INDIAN INSTITUTE OF TECHNOLOGY INDORE

UNDERGRADUATE THESIS

FACE ANTI SPOOFING TECHNIQUES

Authors: Anmol Gomra - 180001007
Shubham Nimesh - 180001054

Supervisor: Dr. Surya Prakash

*Thesis submitted in fulfillment of the requirements for the degree of **Bachelor of Technology***

in the

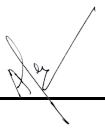
"Department of Computer Science and Engineering"

Declaration of Authorship

I, **Anmol Gomra** and **Shubham Nimesh** declare that this thesis, titled, **"Face anti spoofing"** and the work presented in it is our own. I confirm that:

- This work was done wholly or mainly while in candidature for the BTP project at IIT Indore.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Author-1 Sign: 

Author-2 Sign: 

Date: 27 - 05 - 2022

Certificate

This is to certify that the thesis entitled, "*Face Anti spoofing*" and submitted by **Anmol Gomra** and **Shubham Nimesh** in partial fulfillment of the requirements of **CS 493 B.Tech Project** embodies the work done by them under my supervision.

Supervisor

Dr. Surya Prakash
Associate Professor,
Indian Institute of Technology Indore

Date: 27-May-2022

“*For only in their dreams can men be truly free. ’Twas always thus, and always thus will be. ”*

Tom Schulman

INDIAN INSTITUTE OF TECHNOLOGY INDORE

Abstract

Department of Computer Science and Engineering

Bachelor of Technology

Face anti spoofing

In this new age of computer biometrics, facial recognition is becoming the second most widely deployed biometric authentication method in the world in terms of market quota right after fingerprints. It's used in smartphones, payment methods, and biometric authentication. Each day more and more manufacturers are including face recognition in their products, such as Apple with its Face-ID technology, the banks with the implementation of eKYC solutions for the onboarding process.

Due to this widespread use of face biometrics , it has become more vulnerable to spoofing attacks. Hackers and adversaries try to bypass face biometric systems using certain methods. For the safety of users and to maintain their confidence in online biometric authentication systems, it is today's need to develop such systems which protect biometrics and don't let hackers bypass security systems. The methods which come under this are called **face anti spoofing methods**. These methods prevent false facial verification by using a photo, video, mask or a different substitute for an authorized person's face.

A few advancements have been done in this field, like pixel wise supervision, use of vision transformers etc. Apart from software improvements and deep learning algorithms, specialized hardware is also used to prevent face anti spoofing, like 3d cameras, infrared cameras, 3-D dot projector (used in Apple's most popular device i.e iPhones), and they have proved to be effective at times. But still there is more work to be done, as with the advancements of technology, hackers and adversaries are getting more advanced, and there are several cases when deep fake has been used to bypass Face security systems, posing great threat to users privacy.

In this research project, we have tried to improve upon pixel wise supervision by implementing new neural network architectures such **BiFPN** and **DenseNet**. Moreover in a totally separate experiment we have tried to utilize the famous siamese network in differentiating fake images from real ones. As the project progressed we observed how using BiFPN along with pixel wise supervision provided minor improvements over the vanilla Pixel Wise Supervision. Using our experiments it is also proved how the siamese network can be used as a lightweight method for anti-spoofing where not much data is provided for learning and the task of differentiating fake from real needs to be computationally in-expensive.

Acknowledgements

I would like to thank my B.Tech Project supervisor **Dr. Surya Prakash** for his guidance and constant support in structuring the project and their valuable feedback throughout the course of this project. His overseeing the project meant there was a lot that I learnt while working on it. I thank him for his time and efforts.

Most importantly, I am thankful to **Shubham Nimesh**, my BTP partner without whose constant reminders to get back on the job, writing the Thesis would have taken much longer. Also, I am thankful to my other friends who were a constant source of both motivation and light hearted humor.

I am really grateful to the Institute for the opportunity to be exposed to systemic research, especially Dr. Surya Prakash Lab for providing the necessary hardware utilities to complete the project.
Lastly, I offer my sincere thanks to everyone who helped me complete this project, whose name I might have forgotten to mention.

Table Of Contents

Dedicated to all first graders, on the shores of an unexplored continent, just getting introduced to mathematics.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Thanks to rapid technological advancements, particularly in computer science and electronics. Facial recognition is now, after fingerprints, the second most extensively used biometric authentication method on a global scale in terms of market share. Face recognition is being integrated into more and more goods every day, such as Apple's Face-ID technology and banks' deployment of eKYC solutions for the onboarding process.

Despite the fact that the primary goal of face recognition research has been to improve performance at verification and identification tasks, the security vulnerabilities of face recognition systems have received far less attention in the past, and only in the last few years has some attention been paid to detecting different types of attacks, which consists of detecting whether a biometric trait comes from a living person or is a fake. As a result, the duty of avoiding fraudulent facial verification by utilizing a photo, video, mask, or other substitute for an authorized person's face is defined as Facial anti-spoofing.

1.2 Types of Spoof Attacks

1. **Presentation Attacks**
    - Attackers use a photograph of the user to be impersonated.
    - They use a video of the user to be impersonated.
    - Or hackers can build and use a 3D model of the attacked face, for example, a hyper-realistic mask
    - **Other attacks**: makeup, surgery

2. **Indirect Attacks**
    - Indirect attacks can be performed at the database, that matches, the communication channels, etc. In this type of attack, the attacker needs access to the interior of the system. Attacks can be done at stages such as pre-processing , feature extraction and classification etc.

1.3 Overview on Face Anti Spoofing Techniques

Over the last decade many different techniques have been studied to prevent spoof attacks. It is difficult to categorize one technique better and another one as inferior. Different techniques cater to different case scenarios as per the need arises.
A few of the techniques is listed as below -

1. Eye Blink Detection

One of the most accurate liveness detection tests is eye blink detection. Natural blinking is a simple technique to tell if a face is alive. This is pretty much a simple technique to check responsiveness of the client.
Simple presentation attacks can be avoided using this method such as spoofing using fake photos of the client.

Pros - This method is not too computationally expensive and is relatively fast.
Cons - This attack may not work against video attacks and 3d mask attacks though.

2. The Challenge-Response Technique

Using this technique, during authorization, the client is given a simple challenge of some kind and the client must pass the challenge in order to be authorized or granted access.
The challenge can be as simple as smiling, waving hand , shaking head or even blinking eyes certain number of times.

Pros - This method is not too computationally expensive and is relatively fast.
Cons - Relatively a slower method than others and requires more input from clients.

3. Deep Learning Features: Convolutional Neural Network

Using CNN's, our main focus is to somehow teach a computer to differentiate between a spoof image and an attack image. Somehow we assume that the computer will see a difference between the provided two images which a normal human can't see with naked eye. But what if training data we provided only certain features which represent a certain class of spoof attacks ? In this case if we provide some other image with a different class of attack, the trained neural network will fail to classify the real and spoof. In general it is a good practice to use datasets with large multiple classed spoof images.

Pros - CNN can detect more complex and sophisticated attacks if trained properly
Cons - 1. Much more computationally expensive as training can take a lot of time ranging from hours to days.
2. Overfitting may cause trained network to give wrong results.

In this project, we have chosen CNN's as our base technique for face anti-spoofing. Multiple sub techniques under Deep Learning have been proposed for anti-spoofing. Our experiments in this project mainly focus on the following techniques only :-

1. Face Anti spoofing using Pixel Wise Supervision
2. Face Anti spoofing using Pixel Wise Supervision using DenseNet architecture
3. Face Anti spoofing using Pixel Wise Supervision using Bi-Directional Feature pyramid Network architecture
4. Face Anti spoofing using Siamese Network with triplet loss
5. Face Anti spoofing using Siamese Network with lossless triplet loss

Chapter 2

Literature Survey

2.1 Dual-Cross Central Difference Network for Face Anti-Spoofing

- A Central Difference Convolution (CDC) is used, which is better than its counterpart i.e vanilla convolution at describing fine-grained invariant information, and gives better results than its counterpart.
- The CDC introduces certain features that enhance the representation and generalization capacity.
- CDC is more likely to extract intrinsic spoofing patterns (e.g., lattice artifacts) than vanilla convolution in diverse environments.
- It still have some disadvantages:
    1. In CDC, central gradients from all neighbors are calculated, which is inefficient in both inference and back-propagation stages.
    2. discrepancy among diverse gradient directions, thus redundant and inefficiency introduced.

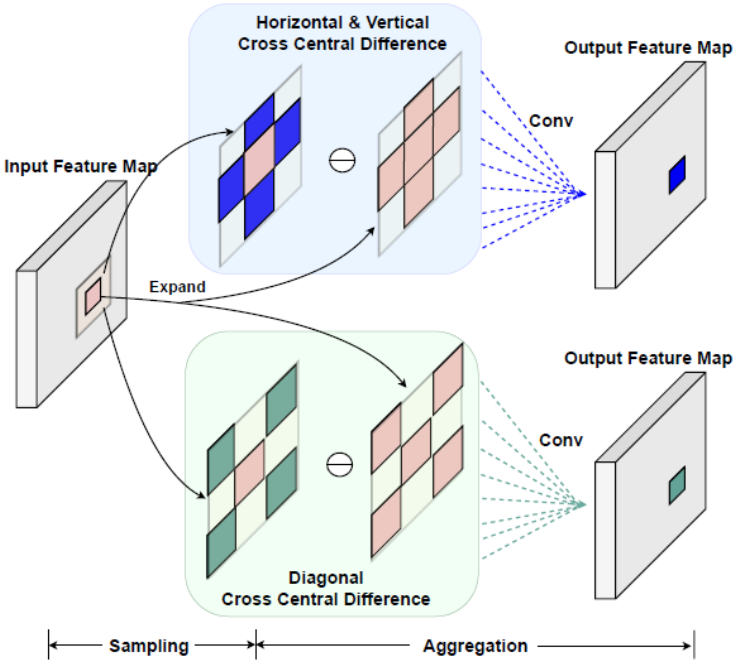


Fig-1 : Cross Central Difference Network

2.2 Pixel-Wise Supervision for Face Anti-Spoofing

- The deep models supervised by traditional binary loss (e.g., `0' for bonafide vs. `1' for PAs) are weak in describing intrinsic and discriminative spoofing patterns.
- Hence, Pixel-wise supervision has been proposed for **face anti spoofing tasks**
- Its motive is to provide more fine-grained pixel supervision.
- In this paper, a pyramid supervision is being proposed, which guides deep models to learn both local and global details from multi-scale feature masks (8x8, 4x4, 2x2 ,1x1 being the scales).
- Two methods have been discussed in this paper. First method is **Binary Mask Supervision** and the second method is **Depth Mask Supervision**.

Methodology

There are two mainstream backbones for pixel-wise supervision based FAS:

1. Classical classification based networks (ResNet [55]) with **binary mask supervision** .
2. Multi-scale fully convolutional network (DepthNet [10]) with **pseudo depth supervision**.

Binary mask supervision:

In this framework, the features are firstly extracted using ResNet50, and then are downsampled into multi-scale representation(8x8,4x4,2x2,1x1). After this ,1x1 convolutions ,with ‘Sigmoid’ function are utilized for features mapping and predicting of the multi-scale binary masks takes place.

The multi-scale binary mask labels provide sufficient pyramid supervision signals for training. Finally,flattening and concatenation of the predicted multi-scale masks takes place and final live/spoof classification is done.

This methodology is further explored in upcoming sections.

Depth mask supervision:

This is similar to Binary mask supervision but instead of generation of binary mask a depth map is generated and then the loss is calculated for training.

Multi-scale fully convolutional network is used in this method

Basic pyramid supervision is used in both of the above methods and we have tried to bring changes to pyramid supervision. This will be further discussed in **chapter 4**.

2.3 On the Effectiveness of Vision Transformers for Zero-shot Face Anti-Spoofing

- This paper has used a vision transformer model based PAD framework.
- It's a type of multi-channel method. Previously, a multichannel face presentation attack detection framework using color, depth, infrared and thermal channels was presented , but it was hardware costly.
- This papert used transfer learning from a pre-trained Vision Transformer model for zero-shot face anti-spoofing tasks.

Methodology

- Pre-processing is done using only the face area.
- Face detection and landmark localization are performed using the MTCNN algorithm. The detected faces are aligned so that the eye centers are horizontally aligned. After this alignment, the images are cropped to a resolution of 224 x 224.
- Vision transformer architecture is used as the backbone.
- An image is divided into patches, and embeddings obtained from the patches are used as the sequence input for the transformer. A sequence of image patches is used as the input followed by transformer layers.
- In this paper, the model is trained with 16 x 16 patches. So input sequence length will be the number of patches H*W/16^2. Alongside a 1D positional embedding is also added to retain position.
- After the transformer layers, an MLP head consisting of a fully connected layer is added for the classification task.

2.4 Bi-Directional Feature Pyramid Network for Pixel-Wise Face Anti-Spoofing

- This paper is basically an improvement over the second paper (i.e., **Pixel-Wise Supervision for Face Anti-Spoofing**). Bi-Directional pyramid supervision is introduced, which is basically an improvement over the multi scaled supervision used in 2nd paper.
- Bi-Directional pyramid supervision provides better usage of features extracted from the images. Main backbone suggested in this paper is **EfficientDet.**
- BiFPN extracts multi-scaled features while also coupled with the EfficientNet feature extractor.

2.5 DenseNet Structure Network for Pixel-Wise Face Anti-Spoofing

- DenseNet was specially developed to improve accuracy caused by the vanishing gradient in high-level neural networks due to the long distance between input and output layers & the information vanishes before reaching its destination.
- It connects the layers internally and provides maximum information flow, thus reducing the chances of diminishing gradient.
- We have used this strategy in place of pyramid supervision which will be further discussed in upcoming sections.

Chapter 3

Objectives

3.1 Objectives of the work

- Our initial focus on our research was primarily on understanding and exploring the assigned topic of face anti-spoofing.
- We have tried to analyze the previous work done in this field, and understand various techniques which are being used to prevent presentation attacks.
- Among the following three papers that were initially assigned to us for reference purposes , the one that caught our eye was the **Revisiting Pixel Wise Supervision for Face Anti Spoofing** :-
    1. Cross Central Difference Networks
    2. **Revisiting Pixel Wise Supervision for Face Anti Spoofing**
    3. Zero Shot Face Anti Spoofing using visual transformers
- In Pixel Wise Supervision we studied the base architecture where we learnt how binary masks were being generated from input images and how binary scores were calculated. Alongside we also learnt how different convolutional layers (AveragePooling Layers and Conv2D Layers) were connected with the final Dense Layer where weight updation is being performed.
- Upon further research we learnt about the following 2 neural network architectures that are being implemented for the task of face anti-spoofing. We have already explained the literature on these networks previously and how they have shown promising positive results.
    1. Bi-Directional Feature Pyramid Networks (Bi-FPN)
    2. DenseNet Architecture
- From here we got our first **2 objectives** where we could experiment our current understanding: -
    1. **Implementing Bi- FPN in the original pixel wise supervision architecture**
    2. **Implementing DenseNet in the original pixel wise supervision architecture**
- In the final stages of our research project, our focus shifted to a few other techniques which do not require too much computational power and could work on small sized datasets.
- We focussed on siamese networks and how they can be used for the task of anti-spoofing. This brought us to our next 2 tasks :-
    3. **Differentiating spoof from fakes using siamese network with triplet loss function.**
    4. **Differentiating spoof from fakes using siamese network with lossless triplet loss function.** Lossless triplet loss is a function which is built on top of basic triplet loss function. It’s working and why we have used it is explained in more detail in the following chapter.

Chapter 4

Design Proposals and Analysis

4.1 Use of Bi-Directional Pyramid Network in Pixel wise supervision.

Initially, the pixel wise supervision (i.e., binary mask supervision and depth map supervision) uses Pyramid supervision techniques for training of the model. However, it was suspected it might be the case that using this supervision technique possibly leaves many minute details, which in turn affects the accuracy and results of the model.

The **bidirectional feature pyramid network** (**BiFPN**) is used in an effort to extract multi-scaled features while also coupled with the **ResNet** feature extractor. While prior works have shown the significance of texture-based features for FAS, we hypothesize that due to the working principle of **BiFPN**, we could potentially extract features that would contain textural information that is important for this task.

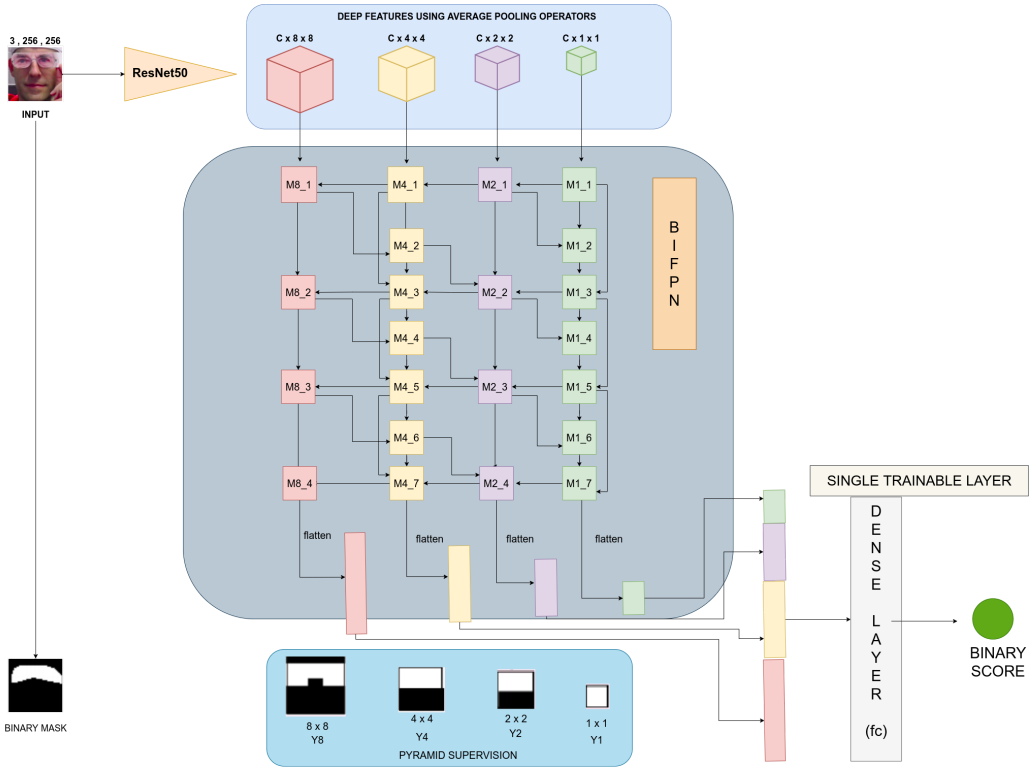


Fig-2 : Our Bi Directional Pyramid network

4.2 Use of Dense Connection Structure with Four layers

In this approach, we have used densenet architecture.

Dense connection architecture (Dense Convolutional Network) is a network architecture that **focuses on deepening deep learning networks while also making them more effective to train by employing shorter connections between layers**.

This is used so as to remove common machine learning training problems such as underfitting, overfitting and diminishing of generated weights.

We hypothesize that due to the working principle of Dense connection structure , we could potentially train the model without facing any problems regarding weight updation and could train even deeper than the regular multi-scale pyramid supervision that we discussed earlier.

The dense connection architecture employed along side multi-scale pyramid supervision is shown as below:-

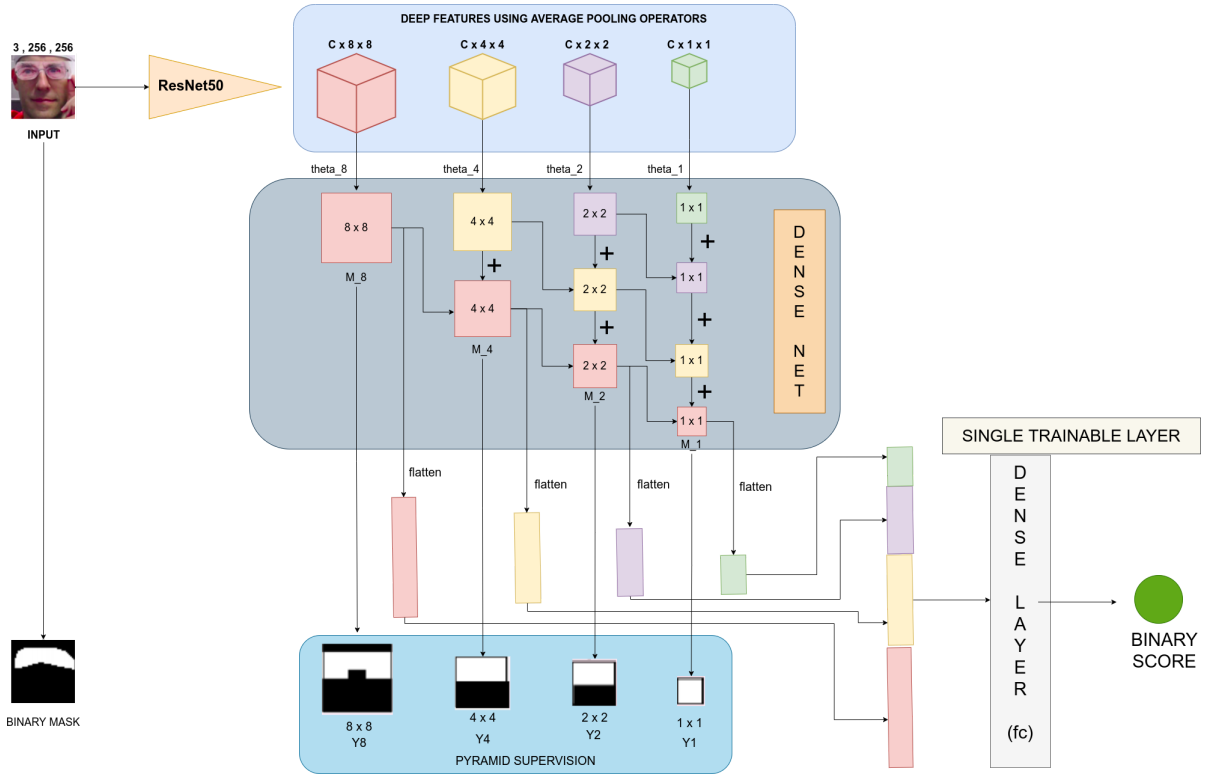


Fig-3 : Conv2D Layers are concatenated together with previous Conv2D layers so as to transfer binary features of the input image in the forward pass. Each forward layer is more enriched with binary features than the previous layer.

4.3 Siamese network

In this approach, we have used the **Siamese Network**.

A Siamese Neural Network is a class of neural network architectures that **contain two or more *identical* subnetworks**. Siamese Neural Networks, works on a similarity function.

There is a comparison made between the inputs to each subnetworks, and works on comparison principle. This approach is different from the two approaches discussed above, as there is no generation of binary mask and no calculation of binary score is performed. One major key difference that can be pointed out in the above 2 approaches and the current approach, lies in the final output of the CNN's. In the case of Pixel Wise Supervision, we get binary scores. We can check what type of attack is employed using the binary score. We definitely can't do this using the Siamese network. For e.g in pixel wise supervision, if we get a binary score of **0.2** , it may be the case that the attack was a **replay attack** or if the binary score was **0.5**, it may be the case the attack was a **partial print attack**. We can categorize attacks based on binary score. In the Siamese network, we pretty much just compare features of two images and give the answer in a yes (1) or no (0) format.

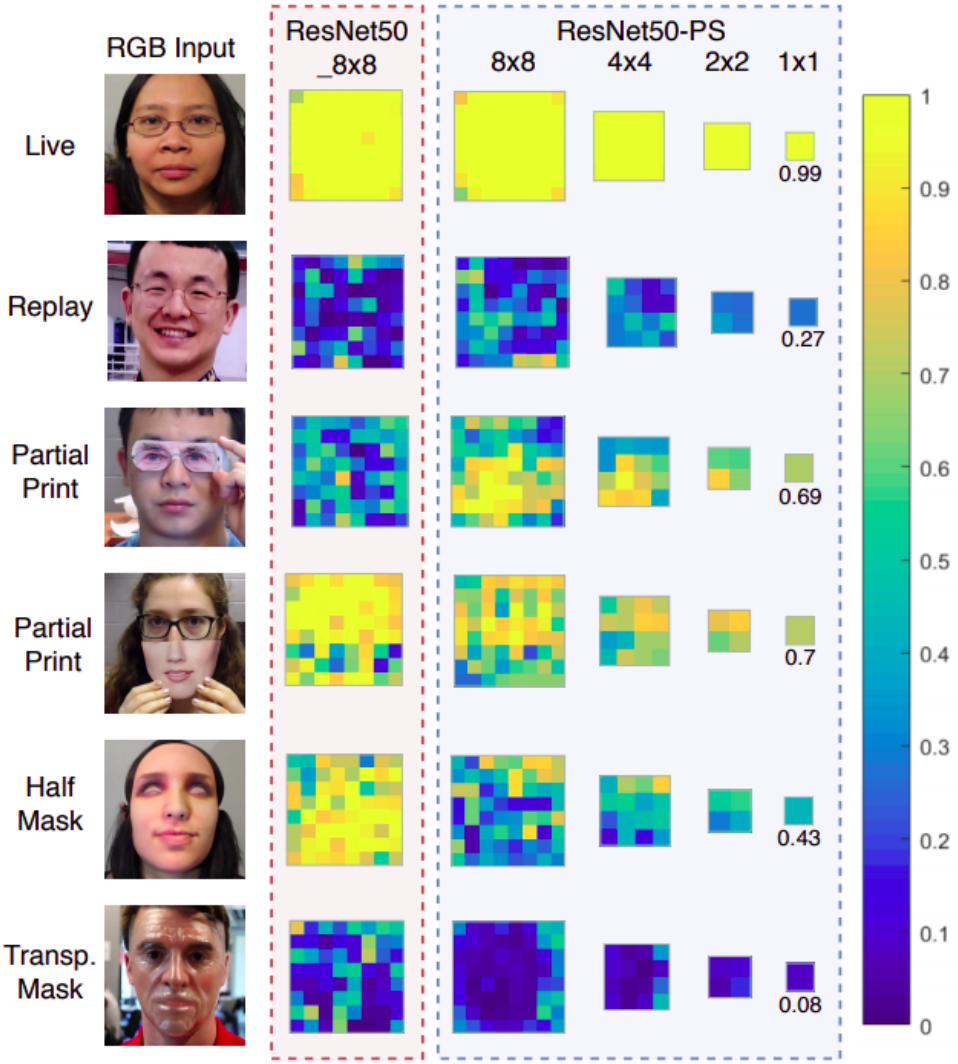


Fig-4 : Prediction of spoof attack class based on binary score generated by pixel wise supervision

In siamese network, we take an input image of a person which is a real image and find out the features of that image, then, we take the same network without performing any updates on weights or biases and input an image of the same person but its spoof and again predict its features.
It is also important to note that the outputs generated in both types of experiments involving Pixel Wise Supervision and Siamese Network are between 0 and 1. But they represent different meanings. In case of Pixel Wise supervision, we get output as binary score because of the sigmoid function we are using in the last dense layer. Each image will have a different binary score on which we can classify the attack type as explained above. We can't do this in the siamese network since output in this case represents the distance between the features of positive (real) and negative (spoof) image, which in turn is just a comparison of similarity of 2 input images. We specify a threshold in this case just to see how low we can get between 0 and 1 so as our network could predict a real image correctly and vice-versa. We can't classify attack type in our experiments involving the Siamese Network.

Now, we compare these two encodings to check whether there is a similarity between the two images. These two encodings act as a latent feature representation of the images. Real and spoof Images with the same person have different features/encodings. Using this, we train our model to identify real and spoof images.

This network is able to learn on low data. It has generally a few different types of loss functions :-

The loss functions are:

1. Contrastive loss
2. Triplet loss
3. Lossless Triplet loss
4. Quadruplet loss

In our experiments which use the Siamese network, we have used only **Triplet loss** and another version of triplet loss called **lossless triplet loss** which is explained further.

There are anchor, positive and negative images. anchor and positive are images of the same person (in our case both anchor as well as positive images are Real images) and negative is a spoof image.

Triplet Loss: Basic concept is that we are trying to minimize the distance between anchor image and the positive image, while at the same time we try to increase the distance between the anchor and negative image, and basically this is the entire learning concept of Siamese Networks.

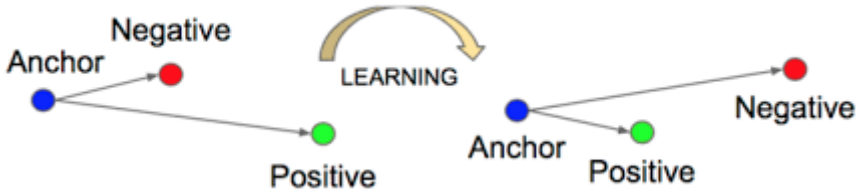


Fig-5 : Siamese network training procedure

$$Loss = \sum_{i=1}^{N} [\, || f_i^a - f_i^p ||_2^2 - || f_i^a - f_i^n ||_2^2 + \alpha \,]$$

*where **a** is an anchor input, **p** is a positive input of the same class **a** and **n** is a negative input of a different class from **a**. **α** is a margin between positive and negative pairs, and **f** is an embedding.*

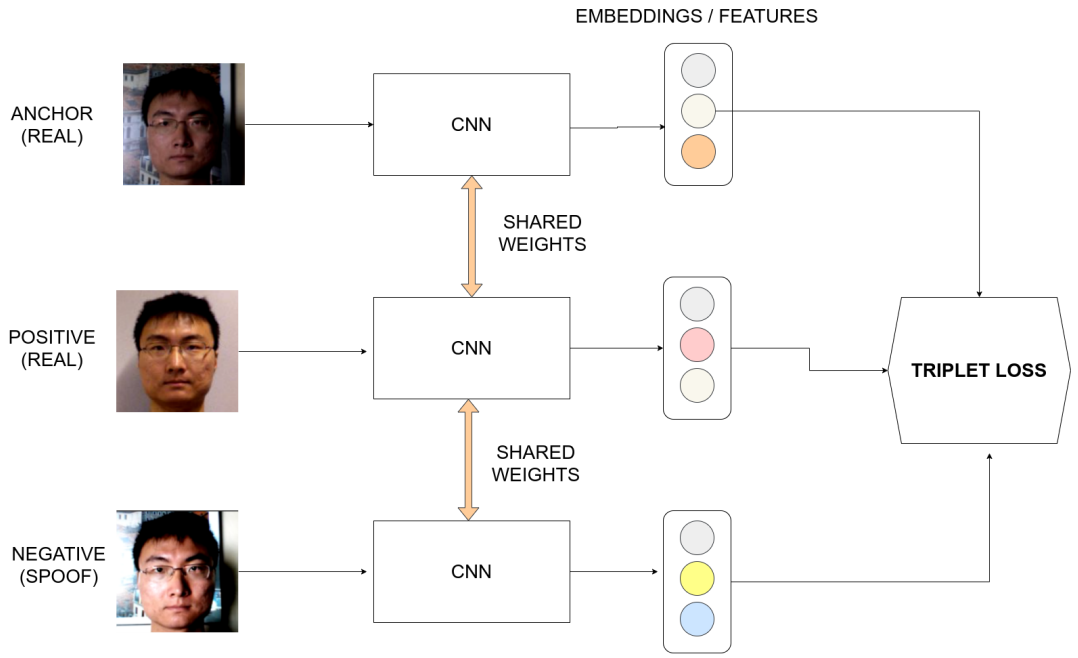


Fig-6 : Siamese network using triplet loss

Lossless Triplet Loss: Lossless Triplet Loss is basically an extension to the triplet loss function. How Lossless Triple Function works can be better explained by discussing a major flaw in the basic triplet loss function that we discussed earlier.

```
def loss_function(x, alpha = 0.2):
    # Triplet Loss function.
    anchor,positive,negative = x
    # distance between the anchor and the positive
    pos_dist = K.sum(K.square(anchor-positive),axis=1)
    # distance between the anchor and the negative
    neg_dist = K.sum(K.square(anchor-negative),axis=1)
    # compute loss
    basic_loss = pos_dist-neg_dist+alpha
    loss = K.mean(K.maximum(basic_loss,0.0))
    return loss
```

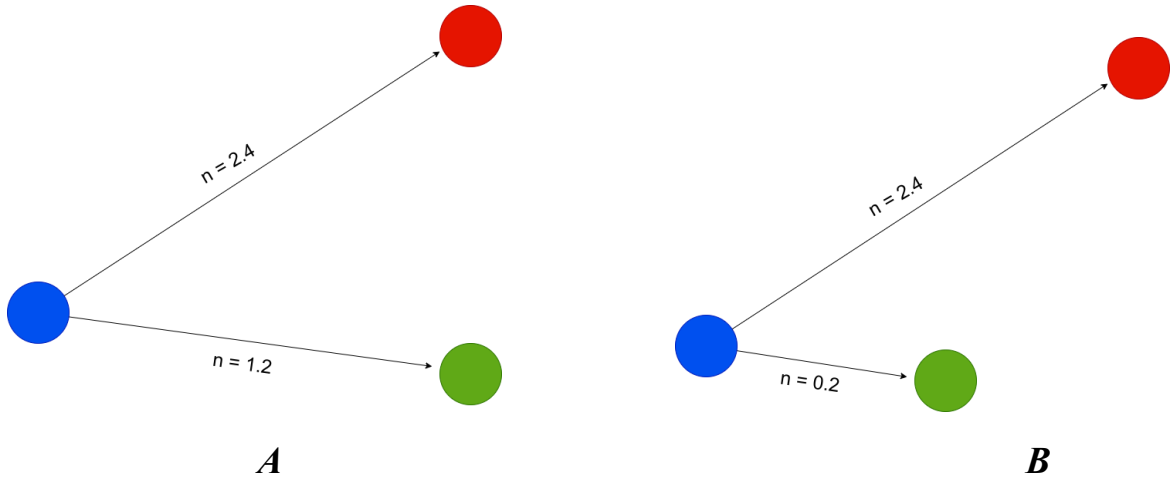
simple triplet loss function

The above code snippet is of a simple triplet loss function. The problem lies in :-

loss = K.maximum(basic_loss,0.0)

According to this code line, every time our loss gets below 0,we lose a lot of information.

If we consider 2 scenarios A and B



A ***B***

$$n = \| f_i^a - f_i^n \|_2^2$$
$$p = \| f_i^a - f_i^p \|_2^2$$
$$\alpha = 0.4$$
$$- - - - - - - - - - -$$
$$A = 1.2 - 2.4 - 0.4 = -1.6$$
$$B = 0.2 - 2.4 - 0.4 = -2.6$$

After the Max function both A and B now return 0 as their loss, which is a clear loss of information. By looking simply, we can say that B is better than A. So how can we improve this ?

To create a loss function that will collect information that has been "lost" below 0. Following some basic geometry, it was discovered that containing the N-dimensional space where the loss is calculated allows for more efficient control. As a result, the first step was to alter the model. The size of the final layer (Embedding layer) had to be kept under control. We can guarantee that each dimension will be between 0 and 1 by employing a sigmoid activation function instead of a linear activation function.

So now the new proposed formula becomes :-

$$Loss = \sum_{i=1}^{N} [(f_i^a - f_i^p)^2 - (f_i^a - f_i^n)^2 + N] \quad \text{... (1)}$$

The code snippet for lossless triplet loss is shown as below :-

```
def lossless_triplet_loss(y_true, y_pred, N = 3, beta=3, epsilon=1e-8):

    anchor,positive,negative = y_pred

    # distance between the anchor and the positive
    pos_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,positive)),1)
    # distance between the anchor and the negative
    neg_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,negative)),1)

    #Non Linear Values
```

```
    # -ln(-x/N+1)
    pos_dist = -tf.math.log(-tf.divide((pos_dist),beta)+1+epsilon)
    neg_dist = -tf.math.log(-tf.divide((N-neg_dist),beta)+1+epsilon)

    # compute loss
    loss = neg_dist + pos_dist

    return loss
```

Here N is the number of dimensions of the embedding vector. This equation is very much similar to our previous loss function, but by having a Sigmoïde and a proper setting for N, we can guarantee that the value will stay above 0.

On further tests, it was found that the error rate was not decreasing steadily and was rather extremely slow. To improve this, we basically need to make the error cost high. This could be easily done using a polynomial cost function instead of a linear one.

The Polynomial cost function looks something like this :-

$$-\ln\left(\frac{-x}{N}+1\right) \quad \ldots\ldots\ldots (2)$$

Employing (2) in (1) we get the final loss function as below :-

$$Loss = \sum_{i=1}^{N}\left[-\ln\left(-\frac{(f_i^a - f_i^p)^2}{\beta}+1+\epsilon\right) - \ln\left(-\frac{N-(f_i^a - f_i^p)^2}{\beta}+1+\epsilon\right)\right]$$

lossless triplet loss function

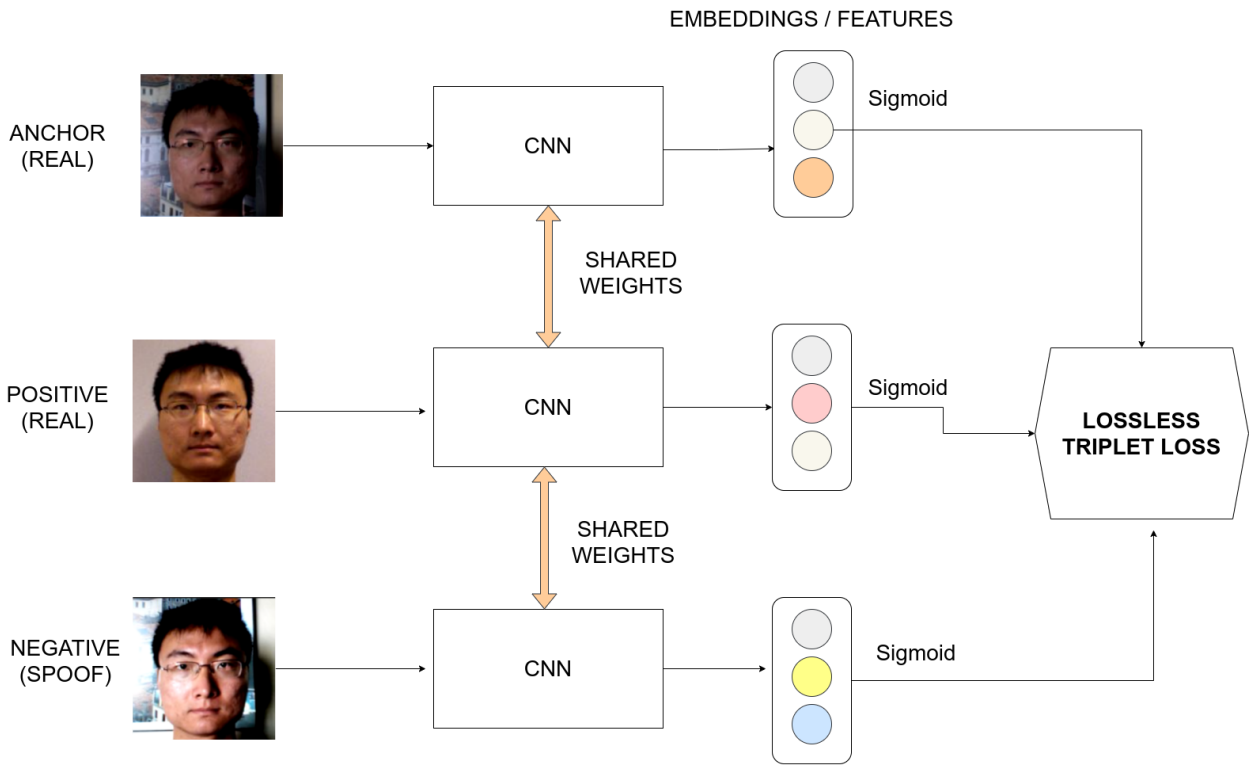


Fig-7 : Siamese network using lossless triplet loss

Chapter 5

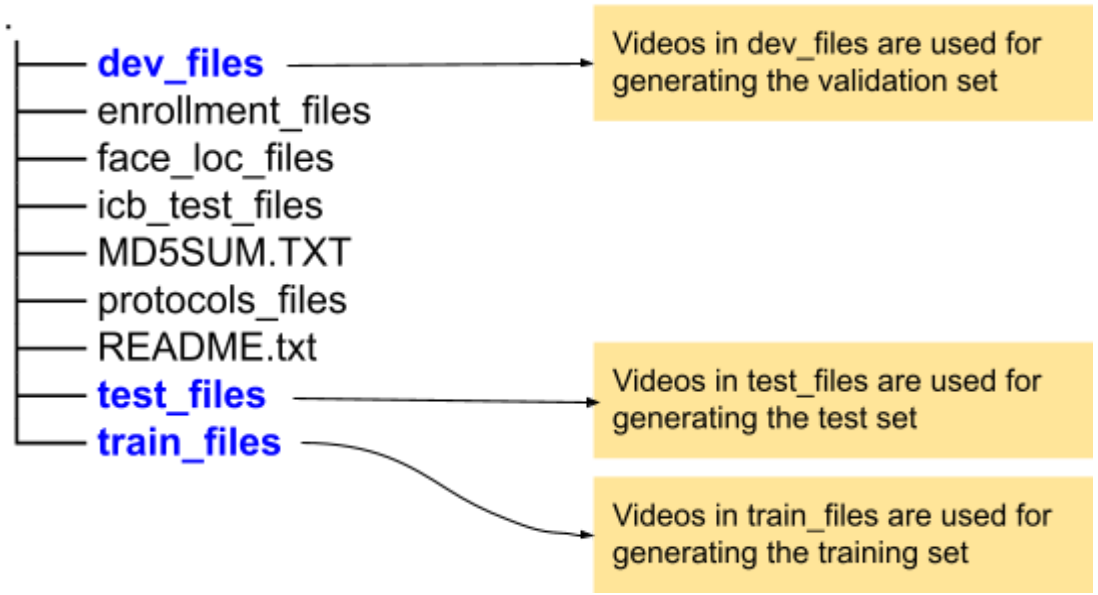
Experiments

5.1 Datasets

For all experiments we have used **Replay Attack Dataset**.

It is important to note that in some of the datasets that we have generated from **Replay Attack Dataset,** we have used the naming convention of folder containing negative images as **attack** or **spoof** interchangeably but both mean exactly the same.

Original Directory Structure for Replay Attack Dataset is as follows :



From the Replay attack we only choose 3 main folders for our use-case :

1. **test_files**
   - test
     - attack
       - fixed – 200 videos
       - hand – 200 videos
     - real – 80 videos

2. **train_files**
   - train
     - attack
       - fixed – 150 videos
       - hand – 150 videos
     - real – 60 videos

3. **dev_files**
   - devel
     - attack
       - fixed – 150 videos
       - hand – 150 videos
     - real – 60 videos

It is important to note that all folders i.e **train_files**, **test_files** and **dev_files** include multiple attack (spoof) and real videos of multiple different people.

We have done 2 types of transformation to Replay Attack Dataset to best fit our experiment needs:

1. The first transformation is done so as to fit the data input requirements for following experiments :-
   - Pixel Wise Supervision using Vanilla Pyramid supervision
   - Pixel Wise Supervision using BiFPN (Bi-Directional Pyramid Network)
   - Pixel Wise Supervision using Dense connection Architecture

   .

   We create training, test and validation sets. The final dataset directory structure is as follows:-

   - **test**
     - attack – 91,799 Images/Frames
     - real – 29,229 Images/Frames
   - **train**
     - attack – 69,215 Images/Frames
     - real – 22,490 Images/Frames
   - **val**
     - attack – 68,467 Images/Frames
     - real – 22,485 Images/Frames

   The process of transformation of videos to corresponding Images/Frames for the test set is shown below. The process is similar for train and validation sets also.

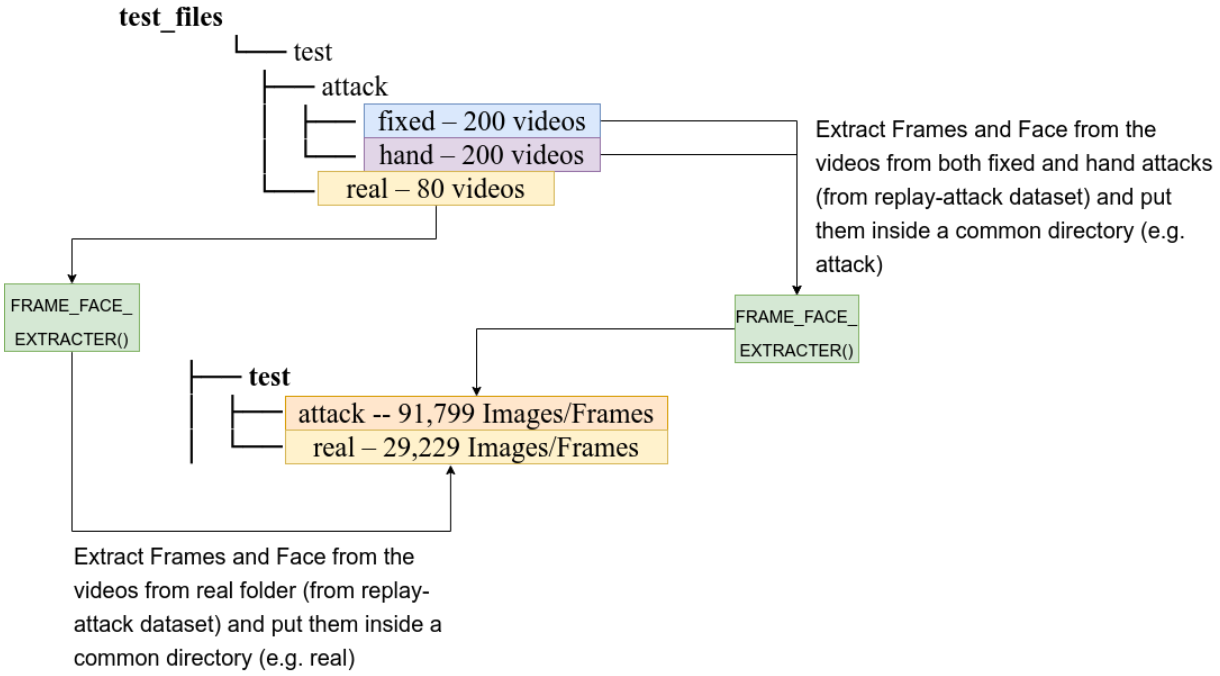


Fig-8 : Transformation of videos to corresponding Images/Frames

Note: The amount frames generated depends upon the methodology used while processing videos. For our needs frames close to 300 per video were more than enough.

2. The second transformation is done so as to fit the data input requirements for following experiments :-

- Anti face spoofing using Siamese Network using triplet loss
- Anti face spoofing using Siamese Network using lossless triplet loss

We create training and test sets only. The final dataset directory structure is as follows :-

.
- **test**
  - images
    - multiple
      - client_009 – 8 to 10 Real Images/Frames of same Client/Person (009)
      - client_011 – 8 to 10 Real Images/Frames of same Client/Person (011)
      - client_019 – 8 to 10 Real Images/Frames of same Client/Person (019)
      - ….
      - ….
    - single – Any number of Images (Real or Spoof) of different clients
- **train**
  - images
    - client001
      - real – 1,500 Real Images/Frames of same Client/Person (001)
      - spoof – 4,700 Spoof Images/Frames of same Client/Person (001)
    - client002
      - real – 1,500 Real Images/Frames of same Client/Person (002)
      - spoof – 4,694 Spoof Images/Frames of same Client/Person (002)
    - client004
    - …..
    - …..
    - …..

The process of transformation of videos to corresponding Images/Frames for the train set is shown in the following diagram:-

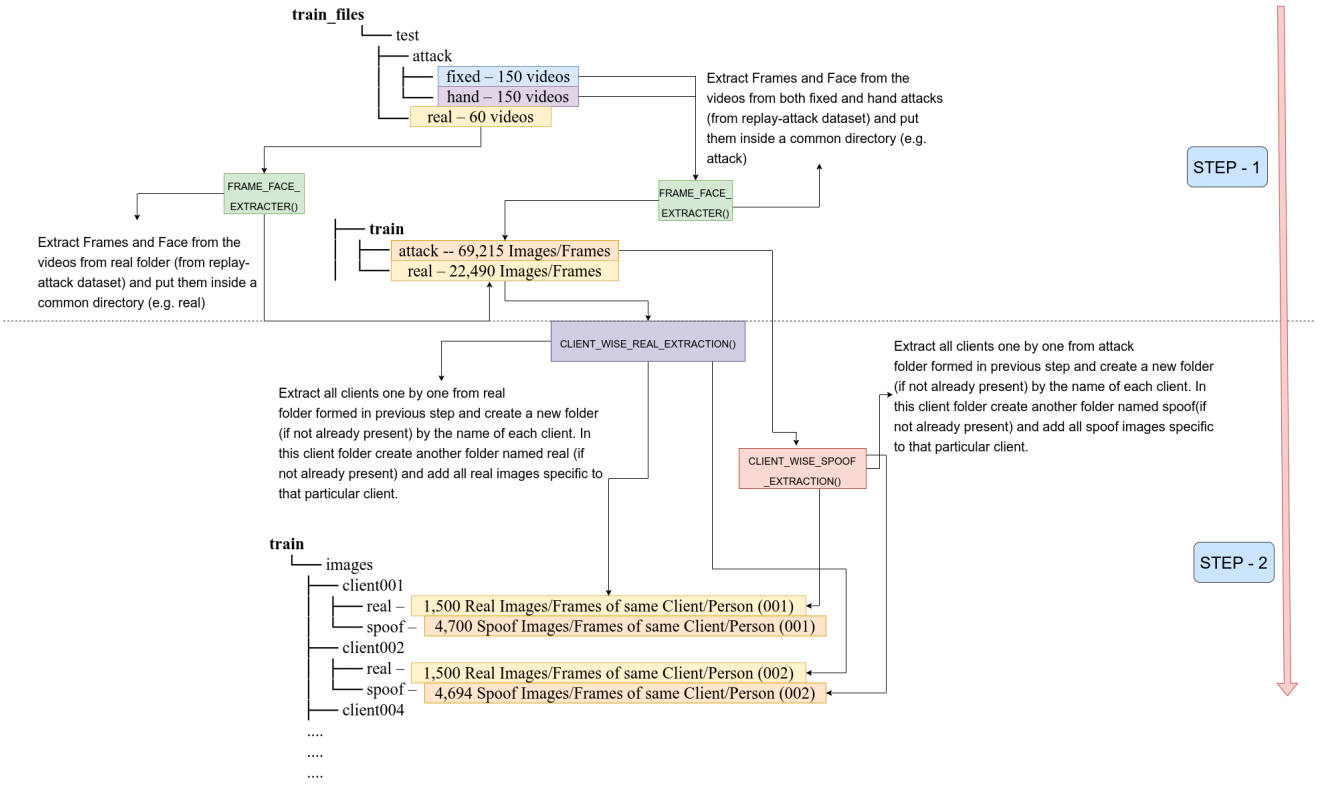


Fig-9 : Transformation of replay dataset to fit siamese network architecture

The Steps 1 and 2 in the above figure are defined as below:

Step-1 : Transform original relay-attack dataset to corresponding real/spoof frames or images.
Step-2: Transform the directory structure of real/spoof frames or images formed in previous step so as to place frames in per client folder structure. Each client folder contains a real and spoof folder, where the real folder consists of real images specific to that particular client and the spoof folder consists of spoof images specific to that particular client .

5.2 Experimental Setup:

For training of all neural networks we have used Google Collaboratory.

Disk Information :-

```
Filesystem      Size  Used Avail Use% Mounted on
-------------------------------------------------
overlay         226G   38G  189G  17% /
tmpfs            64M     0   64M   0% /dev
shm             5.8G     0  5.8G   0% /dev/shm
/dev/root       2.0G  1.2G  812M  59% /sbin/docker-init
tmpfs           6.4G   32K  6.4G   1% /var/colab
/dev/sda1       233G   41G  192G  18% /etc/hosts
tmpfs           6.4G     0  6.4G   0% /proc/acpi
tmpfs           6.4G     0  6.4G   0% /proc/scsi
tmpfs           6.4G     0  6.4G   0% /sys/firmware
```

CPU Specifications :-

```
processor : 0
vendor_id : GenuineIntel
cpu family: 6
model     : 79
model name: Intel(R) Xeon(R) CPU @ 2.20GHz
stepping  : 0
microcode : 0x1
cpu MHz   : 2199.998
cache size: 56320 KB
physical id : 0
siblings : 2
core id : 0
cpu cores : 1
apicid : 0
initial apicid  : 0
fpu : yes
fpu_exception : yes
cpuid level : 13
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht
syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl
xtopology nonstop_tsc eagerfpu pni pclmulqdq ssse3 fma
cx16 sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx
f16c rdrand hypervisor lahf_lm abm 3dnowprefetch
```

```
fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms rtm
rdseed adx xsaveopt



bogomips  : 4399.99
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
--------------------------------------------------------
--
processor : 1
vendor_id : GenuineIntel
cpu family: 6
model : 79
model name: Intel(R) Xeon(R) CPU @ 2.20GHz
stepping  : 0


microcode : 0x1
cpu MHz : 2199.998
cache size: 56320 KB
physical id : 0
siblings  : 2
core id : 0
cpu cores : 1
apicid : 1
initial apicid  : 1
fpu : yes
fpu_exception : yes
cpuid level : 13
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht
syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl
xtopology nonstop_tsc eagerfpu pni pclmulqdq ssse3 fma
cx16 sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx
f16c rdrand hypervisor lahf_lm abm 3dnowprefetch
fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms rtm
rdseed adx xsaveopt
bogomips  : 4399.99
clflush size    : 64
cache_alignment : 64
```

```
address sizes   : 46 bits physical, 48 bits virtual
```

Memory Specs :-

```
MemTotal:       13341992 kB
MemFree:         8670124 kB
MemAvailable:   12753184 kB
Buffers:          115848 kB
Cached:          3865368 kB
SwapCached:            0 kB
Active:          1160124 kB
Inactive:        3077576 kB
Active(anon):     256804 kB
Inactive(anon):      324 kB

Active(file):     903320 kB
Inactive(file):  3077252 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:               772 kB
Writeback:             0 kB
AnonPages:        256548 kB
Mapped:           167332 kB
Shmem:               656 kB
Slab:             377108 kB
SReclaimable:     355904 kB
SUnreclaim:        21204 kB
KernelStack:        3040 kB
PageTables:         4044 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     6670996 kB
Committed_AS:    1194088 kB
VmallocTotal:   34359738367 kB
```

```
VmallocUsed:           0 kB
VmallocChunk:          0 kB
AnonHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       59380 kB
DirectMap2M:     4134912 kB
DirectMap1G:    11534336 kB
```

GPU Specs:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   45C    P8    10W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

5.3 Results and Discussion:

1. Pixel-Wise Supervision Using vanilla pyramid supervision
   Base Dataset - Replay Attack
   In this experiment, while selecting thresholds, it is important to note that if the final **output binary score** is **less than** the selected **threshold**, then the input image is **real** else **spoof**.

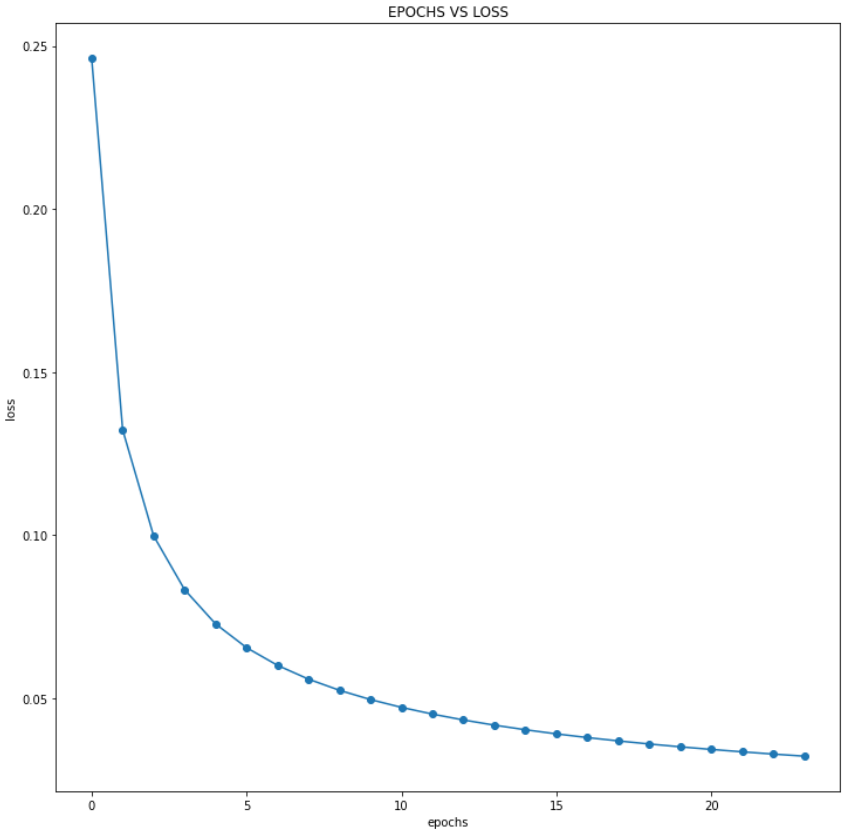


Fig-10 : Epochs vs Loss for vanilla pixel wise supervision

Table - 1 Validation Metrics (vanilla pixel wise supervision)

thresh	apcer	bpcer	acer
0	0	1	0.5
0.1	0.00009	0.011869	0.005979
0.2	0.001125	0.007107	0.004116
0.3	0.003832	0.005033	0.004433
0.4	0.007226	0.004427	0.005826
0.5	0.012225	0.004178	0.008201
0.6	0.017609	0.003754	0.010682
0.7	0.02307	0.003183	0.013126

0.8	0.03018	0.002282	0.016231
0.9	0.043529	0.000886	0.022208
1	1	0	0.5

*** Red represents maximum value** *** Brown represents minimum value**

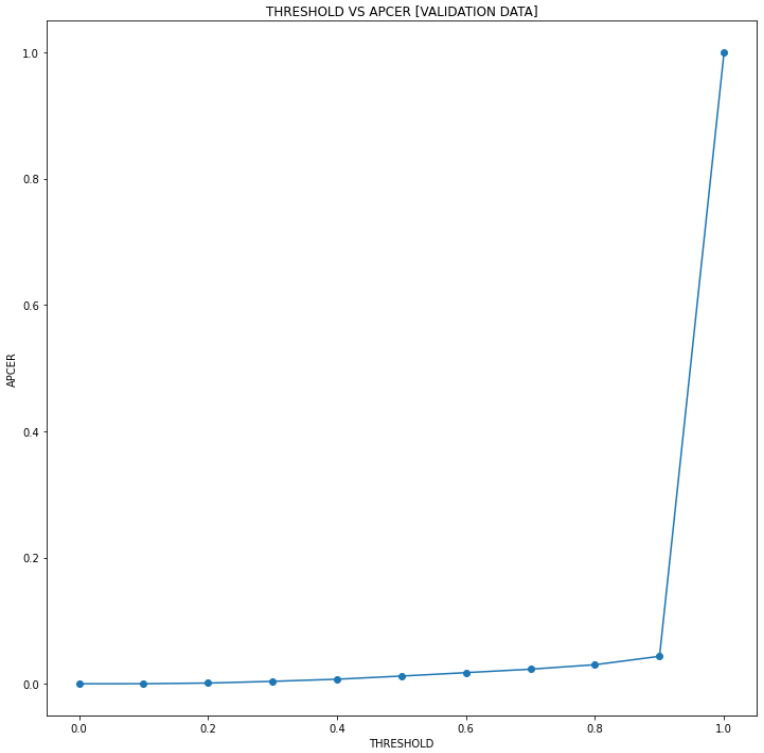


Fig-11 : Threshold vs APCER (validation data) for vanilla pixel wise supervision

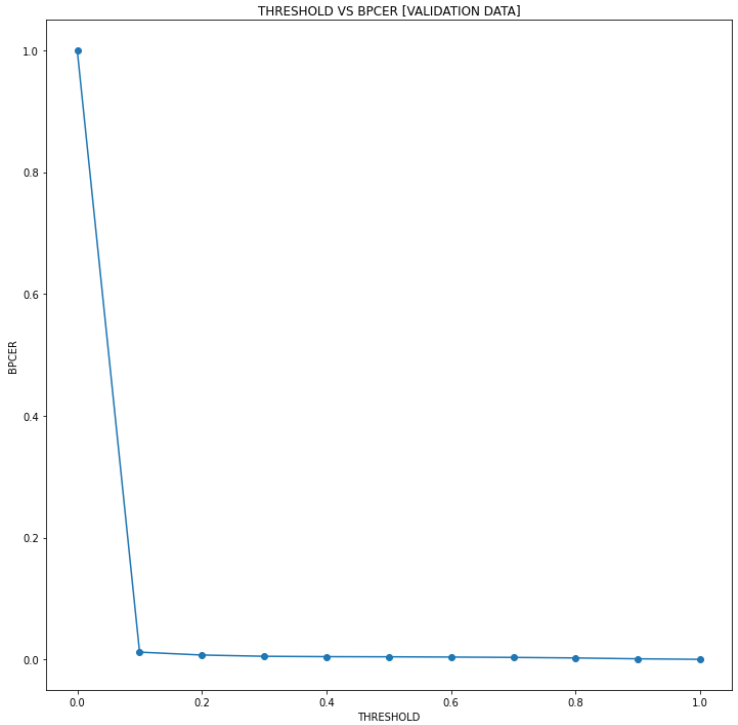


Fig-12 : Threshold vs BPCER (validation data) for vanilla pixel wise supervision

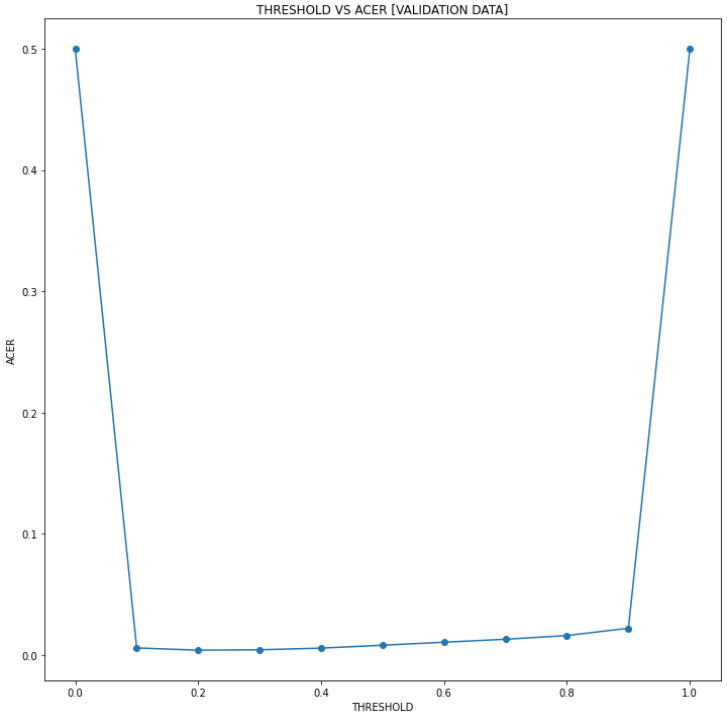


Fig-13 : Threshold vs ACER (validation data) for vanilla pixel wise supervision

Table - 2 Validation Metrics (vanilla pixel wise supervision)

thresh	matrix	recall	precision	f1_score	accuracy
0	[[0 67401] [0 22071]]	nan	0	nan	0.2466805258
0.1	[[66687 801] [2 22174]]	0.99997001	0.9881312233	0.9940153678	0.9910443433
0.2	[[67338 482] [25 22203]]	0.9996288764	0.9928929519	0.9962495284	0.9943696695
0.3	[[67208 340] [85 22095]]	0.9987368671	0.9949665423	0.9968481397	0.9952634629
0.4	[[67466 300] [161 22121]]	0.9976192941	0.9955730012	0.9965950972	0.9948805082
0.5	[[67214 282] [271 21897]]	0.9959842928	0.9958219746	0.9959031271	0.9938325303
0.6	[[67667 255] [393 21925]]	0.9942256832	0.9962456936	0.9952346634	0.9928191489
0.7	[[67023 214] [510 21597]]	0.9924481365	0.9968172286	0.9946278845	0.99189649
0.8	[[67343 154] [669 21498]]	0.9901635006	0.9977184171	0.9939266027	0.9908212884
0.9	[[67645 60] [967 21248]]	0.9859062555	0.9991138025	0.9924660901	0.9885787367
1	[[67302 0] [22106 0]]	0.7527514316	1	0.8589368898	0.7527514316

*** Red represents maximum value** *** Brown represents minimum value**

Table - 3 Test Metrics (vanilla pixel wise supervision)

thresh	apcer	bpcer	acer
0	0	1	0.5
0.1	0	0.017015	0.008507
0.2	0.000035	0.011537	0.005786
0.3	0.000138	0.009631	0.004885
0.4	0.000276	0.008326	0.004301
0.5	0.000554	0.007118	0.003836
0.6	0.000901	0.006223	0.003562
0.7	0.001425	0.005263	0.003344
0.8	0.00253	0.004465	0.003498
0.9	0.006112	0.003487	0.004799
1	1	0	0.5

* **Red represents maximum value** * **Brown represents minimum value**

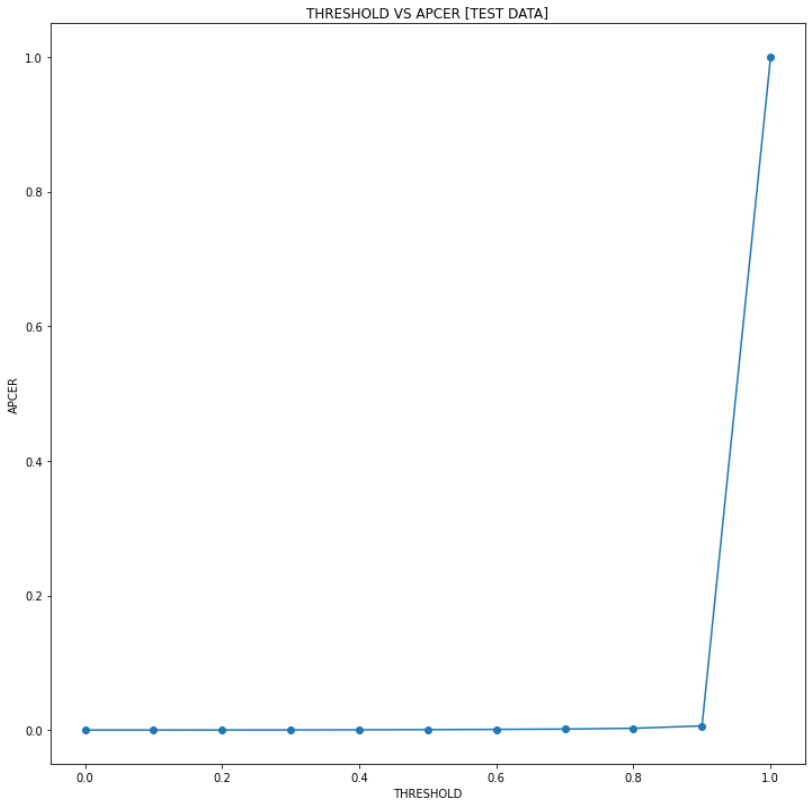


Fig-14 : Threshold vs APCER (test data) for vanilla pixel wise supervision

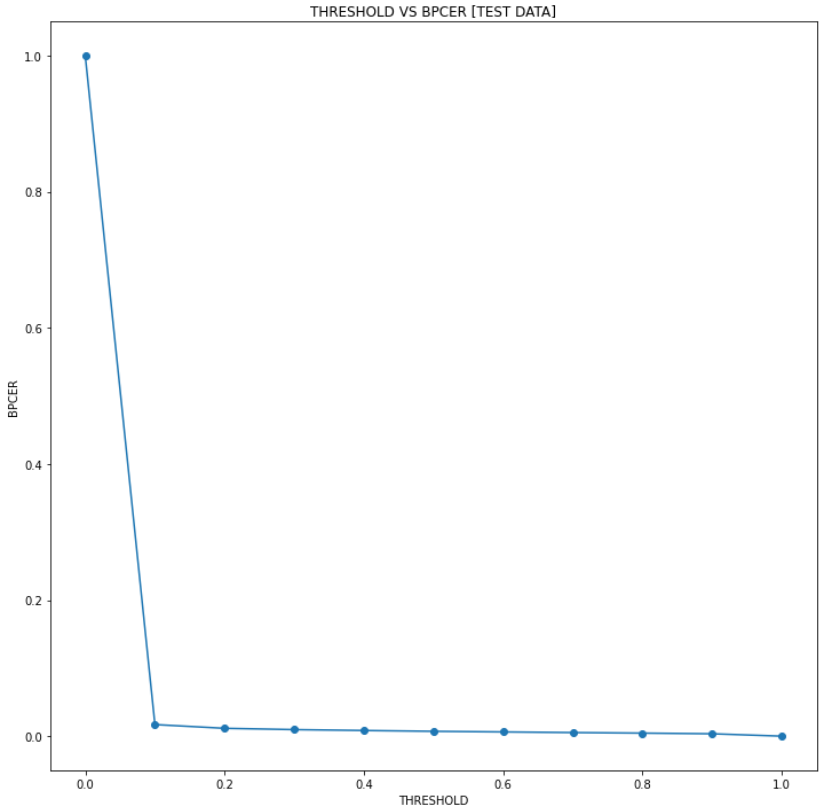


Fig-15 : Threshold vs BPCER (test data) for vanilla pixel wise supervision

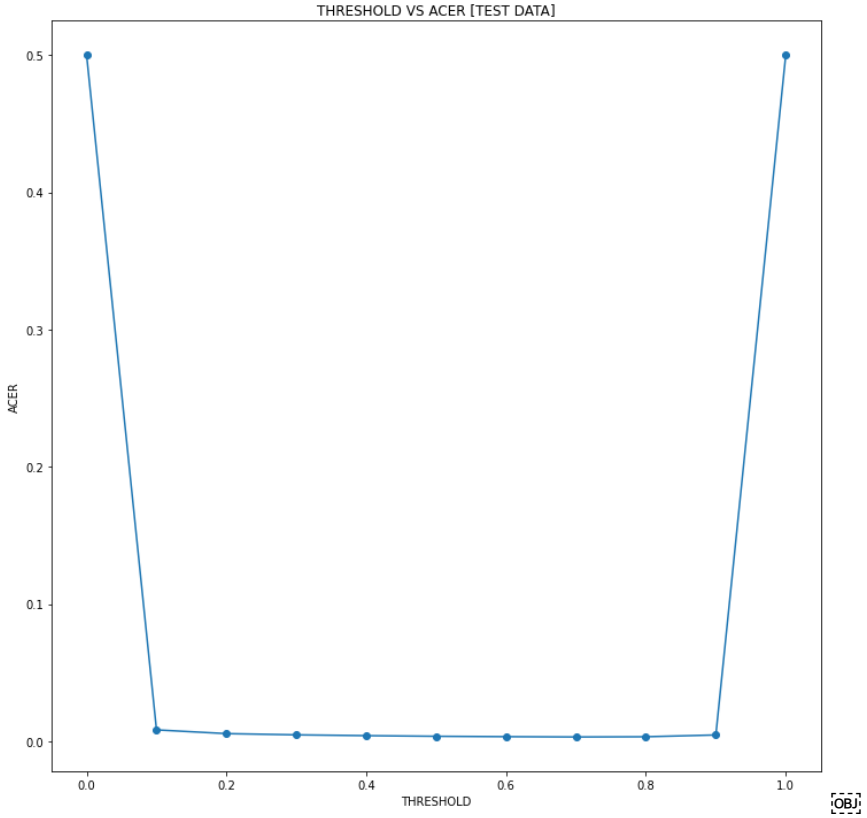


Fig-16 : Threshold vs ACER (test data) for vanilla pixel wise supervision

Table - 4 Test Metrics (vanilla pixel wise supervision)

thresh	matrix	recall	precision	f1_score	accuracy
0	[[0 90716] [0 28900]]	nan	0	nan	0.2416064741
0.1	[[89144 1543] [0 28865]]	1	0.9829854334	0.9914197218	0.9870934823
0.2	[[89445 1044] [1 28806]]	0.9999888201	0.9884626861	0.9941923472	0.9912402763
0.3	[[90181 877] [4 29002]]	0.9999556467	0.9903687759	0.9951391226	0.9926622468
0.4	[[90045 756] [8 28935]]	0.9999111634	0.9916741005	0.995775598	0.9936197221
0.5	[[89970 645] [16 28857]]	0.9998221946	0.9928819732	0.9963399981	0.9944680637
0.6	[[90070 564] [26 28828]]	0.9997114189	0.9937771697	0.9967354617	0.9950622657
0.7	[[89786 475] [41 28738]]	0.9995435671	0.9947374835	0.9971347341	0.9956653226
0.8	[[90291 405] [73 28783]]	0.9991921562	0.9955345329	0.9973599912	0.9960017398
0.9	[[90594 317] [177 28784]]	0.998050038	0.9965130732	0.9972809634	0.9958789375
1	[[90501 0] [28795 0]]	0.7586256035	1	0.8627482757	0.7586256035

* **Red represents maximum value** * **Brown represents minimum value**

2. Pixel-Wise Supervision Using DenseNet Architecture
   Base Dataset - Replay Attack
   In this experiment, while selecting thresholds, it is important to note that if the final **output binary score** is **less than** the selected **threshold**, then the input image is **real** else **spoof**.

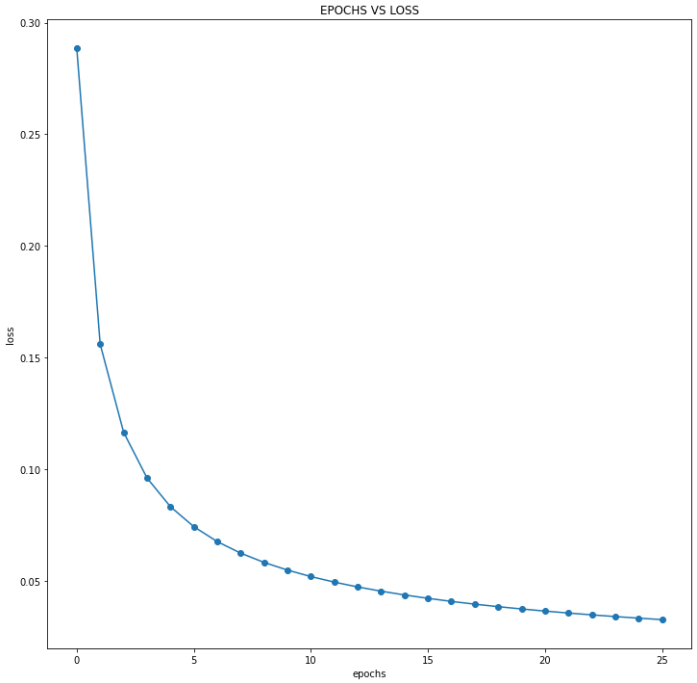


Fig-17 : Epoch vs Loss DenseNet Architecture

Table - 5 Validation Metrics (DenseNet supervision)

thresh	apcer	bpcer	acer
0	0	1	0.5
0.05	0.005975	0.012716	0.009346
0.1	0.012759	0.009338	0.011049
0.15	0.017657	0.007685	0.012671
0.2	0.022823	0.006255	0.014539
0.25	0.026957	0.004765	0.015861
0.3	0.031899	0.00388	0.017889
0.35	0.036706	0.003127	0.019917
0.4	0.041468	0.00239	0.021929
0.45	0.046904	0.001859	0.024382
0.5	0.051307	0.001313	0.02631
0.55	0.056429	0.001003	0.028716
0.6	0.061641	0.000841	0.031241
0.65	0.0682	0.000664	0.034432
0.7	0.075478	0.000502	0.03799
0.75	0.082038	0.000325	0.041181
0.8	0.089541	0.000207	0.044874
0.85	0.097987	0.000118	0.049053

0.9	0.107692	0.000044	0.053868
0.95	0.126202	0.00003	0.063116
1	1	0	0.5

*** Red represents maximum value** *** Brown represents minimum value**

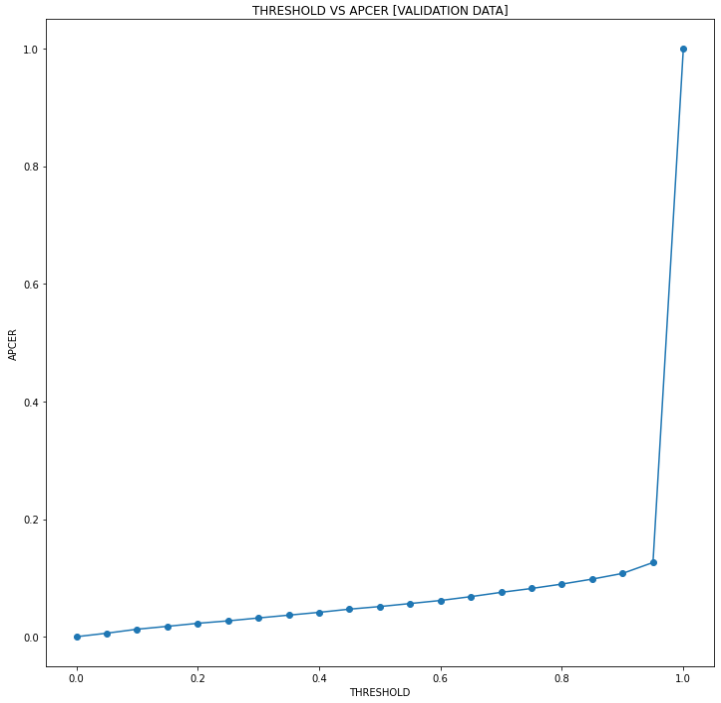


Fig-18 : Threshold vs APCER (validation data) for DenseNet supervision

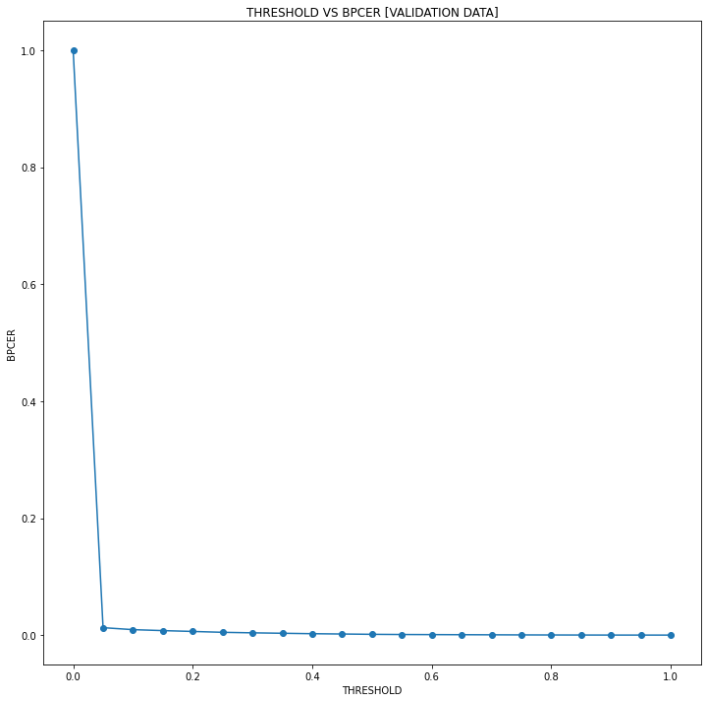


Fig-19 : Threshold vs BPCER (validation data) for DenseNet supervision

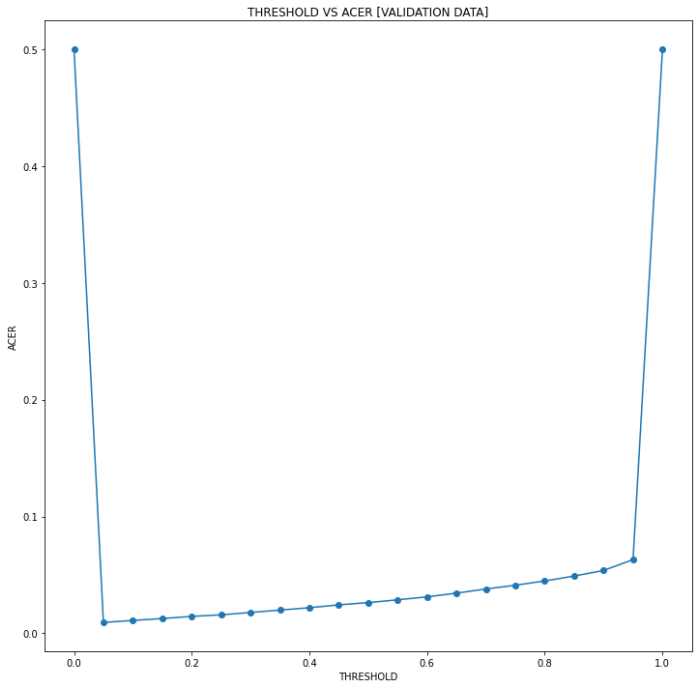


Fig-20 : Threshold vs ACER (validation data) for DenseNet supervision

Table - 6 Validation Metrics (DenseNet supervision)

matrix	thresh	recall	precision	f1_score	accuracy
[[0 67790] [0 22258]]	0	nan	0	nan	0.2471792822
[[66928 862] [133 22125]]	0.05	0.998016731	0.9872842602	0.9926214859	0.9889503376
[[67157 633] [284 21974]]	0.1	0.9957889118	0.9906623396	0.9932190104	0.9898165423
[[67269 521] [393 21865]]	0.15	0.9941917177	0.9923145007	0.9932522222	0.9898498579
[[67366 424] [508 21750]]	0.2	0.9925155435	0.9937453902	0.9931300861	0.9896499645
[[67467 323] [600 21658]]	0.25	0.9911851558	0.9952352854	0.9932060917	0.9897499112
[[67527 263] [710 21548]]	0.3	0.9895950877	0.9961203717	0.9928470083	0.9891946517
[[67578 212] [817	0.35	0.9880546824	0.9968726951	0.9924441018	0.9885727612

21441]]					
[[67628 162] [923 21335]]	0.4	0.9865355721	0.997610267	0.9920420123	0.9879508706
[[67664 126] [1044 21214]]	0.45	0.9848052629	0.9981413188	0.9914284458	0.9870069296
[[67701 89] [1142 21116]]	0.5	0.9834115306	0.998687122	0.9909904635	0.9863295131
[[67722 68] [1256 21002]]	0.55	0.9817912958	0.9989969022	0.990319373	0.9852967306
[[67733 57] [1372 20886]]	0.6	0.9801461544	0.999159168	0.9895613426	0.9841306859
[[67745 45] [1518 20740]]	0.65	0.9780835367	0.9993361853	0.9885956528	0.9826425906
[[67756 34] [1680 20578]]	0.7	0.9758050579	0.9994984511	0.9875096556	0.9809657072
[[67768 22] [1826 20432]]	0.75	0.9737621059	0.9996754684	0.986548652	0.9794776119
[[67776 14] [1993 20265]]	0.8	0.9714343046	0.9997934799	0.9854098968	0.977711887
[[67782 8] [2181 20077]]	0.85	0.9688263797	0.9998819885	0.984109239	0.9756907427
[[67787 3] [2397 19861]]	0.9	0.9658469167	0.9999557457	0.9826054184	0.973347548
[[67788 2] [2809 19449]]	0.95	0.9602107738	0.9999704971	0.9796873984	0.9687833156
[[67790 0] [22258 0]]	1	0.7528207178	1	0.8589819942	0.7528207178

* **Red represents maximum value** * **Brown represents minimum value**

Table - 7 Test Metrics (DenseNet supervision)

thresh	apcer	bpcer	acer
0	0	1	0.5
0.05	0.006603	0.012861	0.009732
0.1	0.009473	0.010119	0.009796
0.15	0.010821	0.008963	0.009892
0.2	0.011513	0.008236	0.009874
0.25	0.012066	0.007542	0.009804
0.3	0.012688	0.007036	0.009862
0.35	0.013103	0.00654	0.009822
0.4	0.014071	0.006056	0.010063
0.45	0.014831	0.005759	0.010295
0.5	0.015385	0.005472	0.010429
0.55	0.016387	0.005142	0.010765
0.6	0.017701	0.004757	0.011229
0.65	0.018842	0.004614	0.011728
0.7	0.020156	0.004349	0.012252
0.75	0.021538	0.004041	0.01279
0.8	0.023336	0.003766	0.013551
0.85	0.025825	0.003501	0.014663
0.9	0.029525	0.003215	0.01637
0.95	0.035367	0.002863	0.019115
1	1	0	0.5

* **Red represents maximum value** * **Brown represents minimum value**

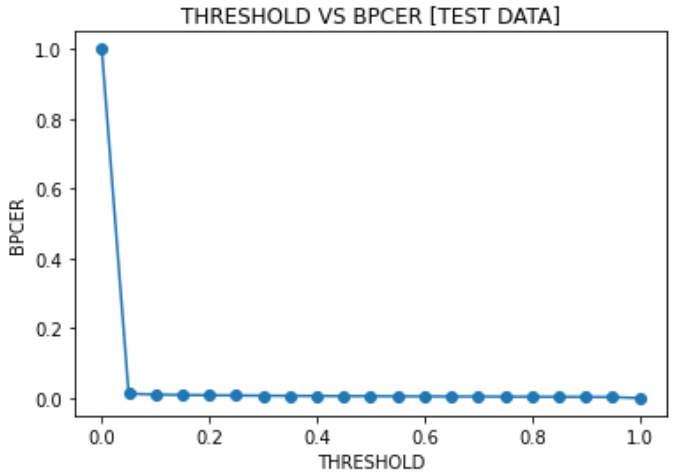


Fig-21 : Threshold vs BPCER (test data) for DenseNet supervision

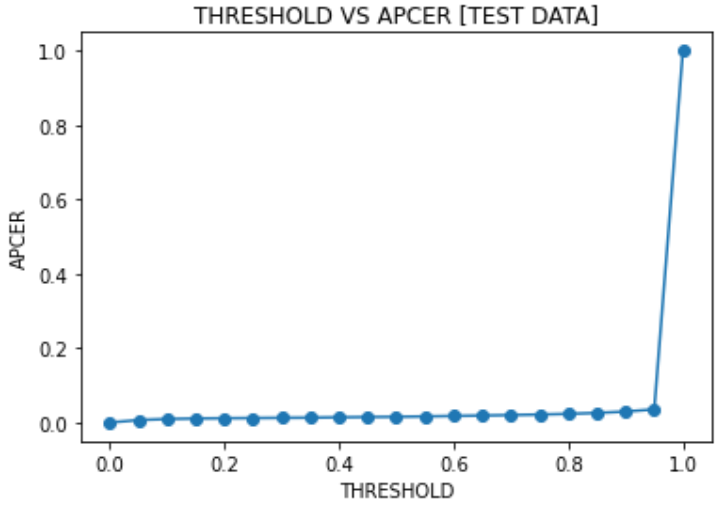


Fig-22 : Threshold vs APCER (test data) for DenseNet supervision

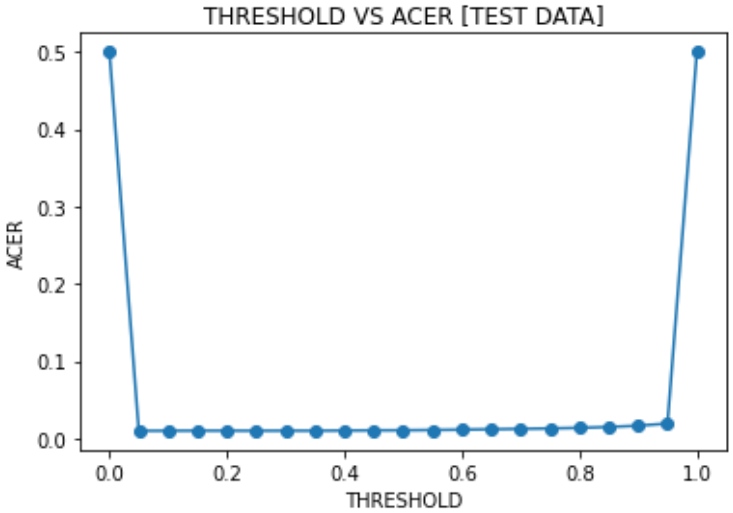


Fig-23 : Threshold vs ACER (test data) for DenseNet supervision

Table - 8 Test Metrics (DenseNet supervision)

matrix	thresh	recall	precision	f1_score	accuracy
[[0 90819] [0 28925]]	0	nan	0	nan	0.2415569882
[[89651 1168] [191 28734]]	0.05	0.9978740455	0.987139255	0.9924776238	0.9886507883
[[89900 919] [274 28651]]	0.1	0.9969614301	0.989880972	0.9934085849	0.9900370791
[[90005	0.15	0.9965344671	0.9910371178	0.99377819	0.9905882549

814] [313 28612]]					
[[90071 748] [333 28592]]	0.2	0.9963165347	0.991763838	0.9940349735	0.9909724078
[[90134 685] [349 28576]]	0.25	0.9961429219	0.9924575254	0.9942968086	0.9913649118
[[90180 639] [367 28558]]	0.3	0.9959468563	0.9929640274	0.9944532051	0.991598744
[[90225 594] [379 28546]]	0.35	0.9958169617	0.9934595184	0.9946368432	0.9918743319
[[90269 550] [407 28518]]	0.4	0.9955114915	0.9939439985	0.9947271275	0.9920079503
[[90296 523] [429 28496]]	0.45	0.9952714246	0.9942412931	0.9947560922	0.992049706
[[90322 497] [445 28480]]	0.5	0.9950973371	0.9945275768	0.9948123754	0.9921332175
[[90352 467] [474 28451]]	0.55	0.99478123	0.9948579042	0.9948195656	0.9921415687
[[90387 432] [512 28413]]	0.6	0.9943673748	0.9952432861	0.9948051376	0.9921165152
[[90400 419] [545 28380]]	0.65	0.9940073671	0.995386428	0.9946964195	0.9919494923
[[90424 395] [583 28342]]	0.7	0.9935938994	0.9956506898	0.9946212313	0.9918325762
[[90452 367] [623 28302]]	0.75	0.9931594839	0.9959589954	0.9945572696	0.9917323624
[[90477 342] [675 28250]]	0.8	0.9925947867	0.9962342682	0.9944111974	0.9915068813
[[90501 318] [747 28178]]	0.85	0.9918135192	0.99649853	0.994150505	0.9911060262

[[90527 292] [854 28071]]	0.9	0.9906545124	0.9967848138	0.9937102086	0.9904295831
[[90559 260] [1023 27902]]	0.95	0.9888296827	0.9971371629	0.9929660473	0.9892854757
[[90819 0] [28925 0]]	1	0.7584430118	1	0.8626301867	0.7584430118

* **Red represents maximum value** * **Brown represents minimum value**

3. Pixel-Wise Supervision Using BiFPN Architecture

Base Dataset - Replay Attack

In this experiment, while selecting thresholds, it is important to note that if the final **output binary score** is **less than** the selected **threshold**, then the input image is **real** else **spoof**.

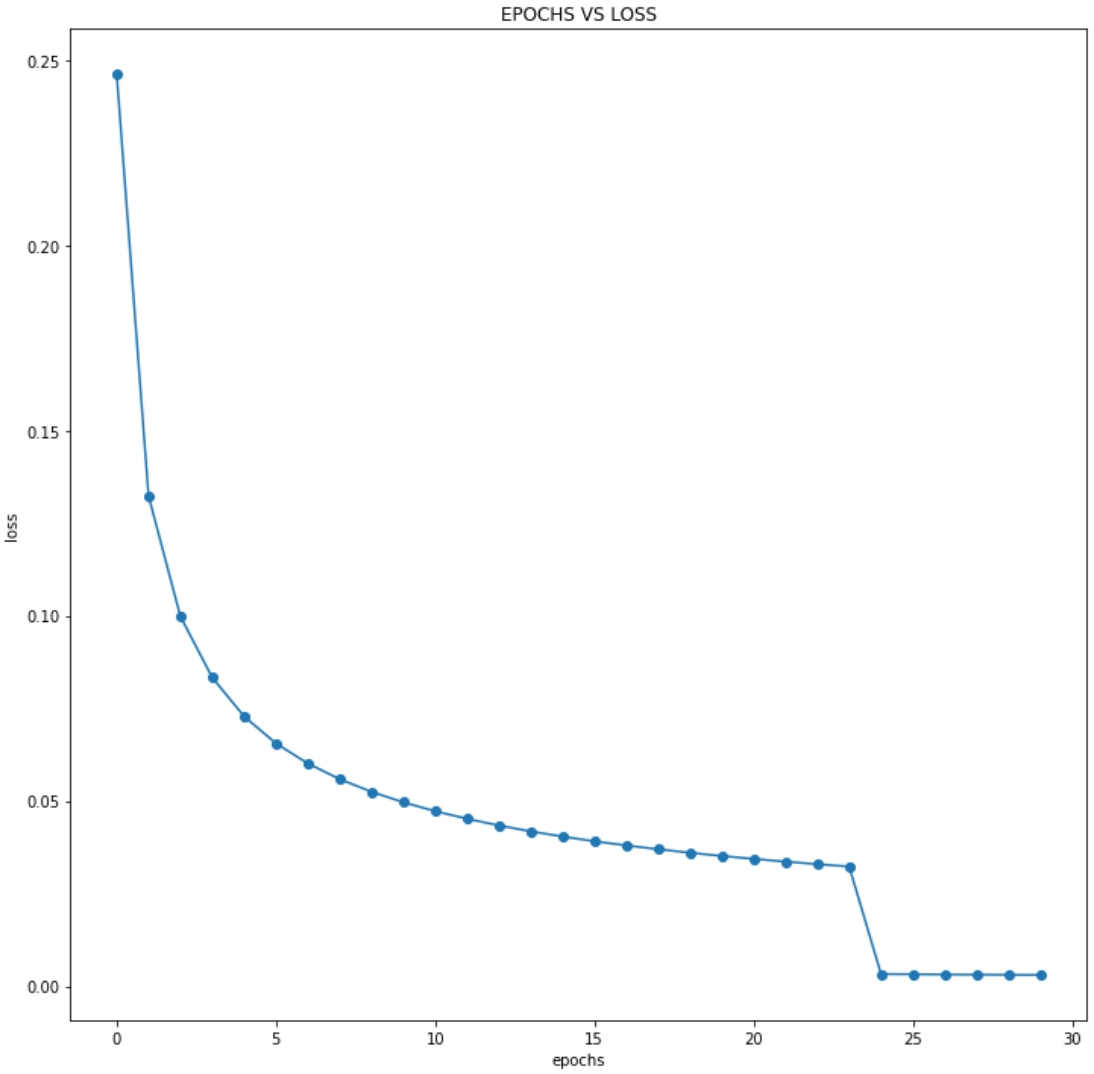


Fig-24 : Epochs vs Loss for BiFPN supervision

Table - 9 Validation Metrics (BiFPN supervision)

thresh	apcer	bpcer	acer
0	0	1	0.5
0.05	0.006904	0.003286	0.005095
0.1	0.005896	0.0001026	0.003011
0.15	0.008473	0.000707	0.00459
0.2	0.008787	0.000545	0.004666
0.25	0.009146	0.000501	0.004823
0.3	0.009908	0.000457	0.005182
0.35	0.010222	0.000442	0.005332
0.4	0.010491	0.000427	0.005459
0.45	0.010715	0.000427	0.005571
0.5	0.011029	0.000383	0.005706
0.55	0.011432	0.000368	0.0059
0.6	0.011657	0.000368	0.006012
0.65	0.012015	0.000354	0.006184
0.7	0.012464	0.000354	0.006409
0.75	0.012777	0.000339	0.006558
0.8	0.013405	0.000324	0.006865
0.85	0.014078	0.000309	0.007193
0.9	0.014974	0.000295	0.007634
0.95	0.01623	0.00028	0.008255
1	1	0	0.5

* **Red represents maximum value**

* **Brown represents minimum value**

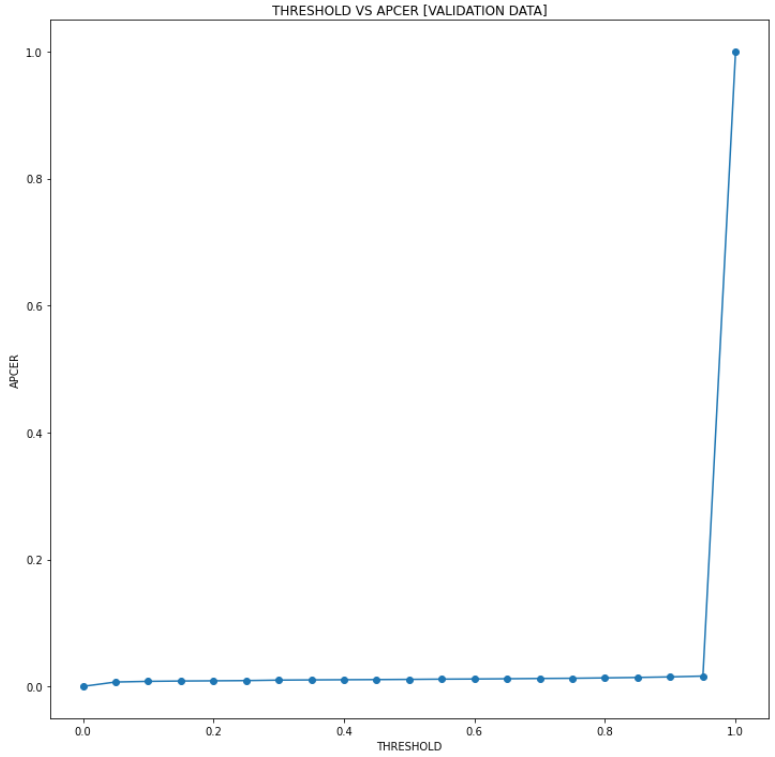


Fig-25 : Threshold vs APCER (validation data) for BiFPN supervision

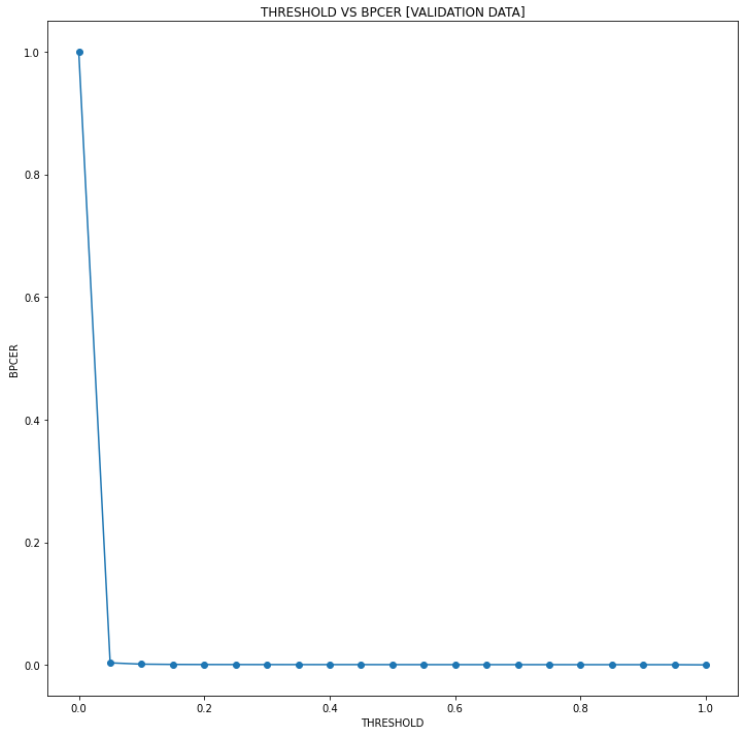


Fig-26 : Threshold vs BPCER (validation data) for BiFPN supervision

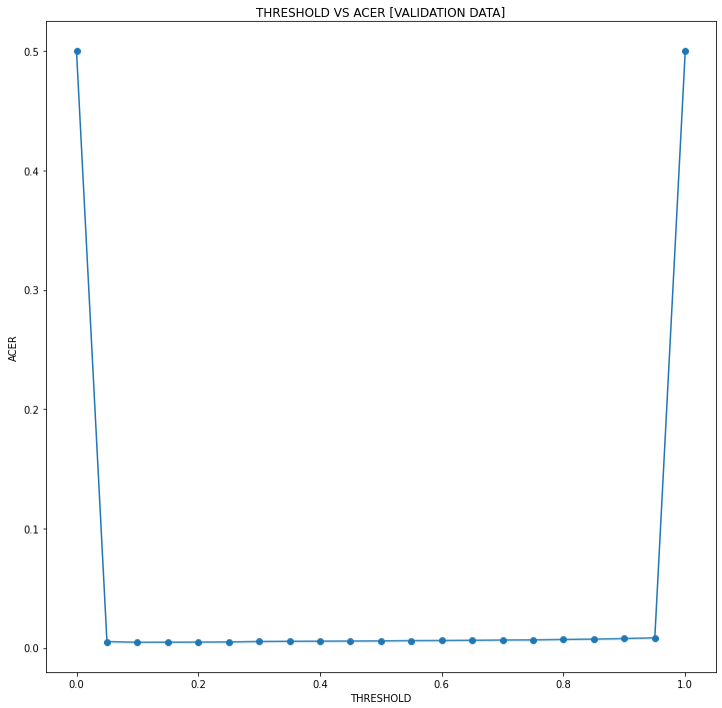


Fig-27 : Threshold vs ACER (validation data) for BiFPN supervision

Table - 10 Validation Metrics (BiFPN supervision)

matrix	thresh	recall	precision	f1_score	accuracy
[[0 67871] [0 22305]]	0	nan	0	nan	0.2473496274
[[67648 223] [154 22151]]	0.05	0.9977286806	0.9967143552	0.9972212599	0.9958192867
[[67791 80] [177 22128]]	0.1	0.9973958333	0.9988212933	0.9981080544	0.9971500177
[[67823 48] [189 22116]]	0.15	0.9972210786	0.999292776	0.9982558525	0.9973718062
[[67834 37] [196 22109]]	0.2	0.9971189181	0.9994548482	0.9982855167	0.9974161639
[[67837 34] [204 22101]]	0.25	0.9970018077	0.9994990497	0.9982488669	0.9973607168
[[67840 31] [221 22084]]	0.3	0.9967529128	0.9995432512	0.9981461319	0.9972054649

[[67841 30] [228 22077]]	0.35	0.9966504576	0.999557985	0.9981021039	0.9971389283
[[67842 29] [234 22071]]	0.4	0.9965626653	0.9995727188	0.9980654226	0.9970834812
[[67842 29] [239 22066]]	0.45	0.9964894758	0.9995727188	0.998028716	0.9970280341
[[67845 26] [246 22059]]	0.5	0.9963871877	0.9996169203	0.997999441	0.9969836764
[[67846 25] [255 22050]]	0.55	0.9962555616	0.9996316542	0.9979407525	0.996894961
[[67846 25] [260 22045]]	0.6	0.9961824215	0.9996316542	0.9979040573	0.9968395138
[[67847 24] [268 22037]]	0.65	0.9960654775	0.999646388	0.9978527201	0.9967618879
[[67847 24] [278 22027]]	0.7	0.9959192661	0.999646388	0.9977793465	0.9966509936
[[67848 23] [285 22020]]	0.75	0.995817005	0.9996611218	0.9977353607	0.9965844571
[[67849 22] [299 22006]]	0.8	0.9956124905	0.9996758557	0.9976400356	0.9964402945
[[67850 21] [314 21991]]	0.85	0.9953934628	0.9996905895	0.9975373985	0.9962850426
[[67851 20] [334 21971]]	0.9	0.9951015619	0.9997053233	0.9973981302	0.9960743435
[[67852 19] [362 21943]]	0.95	0.9946931715	0.9997200572	0.9972002792	0.995774929
[[67871 0] [22305 0]]	1	0.7526503726	1	0.8588710953	0.7526503726

* **Red represents maximum value** * **Brown represents minimum value**

Table - 11 Test Metrics (BiFPN supervision)

thresh	apcer	bpcer	acer
0	0	1	0.5
0.05	0.0036725	0.00325	0.00346125
0.1	0.00394025	0.002413666667	0.003176958333
0.15	0.00409575	0.002028333333	0.003062041667
0.2	0.00421675	0.001896333333	0.003056541667
0.25	0.00431175	0.001819333333	0.003065541667
0.3	0.00442425	0.001742333333	0.003083291667
0.35	0.00449325	0.001731333333	0.003112291667
0.4	0.0045365	0.001720333333	0.003128416667
0.45	0.004623	0.001713	0.003168
0.5	0.00470075	0.001698333333	0.003199541667
0.55	0.004787	0.001683666667	0.003235333333
0.6	0.00485625	0.001665333333	0.003260791667
0.65	0.004934	0.001647	0.0032905
0.7	0.0049945	0.001632333333	0.003313416667
0.75	0.0050895	0.001606666667	0.003348083333
0.8	0.00516725	0.001573666667	0.003370458333
0.85	0.00521925	0.001544333333	0.003381791667
0.9	0.00534875	0.001511333333	0.003430041667
0.95	0.005608	0.001456333333	0.003532166667
1	1	0	0.5

* **Red represents maximum value**

* **Brown represents minimum value**

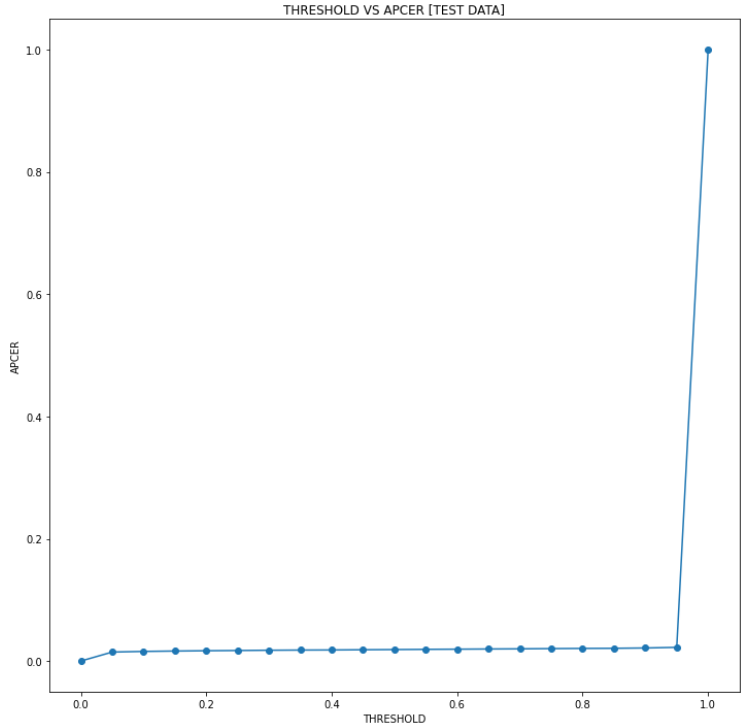


Fig-28 : Threshold vs APCER (test data) for BiFPN supervision

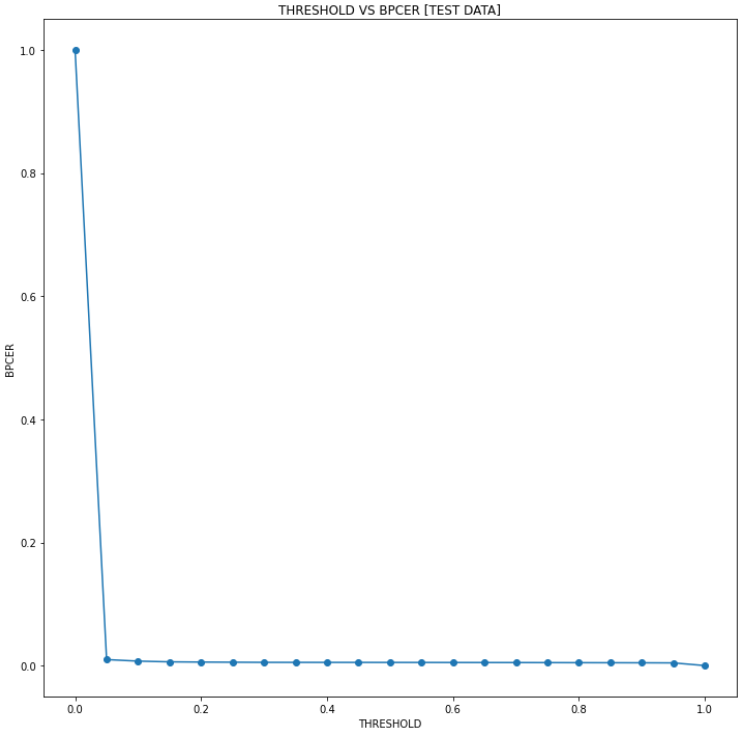


Fig-29 : Threshold vs BPCER (test data) for BiFPN supervision

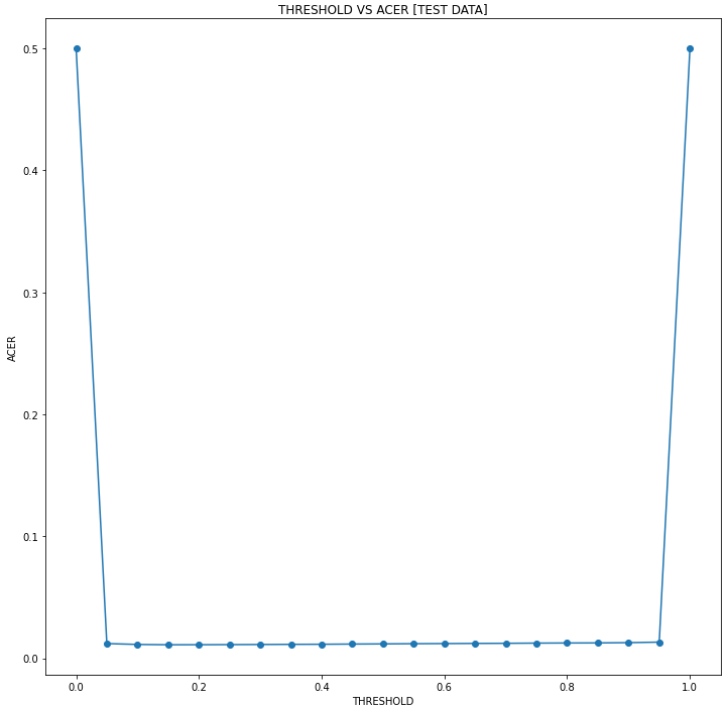


Fig-30 : Threshold vs ACER (test data) for BiFPN supervision

Table - 12 Test Metrics (BiFPN supervision)

matrix	thresh	recall	precision	f1_score	accuracy
[[0 90876] [0 28932]]	0	nan	0	nan	0.2414863782
[[89990 886] [425 28507]]	0.05	0.9952994525	0.9902504512	0.9927685324	0.989057492
[[90218 658] [456 28476]]	0.1	0.994970995	0.9927593644	0.9938639493	0.9907017895
[[90323 553] [474 28458]]	0.15	0.9947795632	0.993914785	0.9943469861	0.9914279514
[[90359 517] [488 28444]]	0.2	0.9946283312	0.9943109292	0.9982706286	0.9971618189
[[90380 496] [499 28433]]	0.25	0.9945091825	0.9945420133	0.9945255976	0.9916950454
[[90401 475] [512 28420]]	0.3	0.9943682422	0.9947730974	0.9945706286	0.9917618189
[[90404 472] [520	0.35	0.9942809379	0.9948061094	0.9945434543	0.9917200855

28412]]					
[[90407 469] [525 28407]]	0.4	0.9942264549	0.9948391214	0.9945326938	0.9917033921
[[90409 467] [535 28397]]	0.45	0.994117259	0.9948611295	0.9944890551	0.9916366186
[[90413 463] [544 28388]]	0.5	0.9940191519	0.9949051455	0.9944619514	0.9915948851
[[90417 459] [554 28378]]	0.55	0.9939101472	0.9949491615	0.9944293829	0.991544805
[[90422 454] [562 28370]]	0.6	0.9938230898	0.9950041815	0.9944132849	0.991519765
[[90427 449] [571 28361]]	0.65	0.9937251368	0.9950592015	0.9943917217	0.9914863782
[[90431 445] [578 28354]]	0.7	0.9936489798	0.9951032176	0.994375567	0.9914613381
[[90438 438] [589 28343]]	0.75	0.9935293924	0.9951802456	0.9943541338	0.9914279514
[[90447 429] [598 28334]]	0.8	0.9934318194	0.9952792817	0.9943546924	0.9914279514
[[90455 421] [604 28328]]	0.85	0.9933669379	0.9953673137	0.9943661198	0.9914446448
[[90464 412] [619 28313]]	0.9	0.9932040007	0.9954663498	0.9943338884	0.9913945646
[[90479 397] [649 28283]]	0.95	0.9928781494	0.9956314098	0.9942528736	0.9912693643
[[90876 0] [28932 0]]	1	0.7585136218	1	0.8626758558	0.7585136218

* **Red represents maximum value** * **Brown represents minimum value**

4. Face Anti - Spoofing using Siamese Network

Base Dataset - Replay Attack

In this experiment, while selecting thresholds, it is important to note that if the final **comparison factor of two input images (distance between spoof and real images)** is **less than** the selected **threshold**, then the input image is **real** else **spoof**. Please also note that outputs of siamese network between 0 and 1 represent different meaning than our outputs of experiments involving pixel wise supervision as explained in 3rd chapter.

EPOCHS VS LOSS

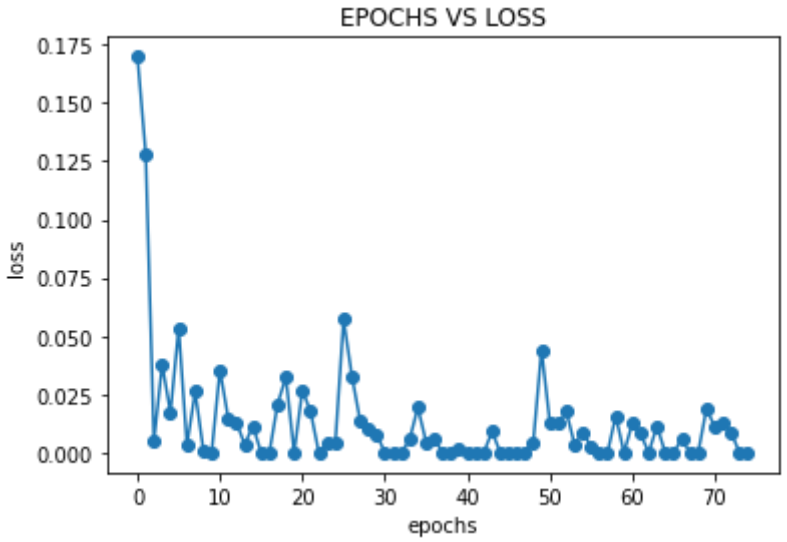


Fig-31 : Epochs vs Loss for Siamese Network

Table - 13 Test Metrics (siamese network supervision)

thresh	apcer	bpcer	acer
0	0	1	0.5
0.05	0	1	0.5
0.1	0	1	0.5
0.15	0	1	0.5
0.2	0	1	0.5
0.25	0	1	0.5
0.3	0	1	0.5
0.35	0	1	0.5
0.4	0	0.833333	0.416667
0.45	0	0.666667	0.333333
0.5	0.166667	0.5	0.333333
0.55	0.166667	0.333333	0.25
0.6	0.166667	0.333333	0.25
0.65	0.166667	0	0.083333
0.7	0.166667	0	0.083333

0.75	0.166667	0	0.083333
0.8	0.166667	0	0.083333
0.85	0.166667	0	0.083333
0.9	0.5	0	0.25
0.95	0.5	0	0.25
1	0.5	0	0.25

*** Red represents maximum value** *** Brown represents minimum value**

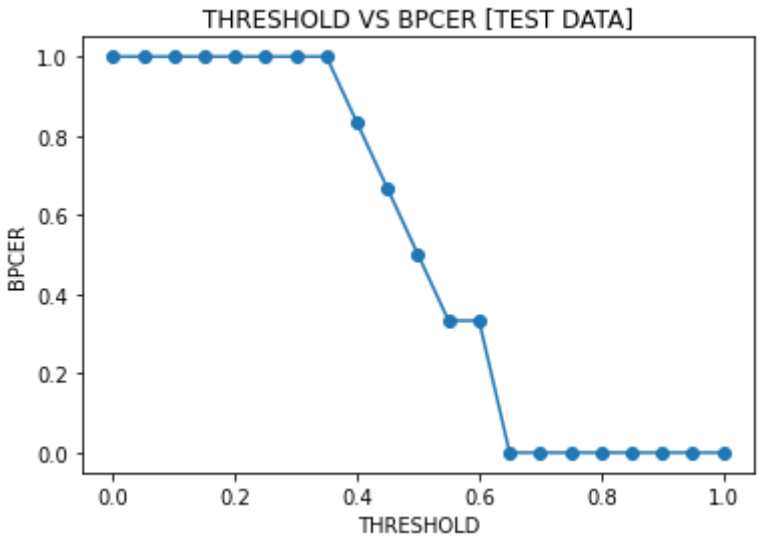


Fig-32 : Threshold vs BPCER (test data) for Siamese Network

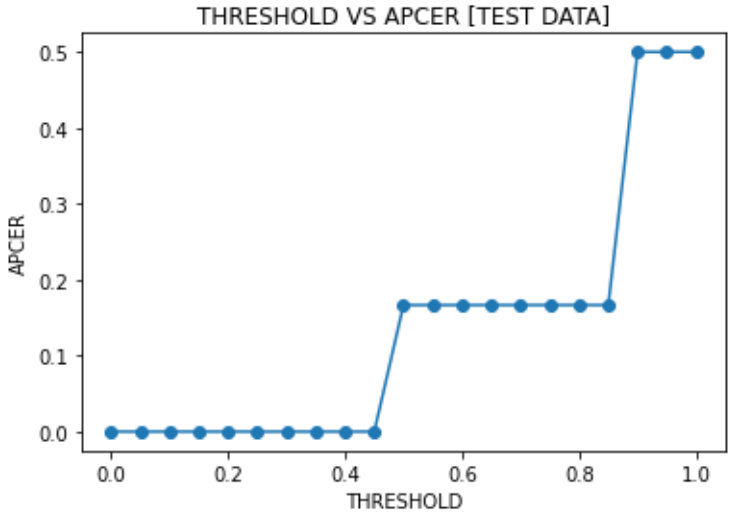


Fig-33 : Threshold vs APCER (test data) for Siamese Network

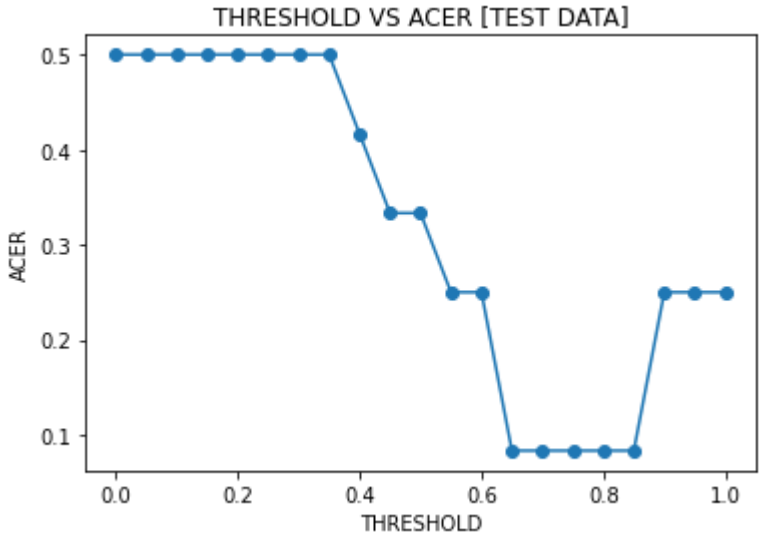


Fig-34 : Threshold vs ACER (test data) for Siamese Network

Table - 14 Test Metrics (siamese network supervision)

matrix	thresh	recall	precision	f1_score	accuracy
[[0 6] [0 6]]	0	nan	0	nan	0.5
[[0 6] [0 6]]	0.05	nan	0	nan	0.5
[[0 6] [0 6]]	0.1	nan	0	nan	0.5
[[0 6] [0 6]]	0.15	nan	0	nan	0.5
[[0 6] [0 6]]	0.2	nan	0	nan	0.5
[[0 6] [0 6]]	0.25	nan	0	nan	0.5
[[0 6] [0 6]]	0.3	nan	0	nan	0.5
[[0 6] [0 6]]	0.35	nan	0	nan	0.5
[[1 5] [0 6]]	0.4	1	0.1666666667	0.2857142857	0.5833333333
[[2 4] [0 6]]	0.45	1	0.3333333333	0.5	0.6666666667
[[3 3] [1 5]]	0.5	0.75	0.5	0.6	0.6666666667
[[4 2] [1 5]]	0.55	0.8	0.6666666667	0.7272727273	0.75

[[4 2] [1 5]]	0.6	0.8	0.6666666667	0.7272727273	0.75
[[6 0] [1 5]]	0.65	0.8571428571	1	0.9230769231	0.9166666667
[[6 0] [1 5]]	0.7	0.8571428571	1	0.9230769231	0.9166666667
[[6 0] [1 5]]	0.75	0.8571428571	1	0.9230769231	0.9166666667
[[6 0] [1 5]]	0.8	0.8571428571	1	0.9230769231	0.9166666667
[[6 0] [1 5]]	0.85	0.8571428571	1	0.9230769231	0.9166666667
[[6 0] [3 3]]	0.9	0.6666666667	1	0.8	0.75
[[6 0] [3 3]]	0.95	0.6666666667	1	0.8	0.75
[[6 0] [3 3]]	1	0.6666666667	1	0.8	0.75

*** Red represents maximum value** *** Brown represents minimum value**

5. Face Anti - Spoofing using Siamese Network with lossless triplet loss
   Base Dataset - Replay Attack
   In this experiment, while selecting thresholds, it is important to note that if the final **comparison factor of two input images (distance between spoof and real images)** is **less than** the selected **threshold**, then the input image is **real** else **spoof**. Please also note that outputs of the siamese network between 0 and 1 represent different meanings than our outputs of experiments involving pixel wise supervision as explained in 3rd chapter.

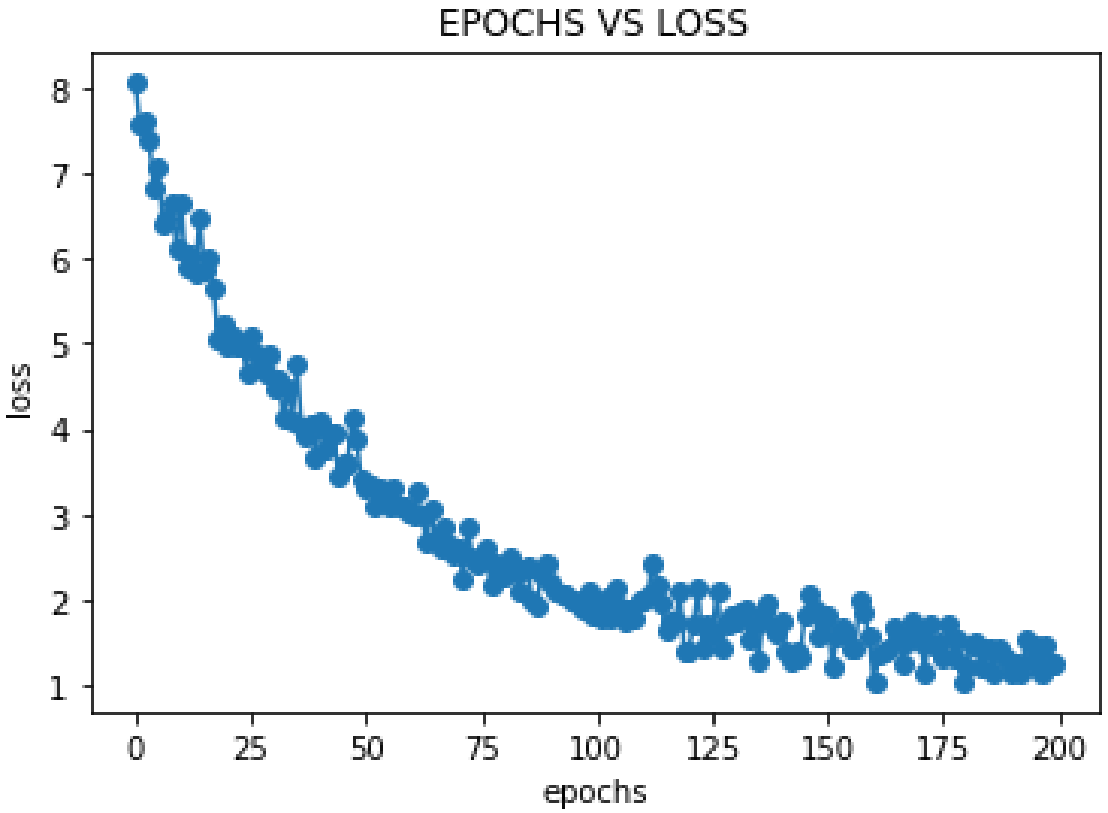


Fig-35 : Epochs vs Loss for Siamese Network w/ lossless triplet loss

Table - 15 Test Metrics (siamese w/ lossless triplet loss network supervision)

thresh	apcer	bpcer	acer
0	0	1	0.5
0.05	0	1	0.5
0.1	0	0.833333	0.416667
0.15	0	0.833333	0.416667
0.2	0	0.666667	0.333333
0.25	0	0	0
0.3	0.186777	0	0.0933885
0.35	0.186777	0	0.0933885
0.4	0.5	0	0.25
0.45	0.666667	0	0.333333
0.5	0.666667	0	0.333333
0.55	0.666667	0	0.333333
0.6	0.666667	0	0.333333
0.65	0.666667	0	0.333333
0.7	0.833333	0	0.416667
0.75	1	0	0.5
0.8	1	0	0.5

0.85	1	0	0.5
0.9	1	0	0.5
0.95	1	0	0.5
1	1	0	0.5

* **Red represents maximum value** * **Brown represents minimum value**

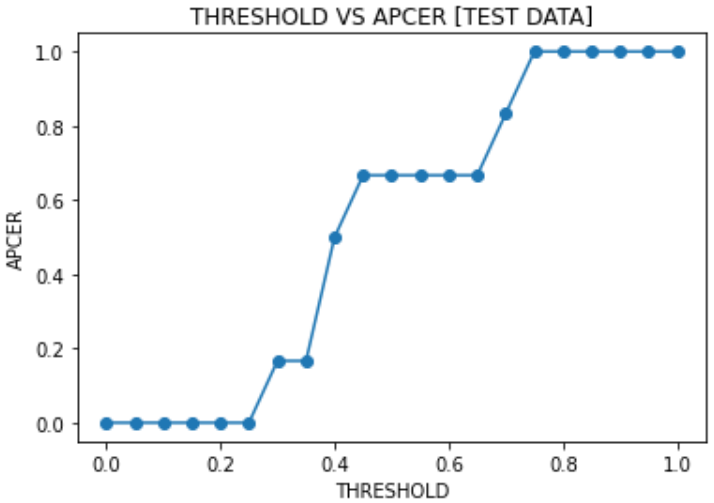


Fig-36 : Threshold vs APCER (test data) for Siamese Network w/ lossless triplet loss

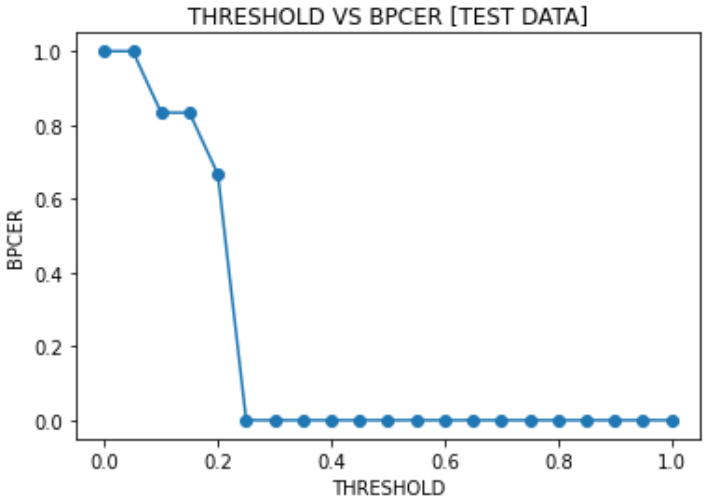


Fig-37 : Threshold vs BPCER (test data) for Siamese Network w/ lossless triplet loss

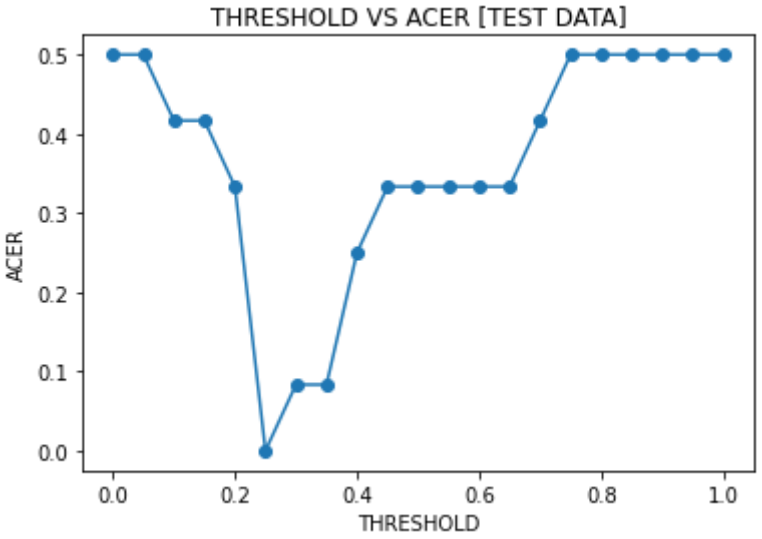


Fig-38 : Threshold vs ACER (test data) for Siamese Network w/ lossless triplet loss

Table - 16 Test Metrics (siamese w/ lossless triplet loss network supervision)

matrix	thresh	recall	precision	f1_score	accuracy
[[0 6] [0 6]]	0	nan	0	nan	0.5
[[0 6] [0 6]]	0.05	nan	0	nan	0.5
[[1 5] [0 6]]	0.1	1	0.1666666667	0.2857142857	0.5833333333
[[1 5] [0 6]]	0.15	1	0.1666666667	0.2857142857	0.5833333333
[[2 4] [0 6]]	0.2	1	0.3333333333	0.5	0.6666666667
[[6 0] [0 6]]	0.25	1	1	1	1
[[6 0] [1 5]]	0.3	0.8571428571	0.96	0.9112245454	0.9045432323
[[6 0] [1 5]]	0.35	0.8571428571	0.96	0.9112245454	0.9045432323
[[6 0] [3 3]]	0.4	0.6666666667	1	0.8	0.75
[[6 0] [4 2]]	0.45	0.6	1	0.75	0.6666666667
[[6 0] [4 2]]	0.5	0.6	1	0.75	0.6666666667

[[6 0] [4 2]]	0.55	0.6	1	0.75	0.6666666667
[[6 0] [4 2]]	0.6	0.6	1	0.75	0.6666666667
[[6 0] [4 2]]	0.65	0.6	1	0.75	0.6666666667
[[6 0] [5 1]]	0.7	0.5454545455	1	0.7058823529	0.5833333333
[[6 0] [6 0]]	0.75	0.5	1	0.6666666667	0.5
[[6 0] [6 0]]	0.8	0.5	1	0.6666666667	0.5
[[6 0] [6 0]]	0.85	0.5	1	0.6666666667	0.5
[[6 0] [6 0]]	0.9	0.5	1	0.6666666667	0.5

* **Red represents maximum value** * **Brown represents minimum value**

Chapter 6

Conclusion and Future Work

We have worked on pixel wise supervision and have introduced two different approaches to it, which are:

1. Bi directional pyramid supervision
2. Dense connection architecture in place of original pyramid supervision

The best overall results (both for **validation** and **test** set) for all of our experiments involving pixel wise supervision are summarized in the following table :-

Table - 17 Most Optimized results from validation metrics of pixel wise supervision (Vanilla , DenseNet , BiFPN)

S.No	Architecture (Ours)	VALIDATION SET			
		THRESHOLD	ACER	ACCURACY	F1 SCORE
1	Pixel-Wise Supervision Using vanilla pyramid supervision	0.3	0.004116	0.9952634629	0.9968481397
2	Pixel Wise Supervision Using DenseNet architecture	0.15	0.009346	0.9898498579	0.9932522222
3	Pixel Wise Supervision Using Bi-FPN architecture	0.2	0.003011	0.9974161639	0.9982855167

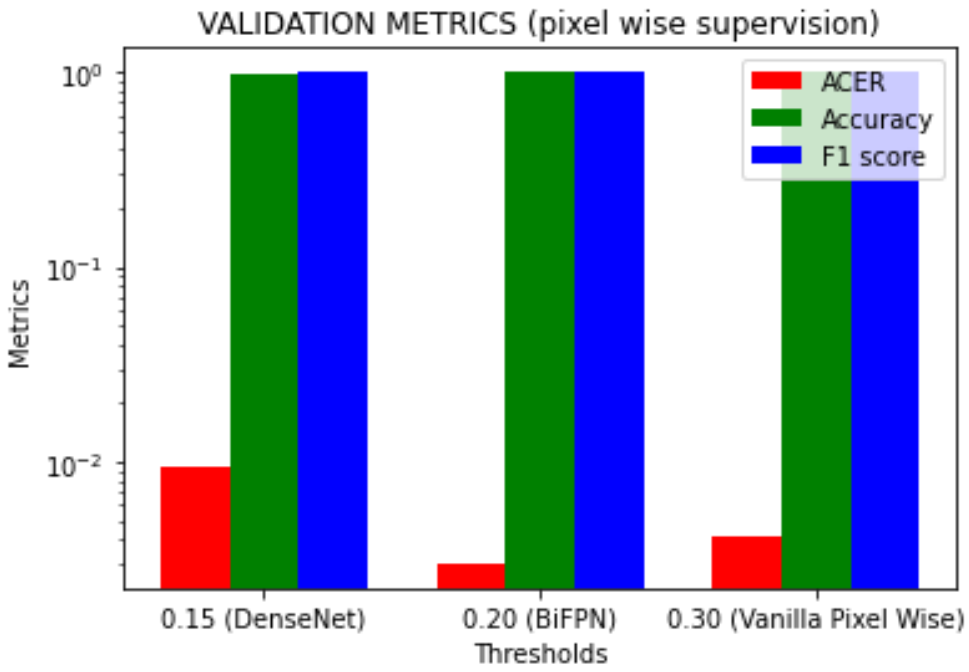


Fig-39 : Validation metrics results for pixel wise supervision

Table - 18 Most Optimized results from test metrics of pixel wise supervision (Vanilla , DenseNet , BiFPN)

S.No	Architecture (Ours)	TEST SET			
		THRESHOLD	ACER	ACCURACY	F1 SCORE
1	Pixel-Wise Supervision Using vanilla pyramid supervision	0.80	0.003344	0.9960017398	0.9973599912
2	Pixel Wise Supervision Using DenseNet architecture	0.55	0.009732	0.9921415687	0.9948195656
3	Pixel Wise Supervision Using Bi-FPN architecture	0.20	0.003056	0.997161818	0.998270628

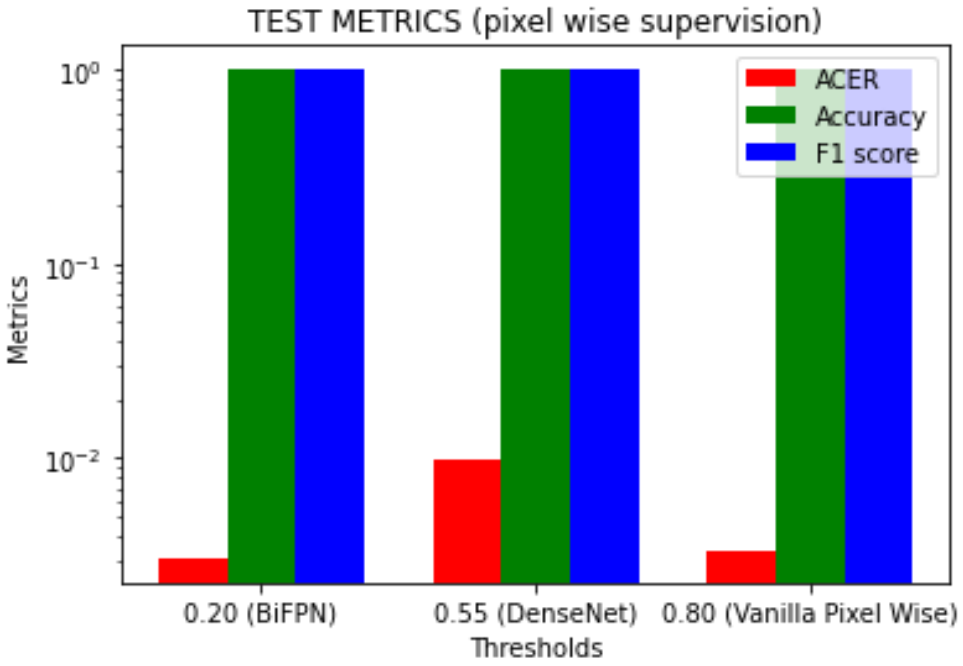


Fig-40 : Test metrics results for pixel wise supervision

By comparing the accuracy values, we have found that:

- Bi Directional Pyramid Supervision approach works slightly better than the original pyramid supervision approach. The reason may be that bidirectional pyramid supervision uses the features more effectively as there are many stages of intermixing at various levels. Due to that we have improved accuracy.
- Dense connection architecture approach works poorly than the original pyramid supervision as there was a decline in the accuracy. This might be because of the fact that there is too much overlap of binary features from previous layers, which lead to distorted binary images, which in turn results in loss of information.

Despite the above results, we can see that the accuracy of the proposed solutions and the original solution do not vary much. We can say these all solutions work almost identically. But for sake of simplicity, we can say the BiFPN solution is the best among them.

Other than pixel wise supervision, we have experimented with Siamese Neural Network for anti face spoofing. In this, we have implemented two approaches which are:

1. using triplet loss function and,
2. using lossless triplet loss function

The best overall results (**test** set) for all of our experiments involving siamese network are summarized in the following table :-

Table - 19 Most Optimized results from test metrics of pixel wise supervision (siamese w/ basic triplet loss and lossless triplet loss)

S.No	Architecture (Ours)	TEST SET			
		THRESHOLD	ACER	ACCURACY	F1 SCORE
1	Siamese Network w/ triplet loss	0.75	0.083333	0.9166666667	0.9230769231
2	Siamese Network w/ lossless triplet loss	0.35	0.0933885	0.9045432323	0.9112245454

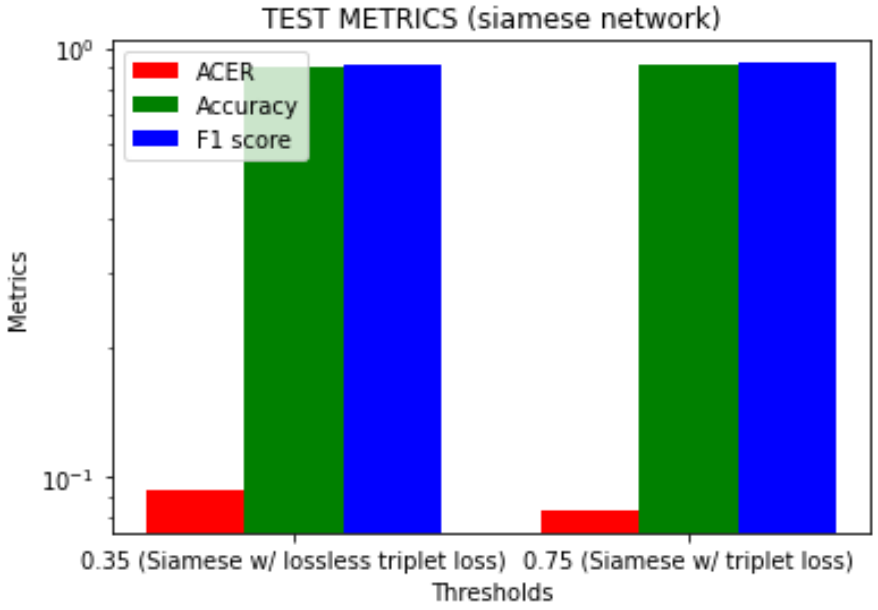


Fig-41 : Test metrics results for siamese networks

While comparing the experiments involving siamese network , the accuracy and ACER values of both the models differ only slightly. Although it is important to add that training siamese network with lossless triplet loss took a longer time and took nearly 200 epochs. For the Replay attack dataset it is safe to say that siamese network with basic triplet loss outperformed lossless triplet loss.

Should we compare models using pixel wise supervision with models involving Siamese Network ?

Comparing the accuracies and ACER values for these two versions of model that we have implemented might not be the best thing in the world. Both of these models worked best on their own and have their own use-cases.

- Models involving Pixel Wise Supervision could be used in advanced cases where along with differentiating spoofs from real, predicting which class of attack was employed is required.
- Models involving siamese network could be used in rather simpler case scenarios where only differentiating spoofs from real is required.

Future work:

Face anti spoofing is best performed using deep learning algorithms. One can try to improve the pre existing algorithms and methods ,like what we have tried to achieve in this project. Other than that, some amalgamation of challenge response technique and deep learning might prove to be useful.
Cross dataset testing with Bi-FPN and DenseNet models could provide some useful results that can help with prediction on a wider range of data.

The technique of the Siamese network can further be helpful in reconstructing lost features of an input image based on whether it is a spoof or real image. For e.g. if identified as real, its features can be enhanced using some other real image so that next time when this image is passed through the same network, the probability that this image will be predicted as real gets high.

References and Citations

- Yu, Zitong & Qin, Yunxiao & Zhao, Hengshuang & Li, Xiaobai & Zhao, Guoying. (2021). Dual-Cross Central Difference Network for Face Anti-Spoofing. 1281-1287. 10.24963/ijcai.2021/177.
- Yu, Zitong & Li, Xiaobai & Shi, Jingang & Xia, Zhaoqiang & Zhao, Guoying. (2021). Revisiting Pixel-Wise Supervision for Face Anti-Spoofing. IEEE Transactions on Biometrics, Behavior, and Identity Science. PP. 1-1. 10.1109/TBIOM.2021.3065526.
- George, Anjith & Marcel, Sébastien. (2021). On the Effectiveness of Vision Transformers for Zero-shot Face Anti-Spoofing. 1-8. 10.1109/IJCB52358.2021.9484333.
- Yang, Wanli & Chen, Yimin & Huang, Chen & Gao, Mingke. (2018). Video-Based Human Action Recognition Using Spatial Pyramid Pooling and 3D Densely Convolutional Networks. Future Internet. 10. 115. 10.3390/fi10120115.
- Roy, Koushik & Rupty, Labiba & Hossain, Md & Sengupta, Shirshajit & Taus, Shehzad & Mohammed, Nabeel. (2021). Bi-FPNFAS: Bi-Directional Feature Pyramid Network for Pixel-Wise Face Anti-Spoofing by Leveraging Fourier Spectra. Sensors. 21. 2799. 10.3390/s21082799.
- O. M. Parkhi, A. Vedaldi, A. Zisserman, Deep Face Recognition, British Machine Vision Conference, 2015.
- Q. Cao, L. Shen, W. Xie, O. M. Parkhi, A. Zisserman, VGGFace2: A dataset for recognising face across pose and age, International Conference on Automatic Face and Gesture Recognition, 2018.
- F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: A Unified Embedding for Face Recognition and Clustering, CVPR, 2015.
- G. Koch, R. Zemel, R. Salakhutdinov, Siamese Neural Networks for One-shot Image Recognition, *ICML deep learning workshop*. Vol. 2. 2015.
- https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62
- https://towardsdatascience.com/building-face-recognition-model-under-30-minutes-2d1b0ef72fda
- https://medium.com/arteos-ai/the-differences-between-sigmoid-and-softmax-activation-function-12adee8cf322
- https://chroniclesofai.com/transfer-learning-with-keras-resnet-50
- https://towardsdatascience.com/lossless-triplet-loss-7e932f990b24
- https://towardsdatascience.com/anti-spoofing-techniques-for-face-recognition-solutions-4257c5b1dfc9