Face PAD using Vision Transformer through transformed input spaces

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science

By

G. C. Charan

19BCE0019

Bhavya Harchandani

19BCE2016

Abhinav Dholi

19BCE2362

Under the guidance of

Dr. Gopinath M.P.

School of Computer Science and Engineering (SCOPE)

VIT, Vellore.

May, 2023

DECLARATION

I hereby declare that the thesis entitled “Face PAD using Vision Transformer Model through transformed input spaces” submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Gopinath M.P.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 20/5/2023

G. C. Charan [illegible]

Bhavya Harchandani [illegible]

Abhinav Dholi [illegible]

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled "Face PAD using Vision Transformer through transformed input spaces" submitted by G. C. Charan-19BCE0019, Bhavya Harchandani-19BCE2016, Abhinav Dholi-19BCE2362, School of Computer Science and Engineering (SCOPE), VIT, for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a record of bonafide work carried out by him / her under my supervision during the period, 01. 07. 2022 to 30.04.2023, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

[illegible]

Place : Vellore

Date : 20/5/2023

Signature of the Guide

Internal Examiner **External Examiner**

Dr. S. Vairamuthu

HOD Department of Software Systems

SCOPE School

Acknowledgement 7
Executive Summary 8
Table of Contents 9
List of Figures 9
List of Tables 10
Abbreviations 11
Symbols and Notations 13
1. Introduction 14
1.1. Theoretical Background 14
1.2. Motivation 15
1.3. Aim of the proposed work 16
1.4. Objective(s) of the proposed work 16
2. Literature Survey 17
2.1. Survey of the existing models/work 17
2.1.1. Conventional Method 17
2.1.2. Deep Learning based Techniques 18
2.1.3. Baseline Methods 19
2.1.3.1. LBP-SVM 19
2.1.3.2. DeepPixBis 20
2.1.3.3. IQM-SVM 20
2.2. Gaps identified in the survey 20
3. Overview of the proposed system 21
3.1. Introduction and Related concepts 21
3.1.1. LBP 21
3.1.2. Vision Transformer (ViT) 23
3.1.3. Data Efficient Image Transformer (DeiT) 26
3.2. Framework and Architecture 28
3.2.1. Datasets Used 28
3.2.1.1. Oulu-NPU 28
3.2.2. Preprocessing 31
3.2.2.1. Frame Extraction 32
3.2.2.2. LBP Texture Extractions 34
3.2.3. Proposed System Model and Implementation 35
4. Proposed System Analysis and Design 40

4.1. Introduction 40
4.2. Requirement Analysis 40
4.2.1. Functional Requirements 40
4.2.1.1. Product Perspective 40
4.2.1.2. Product Features 41
4.2.1.3. User Characteristics 42
4.2.1.4. Assumptions and Dependencies 42
4.2.1.5. Domain Requirements 43
4.2.1.6. User Requirements 43
4.2.2. Non-Functional Requirements 44
4.2.2.1. Product Requirements 44
4.2.2.1.1. Efficiency 44
4.2.2.1.2. Reliability 44
4.2.2.1.3. Portability 44
4.2.2.1.4. Usability 44
4.2.2.2. Organizational Requirements 44
4.2.2.2.1. Implementation Requirements 44
4.2.2.2.2. Engineering Standard Requirements 44
4.2.2.3. Operational Requirements 45
4.2.2.3.1. Economic 45
4.2.2.3.2. Ethical 45
4.2.2.3.3. Legality 45
4.2.2.3.4. Inspectability 45
4.2.3. System Requirements 45
4.2.3.1. Hardware Requirements 45
4.2.3.2. Software Requirements 45
5. Project Demonstration 45
5.1. Face Extraction Code 45
5.2. Texture Filter Extraction 49
5.3. Training the DeiT Vision Transformer 59
5.4. Testing Single Image Implementation 85
6. Results and Discussion 88
6.1. Metrics 95
6.2. Comparison with baseline methods 96

6.3. Conclusion 97

6.4. Future Work 97

7. References 97

Acknowledgement

The project “Face PAD using Vision Transformers through Transformed Input Spaces” was made possible because of inestimable inputs from everyone involved, directly or indirectly. We would first like to thank our guide, Dr. Gopinath M.P., who was highly instrumental in providing not only a required and innovative base for the project but also crucial and constructive inputs that helped make our final product. Our guide has helped us perform research in the specified area and improve our understanding in the area of Deep Learning and Computer Vison Applications and we are very thankful for his support all throughout the project.

We would also like to acknowledge the role of the HOD, Associate Prof. Vairamuthu S, who was instrumental in keeping us updated with all necessary formalities and posting all the required formats and document templates through the mail, which we were glad to have had.

It would be no exaggeration to say that the Dean of SCOPE, Prof. Ramesh Babu K, was always available to clarify any queries and clear the doubts we had during the course of our project.

Finally, we would like to thank the Vellore Institute of Technology, for providing us with a flexible choice and execution of the project and for supporting our research and execution related to the project.

Executive Summary

Facial Recognition is used to verify an identity in biometric systems since they are one of the most distinguishing characteristics. Due to the advancements in social media and networks, photos and videos of a person are easily accessible and the same can be used to cheat a facial recognition system to gain unauthorized access. These attacks are commonly called face spoofing attacks and the most common types are face photo attacks, replay attacks, and 3D mask attacks. We propose the use of vision transformers, a recently developed type of deep learning model for detecting presentation attacks (PADs) that are to deceive the system by presenting fake or manipulated facial images, such as through print or display attacks. In the proposed methodology, we extract LBP texture filters of the facial data for different color spaces and train a model with the DeiT Vision Transformer (ViT), to classify PADs. We provide a detailed analysis of how the proposed model architecture performs when trained with the LBP texture filters of the different color spaces and find the best-performing combination for it. The experiments were performed on the OuluNPU Dataset, and the results were compared to several state-of-the-art baseline methodologies. Our proposed method went on to provide results comparable to these baseline methods. Our method outperformed all the baseline methods for the fourth protocol of the OuluNPU dataset.

Index Terms: Vision Transformer, Spoofing, Deep Learning, CNN, Security, Face, Biometric Systems, PAD, DeiT, RGB, LBP, HSV, YCrCb, Grayscale, LA*B*

Table of Contents

List of Figures

S. No	Image Caption	Page No
1	LBP-SVM for Face PAD	19
2	System Flow	21
3	LBP Algorithm	22
4	LBP-RGB Process Diagram	23
5	Vision Transformer Architecture	24
6	Distillation Procedure	27
7	OULU- Sample labelled images	32
8	Use Case Diagram	35
9	Sequence Diagram	36
10	Model Architecture	39
11	Face Extraction Implementation	49
12	LBP Texture Filter Extraction (RGB)	58
13	Vision Transformer Training	84
14	Single Image Testing	88
15	Testing Results	88
16	Accuracy and Loss for Protocol 1	89
17	ROC Curve for Protocol 1	89
18	Accuracy and Loss for Protocol 2	90
19	ROC Curve for Protocol 2	90
20	Accuracy and Loss for Protocol 3	91
21	ROC Curve for Protocol 3	91
22	Accuracy and Loss for Protocol 4	92
23	ROC Curve for Protocol 4	92
24	Accuracy and Loss for NUAA Grayscale-LBP	94

25	ROC Curve for NUAA Grayscale-LBP	94

List of Tables

S.No.	Table Caption	Page No.
1	OULU NPU Dataset Structure	28
2	OULU NPU Protocols	29
3	NUAA Dataset Structure	31
4	OULU Dataset undersampled	33
5	Test Accuracies for OuluNPU	92
6	Dev and Test accuracies on NUAA Grayscale LBP	94
7	Error Metrics for OuluNPU	95
8	Comparison of the proposed method with baseline methods	96

Abbreviations

Abbreviation	**Full Form**
PAD	Presentation Attack Detection
ViT	Vision Transformer
DeiT	Data Efficient Image Transformers
LBP	Local Binary Patterns
CNN	Convolutional Neural Network
RGB	Red Green Blue
HSV	Hue Saturation View
YCrCb	Green (Y), Blue (Cb), Red (Cr) (digtal video color space)
CIELAB or LA*B*	Commission on Illumination, L* for perceptual lightness and a* and b* for the four unique colors of human vision: magenta, green, blue and yellow
FR	Face Recognition
PAI	Presentation Attack Instrument
NLP	Natural Language Processing
CoALBP	Coaxial Local Binary Patterns
LDA	Linear Discriminant Analysis
DoG	Difference-of-Gaussian
VGGNet	Visual Geometry Group Neural Network
ResNet	Residual Neural Network
LBQ	Local Phase Quantization
FCN	Fully Convolutional Network
LSTM	Long short-term memory
GANs	Generative adversarial networks
RNN	Recurrent Neural Network
SVM	Support Vector Machine
RBF	Radial Basis Function
LBPH	Local Binary Pattern Histogram
MSA	Multihead Self Attention
MLP	Multi Layer Perceptron
GELU	Gaussian Error Linear Unit
KL	Kullback–Leibler divergence loss
LOCO	Leave One Camera Out
APCER	Attack Presentation Classification Error Rate
BPCER	Bonafide Presentation Classification Error Rate
ACER	Average Classification Error Rate
EER	Equal Error Rate
FRR	False Rejection Rate

FAR	False Acceptance Rate
ROC curve	Receiver Operating Characteristic curve
XGBoost	Extreme Gradient Boosting

Symbols and Notations

Symbols and Notations	**Meaning**
$\bigotimes$	Corresponding Point Multiplication
$LBP_{16,2}^{u2}$	LBP Texture Filter with radius 2 and 16 neighbors
$LBP_{8,1}^{u2}$	LBP Texture Filter with radius 1 and 8 neighbors
$LBP_{8,2}^{u2}$	LBP Texture Filter with radius 2 and 8 neighbors
L_{global}	Distillation objective
$L_{global}^{hardDistill}$	Hard-label distillation
L_{CE}	Cross entropy
τ	Temperature for distillation
ψ	Softmax function
λ	Coefficient balancing the Kullback–Leibler divergence loss (KL) and the cross-entropy on ground truth labels

1. Introduction

1.1.Theoretical Background

Faces are one of the most distinguishing characteristics to identify a person, in addition to fingerprints and irises. In this digitally advancing world, intelligent identity authentication mechanisms such as face and fingerprint recognition systems are being used instead of traditional passwords as they are more convenient and secure to use in numerous fields. Large-scale face biometrics-based authentication systems are becoming more common; nowadays, even a low-budget mobile phone has a good-quality face unlock system as an alternative to usual passcodes [4]. Face recognition is also used in education (for tracking attendance), immigration (for border control, particularly when it involves criminals and persons of interest who attempt to cross the border), access control, automobile security, and many such commercial areas. As face recognition is becoming more popular, security threats are also becoming stronger. Due to the advancements in social media and networks, photos and videos of a person are easily accessible and the same can be used to cheat a facial recognition system to gain unauthorized access [4].

An unprotected face recognition (FR) system could be fooled by simply placing artifacts in front of the camera, such as a photograph or video. A presentation attack instrument (PAI) is the artifact used in such an attack. [1]

Presentation attack detection (PAD) systems, as the name implies, are designed to safeguard FR systems from such malicious attempts. Though a wide range of presentation attacks are possible, the majority of research efforts have concentrated on the detection of 2D attacks such as prints and replays, owing to the ease with which such attack instruments can be produced. The majority of PAD research focuses on detecting these attacks using only the RGB spectrum, either using feature-based methods or Convolutional Neural Network (CNN)-based approaches. Several feature-based methods for performing PAD have been proposed over the years, using color, texture, motion, liveliness cues, histogram features [2], local binary pattern [3], [5], and motion patterns [6].

The development of deep learning in the field of computer vision and image classification provided more effective solutions for face spoof detection systems. Convolutional Neural Networks (CNNs) have been widely used in the field of face spoof detection. CNNs that include auxiliary information in the form of binary or depth supervision, in particular, have shown to significantly increase performance [7, 8]. Nevertheless, the maximum of these approaches performs effectively only against 2D attacks, and their efficacy decreases when tested against advanced 3D and partial attacks [10]. Even when dealing with 2D attacks, these models frequently fail to generalize to previously unseen attacks and environments.

1.2.Motivation

The history of ViT can be traced back to the Transformer architecture proposed in the seminal paper [37]. This research revolutionized the field of natural language processing (NLP) by introducing a new type of neural network architecture based on self-attention mechanisms. In 2020, [15] proposed the Vision Transformer (ViT) model, that applied the Transformer architecture to computer vision applications. ViT has various advantages over traditional CNN models. First, it is more efficient than CNNs in terms of computation and memory utilization because it does not rely on expensive convolutional processes. Second, ViT can be trained on large-scale datasets from end-to-end without the requirement for hand-crafted features or pre-training on auxiliary tasks. Third, ViT can learn global relationships between image patches and capture long-range dependencies that CNNs cannot model.

Local Binary Pattern (LBP) is a texture descriptor that has been used in presentation attack detection (PAD) systems [33]. According to surveys on face PAD mechanisms, LBP frequently serves as image descriptors in numerous detection mechanisms as input to a classifier for face PADs [34].

We are using the pre-trained DeiT (Data-Efficient Image Transformers) Vision Transformer model, proposed by [16], which is a more efficiently trained transformer model which requires a lesser amount of data as well as lesser computing resources than the original ViT model, to perform our analysis. We are motivated to provide a complete analysis of the performance of the DeiT ViT model in detecting PADs when trained on different color spaces of LBP.

1.3.Aim of the proposed work

The primary aim of this capstone project is to develop a robust methodology for detecting face spoofing attacks in facial recognition systems using the recently developed vision transformer model. The proposed method aims to provide a reliable and accurate detection mechanism for presentation attacks that are designed to deceive the system by presenting manipulated facial images, such as through print or display attacks. The proposed work also aims to provide a comparative study of different configurations, the model architecture can be used to provide the best results with a detailed analysis of how the architecture performs when the model is trained on different kinds of texture filters to deduce the best-performing configuration for the architecture proposed. Overall, this project provides an innovative approach for detecting face spoofing attacks in facial recognition systems by leveraging the power of deep learning and vision transformer models. The outcomes of this project have significant implications for enhancing the security of biometric systems in various domains, including access control, surveillance, and identity verification.

1.4.Objective(s) of the proposed work

The objective of this capstone project report is to propose and evaluate a novel approach for detecting presentation attacks in facial recognition systems using vision transformers[15] and LBP texture filters[35]. The project's primary objective is to investigate the efficacy of utilizing LBP texture filters for different color spaces and train a model with the DeiT Vision Transformer (ViT)[16] to classify PADs. The proposed method's performance will be assessed using the OuluNPU[38] and NUAA photograph imposter database[58], are commonly used benchmark datasets for face spoofing detection. Furthermore, the study intends to compare the suggested method's performance with several state-of-the-art baseline methods, such as traditional machine learning algorithms and deep learning models, in order to demonstrate the efficacy and superiority of the proposed approach. Overall, the project's purpose is to detect and prevent face spoofing attacks in order to contribute to the development of more secure and robust facial recognition systems.

2. Literature Survey

2.1. Survey of the existing models/work

A thorough literature review was conducted in the domain of face detection. According to the authors[8], biometric systems can be attacked both directly and indirectly. The attackers can use photographs, gelatin fingers, contact lenses, etc., to generate synthetic samples of biometric traits to perform direct attacks (spoofing attacks) through which access to authentication systems can be acquired. In the case of indirect attacks, they obtain information about the internal functionality of the system. They modify the algorithms used to protect biometric templates even further. Because of its low acquisition cost and universality, the face is a promising biometric authentication trait. However, various methods can be used to fool face recognition systems[9].

2.1.1. Conventional Method

A lot of work is being done in the field of spoofing detection in faces employing fixed features. These features could be based on motion, texture, reflectance properties, etc. They use handcrafted features to differentiate between real and spoofed faces. Before training the data with any algorithm, the features are calculated. Techniques like Haar-like features and Linear Discriminant Analysis (LDA) can be used to scan faces in photos and identify spoofing attacks[25]. The co-occurrence of adjacent local binary pattern (CoALBP) technique was proposed in [26] to extract facial texture features to detect spoofed faces. The authors[27] investigated textural differences to detect spoofing by examining the Fourier spectra of 2D and 3D photos. They discovered that the surface reflection characteristic caused a change in the frequency distributions of these images. To distinguish between real and spoofed faces, the authors[28] retrieved hidden facial texture features using Difference-of-Gaussian (DoG) filters. The authors[29] propose an approach for extracting time-spectral descriptors from a video, capturing both spatial and temporal information from a biometric sample. These descriptors have been demonstrated to be effective in detecting various types of attacks in a wide range of environments. Researchers in the field of spoofing detection have presented various motion-based approaches. This involves detecting motions such as eye movements [30]. Other approaches involve evaluating image quality and the property of reflection to distinguish between real and fake faces. To identify spoofing, authors use reflection, color diversity, and blurriness between real and fake faces.

2.1.2. Deep Learning based Techniques

These days, the most widely used approach to deal with spoofing is the use of Convolutional Neural Networks (CNN). CNNs have been used for face presentation attack detection (PAD) in recent years. The use of CNNs for PAD has been shown to be effective in detecting various types of presentation attacks, including printed photos, replay attacks, and 3D masks [18]. One of the most common methodologies for CNN-based PAD is to use a binary classification approach, where the input image is classified as either real or fake [17]. Another methodology is to use a multi-class classification approach, where the input image is classified into one of several classes, such as real, printed photo, or replay attack [17]. In general, CNN-based PAD systems consist of two main components: feature extraction and classification [17]. The feature extraction component extracts features from the input image that are relevant for PAD. The classification component then uses these features to classify the input image as either real or fake. Some of the most used CNN architectures for PAD include VGGNet [19], ResNet [19], and MobileNet [19]. These architectures have been shown to be effective in detecting presentation attacks in various scenarios. However, the research is in progress, and little literature is available that studies the detection of face spoofing using deep learning. One of the approaches discussed in the previous literature relied on dimensionality reduction and feature extraction of input frames using pre-trained weights of convolutional autoencoders, followed by classification using the SoftMax classifier [21]. Authors have also used combinations of texture and illumination features with CNNs, specifically by generating the differences of Gaussian (DoG) and low pass filters which were used to extract texture features and specular reflection features respectively then used to train a CNN with those features and original images by stacking them as multi-channel input [24]. Another approach uses a Deep channel that contains multiple layers of CNNs followed by a pooling layer and a fully connected layer. A Shallow feature unit extracts color texture features, such as LBP, CoALBP, and LBQ, from grey-scale, HSV, and YCbCr color spaces. SiW, ROSE-Youtu, and NUAA Imposter datasets were used by the authors [23]. In another literature, the authors suggested dividing the method into two streams: patch-based CNN and depth-based CNN. They trained a deep neural network end-to-end to learn rich appearance attributes for the patch-based CNN stream, and a fully convolutional network (FCN) to estimate the depth of a face picture for the depth-based CNN stream, then merged both the streams [7]. Another approach employs a pre-trained deep residual network to acquire highly discriminative features, which it then combines with Long Short-Term Memory (LSTM) units to uncover long-range temporal associations of video frames for classification on Replay attack and CASIAFASD datasets [22]. Another deep learning-based method for face PAD is Generative Adversarial Networks (GANs). GANs have been used to generate synthetic face images that can be used to train a classifier for detecting presentation attacks [20]. There are many other deep learning-based methods for face PAD such as Capsule Networks, Recurrent Neural Networks (RNNs), and Autoencoders.

2.1.3. Baseline Methods

2.1.3.1. LBP-SVM

This is the most widely used in face analysis-related problems, such as face recognition, face detection, and facial expression recognition. It has several advantages, including a certain robustness toward illumination variations. This strategy is depicted in Figure 1. On a normalized 64x64 image, three different LBPs were applied: LBPu2 8,2, a uniform circular LBP extracted from an 8-pixel neighborhood with a 2-pixel radius, LBPu216,2, a uniform circular LBP extracted from a 16-pixel neighborhood with a 2-pixel radius, and LBPu2 8,1, a uniform circular LBP extracted from an 8-pixel neighborhood with a 1-pixel radius. Finally, a concatenation of all generated histograms forms an 833-bin/dimension histogram. This histogram is then used as a global micro-texture feature and input into a non-linear (RBF) SVM classifier for PAD.

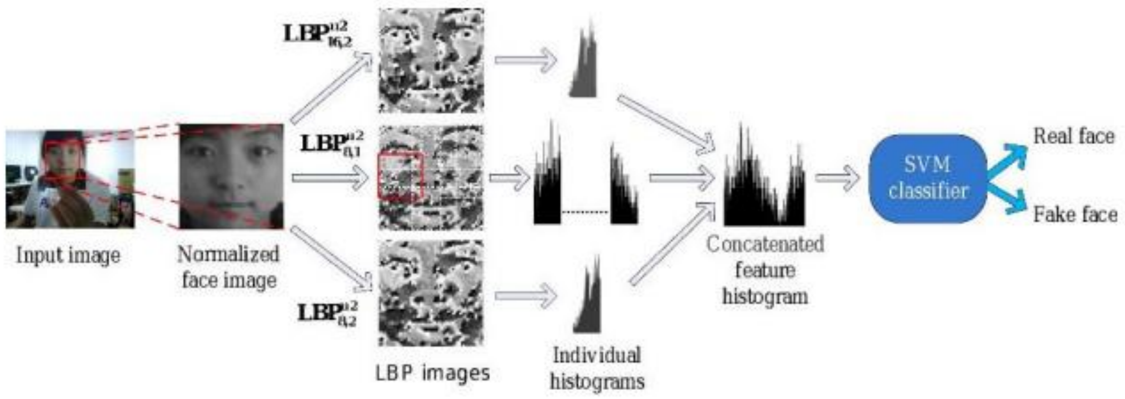


Figure 1: LBP-SVM for Face PAD [42]

2.1.3.2. DeepPixBis

A frame level CNN based framework which does not require temporal features for identifying presentation attacks is proposed in this section. The proposed framework uses a densely connected neural network trained using both binary and pixelwise binary supervision (DeepPixBiS). The framework uses deep pixel-wise supervision, which means that it learns to classify each pixel of the input image as real or fake, based on ground truth masks. The framework uses only frame level information, which means that it does not need any temporal or spatial information from video sequences, making it suitable for deployment in smart devices with minimal computational and time overhead.

2.1.3.3. IQM-SVM

The IQM-SVM method is a presentation attack detection method that uses image quality metrics (IQMs) and support vector machine (SVM). It is used to detect Deepfake videos by treating them as digital presentation attacks. The method is based on the idea that Deepfake videos have different image quality metrics than real videos, and it uses SVM to classify them as real or fake.

2.2.Gaps identified in the survey

Most of the previous approaches to solve this problem have used CNNs. According to [32], one of the drawbacks of CNNs is that they require large amounts of data to learn everything from scratch. CNNs perform better in low data regimes due to their hard inductive bias. Another source states that CNNs are significantly slower due to operations such as maxpool. If the CNN has several layers, the training process takes a lot of time if the computer lacks a good GPU. Other research articles have also stated that ViT has great success with NLP and is now applied to images [15]. The greater disadvantage of a CNN-based approach for detecting face presentation attacks is its sensitivity to adversarial attacks, in which an attacker can produce minor changes to the input image to deceive the model into misclassifying the input as real. Adversarial attacks have the potential to bypass the model's detection capabilities and impact its performance. This issue was raised in a recent work by [57], which showed that CNN-based face anti-spoofing models are vulnerable to adversarial attacks. They demonstrated that attackers can create imperceptible changes that cause the model to misclassify the input, even though the altered photos are visually similar to the originals. Another drawback of CNN-based techniques is their inadequate generalizability to previously unseen attacks or data. CNN-based models are often trained on a specific dataset and may underperform when applied to a variety of attacks or data, resulting in false positives or false negatives. Previous research [32] demonstrated that CNN-based models trained on one dataset may not perform well on another, emphasizing the need for more robust and generalizable models. The prior methods did not produce a more robust model that could perform in more diverse situations, which was a major research gap in these approaches. Recent work has shown that ViT can achieve comparable or even superior performance on image classification tasks compared to CNNs [31].

3. Overview of the proposed system

3.1. Introduction and Related concepts

The process of classification consists of the following major steps, starting from the videos of the dataset, we extract the face frames uniformly using the LBPH face detection algorithm and then save these frames with proper labeling conventions. Further, we create the LBP texture filter from these frames in multiple color spaces such as RGB-LBP, HSV-LBP, YCbCr-LBP, La*b*-LBP, and LBP-grayscale. Next, we feed these individual LBP color space images in the form of feature vectors to the Vision Transformer model (DeiT) for training and testing. Finally, error metrics such as Accuracy, F1 Score, EER, FAR, FRR, and HTER are computed on the test data to evaluate the robustness of the created model.

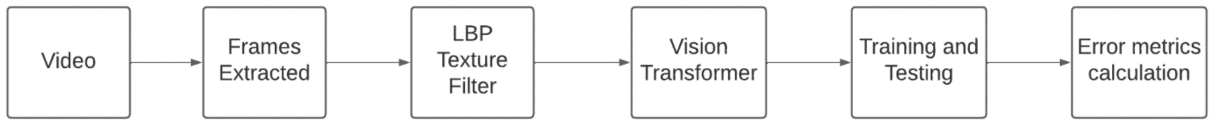


Figure 2: System Flow Diagram

3.1.1. LBP

Local binary patterns (LBP) is a type of visual descriptor used for classification in computer vision. LBP is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.

LBP is initially formed in an eight-pixel neighborhood, with the value of the central pixel serving as a threshold. Any neighbors with values greater than or equal to the center pixel are assigned a value of 1, otherwise, they are set to 0. The values after thresholding (either 0 or 1) would multiply with the corresponding pixel weight, yielding the LBP value as an additive result, which is overwritten to the pixel on which the operation is performed. [35] [Fig 2.]

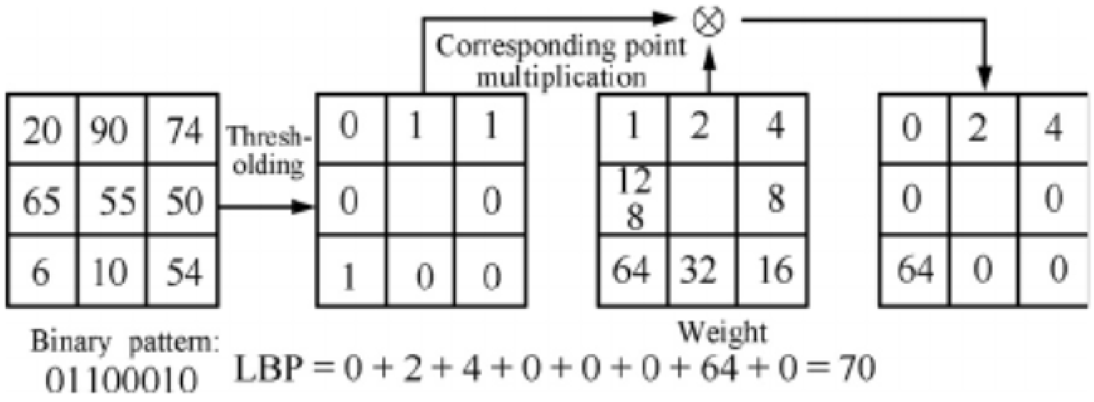


Fig 3: LBP Algorithm [35]

In our methodology, we’ve used LBP on different color spaces, by applying the LBP algorithm separately to each color channel of an image. This results in separate LBP feature maps equivalent to the number of color channels in the provided color space, which can then be combined to create a final feature vector for texture analysis. In Fig 3. LBP algorithm is being performed on RGB color space where, first the R, G, and B channels are separated from the input image, then LBP is applied to those color channels individually then, the transformed color channels are merged into a single LBP transformed RGB image. Further, a similar approach is applied to other color spaces such as HSV, LA*B*, Greyscale, and YCrCb

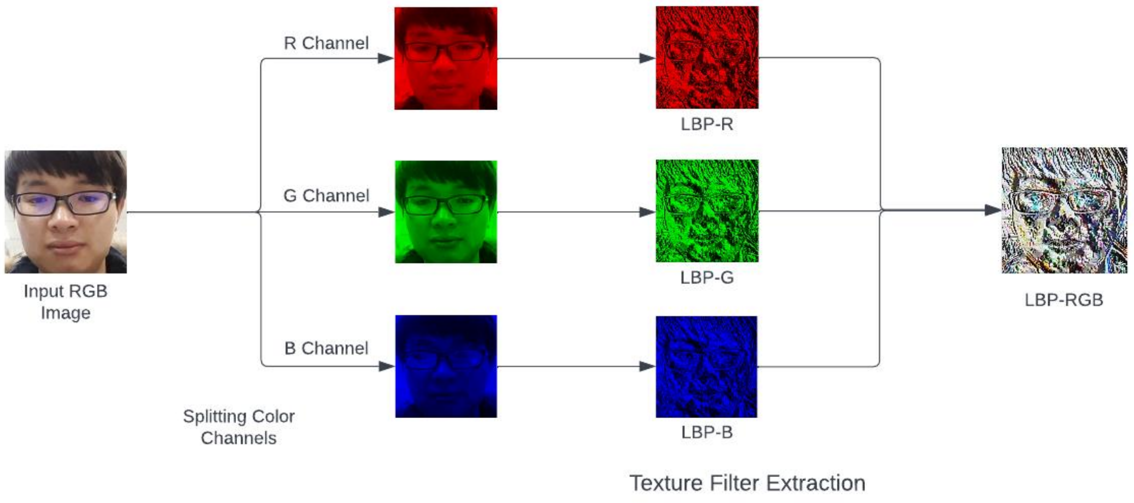


Fig 4: LBP-RGB Process Diagram

3.1.2. Vision Transformer (ViT)

The Vision Transformer, or ViT, is an image classification model that employs a Transformer-like architecture over patches of an image. The image is divided into fixed-size patches, that are then linearly embedded. Position embeddings are then added, and the resulting vector sequence is fed into a standard Transformer encoder. The standard approach of adding an extra learnable "classification token" to the sequence is used to perform classification [15].

ViT can be a revolutionary model in the field of image classification, where previous research has shown that it outperforms, CNN in the terms of accuracy and computational efficiency when trained over a large dataset. Unlike CNNs, ViT uses an attention mechanism to compute the relationship between all pixels in an image. This allows the model to learn global dependencies between different parts of the image. It adds position embeddings to the input image, which encodes the spatial location of each pixel. This allows the model to learn positional relationships between different regions of the image, which is particularly useful for tasks such as image classification [15, 36].

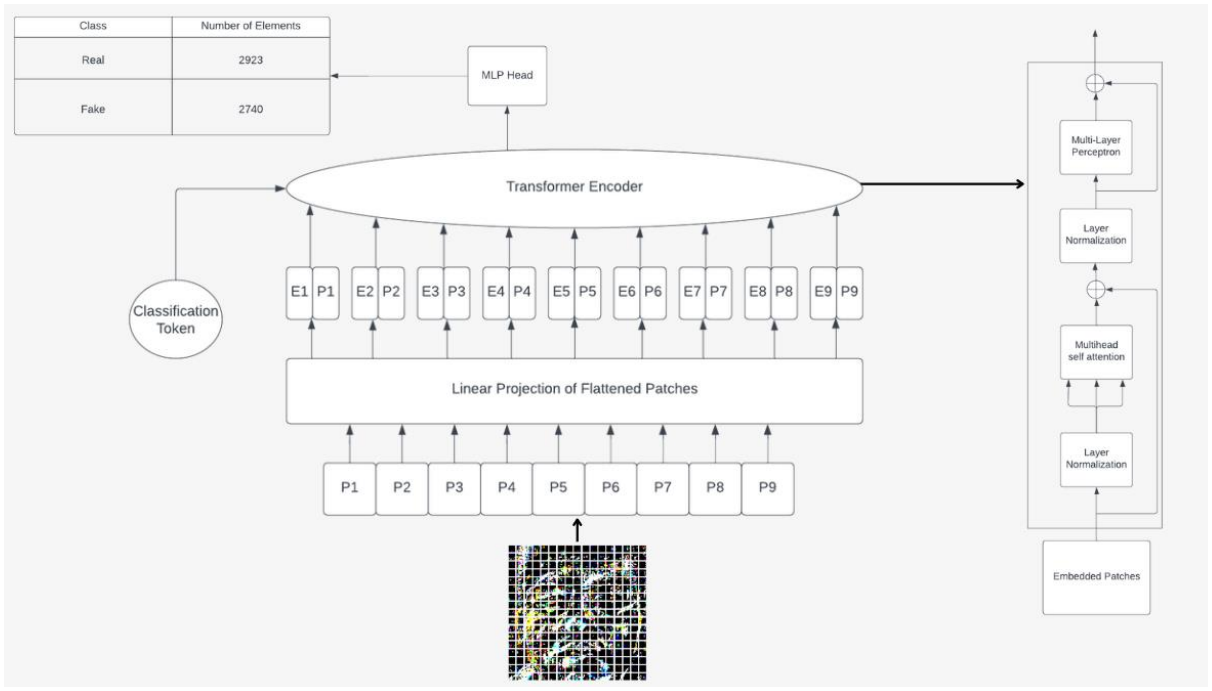


Fig 5: Vision Transformer Architecture

An overview of the model is depicted in Figure 4. A 1D sequence of token embeddings is fed into the standard Transformer. To handle 2D images, the image x in $R^{H \times W \times C}$is reshaped into a sequence of flattened 2D patches , $x_p \in R^{N \times (P^2 \cdot C)}$ where (H, W) is the original image's resolution, C is the number of channels, (P, P) is the resolution of each image patch, and $\mathrm{N} = \frac{HW}{P^2}$ is the resulting number of patches, which also serves as the Transformer's effective input sequence length. Because the Transformer employs a constant latent vector size D across all of its layers, the patches are flattened and map to D dimensions using a trainable linear projection. (Eq. 1).

The last step is adding positional encoding to get the final vector Z. Position embeddings are added to patch embeddings, conventional learnable 1D position embeddings are used [15]. Till this step, "positional encodings" are added to the input embeddings at the bottom of the encoder and decoder stacks. The positional encodings and embeddings have the same dimension d_{model} , therefore the two can be added. There are various learned and fixed positional encodings to choose from [9]. In this work, sine and cosine functions of varying frequencies are used:

$$\mathrm{P}E_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$

$$\mathrm{P}E_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right) \text{ (Eq 1)}$$

Where pos is the position and i is the dimension. That is, each positional encoding dimension corresponds to a sinusoid. The wavelengths are arranged in a geometric sequence from 2π to 10000 · 2π. We selected this function because we expected that it would allow the model to easily learn to attend by relative positions because $\mathrm{P}E_{pos+k}$ can be expressed as a linear function of $\mathrm{P}E_{pos}$ for any fixed offset k [37].

The Transformer Encoder architecture is similar to that described in. It is comprised of several stacks of identical blocks. Each block begins with a Multi-Head Attention layer and ends with a Feed-Forward layer [37].

In the multi-head attention layer, standard qkv self-attention [37] is a key component for the neural architecture. We compute a weighted sum over all values v in an input sequence z ∈ R N×D for each element. The attention weights Aij are deduced by the pairwise similarity of two elements of the sequence and their respective query q i and key k j representations.

$$[\boldsymbol{q}, k, v] = zU_{qkv} \quad U_{qkv} \in R^{D\times 3D_h},$$

$$\mathrm{A} = \mathrm{softmax}\left(qk^{\top}/\sqrt{D_h}\right) \quad A \in R^{N\times N}$$

$$\mathrm{SA}(z) = Av \text{ (Eq 2)}$$

Multihead self-attention (MSA) is an extension of SA in which k self-attention operations, referred to as "heads," are run in parallel and their concatenated outputs are projected. When changing k, D_h (Eq. 3) is typically set to D/k to keep the compute and the number of parameters constant.

$$MSA(z) = [\mathrm{SA}_1(z); \mathrm{SA}_2(z); \cdots; \mathrm{SA}_\mathrm{k}(z)]U_{msa}$$

$$U_{msa} \in R^{k} \cdot D_h \times D \text{ (Eq 3)}$$

To obtain the final vector of embedded dimension D, these attention heads are concatenated and passed through a dense layer [15].

Each of the two sub-layers has a residual connection, which is followed by layer normalization. The model's sub-layers and embedding layers all generate an output of embedded dimension D.

The preceding step's Z vector is passed through the transformer Encoder architecture to produce the context vector C.

Multi-Layer Perceptrons(MLP) contains two-layer with Gaussian Error Linear Unit(GELU). A classification head is implemented for image classification using MLP with one hidden layer at pre-training time and a single linear layer for fine-tuning. ViT's higher layers learn global features, while the lower layers learn both global and local features, which allows ViT to learn more generic patterns [15].

MLP (C Token):

$$z_0 = \left[x_{class}; x_p^1 E; x_p^2 E; \cdots; x_p^N E\right] + E_{pos}, \qquad E \in R^{P^2 \cdot C} \times D, \quad E_{pos} \in R^{N+1} \times D \quad \text{(Eq 4)}$$

$$z'_l = \mathrm{MSA}\left(LN(z_{l-1})\right) + z_{l-1}, \quad l = 1 \cdots L. \text{ (Eq 5)}$$

$$z_l = \mathrm{MLP}\left(LN(z'_l)\right) + z'_l, \quad \mathrm{l} = 1 \cdots L \text{ (Eq 6)}$$

$$y = LN(z_L^0) \text{ (Eq 7)}$$

Once we have our context vector C, we are only interested in the context token for classification purposes. This context token is passed through an MLP head to give us the final probability vector to help predict the class.

3.1.3. Data Efficient Image Transformer (DeiT)

While Vision Transformer, ViT, requires hundreds of millions of images to be pre-trained using external data, it does not generalize well when trained on insufficient amounts of data. [15]. [16] proposes the Data-Efficient Image Transformer, DeiT, as a solution to this problem. While the architecture of DeiT is similar to that of ViT, it is trained on ImageNet in less than 3 days using a single computer and no external data. A teacher-student strategy is introduced with a distillation token [16].

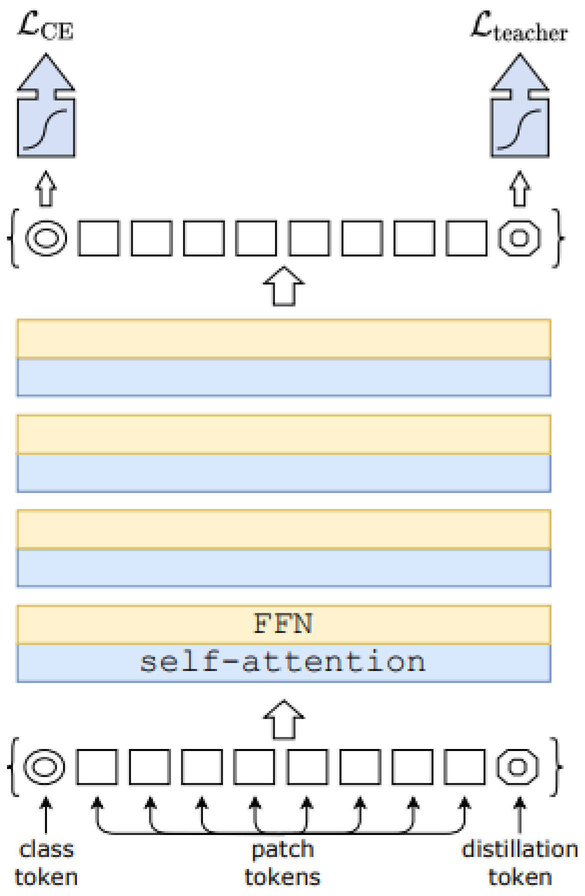


Figure 6: Distillation Procedure [37]

In Figure 5, A new token, the distillation token, is added to the initial embeddings (patches and class token). The distillation token is used similarly as the class token: it interacts with other embeddings through self-attention and is output by the network after the last layer. Its target objective is given by the distillation component of the loss. The distillation embedding allows the model to learn from the output of the teacher, as in a regular distillation, while remaining complementary to the class embedding. Two kinds of distillation equations have been proposed, soft distillation (Eq. 8) and hard label distillation (Eq. 9) [16].

Let Zt be the logits of the teacher model, Zs the logits of the student model. τ is the temperature for the distillation, λ the coefficient balancing the Kullback–Leibler divergence loss (KL) and the cross-entropy on ground truth labels y, and ψ the softmax function. The distillation objective is (Eq. 8)

A variant of distillation has been introduced where the hard decision of the teacher was taken as a true label. Let $y_t = argmax_c Z_t(c)$ be the hard decision of the teacher, the objective associated with this hard-label distillation is: (Eq. 9)

$$\mathcal{L}_{global} = (1-\lambda)\mathcal{L}_{\mathrm{CE}}\left(\psi\left(Z_{\mathrm{s}}\right), y\right) + \lambda\tau^2 \mathrm{KL}\left(\psi\left(Z_{\mathrm{s}}/\tau\right), \psi\left(Z_{\mathrm{t}}/\tau\right)\right) \text{ (Eq 8)}$$

$$\mathcal{L}_{\mathrm{global}}^{\mathrm{hardDistill}} = \frac{1}{2}\mathcal{L}_{\mathrm{CE}}\left(\psi\left(Z_s\right), y\right) + \frac{1}{2}\mathcal{L}_{\mathrm{CE}}\left(\psi\left(Z_s\right), y_{\mathrm{t}}\right) \text{(Eq 9)}$$

3.2.Framework and Architecture

3.2.1. Datasets Used

3.2.1.1. Oulu-NPU

The Oulu-NPU face presentation attack detection database contains 4950 real access and attack videos. The videos were shot in three sessions (Session 1, Session 2 and Session 3) with varying lighting and background environments using the front cameras of six mobile devices (Samsung Galaxy S6 edge, HTC Desire EYE, MEIZU X5, ASUS Zenfone Selfie, Sony XPERIA C5 Ultra Dual, and OPPO N3). Print and video-replay are the presentation attack types assessed in the OULU-NPU database.

	Users	**Real Access**	**Print Attack**	**Video Attack**	**Total**
Training	20	360	720	720	1800
Development	15	270	540	540	1350
Test	20	360	720	720	1800

Table 1: OULU NPU Dataset Structure

For the evaluation of the generalization capability of the face PAD methods, four protocols are used [38].

Protocol I:

The first protocol has been designed to evaluate the generalization of face PAD approaches under previously unseen circumstances, specifically illumination and background environment.

Protocol II:

The second protocol has been designed to evaluate the impact of attacks performed with various printers or displays on the performance of face PAD approaches, which may suffer from new types of artifacts. By introducing a previously unseen print and video-replay attack into the test set, the effect of attack variety is evaluated.

Protocol III:

A Leave One Camera Out (LOCO) procedure is used to investigate the impact of input camera variation. The real and attack videos captured with five smartphones are used to train and tune the algorithms in each iteration, and the generalization of the models is examined using videos recorded with the remaining smartphones.

Protocol IV:

In the final and most difficult protocol, all three factors are considered simultaneously, and the generalization of face PAD methods is evaluated across previously unseen environmental conditions, attacks, and input sensors.

Protocol	**Subset**	**Session**	**Phones**	**Users**	**Attacks Created Using**	**Real videos**	**Attack videos**	**All videos**
	Train	Session 1, 2	6 Phones	1-20	Printer 1, 2; Display 1, 2	240	960	1200
Protocol 1	Dev	Session 1, 2	6 Phones	21-35	Printer 1, 2;	180	720	900

					Display 1, 2			
	Test	Session 3	6 Phones	36-55	Printer 1, 2; Display 1, 2	120	480	600
	Train	Session 1, 2, 3	6 Phones	1-20	Printer 1 ; Display 1	360	720	1080
	Dev	Session 1, 2, 3	6 Phones	21-35	Printer 1 ; Display 1	270	540	810
Protocol 2	Test	Session 1, 2, 3	6 Phones	36-55	Printer 2 ; Display 2	360	720	1080
	Train	Session 1, 2, 3	5 Phones	1-20	Printer 1, 2; Display 1, 2	300	1200	1500
	Dev	Session 1, 2, 3	5 Phones	21-35	Printer 1, 2; Display 1, 2	225	900	1125
Protocol 3	Test	Session 1, 2, 3	1 Phone	36-55	Printer 1, 2; Display 1, 2	60	240	300
	Train	Session 1, 2	5 Phones	1-20	Printer 1; Display 1	200	400	600
Protocol 4	Dev	Session 1, 2	5 Phones	21-35	Printer 1; Display 1	150	300	450

	Test	Session 3	1 Phone	36-55	Printer 2; Display 2	20	40	60

Table 2: OULU NPU Protocols

3.2.1.2. NUAA Photograph Imposter Database

The training and validation samples used in training and validation of the process of model distributed as shown in Figure 2. The dataset contains 12614 photos, including 5105 from the real class and 7509 from the fake class. The printed photo and mobile photo spoof attacks are studied in this research work. Fake and real images from the dataset are used to train and test the model. The data preprocessing applied to the data to deal with missing data from the dataset. Then after the verification of missing data, the next stage was to employ data labeling and encoding for categorical data. This step guarantees that the dependent variable is converted to binary values of 0 and 1. After completing the data preprocessing and label encoding steps, the dataset is separated into two sets. The model was trained with 80% of the data and validated with 20% of the data. [58]

	Training Images	**Development Images**	**Testing Images**	**Total Images**
Real Images	1419	324	3362	5105
Fake Images	1373	375	5761	7509
Total Images	2792	699	9123	12614

Table 3: NUAA Dataset Structure

3.2.2. Preprocessing

In order to prepare the data to feed into the Transformer, firstly the frames with detected faces are extracted from the video clip, and the extracted frames are labeled accordingly. Further, the frames are separated into separate files (train, dev, test), in accordance with the protocols. Next, the training data is balanced according to the classes, by undersampling the number of frames of individual videos, not compromising the variety of the dataset and the protocol. Then, LBP texture filters of all the color spaces mentioned are extracted in order to feed to the transformer for training.

3.2.2.1. Frame Extraction

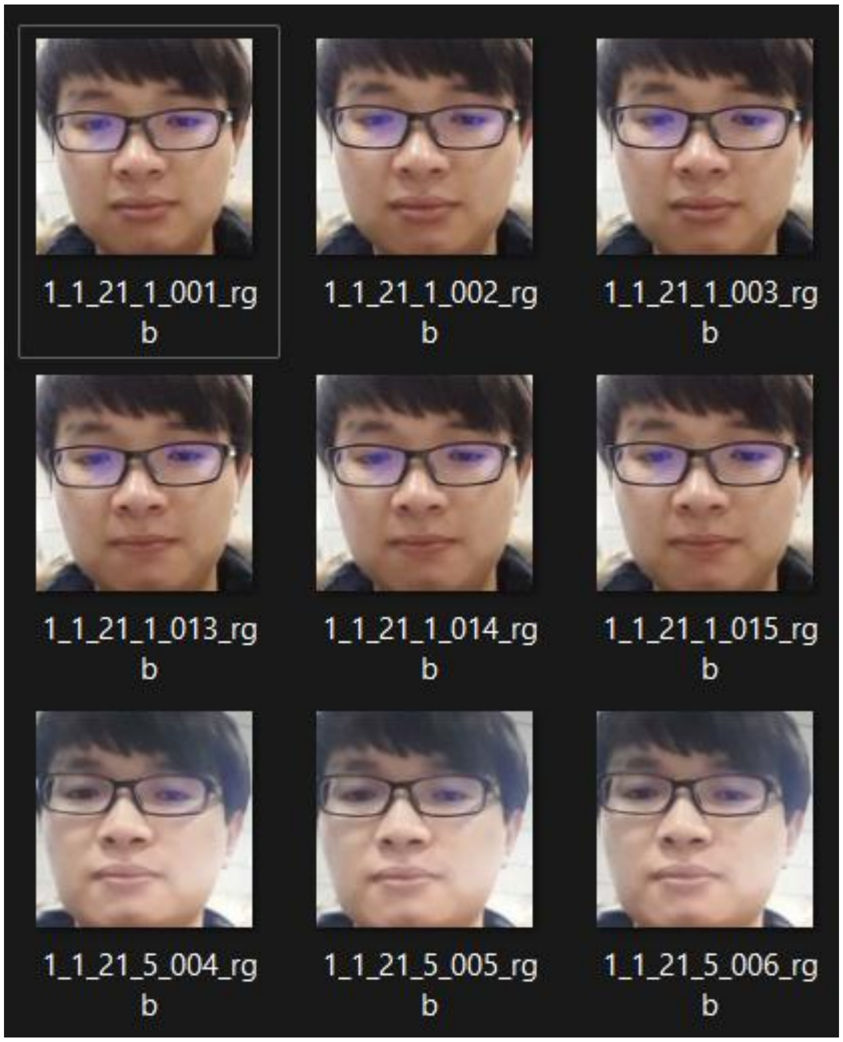


Figure 7: OULU- Sample labelled images

We have used the LBPH Algorithm [39] to detect the faces from the videos, every video is running at 25 fps. Cropped face frames of dimension 128x128 pixels are extracted from the video. Around 10-15 frames are extracted from each video. The labeling convention used for this is as follows '{+1/-1},{folder_number}_{video_name}_{frame_number}_{color_space}'. Here, '+1' denotes the real face, and '-1' denotes the spoofed face. Finally, these frames are separated into the train, dev, and test folders according to the protocols provided.

Undersampling

In most of the cases, we found that the amount of training data for the spoofed image scenario was in higher proportion to the real image scenario, hence making the dataset imbalanced for training it on the transformer. The dataset is undersampled according to the classes, where we undersample the number of frames considered for every spoof scenario video, hence not compromising the protocol provided.

Protocol		**Subset**	**Real Images**	**Fake Images**	**Total Images**
Protocol 1		Train	2923	2740	5663
		Dev	2095	6932	9027
		Test	1579	6179	7758
Protocol 2		Train	4509	4366	8875
		Dev	3271	6046	9317
		Test	4222	7377	11599
Protocol 3	3.1	Train	3625	4294	7919
		Dev	2632	9467	12099
		Test	836	2834	3670
	3.2	Train	3946	4599	8545
		Dev	2882	10126	13008
		Test	547	2089	2636
	3.3	Train	3780	4379	8159
		Dev	2739	9720	12459
		Test	711	2573	3284
	3.4	Train	3710	4506	8216
		Dev	2690	9704	12394
		Test	686	2433	3119
	3.5	Train	3647	4240	7887
		Dev	2638	9283	11921
		Test	833	3091	3924
	3.6	Train	3837	4464	8301
		Dev	2774	9805	12579
		Test	609	2257	2866
Protocol 4	4.1	Train	2339	2536	4865
		Dev	1680	2988	4668
		Test	300	562	862
	4.2	Train	2569	2702	5271

		Dev	1851	3243	5094
		Test	180	433	643
	4.3	Train	2458	2623	5081
		Dev	1755	3149	4904
		Test	261	515	776
	4.4	Train	2402	2583	4985
		Dev	1724	3076	4800
		Test	278	505	783
	4.5	Train	2361	2523	4884
		Dev	1687	2936	4623
		Test	291	585	876
	4.6	Train	2486	2670	5156
		Dev	1778	3068	4846
		Test	239	448	687

Table 4 OULU Dataset Undersampled

3.2.2.2. LBP Texture Extractions

Now, the LBP texture filters of the extracted frames are extracted for different color spaces namely: RGB-LBP, HSV-LBP, LA*B*-LBP, Greyscale-LBP, and YCrCb-LBP. The setup considers the radius of 1 pixel and 8 neighbors in order to extract the filters [35]. Now the dataset is ready to be fed into the transformer for training. For detailed procedure, refer to section 3.1, LBP.

Data Augmentation

The data is augmented with the following parameters before training:

These include random horizontal and vertical flips of images with a probability of 0.5 each for increased data variability during training. Then the input image is resized to have a height and width of 256 pixels. Further, the image is center cropped with a dimension of 224 x 224 and converted to a tensor. The pixel values of the tensor image are then normalized using the ImageNet dataset mean and standard deviation values (mean = (0.485, 0.456, 0.406), std = (0.229, 0.224, 0.225)) in order to provide better model performance.

3.2.3. Proposed System Model and Implementation

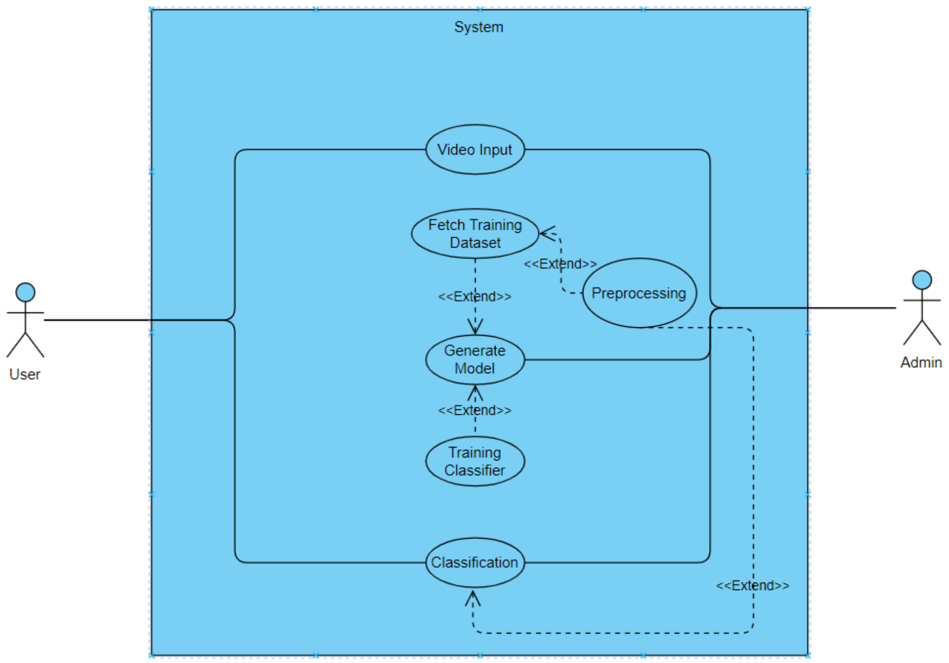


Figure 8: Use Case Diagram

The use case diagram in Figure 7 is made for a possible use case scenario for the framework we have proposed. We have considered two major actors for the complete framework, the first one being an admin, who is mostly the security personnel and the second one is the user, whose video is being classified. Here the admin is responsible for preparing the model and training the classifier. The user is supposed to upload their video through any video input device such as a security camera or a mobile phone camera in certain use cases. The classifier predicts if the video input is spoofed or not.

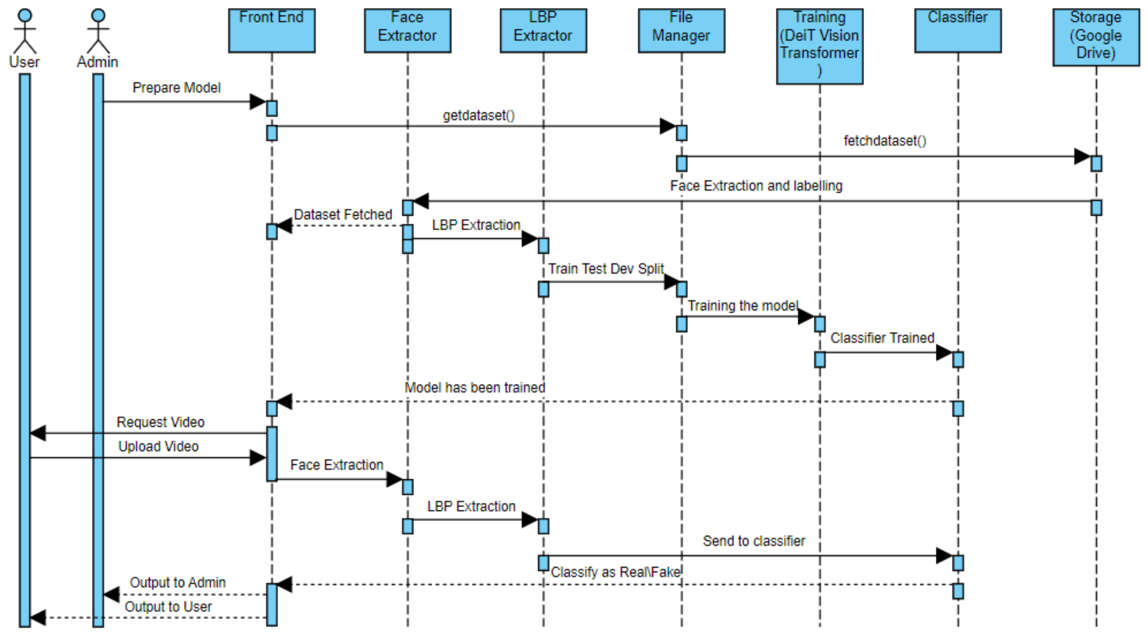


Figure 9: Sequence Diagram

The sequence diagram in Figure 8 depicts the overall workflow of the framework proposed internally as well as how the system would react to the user and the admin throughout. The framework proposed comprises of the following two major processes. First, the admin prepares the classifier by training the model. Here, internally the complete framework is automated to fetch the most updated dataset from the google drive or any storage service is fetched, and the model is retrained. Second, the front-end application asks the user to upload their video, which in the most suitable case would be from a security camera. The system classifies the video fetched from the user as real or spoof and hence, intimates the admin and the user if they've gained access or not.

Model Architecture

We are using the "deit_base_patch16_224" model which refers to a specific variant of the DeiT architecture, where "patch16" indicates the size of the input image patches that the model processes. In the DeiT architecture, input images are divided into non-overlapping patches of size 16x16 pixels, and each patch is then linearly embedded into a 1D vector before being processed by the Transformer layers, and "224" refers to the input image resolution, here, the model is designed to take input images of size 224x224 pixels [16].

While making the observations, the model was trained on the DeiT Transformer for 50 epochs with a learning rate of 0.03. Stochastic gradient descent (SGD) was used as an optimizer for the complete training process, at the momentum of 0.5.

The augmented image is then passed through the three main components of the architecture in the following sequence:

- Pretrained Transformer Layers: This component is in control of the architecture's primary duty, picture classification. It is composed of a series of transformer layers that have been pretrained on a huge dataset of images. These layers extract features from the input image and encode them into a set of high-level representations that capture the semantic meaning of the image.
- Custom Head: This component is added on top of the pretrained transformer layers and is accountable for transforming the output of the transformer layers into a format that can be used for classification. The head is made up of numerous layers described in a Sequential container that are applied to the input features in a sequential order. The following is a breakdown of the several layers employed in the head:

1. BatchNorm1d(768): Batch normalization layer that normalizes the input along the batch dimension. It takes an input size of 768, which is the number of input features coming from the backbone.
2. Linear(n_inputs, 512): Fully connected layer with n_inputs (input features) as input size and 512 as output size. It performs a linear transformation on the input features.
3. Mish(): Activation function called Mish, which is a variant of the popular activation function called Mish (Mish: A Self Regularized Non-Monotonic Neural Activation Function). It introduces a non-linearity to the model to capture complex patterns in the data.
4. Dropout(0.2, inplace=True): Dropout layer that randomly sets a fraction of input elements to 0 during training, with a probability of 0.2. It helps in regularizing the model and preventing overfitting.

5. Linear(512, 256), nn.Linear(256, 128), nn.Linear(128, 64): Additional fully connected layers with decreasing output sizes of 256, 128, and 64 respectively. These layers continue to apply linear transformations to the features with decreasing complexity.
6. Mish(): Mish activation function is used after each of the fully connected layers.
7. nn.Linear(64, len(classes)): Final fully connected layer with an output size equal to the number of classes in the classification task. It produces the logits, which are the raw predictions for each class.

The last fully connected layer produces a set of class predictions that are used to calculate the class label for the input image. The Custom Head is trained to adapt the high-level features extracted by the Transformer Encoder to the specific task at hand. The Custom Head in this case serves to generate class predictions for a specific collection of classes.

Transformer Classifier: This component is responsible for completing the final classification task. It is composed up of a single linear layer that takes the custom head's output as input and provides the final classification scores. This layer's purpose is to map the extracted features to the classes in the output space.

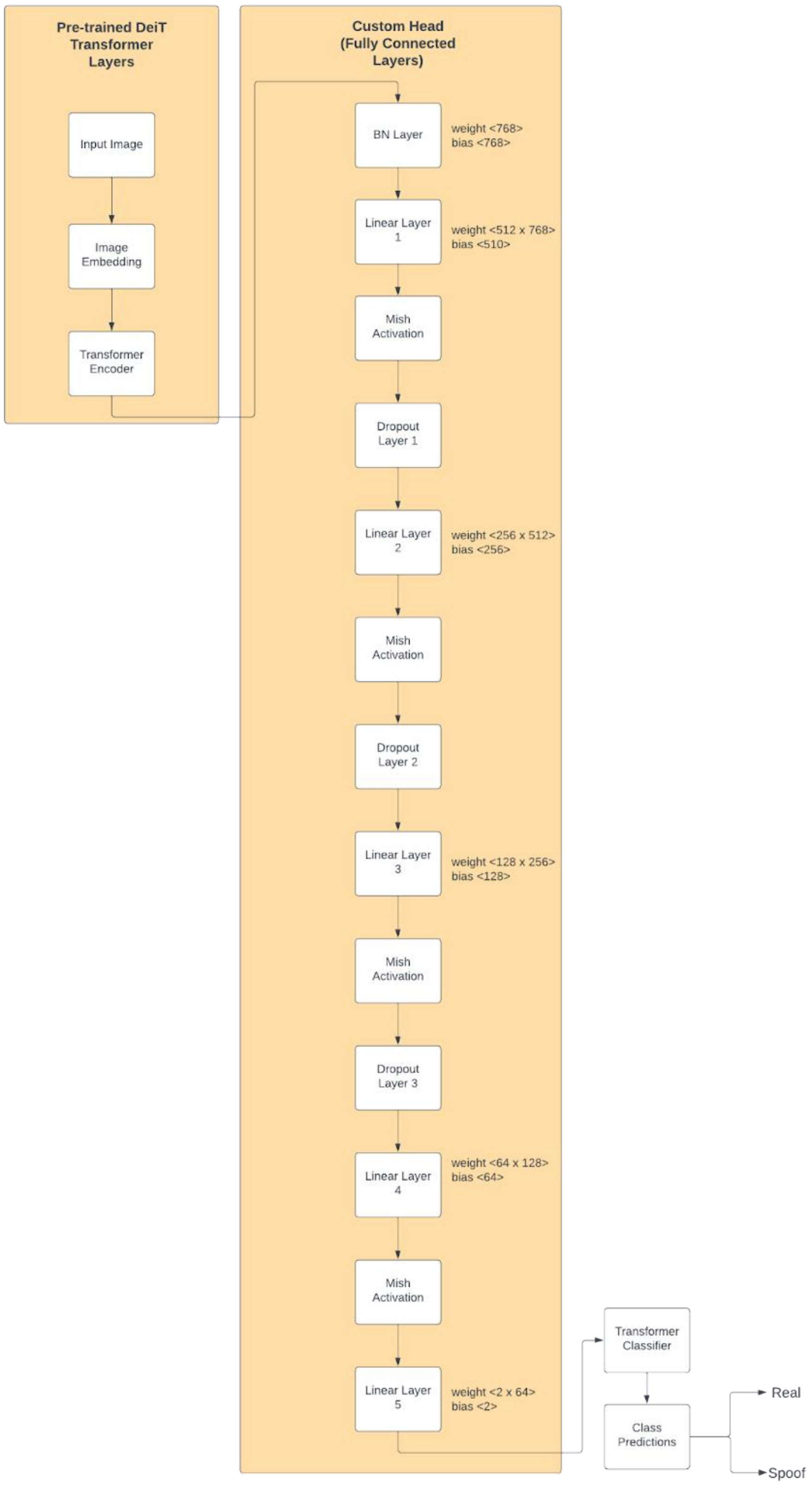


Figure 10: Model Architecture

In summary, the proposed DeiT architecture consists of a pretrained transformer network, a custom head, and a transformer classifier that, when combined, enable high-performance image classification on a variety of datasets. The transformer layers extract features, the custom head transforms features, and the transformer classifier classifies them.

4. Proposed System Analysis and Design

4.1. Introduction

This section describes the analysis and design of vision transformer-based framework for face print attack detection. The proposed system incorporates the extraction of LBP filters in different color spaces and then feeding them to the DeiT model for training and testing. The model is actuated on the face images acquired from the videos of OuluNPU and NUAA Datasets under different environments, parameters, and hardware devices in the form of protocols defined by the datasets which are available in the public domain for research purposes.

4.2. Requirement Analysis

4.2.1. Functional Requirements

4.2.1.1. Product Perspective

The proposed project aims to develop an architecture that can detect face presentation attacks using vision transformer. The system employs a combination of computer vision techniques in which Local Binary Patterns (LBP) of color spaces such as RGB, LAB, YCrCb, HSV, and Grayscale are used as input to the vision transformer to accurately identify real faces and distinguish them from fake ones. The goal of this project is to create a dependable and effective solution for detecting and preventing face spoofing attacks.

The architecture will leverage advanced machine-learning techniques to analyze facial images and detect signs of manipulation or alteration. The system will be able to extract complex elements from images, even those that are not immediately visible to the human eye, by using a vision transformer.

The key objective of this architecture is to detect presentation attacks with high accuracy while minimizing false positives and false negatives. This aids in the prevention of unauthorized access to sensitive systems and data, as well as ensuring that only authorized individuals have access. Furthermore, the solution is easy to use and can be interfaced with existing security systems.

The proposed architecture is developed according to industry standards and best practices and is thoroughly tested to ensure its effectiveness and reliability. The proposed system is a great solution for any organization that values security and data protection since it will provide a robust and scalable solution for tackling face presentation attacks.

Overall, the proposed project is an important addition to the security landscape, adding an additional layer of defense against face spoofing attacks.

4.2.1.2. Product Features

The proposed framework for detecting face presentation attacks employing vision transformer includes a number of powerful features aimed at improving the security and accuracy of facial recognition systems. Among the important aspects are:

1. Multi-color space analysis: To detect face presentation attacks in a wide range of lighting and environmental situations, the system will use a variety of color spaces, including RGB, LAB, YCrCb, HSV, and Grayscale.
2. Local Binary Pattern (LBP) analysis: The system will use LBP analysis to extract facial texture filters and identify patterns that are unique to real faces, enabling the classifier in distinguishing them from fakes.
3. Vision Transformer: The system will employ a cutting-edge vision transformer network that has been pre-trained by Meta, i.e. Data Efficient Image Transformer (DeiT), which is highly effective at identifying patterns in visual data, allowing it to detect subtle differences between real and fake faces while requiring less data to train and fewer computing resources.
4. Comparative Analysis: The proposed methodology is also evaluated in comparison to other state-of-the-art baseline methodologies for identifying face presentation attacks.

4.2.1.3. User Characteristics

The proposed architecture for detecting face presentation attacks using vision transformer is intended for a wide range of users, including security professionals, law enforcement officials, and anyone who needs to authenticate individuals with facial recognition technology.

Some key user characteristics for this product might include:

1. Technical expertise: Users of this product will most likely be familiar with computer vision techniques and machine learning algorithms. They may have prior experience with facial recognition systems and other security technologies, as well as some familiarity with complex software programs.
2. Security expertise: Many users with a background in security or law enforcement will understand the significance of correct identification and will be familiar with a variety of face presentation attacks that can be used to overcome facial recognition systems.
3. Attention to detail: Users of this product must be detail-oriented and able to detect small distinctions between real and fake faces. They may need to analyze photographs or video recordings for extended periods of time, looking for signs of fraud or deception.
4. Collaboration: Depending on the environment in which this product is used, users may need to collaborate actively with others, such as security staff or IT professionals. To achieve their goals, they will need to be able to communicate and coordinate effectively.
5. Flexibility: Users of this product may work in a variety of conditions, from crowded public places to secure facilities. They will need to be able to adapt to changing situations and work effectively in a variety of settings.

Overall, this product's users will be highly competent and experienced professionals committed to upgrading the security and accuracy of their facial recognition systems. They will be familiar with working with complex software applications, have a keen eye for detail, and be able to collaborate well with others.

4.2.1.4. Assumptions and Dependencies

The proposed architecture for detecting face presentation attacks using vision transformer is a complex system that relies on the following key assumptions and dependencies:

1. The success of this project depends on the availability of high-quality datasets for training and testing the vision transformer model. We presume that the Oulu-NPU and NUAA impostor datasets will be used for all experiments in this situation.
2. This project will require access to high-performance computing resources such as GPUs and large amounts of memory. These resources are considered to be available, and the proposed framework will be optimized to make efficient use of them.
3. The proposed architecture is expected to be integrated with existing facial recognition systems in this project. Collaboration with IT professionals and other stakeholders may be required to verify that the system is correctly configured and compatible with existing hardware and software.
4. The proposed architecture is expected to be used in various scenarios, including security checkpoints, access control systems, and other applications requiring accurate identification of individuals. The system must be adaptable enough to different use cases and environments.

4.2.1.5. Domain Requirements

A face presentation attack detection system is supposed to detect distinct types of presentation attacks, such as photo, video, and 3D mask attacks, with high accuracy. Using advanced computer vision techniques and machine learning algorithms, it should be able to detect tiny variations between real and fake faces. The solution should be simple to use and should be compatible with existing facial recognition systems or other security technologies. It should be dependable and accurate, with a low rate of false positives and false negatives. Furthermore, the system should be able to adapt to changing conditions and environments, such as changing lighting and camera angles. Finally, the system should be updated and modified on a regular basis to keep up with new and developing presentation attack approaches.

4.2.1.6. User Requirements

The user of the proposed methodology is supposed to ensure that the complete face of the client should be visible without any obstructions like face masks.

4.2.2. Non-Functional Requirements

4.2.2.1. Product Requirements

4.2.2.1.1. Efficiency

The system should be able to detect face presentation attacks in real time (less than one second) and process a large volume of images efficiently to minimize detection delays.

4.2.2.1.2. Reliability

The system should be able to detect face presentation attacks with high accuracy and handle different types of face presentation attacks, such as print attacks and digital screens. The system should also be robust enough to work in a variety of environments.

4.2.2.1.3. Portability

The system should be able to run on various kinds of hardware configurations, including low-power embedded devices and high-performance servers, and it should also support different operating systems such as Windows, Linux, and macOS.

4.2.2.1.4. Usability

The system should be easy to integrate with other security systems, such as access control systems and surveillance cameras, and it should provide clear and informative error messages in the event of an issue.

4.2.2.2. Organizational Requirements

4.2.2.2.1. Implementation Requirements

To extract the dataset, we used Anaconda Navigator, image processing python libraries along with Jupyter Notebook. To create the texture filters, we used Google Colab Platform and finally to create the model and perform evaluations, we used the Kaggle Notebook environment. We used Google Drive in order to store our datasets.

4.2.2.2.2. Engineering Standard Requirements

Given the domain of our application, no such engineering standards exist.

4.2.2.3. Operational Requirements

4.2.2.3.1. Economic

These include the cost of internet and power supply, cost of renting the computational resources such as the required amount of GPU and RAM for the processing of our models. They also include the cost of resources required to store the data.

4.2.2.3.2. Ethical

The system is developed with ethical factors in mind, such as privacy protection and bias minimization. The system does not violate the freedoms and rights of individuals.

4.2.2.3.3. Legality

Proper licensing for the datasets used to train the model and permissions required from the authorities to gain access to the input face images from the existing security systems .

4.2.2.3.4. Inspectability

The system should provide clear documentation and logs to enable auditing and analysis.

4.2.3. System Requirements

4.2.3.1. Hardware Requirements

● Nvidia P100 or T4 GPU with 16gb GPU Ram

● 32 GB CPU RAM

● Intel(R) Xeon(R) CPU @ 2.20GHz

4.2.3.2. Software Requirements

Anaconda Navigator, Python Libraries, Keras, Scikit Learn and other Machine Learning Libraries, Jupyter Notebook, Kaggle Environment, Google Colab Pro, Google Drive, PyTorch, Matplotlib, NumPy, Pandas, torchvision, oulumetrics and other.

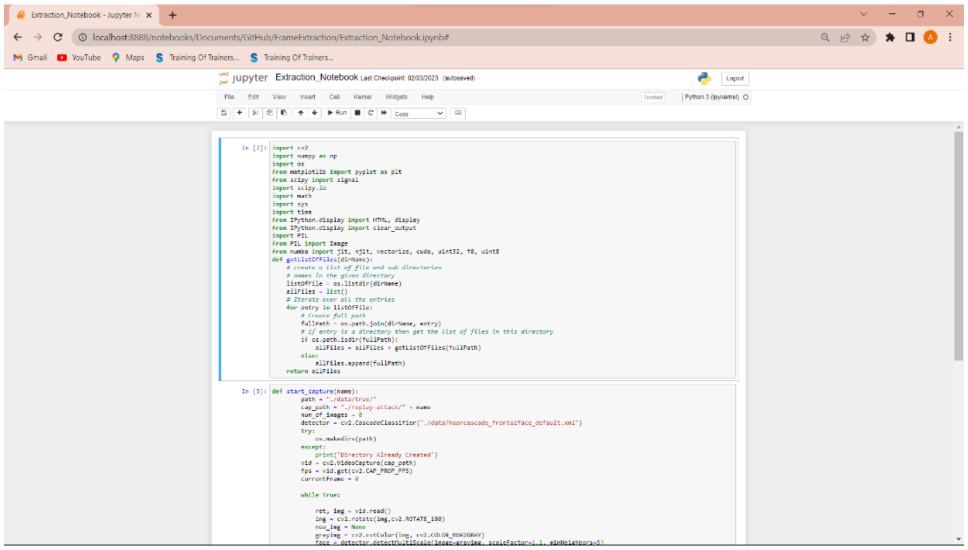
5. Project Demonstration

5.1.Face Extraction Code

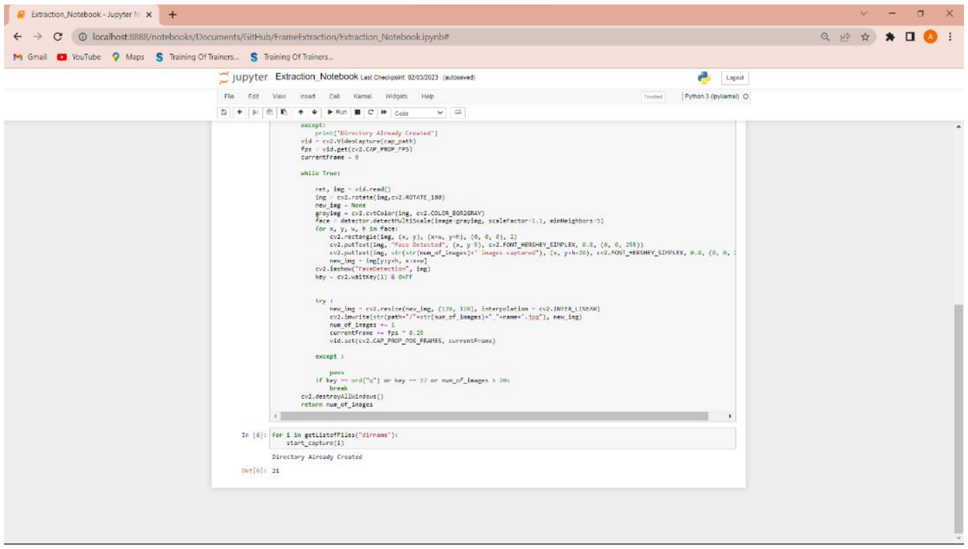
```
#!/usr/bin/env python
```

```
# coding: utf-8

# In[2]:


import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
from scipy import signal
import scipy.io
import math
import sys
import time
from IPython.display import HTML, display
from IPython.display import clear_output
import PIL
from PIL import Image
from numba import jit, njit, vectorize, cuda, uint32, f8, uint8
def getListOfFiles(dirName):
    # create a list of file and sub directories
    # names in the given directory
    listOfFile = os.listdir(dirName)
    allFiles = list()
    # Iterate over all the entries
    for entry in listOfFile:
        # Create full path
        fullPath = os.path.join(dirName, entry)
        # If entry is a directory then get the list of files in this directory
        if os.path.isdir(fullPath):
            allFiles = allFiles + getListOfFiles(fullPath)
        else:
            allFiles.append(fullPath)
    return allFiles


# In[5]:


def start_capture(name):
        path = "./data/true/"
        cap_path = "./replay-attack/" + name
        num_of_images = 0
        detector = cv2.CascadeClassifier("./data/haarcascade_frontalface_default.xml")
        try:
            os.makedirs(path)
        except:
            print('Directory Already Created')
        vid = cv2.VideoCapture(cap_path)
        fps = vid.get(cv2.CAP_PROP_FPS)
        currentFrame = 0

        while True:


            ret, img = vid.read()
            img = cv2.rotate(img,cv2.ROTATE_180)
            new_img = None
            grayimg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            face = detector.detectMultiScale(image=grayimg, scaleFactor=1.1, minNeighbors=5)
            for x, y, w, h in face:
                cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 0), 2)
                cv2.putText(img, "Face Detected", (x, y-5), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0,
255))
                cv2.putText(img, str(str(num_of_images)+" images captured"), (x, y+h+20),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255))
```

```
                new_img = img[y:y+h, x:x+w]
            cv2.imshow("FaceDetection", img)
            key = cv2.waitKey(1) & 0xFF

            try :
                new_img = cv2.resize(new_img, (128, 128), interpolation = cv2.INTER_LINEAR)
                cv2.imwrite(str(path+"/"+str(num_of_images)+"_"+name+".jpg"), new_img)
                num_of_images += 1
                currentFrame += fps * 0.25
                vid.set(cv2.CAP_PROP_POS_FRAMES, currentFrame)

            except :

                pass
            if key == ord("q") or key == 27 or num_of_images > 20:
                break
        cv2.destroyAllWindows()
        return num_of_images

# In[6]:

for i in getListofFiles("dirname"):
    start_capture(i)
```



```
In [2]: import cv2
        import numpy as np
        import os
        from matplotlib import pyplot as plt
        from scipy import signal
        import scipy.io
        import math
        import sys
        import time
        from IPython.display import HTML, display
        from IPython.display import clear_output
        import PIL
        from PIL import Image
        from numba import jit, njit, vectorize, cuda, uint32, f8, uint8
        def getListOfFiles(dirName):
            # create a list of file and sub directories
            # names in the given directory
            listOfFile = os.listdir(dirName)
            allFiles = list()
            # Iterate over all the entries
            for entry in listOfFile:
                # Create full path
                fullPath = os.path.join(dirName, entry)
                # If entry is a directory then get the list of files in this directory
                if os.path.isdir(fullPath):
                    allFiles = allFiles + getListOfFiles(fullPath)
                else:
                    allFiles.append(fullPath)
            return allFiles

In [5]: def start_capture(name):
                path = "./data/true/"
                cap_path = "./replay-attack/" + name
                num_of_images = 0
                detector = cv2.CascadeClassifier("./data/haarcascade_frontalface_default.xml")
                try:
                    os.makedirs(path)
                except:
                    print('Directory Already Created')
                vid = cv2.VideoCapture(cap_path)
                fps = vid.get(cv2.CAP_PROP_FPS)
                currentFrame = 0

                while True:

                    ret, img = vid.read()
                    img = cv2.rotate(img,cv2.ROTATE_180)
                    new_img = None
                    grayimg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                    face = detector.detectMultiScale(image=grayimg, scaleFactor=1.1, minNeighbors=5)
```



```
                except:
                    print('Directory Already Created')
                vid = cv2.VideoCapture(cap_path)
                fps = vid.get(cv2.CAP_PROP_FPS)
                currentFrame = 0

                while True:

                    ret, img = vid.read()
                    img = cv2.rotate(img,cv2.ROTATE_180)
                    new_img = None
                    grayimg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                    face = detector.detectMultiScale(image=grayimg, scaleFactor=1.1, minNeighbors=5)
                    for x, y, w, h in face:
                        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 0), 2)
                        cv2.putText(img, "Face Detected", (x, y-5), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255))
                        cv2.putText(img, str(str(num_of_images)+" images captured"), (x, y+h+20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 
                        new_img = img[y:y+h, x:x+w]
                    cv2.imshow("FaceDetection", img)
                    key = cv2.waitKey(1) & 0xFF


                    try :
                        new_img = cv2.resize(new_img, (128, 128), interpolation = cv2.INTER_LINEAR)
                        cv2.imwrite(str(path+"/"+str(num_of_images)+"_"+name+".jpg"), new_img)
                        num_of_images += 1
                        currentFrame += fps * 0.25
                        vid.set(cv2.CAP_PROP_POS_FRAMES, currentFrame)

                    except :

                        pass
                    if key == ord("q") or key == 27 or num_of_images > 20:
                        break
                cv2.destroyAllWindows()
                return num_of_images

In [6]: for i in getListofFiles("dirname"):
            start_capture(i)

        Directory Already Created

Out[6]: 21
```

Figure 11 Face Extraction Implementation

5.2.Texture Filter Extraction

# -*- coding: utf-8 -*-

"""LBP_RGB.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1JeQmtA-Gzuvw731CRTHU8hGcyeMlhoUM

"""

```
!pip install numba

!find / -iname 'libdevice'

!find / -iname 'libnvvm.so'




#Loading part

from pydrive.auth import GoogleAuth

from pydrive.drive import GoogleDrive

from google.colab import auth

from oauth2client.client import GoogleCredentials

auth.authenticate_user()

gauth = GoogleAuth()

gauth.credentials = GoogleCredentials.get_application_default()

drive = GoogleDrive(gauth)


downloaded = drive.CreateFile({'id': '1nEDHr7mDxV0sfnX83CujlgLIJ9C3WHK1'})   # replace the id
with id of file you want to access

downloaded.GetContentFile('ExtractedRGBOulu_NPU.zip')        # replace the file name with your file

!unzip ExtractedRGBOulu_NPU


import cv2

import numpy as np

import os

from matplotlib import pyplot as plt

from scipy import signal

import scipy.io

import math

import sys

import time
```

```
from IPython.display import HTML, display
from IPython.display import clear_output
import PIL
from PIL import Image
from numba import jit, njit, vectorize, cuda, uint32, f8, uint8
def getListOfFiles(dirName):
    # create a list of file and sub directories
    # names in the given directory
    listOfFile = os.listdir(dirName)
    allFiles = list()
    # Iterate over all the entries
    for entry in listOfFile:
        # Create full path
        fullPath = os.path.join(dirName, entry)
        # If entry is a directory then get the list of files in this directory
        if os.path.isdir(fullPath):
            allFiles = allFiles + getListOfFiles(fullPath)
        else:
            allFiles.append(fullPath)
    return allFiles

# Python program to explain Merging of Channels

# Importing cv2
import cv2
import numpy as np
from matplotlib import pyplot as plt

from numba import int32, float32    # import the types
from numba.experimental import jitclass
```

```
class Human:



    def __init__(self):
        self.img = 0



    def get_pixel(img, center, x, y):

        new_value = 0

        try:
            # If local neighbourhood pixel
            # value is greater than or equal
            # to center pixel values then
            # set it to 1
            if img[x][y] >= center:
                new_value = 1

        except:
            # Exception is required when
            # neighbourhood value of a center
            # pixel value is null i.e. values
            # present at boundaries.
            pass

        return new_value
```

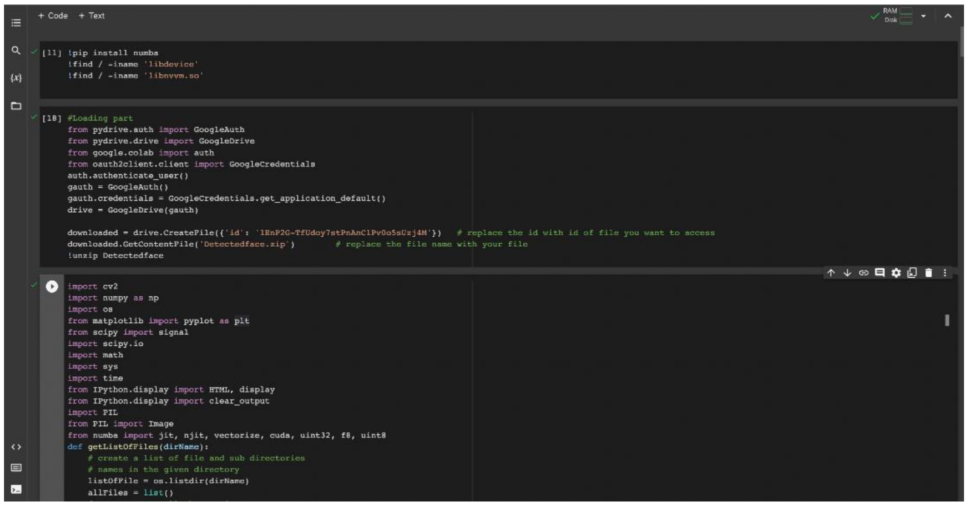
```
# Function for calculating LBP

def lbp_calculated_pixel8(img, x, y):

    center = img[x][y]

    val_ar = []

    # top_left
    val_ar.append(Human.get_pixel(img, center, x - 1, y - 1))

    # top
    val_ar.append(Human.get_pixel(img, center, x - 1, y))

    # top_right
    val_ar.append(Human.get_pixel(img, center, x - 1, y + 1))

    # right
    val_ar.append(Human.get_pixel(img, center, x, y + 1))

    # bottom_right
    val_ar.append(Human.get_pixel(img, center, x + 1, y + 1))

    # bottom
    val_ar.append(Human.get_pixel(img, center, x + 1, y))

    # bottom_left
    val_ar.append(Human.get_pixel(img, center, x + 1, y - 1))

    # left
```

```
        val_ar.append(Human.get_pixel(img, center, x, y - 1))

        # Now, we need to convert binary
        # values to decimal
        power_val = [1, 2, 4, 8, 16, 32, 64, 128]

        val = 0

        for i in range(len(val_ar)):
            val += val_ar[i] * power_val[i]

        return val


    def rgblbp(self, img):

        # Reading the BGR image using imread() function

        strr = img

        image = cv2.imread(img)
        height, width, xx = image.shape

        # Splitting the channels first to generate different
        # single

        # channels for merging as we don't have separate
        # channel images
        b, g, r = cv2.split(image)
```

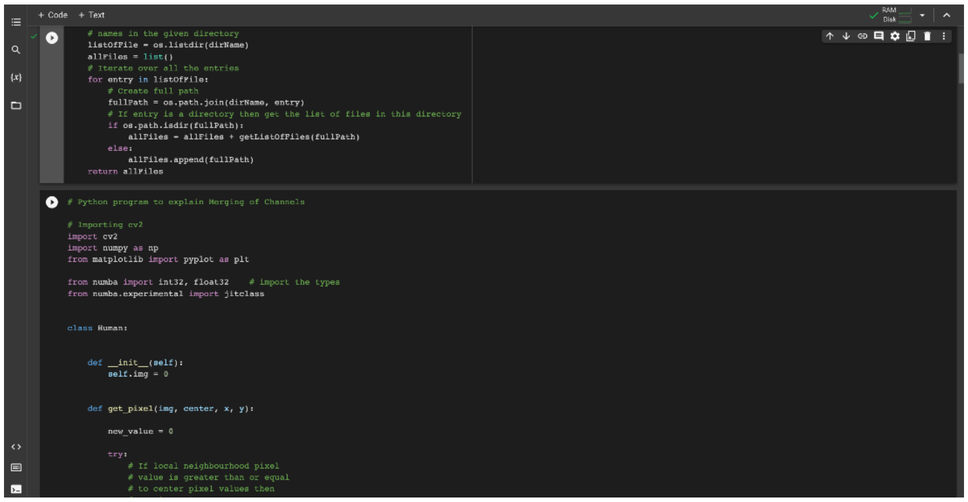
```
# 3 blank rgb
img_lbpr = np.zeros((height, width), np.uint8)
img_lbpg = np.zeros((height, width), np.uint8)
img_lbpb = np.zeros((height, width), np.uint8)
# # Displaying Blue channel image
# cv2.imshow("Model Blue Image", b)

# # Displaying Green channel image
# cv2.imshow("Model Green Image", g)

# # Displaying Red channel image
# cv2.imshow("Model Red Image", r)

# Using cv2.merge() to merge Red, Green, Blue Channels

for i in range(0, height):
    for j in range(0, width):
        img_lbpr[i, j] = Human.lbp_calculated_pixel8(r, i, j)
        img_lbpg[i, j] = Human.lbp_calculated_pixel8(g, i, j)
        img_lbpb[i, j] = Human.lbp_calculated_pixel8(b, i, j)

# into a coloured/multi-channeled image
image_merge = cv2.merge([img_lbpr, img_lbpg, img_lbpb])

path = '/content'
# path = 'D:/OpenCV/Scripts/Images'
directory = r'/content'
os.chdir(directory)
cv2.imwrite(strr, image_merge)
cv2.waitKey(0)
```

```
# cv2.imwrite(os.path.join(path , img[36:44]), image_merge)
# cv2.waitKey(0)

# save in folder dor radius=1,8 nbr




# plt.imshow(image_merge)



# for 16


# print("for 16 neighbour")
# print("...............................................................")


# # 3 blank rgb
# img_lbpr = np.zeros((height, width), np.uint8)
# img_lbpg = np.zeros((height, width), np.uint8)
# img_lbpb = np.zeros((height, width), np.uint8)


# for i in range(0, height):
#     for j in range(0, width):
#         img_lbpr[i, j] = Human.lbp_calculated_pixel8(r, i, j)
#         img_lbpg[i, j] = Human.lbp_calculated_pixel8(g, i, j)
#         img_lbpb[i, j] = Human.lbp_calculated_pixel8(b, i, j)


# # into a coloured/multi-channeled image
# image_merge = cv2.merge([img_lbpr, img_lbpg, img_lbpb])
```

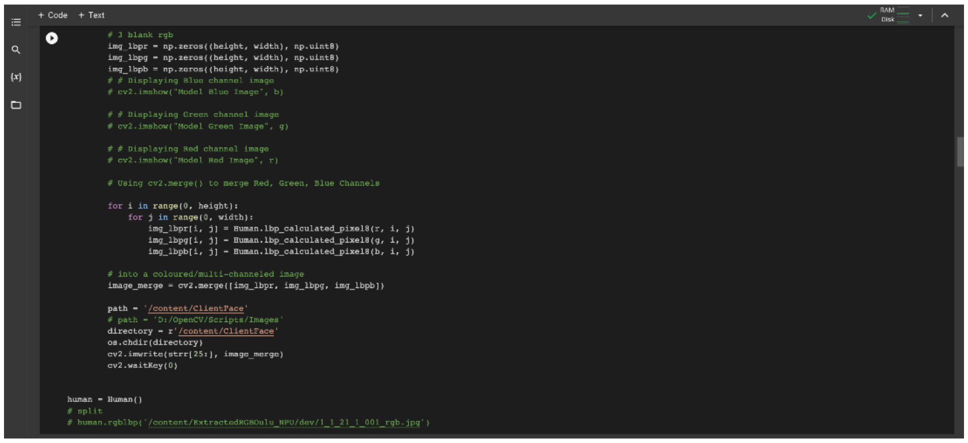
```
    # plt.imshow(image_merge)
    # print("for 16 neighbour")
    # print("............................................................")



    # Displaying Merged RGB image


    # cv2.imshow("RGB_Image", image_merge)


    # # Waits for user to press any key
    # cv2.waitKey(0)



human = Human()
# split
# human.rgblbp('/content/ExtractedRGBOulu_NPU/dev/1_1_21_1_001_rgb.jpg')


human.rgblbp("/content/Fake/fake.jpg")


path_list = getListOfFiles("/content/True")


# len(path_list)
for i in path_list:
  human.rgblbp(i)
```



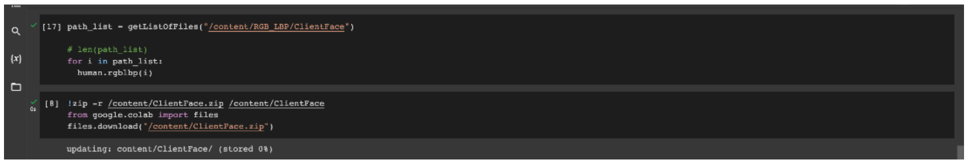
```
[11] !pip install numba
     !find / -iname 'libdevice'
     !find / -iname 'libnvvm.so'

[18] #Loading part
     from pydrive.auth import GoogleAuth
     from pydrive.drive import GoogleDrive
     from google.colab import auth
     from oauth2client.client import GoogleCredentials
     auth.authenticate_user()
     gauth = GoogleAuth()
     gauth.credentials = GoogleCredentials.get_application_default()
     drive = GoogleDrive(gauth)

     downloaded = drive.CreateFile({'id': '1EnP2G-TfUdoy7stPnAnClPv0o5sUzj4M'})   # replace the id with id of file you want to access
     downloaded.GetContentFile('Detectedface.zip')        # replace the file name with your file
     !unzip Detectedface

     import cv2
     import numpy as np
     import os
     from matplotlib import pyplot as plt
     from scipy import signal
     import scipy.io
     import math
     import sys
     import time
     from IPython.display import HTML, display
     from IPython.display import clear_output
     import PIL
     from PIL import Image
     from numba import jit, njit, vectorize, cuda, uint32, f8, uint8
     def getListOfFiles(dirName):
         # create a list of file and sub directories
         # names in the given directory
         listOfFile = os.listdir(dirName)
         allFiles = list()
```



```
         # names in the given directory
         listOfFile = os.listdir(dirName)
         allFiles = list()
         # Iterate over all the entries
         for entry in listOfFile:
             # Create full path
             fullPath = os.path.join(dirName, entry)
             # If entry is a directory then get the list of files in this directory
             if os.path.isdir(fullPath):
                 allFiles = allFiles + getListOfFiles(fullPath)
             else:
                 allFiles.append(fullPath)
         return allFiles

     # Python program to explain Merging of Channels

     # Importing cv2
     import cv2
     import numpy as np
     from matplotlib import pyplot as plt

     from numba import int32, float32    # import the types
     from numba.experimental import jitclass


     class Human:


         def __init__(self):
             self.img = 0


         def get_pixel(img, center, x, y):

             new_value = 0

             try:
                 # If local neighbourhood pixel
                 # value is greater than or equal
                 # to center pixel values then
```



```
             # 3 blank rgb
             img_lbpr = np.zeros((height, width), np.uint8)
             img_lbpg = np.zeros((height, width), np.uint8)
             img_lbpb = np.zeros((height, width), np.uint8)
             # # Displaying Blue channel image
             # cv2.imshow("Model Blue Image", b)

             # # Displaying Green channel image
             # cv2.imshow("Model Green Image", g)

             # # Displaying Red channel image
             # cv2.imshow("Model Red Image", r)

             # Using cv2.merge() to merge Red, Green, Blue Channels

             for i in range(0, height):
                 for j in range(0, width):
                     img_lbpr[i, j] = Human.lbp_calculated_pixel8(r, i, j)
                     img_lbpg[i, j] = Human.lbp_calculated_pixel8(g, i, j)
                     img_lbpb[i, j] = Human.lbp_calculated_pixel8(b, i, j)

             # into a coloured/multi-channeled image
             image_merge = cv2.merge([img_lbpr, img_lbpg, img_lbpb])

             path = '/content/ClientFace'
             # path = 'D:/OpenCV/Scripts/Images'
             directory = r'/content/ClientFace'
             os.chdir(directory)
             cv2.imwrite(strr[25:], image_merge)
             cv2.waitKey(0)


     human = Human()
     # split
     # human.rgblbp('/content/ExtractedRGBOulu_NPU/dev/1_1_21_1_001_rgb.jpg')
```



```
[17] path_list = getListOfFiles("/content/RGB_LBP/ClientFace")

     # len(path_list)
     for i in path_list:
       human.rgblbp(i)

[8]  !zip -r /content/ClientFace.zip /content/ClientFace
     from google.colab import files
     files.download("/content/ClientFace.zip")

     updating: content/ClientFace/ (stored 0%)
```

Figure 12: LBP Texture Filter Extraction (RGB)

Note: Similar process was done for other color spaces as well

5.3.Training the DeiT Vision Transformer

```
#!/usr/bin/env python

# coding: utf-8


# In[1]:




import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os

import torch

import torchvision

from torchvision import datasets

from torchvision import transforms as T # for simplifying the transforms

from torch import nn, optim

from torch.nn import functional as F

from torch.utils.data import DataLoader, sampler, random_split

from torchvision import models

## Now, we import timm, torchvision image models

get_ipython().system('pip install timm # kaggle doesnt have it installed by default')

import timm

from timm.loss import LabelSmoothingCrossEntropy # This is better than normal nn.CrossEntropyLoss

# remove warnings
```

```
import warnings

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt

get_ipython().run_line_magic('matplotlib', 'inline')

import sys

from tqdm import tqdm

import time

import copy

import sklearn.metrics

from sklearn.metrics import confusion_matrix, f1_score

get_ipython().system('pip install oulumetrics')

import oulumetrics

get_ipython().system('pip install keras')

get_ipython().system('pip install tensorflow-addons')

get_ipython().system('pip install gdown')

import subprocess



# # Dataset Extraction



# In[2]:




protocol = 'NUAA'

color_space = 'YCrCb'
```

```
# In[3]:




get_ipython().system('gdown --id 160ubWnc6YlKrSGeDCCT4EQDaNH2l_dT4')


# specify the file name and destination directory
file_name = f'{protocol}_{color_space}_Seperated'
# construct the command to unzip the file
command = ['unzip','-q', file_name]
# execute the command
subprocess.run(command)



# # Dataset Loading


# In[4]:



dataset_path = f"/kaggle/working/{file_name}"



# In[5]:
```

```
def get_data_loaders(data_dir, batch_size, train = False):
    if train:
        #train
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),
#             T.RandomApply(torch.nn.ModuleList([T.ColorJitter()]), p=0.25),
            T.Resize(256),
            T.CenterCrop(224),
            T.ToTensor(),
            T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)), # imagenet means
            T.RandomErasing(p=0.20, value='random')
        ])
        train_data = datasets.ImageFolder(os.path.join(data_dir, "train/"), transform = transform)
        train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True, num_workers=4)
        return train_loader, len(train_data)
    else:
        # val/test
        transform = T.Compose([ # We dont need augmentation for test transforms
            T.Resize(256),
            T.CenterCrop(224),
            T.ToTensor(),
            T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)), # imagenet means
        ])

        val_data = datasets.ImageFolder(os.path.join(data_dir, "dev/"), transform=transform)

        test_data = datasets.ImageFolder(os.path.join(data_dir, "test/"), transform=transform)

        val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=True, num_workers=4)

        test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=True, num_workers=4)

        return val_loader, test_loader, len(val_data), len(test_data)



# In[6]:



def get_classes(data_dir):

    all_data = datasets.ImageFolder(data_dir)

    return all_data.classes



# In[7]:



(train_loader, train_data_len) = get_data_loaders(dataset_path, 128, train=True)

(dev_loader, test_loader, dev_data_len, test_data_len) = get_data_loaders(dataset_path, 32, train=False)



# In[8]:
```

```
classes = get_classes(f"/kaggle/working/{file_name}/train/")

print(classes, len(classes))



# In[9]:



dataloaders = {

    "train": train_loader,

    "val": dev_loader

}

dataset_sizes = {

    "train": train_data_len,

    "val": dev_data_len

}



# In[10]:



print(len(train_loader), len(dev_loader), len(test_loader))



# In[11]:
```

```
print(train_data_len, dev_data_len, test_data_len)


# In[12]:


# now, for the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device


# In[13]:


# model_names = timm.list_models(pretrained=True)
# print(model_names)


# # Model Parameters

# In[14]:
```

```
pretrained_model_name = 'deit_base_patch16_224'   #cait_M48 cait_XXS36_224 cait_S24_224
deit_base_distilled_patch16_224 deit_base_patch16_224 deit_tiny_distilled_patch16_224

no_epochs = 50




# In[15]:





learning_rate = 0.03

model = torch.hub.load('facebookresearch/deit:main', pretrained_model_name, pretrained=True)


for param in model.parameters(): #freeze model

    param.requires_grad = False


n_inputs = model.head.in_features

model.head = nn.Sequential(


#     nn.BatchNorm1d(192),

#     nn.Linear(n_inputs, 512),

#     nn.ReLU(),

# #     nn.BatchNorm1d(512),

#     nn.Dropout(0.2, inplace=True),

#     nn.Linear(512, 256),

#     nn.ReLU(),
#     nn.Dropout(0.2, inplace=True),

#     nn.Linear(256, 128),

#     nn.ReLU(),

#     nn.Dropout(0.2, inplace=True),

#     nn.Linear(128, 64),

#     nn.ReLU(),

# #     nn.Dropout(0.2, inplace=True),


#     nn.Linear(64, len(classes))

    ############################################

    #nn.BatchNorm1d(768),

    nn.Linear(n_inputs, 512),

    nn.Mish(),

#     nn.BatchNorm1d(512),

    nn.Dropout(0.2, inplace=True),

    nn.Linear(512, 256),

    nn.Mish(),

    nn.Dropout(0.2, inplace=True),

    nn.Linear(256, 128),

    nn.Mish(),

    nn.Dropout(0.2, inplace=True),

    nn.Linear(128, 64),

    nn.Mish(),

#     nn.Dropout(0.2, inplace=True),
    nn.Linear(64, len(classes))
    ##############################################


#     nn.BatchNorm1d(192),
# #     nn.Linear(n_inputs, 512),
#     nn.PReLU(),
# #     nn.BatchNorm1d(512),
#     nn.Dropout(0.2, inplace=True),
# #     nn.Linear(512, 256),
#     nn.PReLU(),
#     nn.Dropout(0.2, inplace=True),
# #     nn.Linear(256, 128),
#     nn.PReLU(),
#     nn.Dropout(0.2, inplace=True),
# #     nn.Linear(128, 64),
#     nn.PReLU(),
    #################################
#     nn.Linear(n_inputs, 512),
#     nn.ReLU(),
#     nn.Dropout(0.3),
#     nn.Linear(512, len(classes))
)


# model.head = head_params
model = model.to(device)
```

```
# print(model)

# parameters



# In[16]:



criterion = LabelSmoothingCrossEntropy()

criterion = criterion.to(device)

optimizer = optim.SGD(model.head.parameters(), lr=learning_rate, momentum=0.5)



# In[17]:



# lr scheduler

exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)



# In[18]:



train_accs = []

val_accs = []

train_losses = []
```

```
val_losses=[]


def train_model(model, criterion, optimizer, scheduler, num_epochs):
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print("-"*10)

        for phase in ['train', 'val']: # We do training and validation phase per epoch
            if phase == 'train':
                model.train() # model to training mode
            else:
                model.eval() # model to evaluate

            running_loss = 0.0
            running_corrects = 0.0

            for inputs, labels in tqdm(dataloaders[phase]):
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()
                with torch.set_grad_enabled(phase == 'train'): # no autograd makes validation go faster
                    outputs = model(inputs)
#                  print(outputs)
                    _, preds = torch.max(outputs,1) # used for accuracy
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            if phase == 'train':
                scheduler.step() # step at end of epoch

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc =  running_corrects.double() / dataset_sizes[phase]
            epoch_acc = epoch_acc.cpu()
            if phase == 'val':
                val_accs.append(epoch_acc)
                val_losses.append(epoch_loss)
            else:
                train_accs.append(epoch_acc)
                train_losses.append(epoch_loss)
            print("{} Loss: {:.4f} Acc: {:.4f}".format(phase, epoch_loss, epoch_acc))


            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict()) # keep the best validation accuracy model
        print()
    time_elapsed = time.time() - since # slight error
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
    print("Best Train Acc: {:.4f}".format(max(train_accs)))
    print("Best Val Acc: {:.4f}".format(best_acc))

    model.load_state_dict(best_model_wts)
    return model



# # Model Training

# In[19]:


model_ft = train_model(model, criterion, optimizer, exp_lr_scheduler,num_epochs=no_epochs)


# In[20]:
```

```
# save the model

torch.save(model_ft.state_dict(), "Deit_Model.pth")




# # Graphs


# In[21]:




x = [i for i in range(no_epochs)]

# plot lines

plt.plot(x, train_accs, label = "Training Accuracy")

plt.plot(x, val_accs, label = "Validation Accuracy")

plt.xlabel("Epoch Number")

plt.ylabel("Accuracy")

plt.legend()

plt.show()




# In[22]:
```

```
x = [i for i in range(no_epochs)]

# plot lines

plt.plot(x, train_losses, label = "Training Loss")

plt.plot(x, val_losses, label = "Validation Loss")

plt.xlabel("Epoch Number")

plt.ylabel("Loss")

plt.legend()

plt.show()




# # Test and Dev Accuracies


# In[23]:




dev_loss = 0.0

class_correct = list(0 for i in range(len(classes)))

class_total = list(0 for i in range(len(classes)))

model_ft.eval()

dev_predictions = []

dev_targets = []


for data, target in tqdm(dev_loader):

    data, target = data.to(device), target.to(device)

    with torch.no_grad(): # turn off autograd for faster testing
```

```
        output = model_ft(data)

        loss = criterion(output, target)

    dev_loss = loss.item() * data.size(0)

    _, pred = torch.max(output, 1)

    pred_list = np.squeeze(pred.cpu().numpy())

    target_list = np.squeeze(target.cpu().numpy())

    dev_predictions.extend(pred_list)

    dev_targets.extend(target_list)

    correct_tensor = pred.eq(target.data.view_as(pred))

    correct = np.squeeze(correct_tensor.cpu().numpy())

#     print('target',target)

#     print('##################################################')

#     print('output',output)

#     print('##################################################')

#     print('pred',pred)

#     print('##################################################')

#     print('correct_tensor',correct_tensor)

#     print('##################################################')

#     print('correct',correct)

    if len(target) == 32:

        for i in range(32):

            label = target.data[i]

            class_correct[label] += correct[i].item()

            class_total[label] += 1
```

```
# print('class_correct',class_correct)
# print('##################################################')
# print('class_total',class_total)


dev_loss = dev_loss / dev_data_len
print('Dev Loss: {:.4f}'.format(dev_loss))
for i in range(len(classes)):
    if class_total[i] > 0:
        print("Dev Accuracy of %5s: %2d%% (%2d/%2d)" % (
            classes[i], 100*class_correct[i]/class_total[i], np.sum(class_correct[i]), np.sum(class_total[i])
        ))
    else:
        print("Dev accuracy of %5s: NA" % (classes[i]))
print("Dev Accuracy of %2d%% (%2d/%2d)" % (
            100*np.sum(class_correct)/np.sum(class_total), np.sum(class_correct), np.sum(class_total)
        ))



# In[24]:




test_loss = 0.0
class_correct = list(0 for i in range(len(classes)))
class_total = list(0 for i in range(len(classes)))
model_ft.eval()
```

```
test_predictions = []
test_targets = []

for data, target in tqdm(test_loader):
    data, target = data.to(device), target.to(device)
    with torch.no_grad(): # turn off autograd for faster testing
        output = model_ft(data)
        loss = criterion(output, target)
    test_loss = loss.item() * data.size(0)
    _, pred = torch.max(output, 1)
    pred_list = np.squeeze(pred.cpu().numpy())
    target_list = np.squeeze(target.cpu().numpy())
    test_predictions.extend(pred_list)
    test_targets.extend(target_list)
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.cpu().numpy())
#     print('target',target)
#     print('##################################################')
#     print('output',output)
#     print('##################################################')
#     print('pred',pred)
#     print('##################################################')
#     print('correct_tensor',correct_tensor)
#     print('##################################################')
#     print('correct',correct)
```

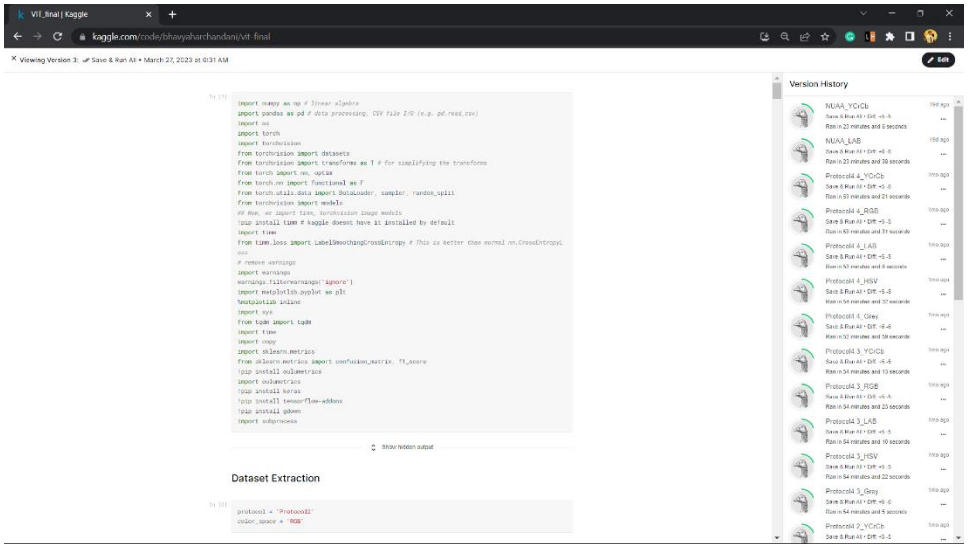
```
    if len(target) == 32:
        for i in range(32):
            label = target.data[i]
            class_correct[label] += correct[i].item()
            class_total[label] += 1

# print('class_correct',class_correct)
# print('##################################################')
# print('class_total',class_total)

test_loss = test_loss / test_data_len
print('Test Loss: {:.4f}'.format(test_loss))
for i in range(len(classes)):
    if class_total[i] > 0:
        print("Test Accuracy of %5s: %2d%% (%2d/%2d)" % (
            classes[i], 100*class_correct[i]/class_total[i], np.sum(class_correct[i]), np.sum(class_total[i])
        ))
    else:
        print("Test accuracy of %5s: NA" % (classes[i]))
print("Test Accuracy of %2d%% (%2d/%2d)" % (
            100*np.sum(class_correct)/np.sum(class_total), np.sum(class_correct), np.sum(class_total)
        ))



# # Save the model
# In[25]:


example = torch.rand(1, 3, 224, 224)
traced_script_module = torch.jit.trace(model.cpu(), example)
traced_script_module.save("vit_model.pt")


# In[26]:


# print('Target Labels: ',len(targets))
# print('Prediction Labels: ',len(predictions))


# # Other Metrics

# In[27]:


#EER
def compute_eer(label, pred, positive_label=1):
    # all fpr, tpr, fnr, fnr, threshold are lists (in the format of np.array)
    fpr, tpr, threshold = sklearn.metrics.roc_curve(label, pred)
    fnr = 1 - tpr


    # the threshold of fnr == fpr
    eer_threshold = threshold[np.nanargmin(np.absolute((fnr - fpr)))]


    # theoretically eer from fpr and eer from fnr should be identical but they can be slightly differ in reality
    eer_1 = fpr[np.nanargmin(np.absolute((fnr - fpr)))]
    eer_2 = fnr[np.nanargmin(np.absolute((fnr - fpr)))]


    # return the mean of eer from fpr and from fnr
    eer = (eer_1 + eer_2) / 2


    return eer


test_EER = compute_eer(test_targets,test_predictions)
dev_EER = compute_eer(dev_targets,dev_predictions)


print('The EER for Dev is: {:.2f}'.format(dev_EER*100))
print('The EER for Test is: {:.2f}'.format(test_EER*100))




# In[28]:




#HTER
```

```
def calculate_HTER(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()
    far = fp / (fp + tn)
    frr = fn / (tp + fn)
    hter = (far + frr) / 2
    return hter


dev_hter = calculate_HTER(dev_targets, dev_predictions)
print('The HTER for Dev is: {:.2f}'.format(dev_hter*100))


test_hter = calculate_HTER(test_targets, test_predictions)
print('The HTER for Test is: {:.2f}'.format(test_hter*100))



# In[29]:



#APCER, BPCER, ACER
test_attack_types = [ (1 if x==1 else 2) for x in test_targets]
dev_attack_types = [ (1 if x==1 else 2) for x in dev_targets]



# returns the metrics APCER, BPCER and ACER
test_apcer, test_bpcer, test_acer = oulumetrics.calculate_metrics(test_attack_types, test_predictions)
```

```
dev_apcer, dev_bpcer, dev_acer = oulumetrics.calculate_metrics(dev_attack_types, dev_predictions)


print('######## Dev Oulu Metrics ##########')

print('APCER: {:.2f}'.format(dev_apcer*100))

print('BPCER: {:.2f}'.format(dev_bpcer*100))

print('ACER: {:.2f}'.format(dev_acer*100))


print('######## Test Oulu Metrics ##########')

print('APCER: {:.2f}'.format(test_apcer*100))

print('BPCER: {:.2f}'.format(test_bpcer*100))

print('ACER: {:.2f}'.format(test_acer*100))





# In[30]:




# Compute confusion matrix

cm = confusion_matrix(test_targets, test_predictions)

print("Confusion Matrix:")

print(cm)


# Compute F1 score

f1 = f1_score(test_targets, test_predictions)
```

```
print("F1 Score:", f1)



# In[31]:



from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt


# Compute ROC curve
fpr, tpr, thresholds = roc_curve(test_targets, test_predictions)


# Plot ROC curve
plt.plot(fpr, tpr, color='blue', label='ROC Curve')
plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()


# # Add predicted labels to plot
# for i, threshold in enumerate(thresholds):
#     plt.annotate(threshold, (fpr[i], tpr[i]), textcoords="offset points", xytext=(0,10), ha='center')


# for i, label in enumerate(test_predictions):
```

```
#     if label == 1:
#         plt.plot(fpr[i], tpr[i], 'o', color='green')
#     else:
#         plt.plot(fpr[i], tpr[i], 'o', color='red')


# plt.show()
```



Viewing Version 3: Save & Run All • March 27, 2023 at 6:31 AM

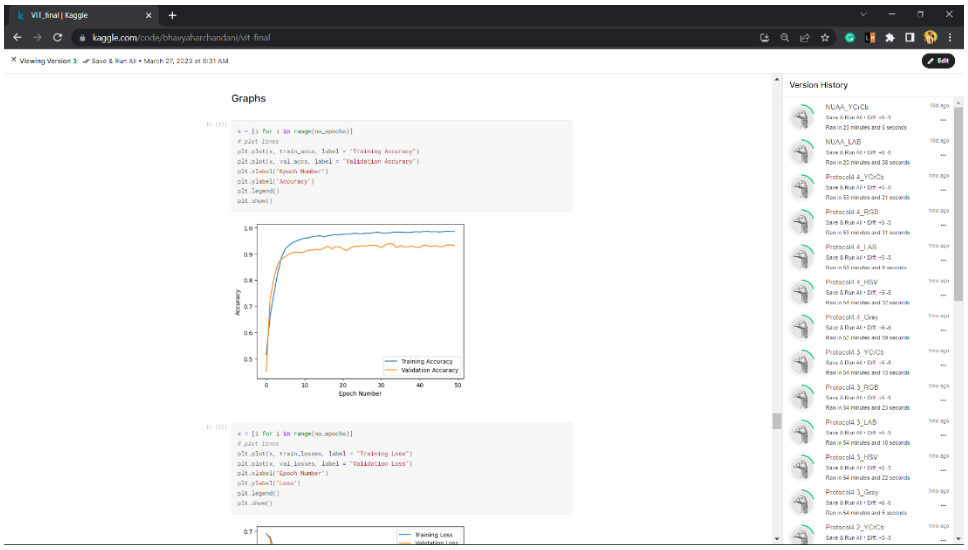
```
In [1]:
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import torch
import torchvision
from torchvision import datasets
from torchvision import transforms as T # for simplifying the transforms
from torch import nn, optim
from torch.nn import functional as F
from torch.utils.data import DataLoader, sampler, random_split
from torchvision import models
## Now, we import timm, torchvision image models
!pip install timm # kaggle doesnt have it installed by default
import timm
from timm.loss import LabelSmoothingCrossEntropy # This is better than normal nn.CrossEntropyL
oss
# remove warnings
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
%matplotlib inline
import sys
from tqdm import tqdm
import time
import copy
import sklearn.metrics
from sklearn.metrics import confusion_matrix, f1_score
!pip install oulumetrics
import oulumetrics
!pip install keras
!pip install tensorflow-addons
!pip install gdown
import subprocess
```

Dataset Extraction

```
In [2]:
protocol = 'Protocol2'
color_space = 'RGB'
```

Version History

NUAA_YCrCb Save & Run All • Diff: +5 -5 Ran in 23 minutes and 6 seconds	19d ago
NUAA_LAB Save & Run All • Diff: +6 -6 Ran in 23 minutes and 38 seconds	19d ago
Protocol4.4_YCrCb Save & Run All • Diff: +5 -6 Ran in 53 minutes and 21 seconds	1mo ago
Protocol4.4_RGB Save & Run All • Diff: +5 -5 Ran in 53 minutes and 31 seconds	1mo ago
Protocol4.4_LAB Save & Run All • Diff: +5 -5 Ran in 53 minutes and 6 seconds	1mo ago
Protocol4.4_HSV Save & Run All • Diff: +5 -5 Ran in 54 minutes and 32 seconds	1mo ago
Protocol4.4_Grey Save & Run All • Diff: +6 -6 Ran in 52 minutes and 58 seconds	1mo ago
Protocol4.3_YCrCb Save & Run All • Diff: +5 -5 Ran in 54 minutes and 13 seconds	1mo ago
Protocol4.3_RGB Save & Run All • Diff: +5 -5 Ran in 54 minutes and 23 seconds	1mo ago
Protocol4.3_LAB Save & Run All • Diff: +5 -5 Ran in 54 minutes and 10 seconds	1mo ago
Protocol4.3_HSV Save & Run All • Diff: +5 -5 Ran in 54 minutes and 22 seconds	1mo ago
Protocol4.3_Grey Save & Run All • Diff: +6 -6 Ran in 54 minutes and 5 seconds	1mo ago
Protocol4.2_YCrCb Save & Run All • Diff: +5 -5	1mo ago



Viewing Version 3: Save & Run All • March 27, 2023 at 6:31 AM

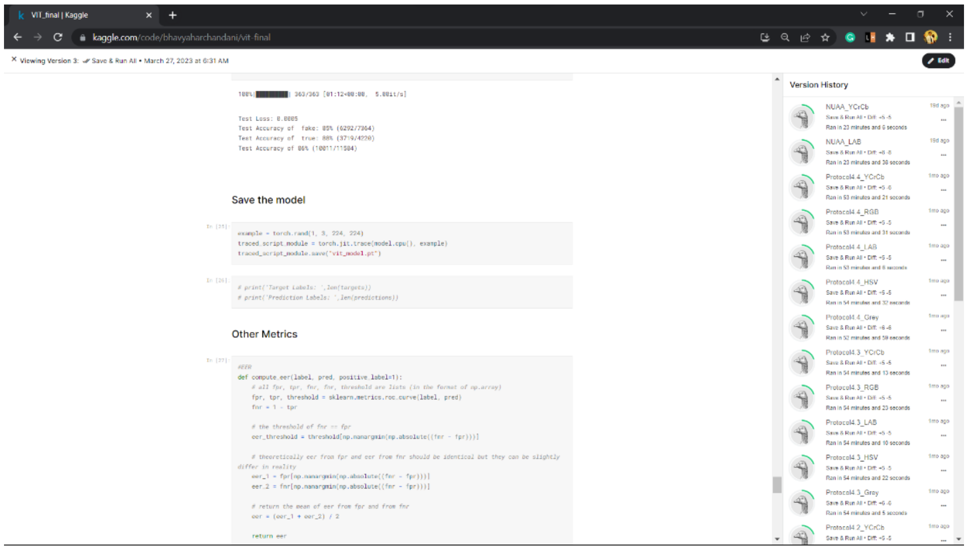
Graphs

```
In [21]:
x = [i for i in range(no_epochs)]
# plot lines
plt.plot(x, train_accs, label = "Training Accuracy")
plt.plot(x, val_accs, label = "Validation Accuracy")
plt.xlabel("Epoch Number")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
In [22]:
x = [i for i in range(no_epochs)]
# plot lines
plt.plot(x, train_losses, label = "Training Loss")
plt.plot(x, val_losses, label = "Validation Loss")
plt.xlabel("Epoch Number")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Version History

NUAA_YCrCb Save & Run All • Diff: +5 -5 Ran in 23 minutes and 6 seconds	19d ago
NUAA_LAB Save & Run All • Diff: +6 -6 Ran in 23 minutes and 38 seconds	19d ago
Protocol4.4_YCrCb Save & Run All • Diff: +5 -6 Ran in 53 minutes and 21 seconds	1mo ago
Protocol4.4_RGB Save & Run All • Diff: +5 -5 Ran in 53 minutes and 31 seconds	1mo ago
Protocol4.4_LAB Save & Run All • Diff: +5 -5 Ran in 53 minutes and 6 seconds	1mo ago
Protocol4.4_HSV Save & Run All • Diff: +5 -5 Ran in 54 minutes and 32 seconds	1mo ago
Protocol4.4_Grey Save & Run All • Diff: +6 -6 Ran in 52 minutes and 58 seconds	1mo ago
Protocol4.3_YCrCb Save & Run All • Diff: +5 -5 Ran in 54 minutes and 13 seconds	1mo ago
Protocol4.3_RGB Save & Run All • Diff: +5 -5 Ran in 54 minutes and 23 seconds	1mo ago
Protocol4.3_LAB Save & Run All • Diff: +5 -5 Ran in 54 minutes and 10 seconds	1mo ago
Protocol4.3_HSV Save & Run All • Diff: +5 -5 Ran in 54 minutes and 22 seconds	1mo ago
Protocol4.3_Grey Save & Run All • Diff: +6 -6 Ran in 54 minutes and 5 seconds	1mo ago
Protocol4.2_YCrCb Save & Run All • Diff: +5 -5	1mo ago



```
100%|██████████| 363/363 [01:12<00:00,  5.00it/s]

Test Loss: 0.0005
Test Accuracy of  fake: 85% (6292/7364)
Test Accuracy of  true: 88% (3719/4220)
Test Accuracy of 86% (10011/11584)
```

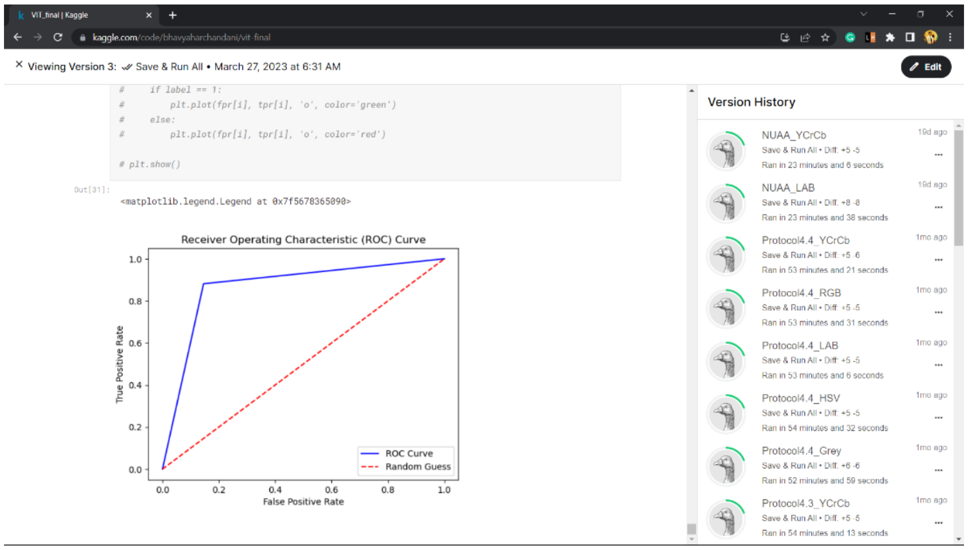
Save the model

```
In [25]:
example = torch.rand(1, 3, 224, 224)
traced_script_module = torch.jit.trace(model.cpu(), example)
traced_script_module.save("vit_model.pt")
```

```
In [26]:
# print('Target Labels: ',len(targets))
# print('Prediction Labels: ',len(predictions))
```

Other Metrics

```
In [27]:
#EER
def compute_eer(label, pred, positive_label=1):
    # all fpr, tpr, fnr, fnr, threshold are lists (in the format of np.array)
    fpr, tpr, threshold = sklearn.metrics.roc_curve(label, pred)
    fnr = 1 - tpr

    # the threshold of fnr == fpr
    eer_threshold = threshold[np.nanargmin(np.absolute((fnr - fpr)))]

    # theoretically eer from fpr and eer from fnr should be identical but they can be slightly
differ in reality
    eer_1 = fpr[np.nanargmin(np.absolute((fnr - fpr)))]
    eer_2 = fnr[np.nanargmin(np.absolute((fnr - fpr)))]

    # return the mean of eer from fpr and from fnr
    eer = (eer_1 + eer_2) / 2

    return eer
```



```
#     if label == 1:
#         plt.plot(fpr[i], tpr[i], 'o', color='green')
#     else:
#         plt.plot(fpr[i], tpr[i], 'o', color='red')

# plt.show()
```

```
Out[31]:
<matplotlib.legend.Legend at 0x7f5678365090>
```

Figure 13: Vision Transformer Training

5.4. Testing Single Image Implementation

#!/usr/bin/env python

# coding: utf-8

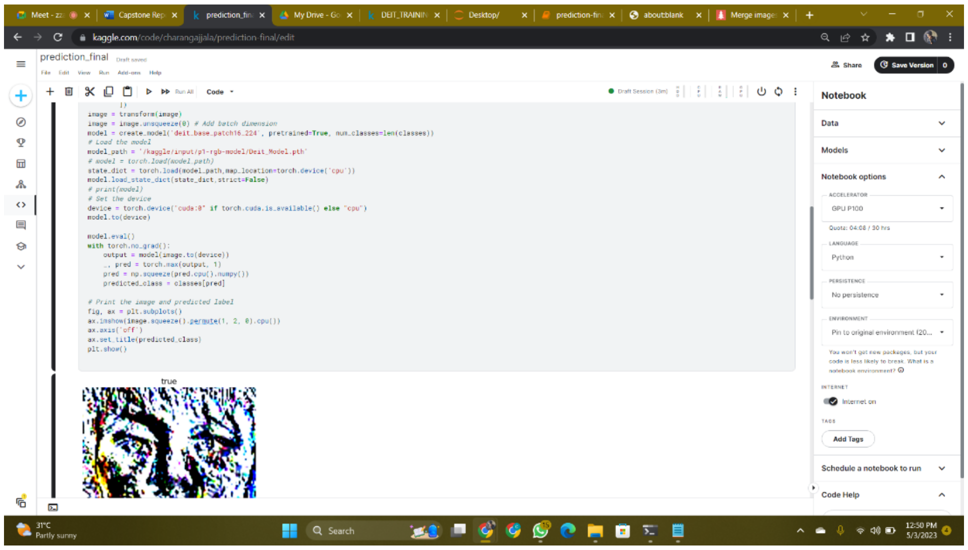
# In[1]:

import torch

import numpy as np

```
from PIL import Image

from torchvision import transforms as T

from timm.models import create_model

import matplotlib.pyplot as plt

get_ipython().system('pip install gdown')

get_ipython().system('gdown --id 1uZbrac0JBZaIREPI2TJgQGVpSqb8IQz2')

get_ipython().system('unzip -q prediction_images'

# In[8]:




#
"C:\Users\Charan\Desktop\Capstone\Data\ColorSpace_Datasets\Oulu_Dataset\Oulu_Dataset\Charan\prot
ocol1\Protocol1_RGB_Seperated_undersampled\test\fake\1_3_36_2_001_rgb.jpg"

# Load the image

image_path = '/kaggle/working/prediction_images/Real.jpg'

image = Image.open(image_path)

# Define the classes

classes = ['fake','true']

# Preprocess the image

transform  = T.Compose([ # We dont need augmentation for test transforms

        T.Resize(256),

        T.CenterCrop(224),

        T.ToTensor(),

        T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)), # imagenet means

    ])
```

```
image = transform(image)

image = image.unsqueeze(0) # Add batch dimension

model = create_model('deit_base_patch16_224', pretrained=True, num_classes=len(classes))

# Load the model

model_path = '/kaggle/input/p1-rgb-model/Deit_Model.pth'

# model = torch.load(model_path)

state_dict = torch.load(model_path,map_location=torch.device('cpu'))

model.load_state_dict(state_dict,strict=False)

# print(model)

# Set the device

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model.to(device)

model.eval()

with torch.no_grad():

    output = model(image.to(device))

    _, pred = torch.max(output, 1)

    pred = np.squeeze(pred.cpu().numpy())

    predicted_class = classes[pred]

# Print the image and predicted label

fig, ax = plt.subplots()

ax.imshow(image.squeeze().permute(1, 2, 0).cpu())

ax.axis('off')

ax.set_title(predicted_class)

plt.show()
```


kaggle.com/code/charangajjala/prediction-final/edit
prediction_final
Notebook
Data
Models
Notebook options
Schedule a notebook to run
Code Help
true

```
        ])
image = transform(image)
image = image.unsqueeze(0) # Add batch dimension
model = create_model('deit_base_patch16_224', pretrained=True, num_classes=len(classes))
# Load the model
model_path = '/kaggle/input/p1-rgb-model/Deit_Model.pth'
# model = torch.load(model_path)
state_dict = torch.load(model_path,map_location=torch.device('cpu'))
model.load_state_dict(state_dict,strict=False)
# print(model)
# Set the device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)

model.eval()
with torch.no_grad():
    output = model(image.to(device))
    _, pred = torch.max(output, 1)
    pred = np.squeeze(pred.cpu().numpy())
    predicted_class = classes[pred]

# Print the image and predicted label
fig, ax = plt.subplots()
ax.imshow(image.squeeze().permute(1, 2, 0).cpu())
ax.axis('off')
ax.set_title(predicted_class)
plt.show()
```

ACCELERATOR: GPU P100
LANGUAGE: Python
PERSISTENCE: No persistence
ENVIRONMENT: Pin to original environment
INTERNET: Internet on

Figure 14: Single Image Testing

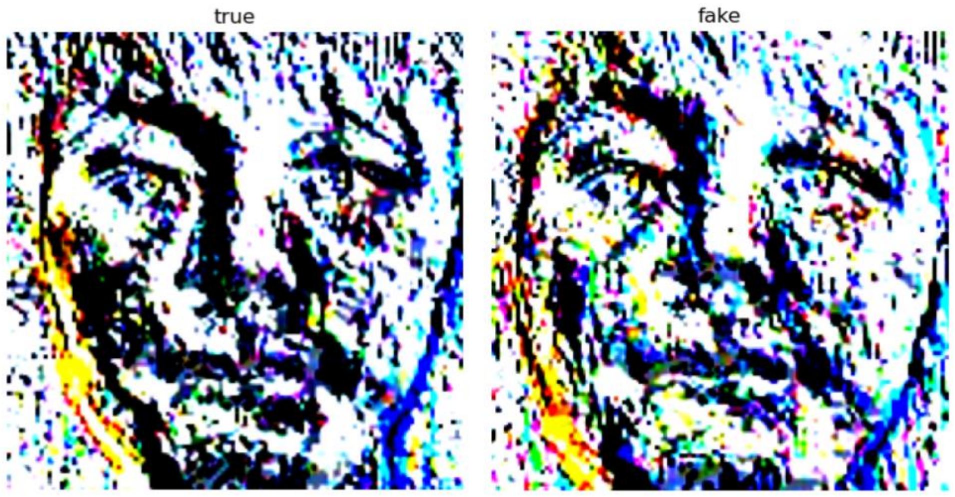


Figure 15: Testing Results

6. Results and Discussion

The evaluation of the experiment is done on the basis of the testing accuracy on the OuluNPU Dataset, a detailed display of the testing results for the five color spaces namely, RGB-LBP, HSV-LBP, Grayscale-LBP, YCrCb-LBP, and LA*B*-LBP is provided in the [TABLE 5]. Further, in order to evaluate the robustness of the model, ISO/IEC 30107-3 metrics [40] are calculated for every configuration. We have displayed the best values received amongst all the color spaces in the [TABLE 6].

- Protocol 1

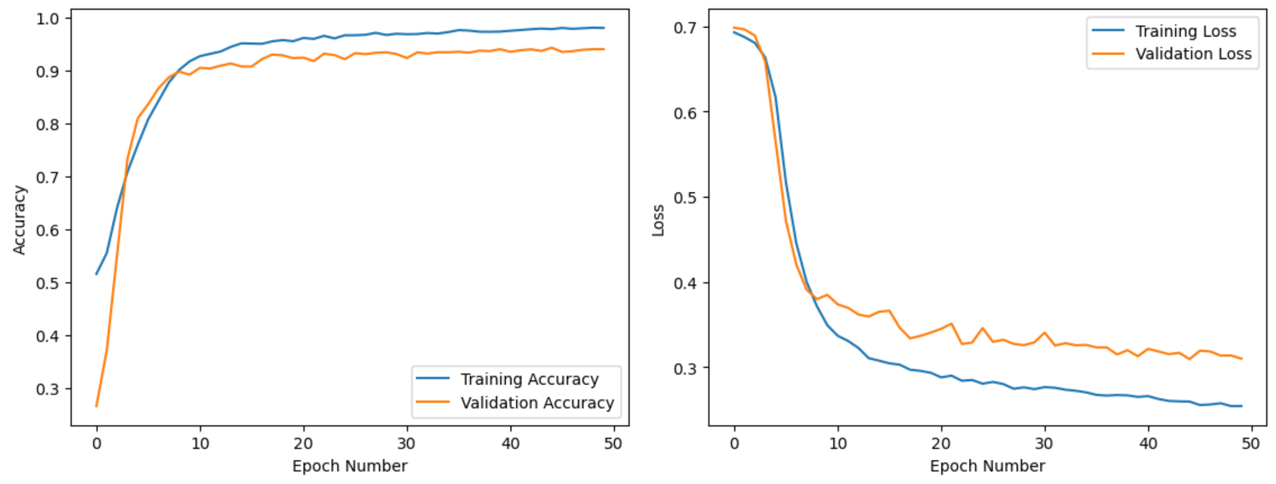


Figure 16: Accuracy and Loss for Protocol 1

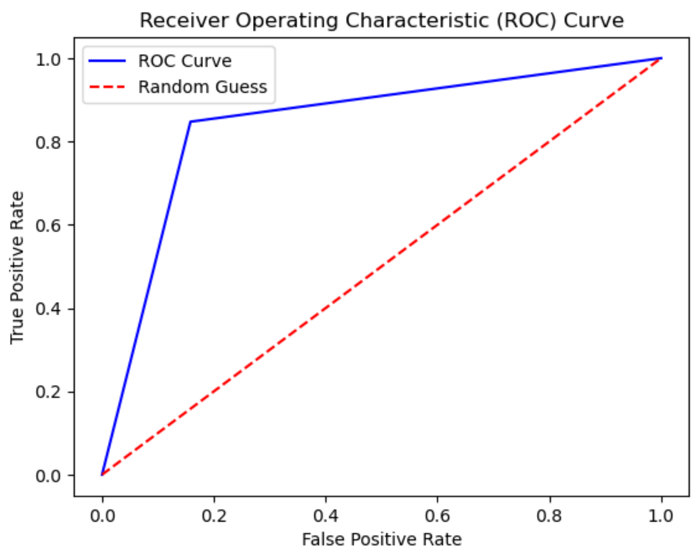


Figure 17: ROC Curve for Protocol 1

- Protocol 2

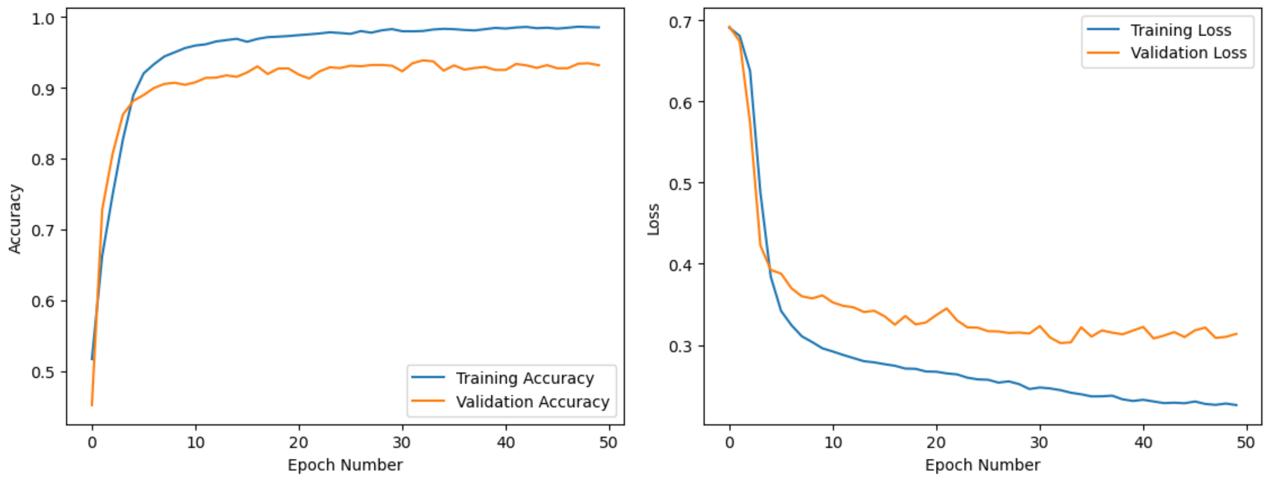


Figure 18: Accuracy and Loss for Protocol 2

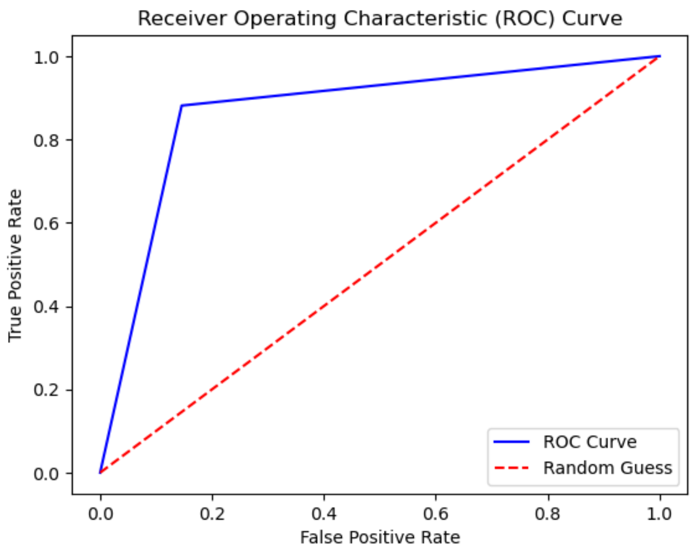


Figure 19: ROC Curve for Protocol 2

- Protocol 3

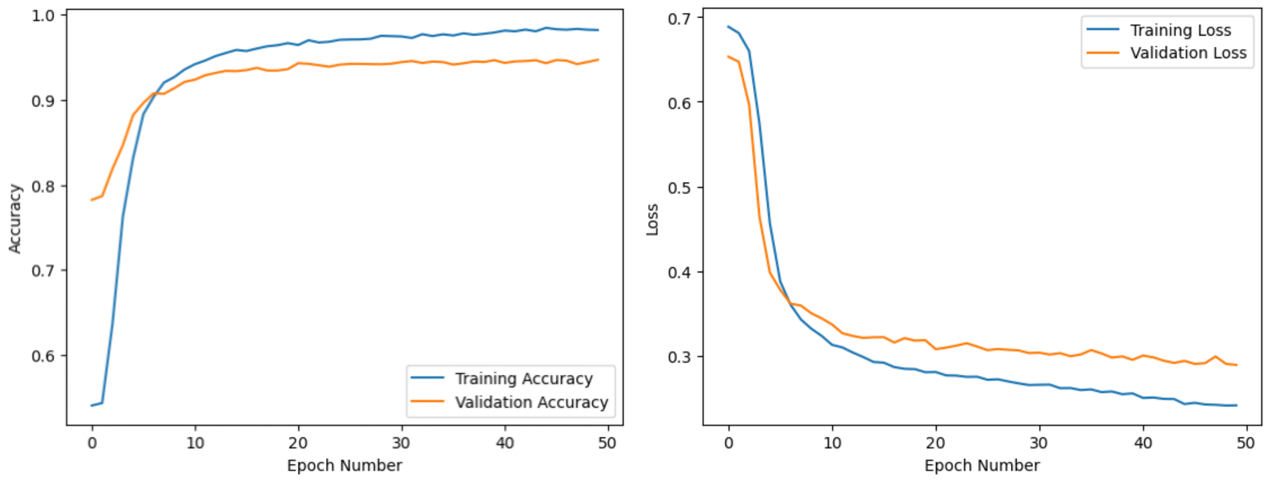


Figure 20: Accuracy and Loss for Protocol 3

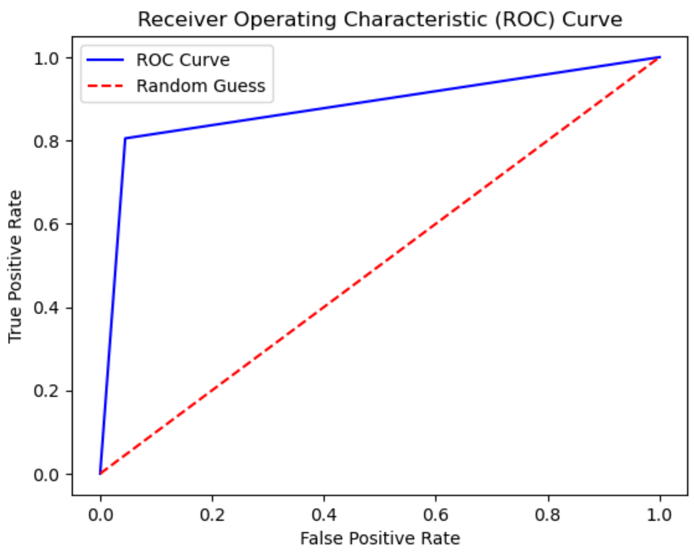


Figure 21: ROC Curve for Protocol 3

- Protocol 4

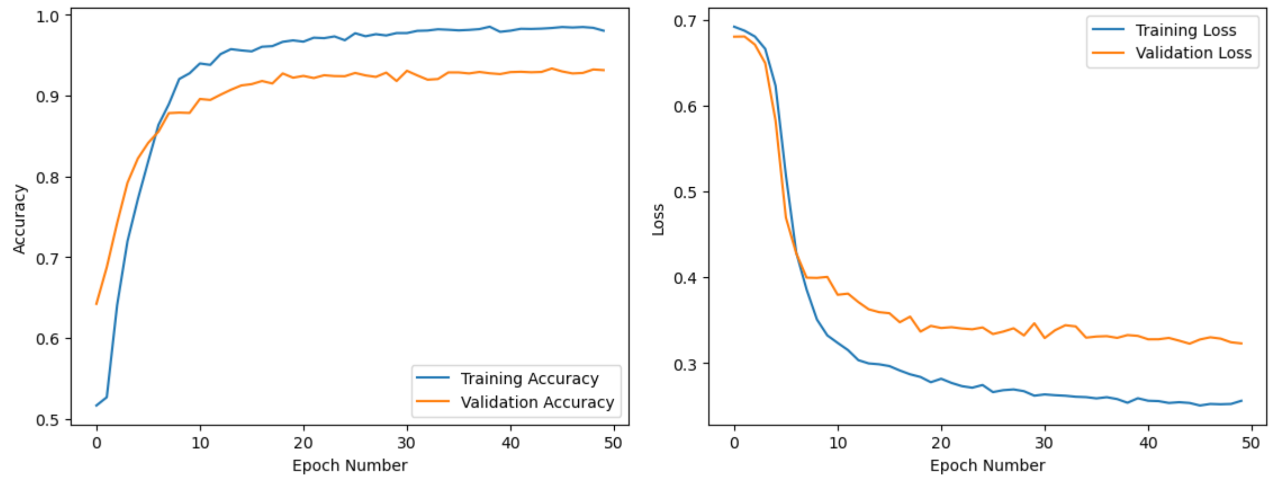


Figure 22: Accuracy and Loss for Protocol 4

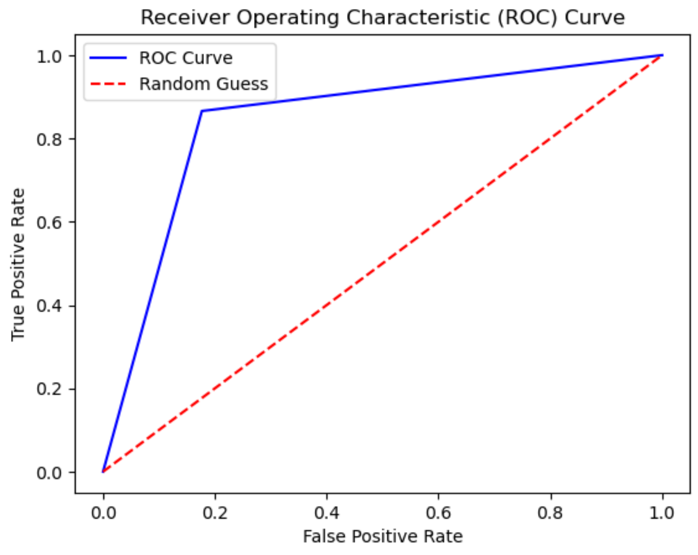


Figure 23: ROC Curve for Protocol 4

Protocol Name	**Color space**	**Real Detection Accuracy**	**Spoof Detection Accuracy**	**Testing Accuracies**	**F1 Score**
Protocol 1	RGB-LBP	0.84	0.84	0.84	0.69
	HSV-LBP	0.64	0.77	0.75	0.51
	Grayscale-LBP	0.68	0.69	0.69	0.48

	YCrCb-LBP	0.83	0.72	0.74	0.57
	LA*B*-LBP	0.82	0.79	0.79	0.63
	RGB-LBP	0.88	0.85	0.86	0.83
	HSV-LBP	0.52	0.81	0.7	0.56
	Grayscale-LBP	0.69	0.83	0.78	0.7
	YCrCb-LBP	0.8	0.8	0.8	0.75
Protocol 2	LA*B*-LBP	0.77	0.77	0.77	0.72
	RGB-LBP	0.78 ± 0.11	0.925± 0.025	0.915± 0.025	0.775 ± 0.055
	HSV-LBP	0.46± 0.11	0.76± 0.15	0.74± 0.07	0.45 ± 0.15
	Grayscale-LBP	0.58± 0.12	0.775± 0.155	0.77± 0.07	0.595 ± 0.125
	YCrCb-LBP	0.59± 0.26	0.91± 0.06	0.86± 0.04	0.615 ± 0.135
Protocol 3	LA*B*-LBP	0.575± 0.165	0.845± 0.115	0.845±0.095	0.6085 ± 0.069
	RGB-LBP	0.84± 0.09	0.74± 0.19	0.725± 0.105	0.685 ± 0.095
	HSV-LBP	0.69± 0.18	0.735± 0.065	0.72± 0.09	0.625 ± 0.135
	Grayscale-LBP	0.665± 0.145	0.58± 0.25	0.635± 0.135	0.575 ± 0.085
	YCrCb-LBP	0.655± 0.285	0.725± 0.185	0.725± 0.065	0.615 ± 0.135
Protocol 4	LA*B*-LBP	0.85± 0.07	0.565± 0.135	0.645± 0.095	0.615 ± 0.065

Table 5: Test Accuracies for OuluNPU

We also implemented our methodology on the NUAA Photograph Imposter database[58] with the Grayscale-LBP configuration and the following are the results we obtained from it:

Graphs for Grayscale-LBP:

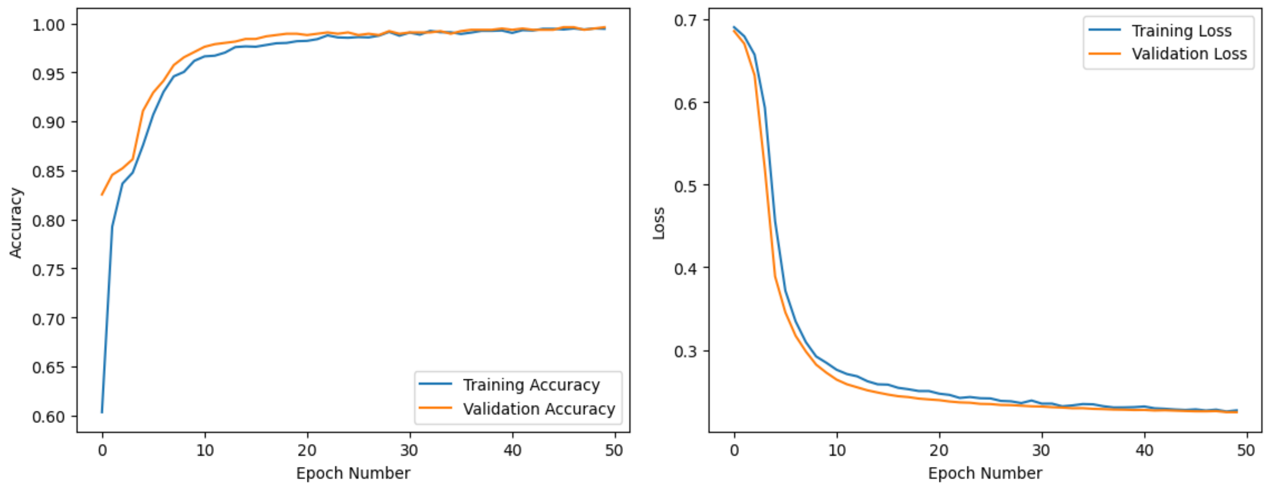


Figure 24: Accuracy and Loss for NUAA Grayscale-LBP

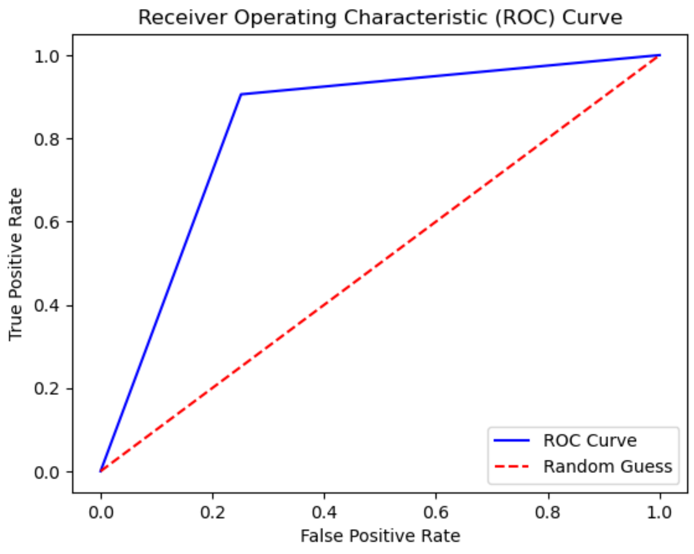


Figure 25: ROC Curve for NUAA Grayscale-LBP

	Dev Accuracy	**Test Accuracy**
Client Face	0.99	0.90
Imposter Face	0.99	0.74
Net Accuracy	0.99	0.80

F1 Score	-	0.78

Table 6: Dev and Test accuracies on NUAA Grayscale LBP

6.1.Metrics

For the evaluations in the Oulu-NPU dataset, we have used the ISO/IEC 30107-3 metrics [40], Attack Presentation Classification Error Rate (APCER), and Bonafide Presentation Classification Error Rate (BPCER) along with the Average Classification Error Rate (ACER) in the eval set. Additionally, the equal error rate (EER) is also reported in the eval set. Tt is the point where the false rejection rate (FRR) is equal to false acceptance rate (FAR) in receiver operating characteristic (ROC) curve.

Protocol Name	**Error Metric**	**Values**	**Best Performing Color Space**
Protocol 1	**APCER(%)**	15.81	RGB-LBP
	BPCER(%)	15.26	
	ACER(%)	15.54	
Protocol 2	**APCER(%)**	14.57	RGB-LBP
	BPCER(%)	11.87	
	ACER(%)	13.22	
Protocol 3	**APCER(%)**	10.2 ± 5.61	RGB-LBP
	BPCER(%)	21.945 ± 11.145	
	ACER(%)	14.56 ± 4.64	
Protocol 4	**APCER(%)**	30.535 ± 14.015	RGB-LBP
	BPCER(%)	15.145 ± 8.955	

	ACER(%)	24.935 ± 9.395	

Table 7 Error Metrics for OuluNPU

6.2.Comparison with baseline methods

Protocol Name	Method	APCER(%)	BPCER(%)	ACER(%)
	LBP-SVM	12.92	51.67	32.29
	IQM-SVM	19.17	30.83	25
	DeepPixBis	0.83	0	0.42
Protocol 1	**Proposed Method**	**15.81**	**15.26**	**15.54**
	LBP-SVM	30	20.28	25.14
	IQM-SVM	12.5	16.94	14.72
	DeepPixBis	11.39	0.56	5.97
Protocol 2	**Proposed Method**	**14.57**	**11.87**	**13.22**
	LBP-SVM	28.5 ± 33.05	23.33 ± 17.99	25.92 ± 11.25
	IQM-SVM	21.94 ± 9.99	21.95 ± 16.79	21.95 ± 8.09
	DeepPixBis	11.67 ± 19.57	10.56 ± 14.06	11.11 ± 9.4
Protocol 3	**Proposed Method**	**10.2 ± 5.61**	**21.945 ± 11.145**	**14.56 ± 4.64**
	LBP-SVM	41.67 ± 27.03	55.0 ± 21.21	48.33 ± 6.07
	IQM-SVM	34.17±25.89	39.17±23.35	36.67±12.13
	DeepPixBis	36.67±29.67	13.33±16.75	25.0±12.67
Protocol 4	**Proposed Method**	**30.535 ± 14.015**	**15.145 ± 8.955**	**24.935 ± 9.395**

Table 8 Comparison of the proposed method with baseline methods

On comparing our proposed methodology with baseline methods on the basis of the ISO/IEC 30107-3 metrics [40], we find that our method performs competitively for protocols 1, 2, and 3 to the baseline methods. Our method outperforms all the baseline methods for protocol 4 of the OuluNPU Dataset [38] which can be seen in [TABLE 8]. The protocol 4 of the OuluNPU Dataset is the most difficult protocol which considers changes in illumination, changes in printers and display, as well as variations in camera to justify sensor interoperability together. Hence making our proposed method most feasible for different environments.

6.3.Conclusion

In this work, we have presented a novel methodology for Face Presentation Attack Detection (PAD). We created an architecture where we made use of the DeiT Vision Transformer [16] to train the LBP transformed spaces of the OuluNPU Dataset [38] with different color spaces. We analysed the working of the model created when different LBP color spaces were used. We were able to find that the model performs the best when trained on the RGB-LBP color space in comparison to the other color spaces mentioned above. We compared our proposed method to other baseline methodologies for Face PAD for the OuluNPU Dataset, where it was found that our method provides comparable results for the Protocols 1, 2 and 3 of the dataset and outperforms all the baseline methods for Protocol 4.

6.4.Future Work

Additional work can be done on the provided methodology in order to increase the accuracy of the model. One of the approaches we would like to explore with this model is by boosting it using XGBoost. Furthermore, the model can be tested on more datasets, and some cross-database testing can provide more information about the model's performance in unknown environments.

7. References

1. Anjos, André & Günther, Manuel & Pereira, Tiago & Korshunov, Pavel & Mohammadi, Amir & Marcel, Sébastien. (2017). Continuously Reproducing Toolchains in Pattern Recognition and Machine Learning Experiments.
2. Z. Boulkenafet, J. Komulainen and A. Hadid, "Face anti-spoofing based on color texture analysis," 2015 IEEE International Conference on Image Processing (ICIP), Quebec City, QC, Canada, 2015, pp. 2636-2640, doi: 10.1109/ICIP.2015.7351280.
3. Komulainen, Jukka & Hadid, Abdenour & Pietikäinen, Matti. (2011). Face spoofing detection from single images using micro-texture analysis. International Joint Conference on Biometrics. 10.1109/IJCB.2011.6117510.

4. Li, L., Correia, P.L. and Hadid, A. (2018), Face recognition under spoofing attacks: countermeasures and research directions. IET Biom., 7: 3-14 . https://doi.org/10.1049/iet-bmt.2017.0089
5. Chingovska, Ivana, André Anjos, and Sébastien Marcel. "On the effectiveness of local binary patterns in face anti-spoofing." 2012 BIOSIG-proceedings of the international conference of biometrics special interest group (BIOSIG). IEEE, 2012.
6. Anjos, André & Marcel, Sébastien. (2011). Counter-Measures to Photo Attacks in Face Recognition: a public database and a baseline. IAPR IEEE International Joint Conference on Biometrics. 10.1109/IJCB.2011.6117503.
7. Y. Atoum, Y. Liu, A. Jourabloo and X. Liu, "Face anti-spoofing using patch and depth-based CNNs," 2017 IEEE International Joint Conference on Biometrics (IJCB), Denver, CO, USA, 2017, pp. 319-328, doi: 10.1109/BTAS.2017.8272713.
8. George, Anjith, and Sébastien Marcel. "Deep pixel-wise binary supervision for face presentation attack detection." 2019 International Conference on Biometrics (ICB). IEEE, 2019.
9. George, Anjith, et al. "Biometric face presentation attack detection with multi-channel convolutional neural network." IEEE Transactions on Information Forensics and Security 15 (2019): 42-55.
10. Liu, Yaojie, et al. "Deep tree learning for zero-shot face anti-spoofing." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.
11. Nikisins, Olegs, Anjith George, and Sébastien Marcel. "Domain adaptation in multi-channel autoencoder based features for robust face anti-spoofing." 2019 International Conference on Biometrics (ICB). IEEE, 2019.
12. George, Anjith, and Sébastien Marcel. "Cross modal focal loss for rgbd face anti-spoofing." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021.
13. George, Anjith, and Sébastien Marcel. "On the effectiveness of vision transformers for zero-shot face anti-spoofing." 2021 IEEE International Joint Conference on Biometrics (IJCB). IEEE, 2021.
14. F. Zhuang et al., "A Comprehensive Survey on Transfer Learning," in Proceedings of the IEEE, vol. 109, no. 1, pp. 43-76, Jan. 2021, doi: 10.1109/JPROC.2020.3004555.

15. Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).
16. Touvron, Hugo, et al. "Training data-efficient image transformers & distillation through attention." International conference on machine learning. PMLR, 2021.
17. Benlamoudi, Azeddine & Bekhouche, Salah Eddine & Maarouf, Korichi & Bensid, Khaled & Ouahabi, A. & Hadid, Abdenour & taleb-ahmed, Abdelmalik. (2022). Face Presentation Attack Detection Using Deep Background Subtraction. Sensors. 22. 10.3390/s22103760.
18. Almeida, Waldir R., et al. "Detecting face presentation attacks in mobile devices with a patch-based CNN and a sensor-aware loss function." PloS one 15.9 (2020): e0238058.
19. Y. Zhang, M. Zhao, L. Yan, T. Gao and J. Chen, "CNN-Based Anomaly Detection For Face Presentation Attack Detection With Multi-Channel Images," 2020 IEEE International Conference on Visual Communications and Image Processing (VCIP), Macau, China, 2020, pp. 189-192, doi: 10.1109/VCIP49819.2020.9301818.
20. Brownlee, Jason. "A Gentle Introduction to Deep Learning for Face Recognition." Machine Learning Mastery (2019).
21. Arora, Shefali, M. P. S. Bhatia, and Vipul Mittal. "A robust framework for spoofing detection in faces using deep learning." The Visual Computer (2021): 1-12.
22. Tu, Xiaokang & Fang, Yuchun. (2017). Ultra-deep Neural Network for Face Anti-spoofing. 686-695. 10.1007/978-3-319-70096-0_70.
23. Hashemifard, Kooshan & Akbari, Mohammad. (2021). A Compact Deep Learning Model for Face Spoofing Detection.
24. Gu, Fei & Xia, Zhihua & Fei, Jianwei & Yuan, Chengsheng & Zhang, Qiang. (2020). Face spoof detection using feature map superposition and CNN. International Journal of Computational Science and Engineering. 22. 355. 10.1504/IJCSE.2020.107356.
25. N. Erdogmus and S. Marcel, "Spoofing in 2D face recognition with 3D masks and anti-spoofing with Kinect," 2013 IEEE Sixth International Conference on Biometrics: Theory, Applications and Systems (BTAS), Arlington, VA, USA, 2013, pp. 1-6, doi: 10.1109/BTAS.2013.6712688.
26. D. Gragnaniello, G. Poggi, C. Sansone and L. Verdoliva, "An Investigation of Local Descriptors for Biometric Spoofing Detection," in IEEE Transactions on Information Forensics and Security, vol. 10, no. 4, pp. 849-863, April 2015, doi: 10.1109/TIFS.2015.2404294.

27. Li, Jiangwei & Tan, Tieniu & Jain, Anil. (2004). Live Face Detection Based on the Analysis of Fourier Spectra. Proceedings of SPIE - The International Society for Optical Engineering. 5404. 296-303. 10.1117/12.541955.
28. Mahitha. “Face Spoof Detection Using Machine Learning with Colour Features.” (2018).
29. Tan, Xiaoyang, et al. "Face Liveness Detection from a Single Image with Sparse Low Rank Bilinear Discriminative Model." ECCV (6) 6316 (2010): 504-517.
30. Bharadwaj, Samarth, et al. Face anti-spoofing via motion magnification and multifeature videolet aggregation. 2014.
31. Raghu, Maithra, et al. "Do vision transformers see like convolutional neural networks?." Advances in Neural Information Processing Systems 34 (2021): 12116-12128.
32. Joshi, Soumya, et al. "Issues in training a convolutional neural network model for image classification." Advances in Computing and Data Sciences: Third International Conference, ICACDS 2019, Ghaziabad, India, April 12–13, 2019, Revised Selected Papers, Part II 3. Springer Singapore, 2019.
33. Peng, Fei, Shao-hua Meng, and Min Long. "Presentation attack detection based on two-stream vision transformers with self-attention fusion." Journal of Visual Communication and Image Representation 85 (2022): 103518.
34. Sharma, Deepika, and Arvind Selwal. "A survey on face presentation attack detection mechanisms: hitherto and future perspectives." Multimedia Systems (2023): 1-51.
35. Ke-Chen, Song, et al. "Research and perspective on local binary pattern." Acta Automatica Sinica 39.6 (2013): 730-744.
36. Paul, Sayak, and Pin-Yu Chen. "Vision transformers are robust learners." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 36. No. 2. 2022.
37. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
38. Boulkenafet, Zinelabinde, et al. "OULU-NPU: A mobile face presentation attack database with real-world variations." 2017 12th IEEE international conference on automatic face & gesture recognition (FG 2017). IEEE, 2017.
39. Singh, Abhishek Pratap, et al. "face recognition system based on LBPH algorithm." Int. J. Eng. Adv. Technol 8.5 (2019): 26-30.

40. Busch, Christoph. "Standards for biometric presentation attack detection." Handbook of Biometric Anti-Spoofing: Presentation Attack Detection and Vulnerability Assessment. Singapore: Springer Nature Singapore, 2023. 571-583.
41. Ming, Zuheng, et al. "A survey on anti-spoofing methods for facial recognition with rgb cameras of generic consumer devices." Journal of Imaging 6.12 (2020): 139.
42. Korshunov, Pavel, and Sébastien Marcel. "Deepfakes: a new threat to face recognition? assessment and detection." arXiv preprint arXiv:1812.08685 (2018).
43. Watanabe, Kota, Koichi Ito, and Takafumi Aoki. "Spoofing Attack Detection in Face Recognition System Using Vision Transformer with Patch-wise Data Augmentation." 2022 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC). IEEE, 2022.
44. Chen, Haonan, et al. "A cascade face spoofing detector based on face anti-spoofing R-CNN and improved Retinex LBP." IEEE Access 7 (2019): 170116-170133.
45. Wang, Pichao, et al. "Kvt: k-nn attention for boosting vision transformers." Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV. Cham: Springer Nature Switzerland, 2022.
46. Wang, Pichao, et al. "Kvt: k-nn attention for boosting vision transformers." Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV. Cham: Springer Nature Switzerland, 2022.
47. Song, Xiao, et al. "Discriminative representation combinations for accurate face spoofing detection." Pattern Recognition 85 (2019): 220-231.
48. Huang, Di, et al. "Local binary patterns and its application to facial image analysis: a survey." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 41.6 (2011): 765-781.
49. Hashemifard, Seyedkooshan, and Mohammad Akbari. "A compact deep learning model for face spoofing detection." arXiv preprint arXiv:2101.04756 (2021).
50. Samrity Saini, Kiranpreet Kaur. "KNN Classification for the Face Spoof Detection." IJSER Volume10,Issue 3, 2019
51. Nagpal, Chaitanya, and Shiv Ram Dubey. "A performance evaluation of convolutional neural networks for face anti spoofing." 2019 International Joint Conference on Neural Networks (IJCNN). IEEE, 2019.

52. Zhang, Meigui, Kehui Zeng, and Jinwei Wang. "A survey on face anti-spoofing algorithms." Journal of Information Hiding and Privacy Protection 2.1 (2020): 21.
53. Liao, Chen-Hao, et al. "Domain Invariant Vision Transformer Learning for Face Anti-Spoofing." Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. 2023
54. Li, Lei, et al. "An original face anti-spoofing approach using partial convolutional neural network." 2016 Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA). IEEE, 2016
55. Li, Xiaobai, et al. "Generalized face anti-spoofing by detecting pulse from face videos." 2016 23rd International Conference on Pattern Recognition (ICPR). IEEE, 2016
56. Wang, Zezheng, et al. "Deep spatial gradient and temporal depth learning for face anti-spoofing." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020
57. Khamaiseh, Samer Y., et al. "Adversarial Deep Learning: A Survey on Adversarial Attacks and Defense Mechanisms on Image Classification." IEEE Access (2022).
58. Tan, Xiaoyang, et al. "Face Liveness Detection from a Single Image with Sparse Low Rank Bilinear Discriminative Model." ECCV (6) 6316 (2010): 504-517.