T.C.
MANİSA CELAL BAYAR UNIVERSITY
GRADUATE SCHOOL OF

MASTER THESIS
DEPARTMENT of COMPUTER ENGINEERING

FACE PRESENTATION ATTACK DETECTION BY DEEP LEARNING

Muhammed SELAMCIOĞLU

Supervisor
Prof. Dr. Muhammet Gökhan ERDEM

MANİSA–2024

MUHAMMED SELAMCIOĞLU	**FACE PRESENTATION ATTACK DETECTION BY DEEP LEARNING**	**2024**

COMMITMENT

I declare that this thesis was written in Manisa Celal Bayar University, Faculty of Engineering, Department of Computer Engineering in accordance with academic and ethical rules, and all the literature information used is included in the thesis with reference.

Signature
Muhammed SELAMCIOĞLU

CONTENTS

LIST of ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Networks
CNN	Convolutional Neural Network
CI	Counterfactual Intervention
DL	Deep Learning
ED-LBP	Equilibrium Difference Local Binary Pattern
GPUs	Graphics Processing Units
ILSVRC	ImageNet Large Scale Image Recognition Competition
ML	Machine Learning
MLP	Multilayer Perceptron
MS	MixStyle
NN	Neural Network
PAD	Face Presentation Attack Detection
rPPG	remote photoplethysmography
RNN	Recurrent Neural Networks
SAFPAD	Salience-aware face presentation attack detection
SVM	Support Vector Machine
WMCA	Wide Multi-Channel Presentation Attack

LIST of FIGURES

LIST of TABLES

ACKNOWLEDGEMENT

Today, deep learning is used extensively in many studies. In this thesis, face presentation attack detection has been realized with deep learning algorithms to detect attackers who threaten our biometric security and to increase the reliability of these systems.

First of all, I would like to thank my advisor Prof. Dr. Muhammet Gökhan ERDEM for his interest, support and patience throughout this study. In addition, sharing his experience and superior knowledge in the field of Deep learning and Artificial intelligence with me throughout the study has been the biggest support for me to finish this thesis.

I would also like to thank the dear faculty members who gave lectures in our department and provided me with valuable information during the course process.

Finally, I would like to thank my family, my wife and loved ones. I am grateful for your patience, motivation and understanding during this process.

Muhammed SELAMCIOĞLU
Manisa, 2024

ABSTRACT

M. Sc. Thesis

Muhammed SELAMCIOĞLU

Manisa Celal Bayar University
Graduate School
Department of Computer Engineering

Supervisor: Prof. Dr. Muhammet Gökhan ERDEM

Today, with the development of technology, artificial intelligence algorithms have started to be used in many fields. Artificial intelligence algorithms, which we use in many fields such as medical, weather forecasting, digital assistants, autonomous vehicles and security systems, are becoming more widespread and spreading to different fields day by day. Biometric recognition systems, which are the subject of this study, are actively used in many areas such as banking, health, airports and border control, and mobile phones.

This study investigates the effectiveness of deep learning algorithms in face presentation attack detection. Attackers who threaten the security of facial recognition systems try to bypass these systems by using fake faces (e.g. photos, videos or masks). The aim of this study is to examine the success of deep learning-based methods in detecting such attacks.

In this study, two different convolutional neural network models are used and different methods are added to these models to compare the problem solving performance of convolutional neural network models. The models are trained on the OULU-NPU dataset and the results obtained by using different data sampling methods on the dataset are discussed. The deep learning models, dataset and methodologies used in the thesis are described in detail, and the model training and testing processes are discussed in detail. For performance evaluation, the results obtained in the light of Half of Total Error Rate (HTER), Attack Presentation Classification Error Rate (APCER) and Bonafide Presentation Classification Error Rate (BPCER) metrics are discussed. The results show that the models are successful in detecting face presentation attacks.

In conclusion, this thesis demonstrates that deep learning techniques are a powerful tool for face-presentation attack detection and can play an important role in improving the reliability of biometric security systems.

Keywords: Face Presentation Attack Detection, Deep Learning, Artificial Intelligence, Biometric recognition systems, Convolutional Neural Network

2024, 59 pages

ÖZET

Yüksek Lisans Tezi

Muhammed SELAMCIOĞLU

Manisa Celal Bayar Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Muhammet Gökhan ERDEM

Günümüzde teknolojinin gelişmesiyle beraber yapay zeka algoritmaları birçok alanda kullanılmaya başlanmıştır. Medikal, hava durumu tahminleri, dijital asistanlar, otonom araçlar ve güvenlik sistemleri gibi birçok alanda kullandığımız yapay zeka algoritmaları gün geçtikçe daha çok yaygınlaşmakta ve farklı alanlara yayılmaktadır. Bu çalışmanın da konusu olan biyometrik tanıma sistemleri; bankacılık, sağlık, havaalanları ve sınır denetimi, mobil telefonlar gibi birçok alanda aktif olarak kullanılmaktadır.

Bu çalışmada yüz sunum saldırı tespiti alanında derin öğrenme algoritmalarının etkinliği araştırılmaktadır. Yüz tanıma sistemlerinin güvenliğini tehtit eden saldırganlar bu sistemleri sahte yüzler kullanarak (örneğin fotoğraf, video veya maskeler) aşmaya çalışmaktadır. Yapılan çalışmada derin öğrenme tabanlı yöntemlerin bu türdeki saldırıları tespit etmedeki başarısını incelemektir.

Çalışmada iki farklı evrişimli sinir ağı modeli kullanılmış ve bu modellere farklı metotlar eklenerek evrişimli sinir ağı modellerinin problem çözme performansları kıyaslanmıştır. Modeller OULU-NPU veri seti üzerinde eğitilmiş ve veriseti üzerinde farklı veri arttırma yöntemleri kullanılarak elde edilen sonuçlar tartışılmıştır. Tezde kullanılan derin öğrenme modelleri, veriseti ve metodolojiler ayrıntılı olarak açıklanmış, model eğitimi ve test süreçleri kapsamlı olarak ele alınmıştır. Performans değerlendirmesi için Toplam Hata Oranının Yarısı (ing. HTER), Saldırı Sunumu Sınıflandırma Hata Oranı (ing. APCER) ve Gerçek Sunum Sınıflandırma Hata Oranı (ing. BPCER) metrikleri ışığında elde edilen sonuçlar tartışılmıştır. Elde edilen bulgular ele alınan modellerin yüz sunum saldırılarını tespit etmede başarılı olduğunu göstermiştir.

Sonuç olarak bu tez derin öğrenme tekniklerinin yüz sunum saldırı tespiti için güçlü bir araç olduğunu ve biyometrik güvenlik sistemlerinin güvenilirliğini artırmada önemli bir rol oynayabileceğini ortaya koymaktadır.

Anahtar Kelimeler: Sahte yüz sunum saldırı tespiti, Derin öğrenme, Yapay zeka, biyometrik tanıma sistemleri, Evrişimli sinir ağları

2024, 59 Sayfa

1. INTRODUCTION

In today's century, with the development of technology, it is vital to ensure the security and effectiveness of biometric systems such as face recognition. Face recognition technology is actively used in many areas such as banking, border control, and mobile device security. Facial recognition systems have become a preferred option for authentication and identification processes due to their non-intrusive nature and ease and convenience. However, with the developing technology and widespread use of social media, it is becoming easier for attackers to obtain digital photographs of individuals, which poses a great risk to security systems. Attackers can use the obtained images to modify or falsify these images to deceive facial recognition systems and reduce the reliability of the systems.

Presentation attacks, where an attacker tries to fool the biometric system by presenting a fake face, are a major concern. These attacks can be carried out using different methods such as printed photos (print attacks), 3D masks (mask attacks) or digital videos (replay attacks). These deceptive tactics can easily bypass simple facial recognition systems and require the development of robust countermeasures to protect the security and reliability of biometric authentication. Effective Presentation Attack Detection (PAD) is crucial to protect facial recognition systems against such threats. Despite numerous efforts to develop distinctive features for PAD, achieving high accuracy in detecting various types of attacks remains a challenging and unsolved problem. PAD methods can be categorized into hardware-based and software-based approaches. Hardware solutions often include additional devices to detect liveness or user interaction challenges, making systems more complex and expensive. On the other hand, software-based methods, especially those focused on texture analysis from single RGB images, offer a more practical and cost-effective solution.

In recent years, with the significant development of artificial intelligence (AI) algorithms and their widespread use in many areas of our lives, remarkable progress has been made, especially in the field of deep learning (DL). With these advances, the accuracy, speed and reliability of face recognition systems are also increasing. The ability to extract complex features from face images using DL algorithms such as convolutional neural networks (CNN) and to learn these features automatically has

increased their usability in this field. CNN have been used to improve the performance of face recognition systems.

However, advancing AI technologies are also changing and improving the methods used by attackers. One of the most prominent instances is the technique known as Deepfake, which enables attackers to produce incredibly lifelike fake photos and videos that fool facial recognition systems. The difficulty in distinguishing these deepfake attacks from the real face underscores the need for advanced PAD techniques that can detect these attacks.

In Section II, the development process of AI and techniques used in this study is explained and the infrastructure of two different DL algorithms developed for solving the problem is prepared. In particular, the CNN architecture used to solve the problem is analyzed and explained in detail. In Section III, the literature on the subject is reviewed and how different techniques and methods contribute to the solution of the problem, the metrics of the method used are examined and compared. In Section IV we provide detailed information about the preparation of the dataset used in our study, how various data preprocessing is done, how the two separate CNN model structures used in solving the problem are created as a base, and how the MixStyle (MS) and Counterfactual Intervention (CI) methods integrated into the models are applied. In the following Section, we present the results of the two models tested on the OULU-NPU dataset with different combinations. In the final Section VI, we discuss and conclude these results.

2. BACKGROUND OF STUDY

2.1. Artificial Intelligence

Compared to other living beings, human beings stand out with their intellect. It is a creature that tries to solve the problems it encounters with its mind and develops different techniques and applications. With the development of the first digital computer in the 1940s, humans tried to solve problems with the algorithms they developed, but the necessity to develop a different algorithm for each problem and the fact that this was done with humans every time forced them to develop a different system. This is why scientists have developed AI, a system that can simulate human mental functions such as problem-solving, learning, and adapting to new information. The complexity of the human brain and the functioning of the neuron networks in the brain inspired scientists and AI has continuously evolved from the 1950s to the present day.

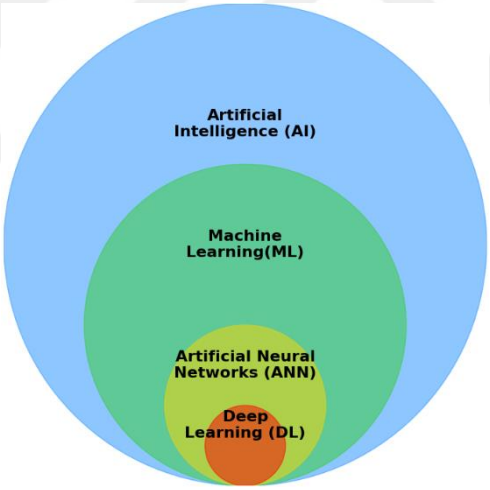


Figure 2.1 Relations of AI, ML, ANN and DL

2.2. Artificial Intelligence Applications

The areas of use of AI have expanded so much today that it is used in almost every field of humanity and makes our lives easier. We can list some of the areas of use of AI as follows:

- Health
- Autonomous vehicles

- Weather forecasts
- Digital assistants
- Game industry
- Workforce management systems
- Security systems

2.3. Historical relation of Machine learning, ANN and Deep learning

Machine learning (ML) is a sub-branch of AI that analyzes the structured data given as input to the system. It can learn on its own using this data and apply what it has learned to achieve the correct result using various algorithms. The foundations of ML began in 1943 when Walter Pitts and Warren McCulloch presented the first mathematical model of neural networks (NN) in a scientific paper [1]. Then in 1949, Donald Hebb's book "The Organization of Behavior", which theorized how behavior is related to NN and brain activity, strengthened the foundations of ML development [2].

In 1950, Alan Turing created the Turing test to determine whether a computer has real intelligence. In order for the test to be successful, the computer had to convince a human that it was like him, and while working at the University of Manchester, he discussed this principle in his paper Computing Machinery and Intelligence, beginning with the question 'Can machines think?'[3]. With the help of these developments, the first computer game that played itself and learned moves as it played was designed by Arthur Samuel in 1952. Then, in 1957, Frank Rosenblatt designed the first NN for computers, ***the perceptron***, which simulated the thinking processes of the human brain [4]. In 1967, an important step was taken in the field of ML with the writing of the "nearest neighbor" algorithm, which could be used to create the fastest and shortest route for traveling salesmen.

Considering the developing technology, due to reasons of increasing data size, increasing complexity of problems, and increasing processing power needs, traditional ML algorithms have become inadequate in solving modern problems based on human-determined features. As a solution to this problem, Deep Learning (DL) came to rise. It is a subfield of ML that enables computers to learn in a way that mimics the human brain using large-scale data. It involves training the machine on large datasets to

perform more complex tasks, such as image and speech recognition. DL models can learn them and make accurate decisions with minimal human intervention.

The foundations of DL go back to the foundations of ML with NN models such as the McCulloch-Pitts model in the 1940s and Frank Rosenblatt's Perceptron in the 1950s. By the 1970s and 1980s, developments in DL were limited due to the limited success of research on NN and the limited computational power of the hardware available at the time. In 1989, Yann LeCunn expanded convolutional net with back-propagation net to predict handwritten digits with high accuracy by concentrating on automatic learning of the images instead of the hand-designed preprocessing used in previous studies [5]. Starting from 1986, David Rumelhart, Geoffrey Hinton and Ronal Williams introduced the idea of Backpropagation through time (BPtt) for training Recurrent Neural Nets (RNN) for handling sequential data. But vanishing and exploding gradient problems lead Sepp Hochreiter and Jürgen Schmidhuber introduced a new NN model called Long short-term memory (LSTM), in 1997, to overcome the long-term learning difficulties of RNN [6]. LSTMs are preferred over traditional RNNs for these tasks because they can effectively capture long-term dependencies and manage the flow of information through their gating mechanisms (input gate, forget gate, and output gate). This allows them to remember important information over longer periods and forget irrelevant information, making them highly effective for sequential data processing. Although NN algorithms were advantageous in this period, they were not used due to the inadequacy of the hardware. Due to the computational cost simpler models such as support vector machines (SVM), were preferred until the early 2000s.

In the early 2000s, with the increase in computational power (i.e. new CPU architectures) and the use of graphics processing units (GPUs) in calculations, the computing speed increased thousands of times. This led the NN models started to rival the classical ML models (like SVM) again in this period. Then, in 2000, Igor Aizenberg and his colleagues coined the term "***Deep Learning***" [7]. In 2006, Geoffrey Hinton published a paper showing how a multilayer feed-forward NN can effectively train one layer at each iteration and then fine-tune it with a supervised back-propagation method [8]. In 2012, AlexNet, developed by Ilya Sutskever, Geoffrey Hinton, and Alex Krizhevsky, was the first deep neural network to win the ImageNet Large Scale Image Recognition Competition (ILSVRC). AlexNet, which took five to

six days to train on two GTX 580 3GB GPUs, broke new ground in the field of DL with its use of new methods such as ReLu activation function, dropout, max pooling, and the design of 8 layers [9]. In 2014, VGGNet, a CNN model developed by Karen Simonyan and Andrew Zisserman from Oxford University, has a deeper and more complex architecture inspired by the AlexNet model [10]. ResNet, developed by the Microsoft Research team in 2015, is considered one of the most important models in the field of DL. In this model, a new technique called residual connections was used, where these connections learn residuals (the difference between the desired output and the actual output) rather than directly learning the mapping from input to output. ResNet, which was designed to be deeper and more complex than previous models such as VGGNet and AlexNet, won the ILSVRC in 2015 with an error rate of 23% and increased the confidence in DL studies [11].

With all these developments, ML and DL algorithms continue to evolve rapidly and cover a wide range of techniques including computer vision, natural language processing, model identification, and predictive maintenance. These applications can be used both with and without human intervention.

2.4. Artificial Neural Networks (ANN)

Artificial neural networks (ANN) are an artificial model inspired by the human brain and designed to mimic the way the brain processes information. The most basic units that make up an ANN are artificial nerve cells, which are like biological nerve cells (neurons). Artificial neurons are the building blocks of ANN that receive and process input signals and transfer the output signal to the next layer. Each layer is fully or partially connected with the next layer. These connections transfer information from one layer to another. Each individual connection on the neuron in a layer has a value called a weight which determines how the information should be processed. In Figure 2.2, we can see an example of ANN architecture having 3 layers of neurons: input layer, hidden layers, and output layer.

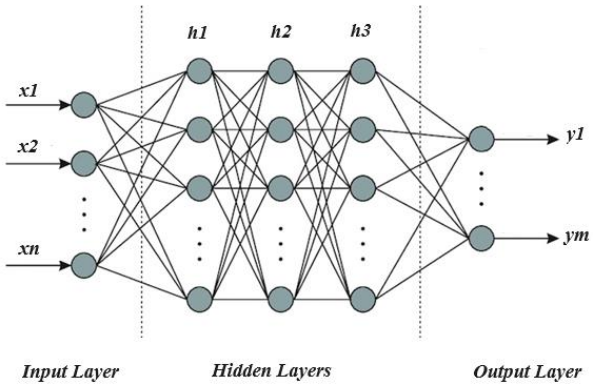


Figure 2.2 An example of ANN Architecture

Input layer: Information from the outside world enters ANN with the help of the input layer. In Figure 2.2, $\{x_1,..,x_n\}$ are the inputs of ANN architecture. The information entering through the Input Layer is processed, analyzed or categorized by input nodes and transmitted to the next layer.

Hidden layer: The transferred data from Input Layer comes to the Hidden Layer. Depending on the solution of the problem, the complexity of ANN may increase the number of hidden layers making the ANN architecture carrying more than one hidden layer in an ANN. In the Figure 2.2. we are observing that there are 3 hidden layers noted with the labels of h_1, h_2, and h_3. Hidden layer may receive their inputs from the input layer or from the previous hidden layers. Each hidden layer processes the data transferred from preceding layers and produces outputs for the next layer.

Output layer: The output layer provides the final outcome of processing data once it has passed through all associated layers. Depending on the problem, there may be one or more nodes in the output layer, like multiple outputs $\{y_1,..,y_m\}$ of the example ANN architecture given in Figure 2.2. The output values may vary with the type of problem. If the ANN is used for classification purposes, the outputs are going to be either 0 or 1 (indication of the membership to the observed class). If the model is used for prediction, the outputs are going to represent the value of the expectation.

Weight and Bias: In ANN, weight and bias are two important elements that affect the functioning and decision-making mechanism of a neuron. Each neuron in a layer of an ANN can focus on a distinctive feature of the model. The weight values in each neuron play an important role in the transition of the data from the neuron to the next layer. A higher weight value of the connection means more influence of the neuron on the connected neuron. Bias can be defined as special weight of a constant

input of value 1. It helps models to shift the activation function by adjusting the output of the NN and enabling the network to make accurate predictions/classifications [12]. The usage of weights and bias can be explained by a single neural model given in Figure 2.3 and the model related mathematical formulation shown in s equations (2.1a and 2.1b).

$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \ldots\ldots + b \quad (2.1a)$$

$$y = f(x, w + b) = f(z) \quad (2.1b)$$

where x_i are inputs, w_i are the weights, b is bias, y is the output and f is the activation function. Figure 2.3 shows this single neuron (f activation function is seen as "tanh" function).

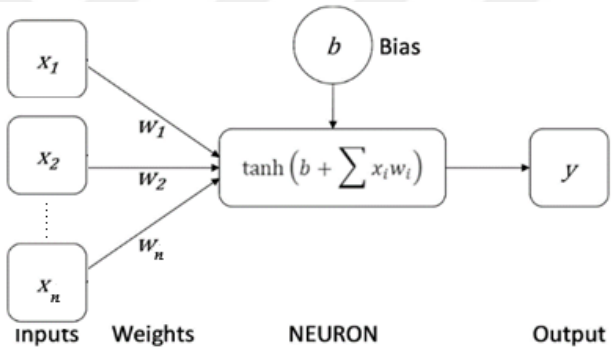


Figure 2.3 The structure of a single artificial neuron.

2.5. Types of Artificial Neural Networks

ANN is a set of mathematical models inspired by the human brain that are used to solve complex problems. Many types of ANN can be customized for different problems and data types, ranging from the simplest perceptron model to the more complex CNN based DL models. Different types of ANNs are used in wide range of application fields such as computer vision, natural language processing, voice analysis, and video analysis and image/video generation.

2.5.1. The Perceptron Model

The Perceptron model is the oldest ANN model, originating in 1943, based on the Walter Pitts and Warren McCulloch neuron model (MCP), and developed by Frank Rosenblatt to improve the single-neuron MCP model to calculate optimal weights for binary classifications. The Perceptron model can take multiple inputs, such as numeric, categorical or textual data. Each input is assigned to a vector x. For each input x, the weight vector w is modified where w is randomly assigned at the initial stage of training the model. The weights determine how important the inputs are to the model. Each x-value is multiplied by the vector w and summed to produce a scalar value defined as the z-vector, just like in equations (2.1a and 2.1b). The generated z value is sent to the ***Heaviside step function (i.e. Hardlimiter)*** , which is known as activation function of Perceptron model. In the view of this function, if the z is greater than certain threshold Φ (default is Φ=0) it converts the z value into a value 0 or 1 for classification.

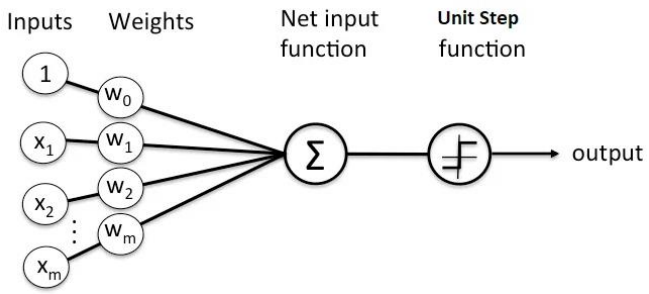


Figure 2.4 Perceptron Model [13]

2.5.2. Multilayer Perceptron Model (MLP)

The inability of the Perceptron model to solve complex problems and the fact that it can only be used on linearly separable problems (for instance, XOR problem of two inputs can not be solved by Perceptron) led to the development of the Multilayer Perceptron (MLP) model. Unlike the Perceptron model, the MLP has more than one layers There is a limitation which the Perceptron model uses only the Heaviside (or Step/Hardlimiter) activation function. But, In MLP, we are not limited to a single activation function, we have other choices like Sigmoid, Tanh, ReLU, GeLU, or new activations like Swish. The MLP is classified as a feed-forward algorithm and, like the

Perceptron, the inputs are combined into a weighted sum with initial weights and passed through the activation function. However, unlike Perceptron, each linear combination is propagated to the next layer. That is, the outputs of the previous layer are used as the inputs of the next layer and so on until the output layer. If this iteration is completed in one go and the calculated weights are passed to the output layer, the model learns from the output expectations and updates its weights to minimize the cost function using the Backpropagation algorithm.

Backpropagation Algorithm:

Backpropagation Algorithm, discovered and developed by Paul Werbos in 1974, is a learning mechanism that allows the MLP to iteratively adjust the weights in the network to minimize the cost function [14]. As we can see from Figure 2.5, in the backpropagation algorithm, the error occurred at the output layer is propagated on the weights in the reverse direction (i.e. from the output layer to the input layer). The weights are correct themselves (update) during this process to learn the error at the output layer. The updates of the output layer weights are done by taking the derivatives of the error function (like MSE, Mean Square Error, in eq (2.2)) according to the related weight given in eq (2.3)). At each inner layer of the architecture, Chain Rule is used to get the reflected derivatives of the error function at inner neurons. For the back propagation algorithm to work properly, activation function(s), cost functions (MSE, Cross Entropy) must be differentiable [15].

$$\mathrm{MSE} = \frac{1}{\mathrm{n}} * \sum_{i=1}^{n} (y - \hat{\mathrm{y}})^2 \tag{2.2}$$

Output layer weights are updated according to the eq (2.3):

$$\mathrm{v(t)} = \mathrm{v(t-1)} - \eta \frac{\mathrm{d(MSE)}}{\mathrm{dv(t-1)}} \tag{2.3}$$

Inner layer weights are updated according to the Chain Rule given in eq (2.4)

$$\frac{\mathrm{d(MSE)}}{\mathrm{dw(t-1)}} = \frac{\mathrm{d(MSE)}}{\mathrm{dv(t-1)}} * \frac{\mathrm{dv(t-1)}}{\mathrm{dw(t-1)}} \tag{2.4}$$

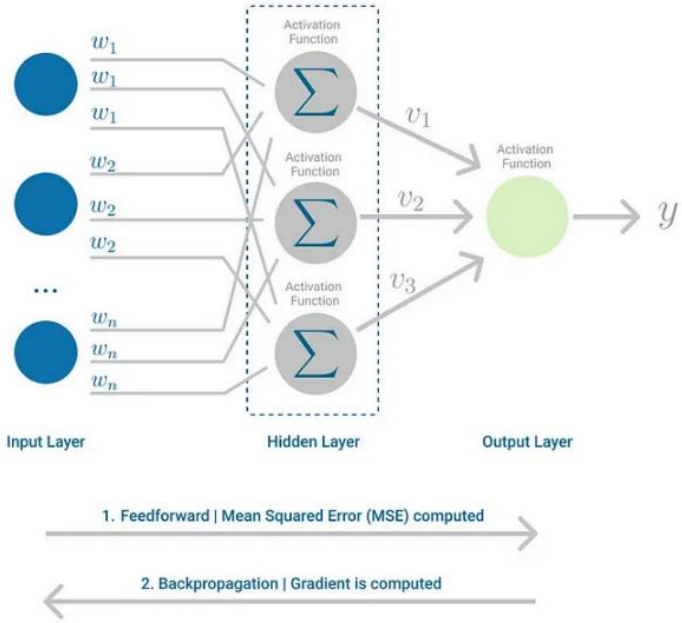


Figure 2.5 MLP, highlighting the Feedforward and Backpropagation steps [15]

2.5.3. Convolutional neural networks (CNN):

Although CNNs are categorized in MLP types of neural networks, they differ from them in the manner of connections. In traditional MLP architecture, layers are fully connected to the next layer. This makes the architecture carrying massive number of trainable hyperparameters. The main problem that arises in this cumbersome architecture is computational complexity. In each iteration of learning, thousands or millions of parameters are updated. CNN is developed as a solution to this problem by using the mathematical operation of convolution. In convolution, not all neurons of one layer are fully connected to the next layer, but neighboring neurons are reflected to specific neurons at the next layer. So with the help of the "neighboring approach", a limited number of parameters, instead of thousands/millions, are enough for learning.

CNNs are based on the research of two neurophysiologists, David Hubel and Torsten Wiesel, on how the mammalian visual system works [16]. The mammalian brain is a complex network capable of processing vision. The retina, visual cortex, and other areas in the eye have evolved to process visual information. Vision is the processing and interpretation of images from the retina, and nerve cells in the brain

specialize in identifying different features during the processing of visual information. For example, some neurons can only respond to certain directions, while others can only respond to a certain level of contrast. This difference is the basis of CNN. Convolution layers in CNN operate by applying filters to detect specific patterns in an image. This process is like the activity of nerve cells in the visual cortex of biological NNs. Pooling layers are used to reduce processing complexity and obtain features that are resistant to translations. Finally, fully connected layers (i.e. traditional MLP structure) are used to classify the extracted features.

CNNs are used in many application areas such as image classification, object detection, object tracking, medical image processing, advanced driver assistance systems (ADAS) in self-driving vehicles, natural language processing, and face recognition and biometric authentication, which are related to the topic of this thesis. The main goal of CNN used in all these applications is to extract ***meaningful local features*** from an input variable such as an image at the input layer and to combine these features to form new features fed up into the next layer (Figure 2.6).

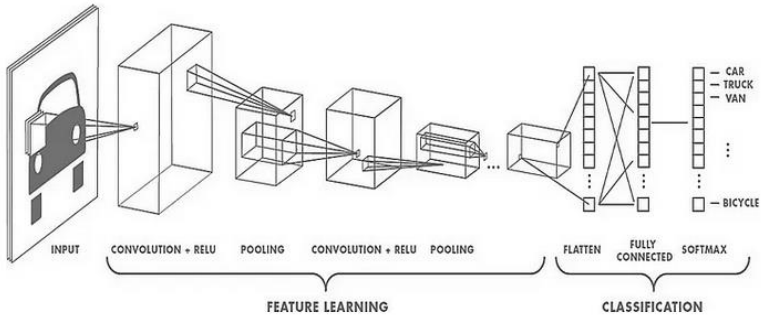


Figure 2.6 General architecture of CNN

Input Layer:

In CNN, the input layer is the layer where data is input to the network. Depending on the application and problem, the input of CNN can be image, audio, or video. In image processing applications, the input layer usually contains a tensor containing the pixel values of the image. This tensor is given to the model as 3-channel RGB or single channel (gray). The input of the CNN is set to a fixed size, although it varies in different models, for example, in some models, The input may be 224x224x3 (ResNet), while in another model it may be 229x229x3 (Inception). The uniform size of the inputs in the input layer is important for the attributes to be extracted throughout

the model, so the inputs are resized to a fixed width and height value when the dataset is preprocessed before being fed into the model.

Output Layer:

The output layer is the last layer of the model and produces the output of the model. The output layer varies according to the type of application and the purpose of the model. For example, in classification problems, there are as many neurons in the output layer as there are classes. These neurons determine the probability that the input belongs to a certain class. In regression problems, there is usually a single neuron in the output layer, and it represents the continuous value predicted by the model.

Convolution Layer:

The convolution layer is the most important layer of the network, which gives the name to CNN and takes on the learning load of the model. The convolution layer is the layer where feature maps are extracted by applying a convolution operator on the input. In the convolution layer, different kernels are applied to the input. These kernels are used to extract different features from the input data. In Figure 2.7, the effects of applying the well-known kernels can be seen. Edge detector kernel is the most famous one used to extract object borders, whereas the others can apply to blur the input [17].

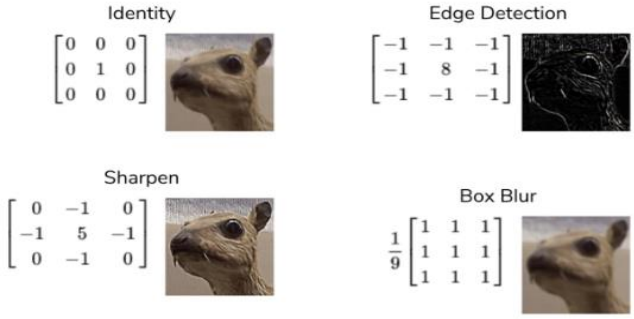


Figure 2.7 Special kernels applying on image [17]

Typical kernel sizes are 2x2, 3x3, 5x5, or 7x7. The convolution process is the application of the filter on the input data by shifting it by a certain stride. Also, a technique called padding can be used. Padding is empty pixels added to the edges of the input data. This allows the convolution process to preserve the size of the input data and helps to control the output size. At each shift step, the filter is applied to a region of the input data and a dot product is performed. The dot product results are

summed to obtain output data. The size of the output as a result of the convolution process is expressed by the following equation:

$$\text{Output}_{\text{size}} = \frac{\text{Input}_{\text{width}} - \text{Kernel}_{\text{size}} + 2 \times \text{padding}}{\text{Stride}} + 1 \tag{2.5}$$

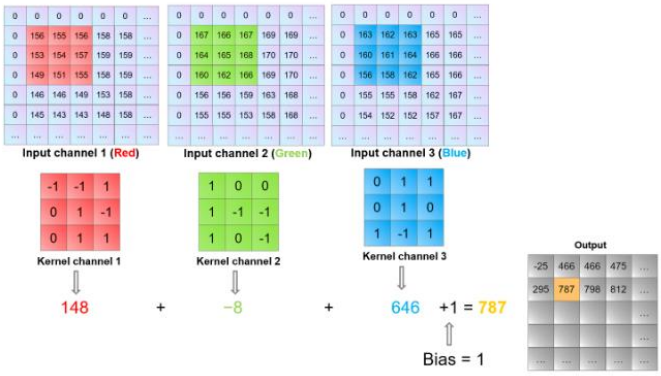


Figure 2.8 Convolution operation on an input image [18].

Activation function

Activation functions are added at the end of each layer during forward propagation, and although they increase the computational load in the CNN model, they allow the network to learn more complex and higher-level features by breaking linearity. For example, if we assume that we have a network without an activation function, then each neuron will perform a linear transformation on the inputs using weights and biases. In this case, it does not matter how many layers and complexity we have in our network because all layers will behave as a single layer regressor and learning will be limited. In this case, even though the network is simplified and the computational load is reduced, the network cannot solve a complex problem and only acts as a linear regression model.

Three types of activation functions can be used in CNN:

i. Binary step Function

With the binary step function, a threshold value is used to determine whether a neuron is activated or not. In this case, the input to the activation function is compared with a threshold value and if the neuron is greater than the threshold value, it is activated, if the value of the neuron is less than the threshold value, the neuron becomes

inactive and is not transmitted to the next layer. In this case, the binary step function cannot be used to solve a multiclass problem and backpropagation is problematic because the gradient of the step function is 0.

Figure 2.9 Binary Step Function [19]

ii. Linear Activation Function

The linear activation function does not affect the weighted sum of the input and outputs the input exactly as it is (Figure 2.10). Therefore, it does not matter how many layers the model has and how complex it is, since the model behaves as a single layer, it cannot solve complex problems. Additionally, since the function's derivative is a constant and unrelated to the input x, backpropagation is not feasible.

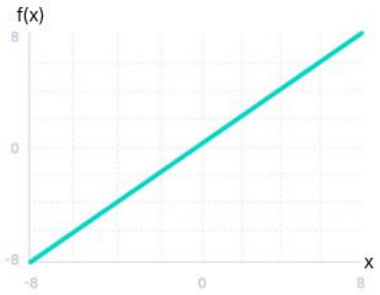


Figure 2.10 Linear activation function [19]

iii. Non-Linear activation function

For example, the complexity of the objects and features in an image cannot be expressed linearly, so using nonlinear functions such as ReLu, sigmoid, Softmax, and Swish as activation functions allows the network to better model such complexities. For example, the ReLu function (Figure 2.11) as activation takes the input value (x)

and returns the maximum value between 0 and the input value. So, if the input is negative, the ReLU function outputs zero, whereas if the input is positive, it returns the input value directly. The graph of this function has a region parallel to 0, which is a straight line in areas where the input value is negative and shows a linear increase when the input value is positive.

$$f(x) = \max(0, x) \tag{2.6}$$

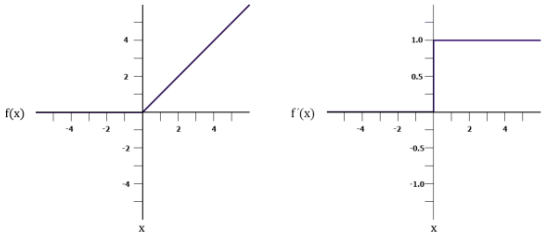


Figure 2.11 The ReLU Activation Function and its Derivative

Due to this feature, ReLu breaks the linearity of the network, allowing more complex features to be learned. In addition, since it provides a non-zero gradient for positive inputs, it ensures that the gradients are different from 0 during backpropagation and that these gradients are transmitted more effectively to higher layers of the network [19].

Pooling Layer:

The pooling layer is a commonly used layer in CNN. The pooling layer reduces the size of the feature maps obtained after the convolution layer and helps the network learn more generalizable features. Since the size of the output obtained with the pooling layer is reduced, it reduces the computational burden of the network while at the same time reducing the risk of overfitting. It can be used in two ways: maximum pooling and average pooling. As with the convolution layer, pooling filters can be applied at different kernel sizes. When applied by sliding on the features, the maximum pooling method like in Figure 2.12, translates the highest value in the portion it covers, while in average pooling it translates the average value.

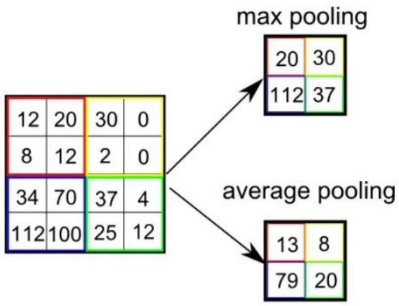


Figure 2.12 Types of Pooling [20]

3. LITERATURE REVİEW

Salience-aware face presentation attack detection (SAFPAD), discussed in [29], is a notable contribution to the field of face Presentation Attack Detection (PAD). SAFPAD uses a Deep Reinforcement Learning (DRL) approach to find and focus on distinctive salient regions in face images to distinguish real faces from presentation attacks. By combining high-level features and local patches in face photos, the SAFPAD method significantly improves the accuracy of PAD by effectively reducing redundancy. After extensive testing on public datasets, SAFPAD outperforms other approaches. On the Replay-Attack dataset, it yields a staggering *Average Classification Error Rate* (ACER) of 22.2%. This approach significantly outperformed previous methods by reducing cross-testing errors by 5.4% relative to the state-of-the-art.

In [30], based on the success of CNNs in various computer vision tasks, it was predicted that CNNs can also be successful in PAD applications. However, this work explores the concept of partial feature extraction for face recognition rather than relying on the last connected layer of CNNs. In the proposed method, a CNN model (VGG-face) used as a face recognition model is adapted to the anti-spoofing problem. The number of outputs in the fully connected layer of the model is reduced from 2622 to 2 and adapted to the model. Then, potential overfitting issues are mitigated by applying a block principal component analysis (PCA) technique to reduce the feature dimensionality. Finally, a SVM classifier is used to distinguish between real and fake faces. Experimental evaluations on publicly available datasets such as Replay-Attack and CASIA show that the proposed method provides improvements over existing state-of-the-art techniques, giving 2.9% *Equal Error Rate* (EER) and 4.5% EER respectively.

In [31], a CNN-based approach for PAD is presented. Wide Multi-Channel Presentation Attack (WMCA) model is designed instead of the traditional featured CNN models. With WMCA, 2D and 3D presentation attacks including both impersonation and obfuscation techniques are studied. Instead of a software-based and texture-based approach, different data from various channels such as color, depth, infrared, thermal, and infrared are considered along with hardware. With Intel RealSense SR300 sensor and Seek Thermal Compact PRO sensor hardware, the

dataset was collected by considering different environmental conditions, light, background, etc., and the dataset was created and studied with fake presentation types such as print, replay, rigid mask, paper mask, etc. Through extensive experiments, we have achieved an ACER of 0.3% on the newly introduced WMCA dataset, demonstrating superior performance against baseline methods.

[32] purposes (CNN)-based framework for PAD, integrating deep pixel-wise supervision. Since the proposed model only works on frame-level information, it requires minimal computational and time overhead, making it suitable for use in smart devices. The proposed model is inspired by the DenseNet architecture (has the first eight layers as a pre-trained model trained in the ImageNet dataset, followed by two dense blocks and two transition blocks. At the end of the structure, a fully connected MLP-type layer is added for classification purposes). The framework is a densely connected NN trained using binary and pixel-wise supervision. The model was trained on various public datasets and outperformed state-of-the-art methods with 0% HTER and 0.42% ACER on the Replay Mobile dataset (in Protocol-1 of the OULU-NPU dataset).

In [33], using a light field camera, a model was created that records not only the intensity but also the direction of the rays coming from the camera, obtaining multiple depth and focus points in a single shot and classifying using the variation of these depth and focus points. The model includes Face Detection and pre-processing, Focus Measure, Estimating the variation in focus, and Classification steps. In the focus measure step, methods such as gradient-based, statistical-based, transformation-based and image characteristic-based were tested and compared. In the study, images of 80 people were recorded with a light field camera, and the model was trained with these images. It stands out among the state-of-the-art methods with an APCER value of 3.88% and an ACER value of 5.27%.

The method proposed in [34] describes a CNN-RNN model designed to estimate face depth with pixel-wise supervision and remote photoplethysmography (rPPG) signals with sequential supervision. In general, the model is a combination of 2 different architectures. A CNN, which resembles a typical ResNet architecture, and 2 branches to estimate the depth and feature map at the output of the model. An RNN model with 100 hidden neurons and 1 fully connected layer to predict the rPGG signal. By combining the depth and rPGG signals from these two separate architectures, the

model effectively distinguishes between real and fake faces. The promised model is tested on various public datasets and achieves 1.6% APCER, BPCER, and ACER on Protocol-1 of the OULU-NPU dataset.

In the study [35], which aims to find a solution to PAD with a new texture descriptor called Equilibrium Difference Local Binary Pattern (ED-LBP), the inconsistency of adjacent pixels in face images is taken into account and this inconsistency is encoded into LBP. The method also computes feature histograms on each image band separately. The model combines the ED-LBP histograms obtained from various color spaces into a unified feature vector and classification is performed using SVM at the final layer. The method has been tested on publicly available datasets and has achieved remarkable success 8.33 % ACER on the Protocol-1 of the OULU-NPU dataset among state-of-the-art methods.

4. MATERIALS AND METHODS

4.1. Dataset Description

This study uses the OULU-NPU dataset, which was created specifically for the detection of mobile face presentation attacks [21]. The dataset was created in three different environments under different ambient lights to account for real-world variations. To examine the print and replay attack types, two different printers and displays were used. Videos that make up the entire dataset were recorded using six different mobile devices' front cameras, and captured videos were categorized into two classes: fake and real. Three subsets were created from the 55 subjects' videos for testing, development, and training purposes. The dataset distribution is shown in Figure 4.1. The content and diversity of the dataset were suitable for solving the problem related to the thesis topic, and it was thought that it would be sufficient in size, so no extra dataset was needed.

	Users	Real access	Print attacks	Video attacks	Total
Training	20	360	720	720	1800
Development	15	270	540	540	1350
Test	20	360	720	720	1800

Figure 4.1 OULU-NPU Dataset Distribution

4.2. Building Dataset

The CNN model that we use to solve the problem of face presentation attack uses fixed-size images in the input layer. The OULU-NPU dataset consists of video images with consecutive frames, so it is necessary to extract image frames containing face data from the dataset camera views. In order to achieve this goal, all videos in three separate folders of the original dataset in two separate categories (bonafide and attack) were visited using the relevant Python libraries. 30 frames (samples) taken from each video. The faces were detected on them. Then, the detected face images

were resized to 224x224 and were used to create a new dataset placed under the relevant folders as Train, Test, and Development for using in future CNN models training. This new dataset is supported with the relevant CSV files using Python libraries. The CSV files contain the image file paths, and the class labels for them. All these steps and the flow diagram of our work are shown in Figure 4.2.

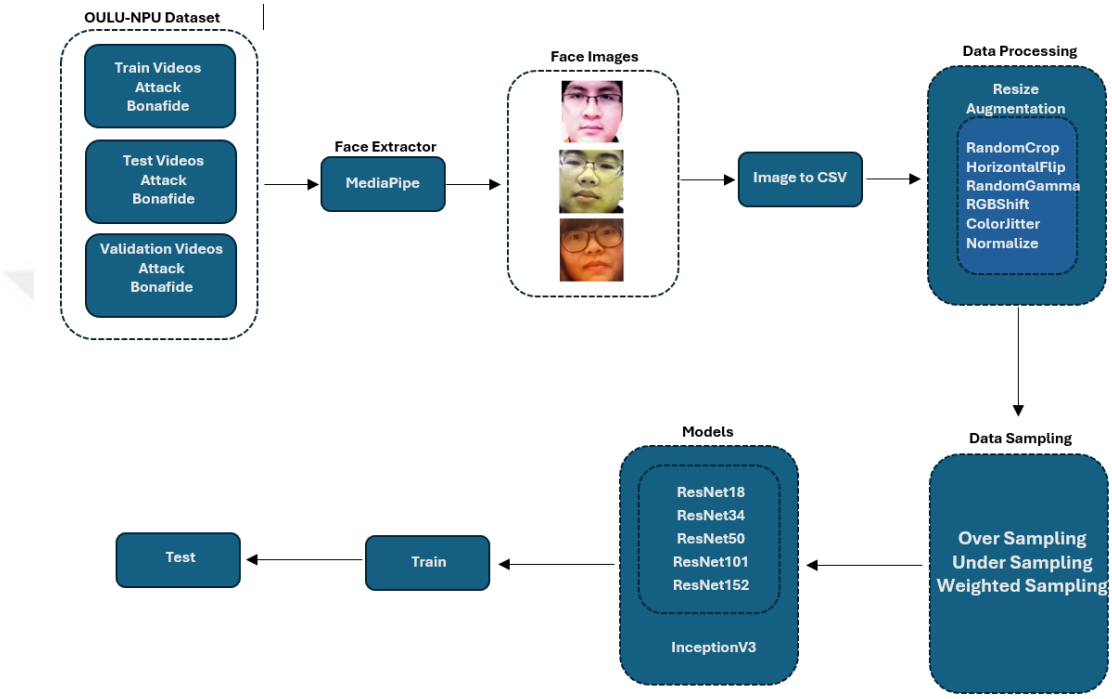


Figure 4.2 Our Workflow

4.3. Face Detection

Today, face detection can be done with the help of many algorithms. These algorithms differ from each other in terms of computational load and detection accuracy. In this study, two different face detection algorithms are tested for dataset generation and compared with each other. First of all, an algorithm was designed with the MTCNN algorithm, which is frequently used in the literature, for face detection and saving the detected face in the relevant folders, but both the slow operation of the algorithm and the long time it took to create the dataset and the detection of different objects instead of faces in some frames of the videos in the original dataset, as shown in Figure 4.3, led to the need to use a different algorithm in the study. ***Mediapipe's***

face detection algorithm BlazeFace model was used as a second algorithm in our study and the dataset was created as desired without any problems. This is a new contribution to the PAD studies in the literature.

Figure 4.3 Detection samples from MTCNN and Mediapipe

4.3.1. Mediapipe's BlazeFace Algorithm

Mediapipe is a set of libraries and tools developed by Google AI for the rapid implementation of AI and ML techniques [22]. In this study, the face detection algorithm supported by Mediapipe was used. In the face detection algorithm, Mediapipe uses the ***BlazeFace algorithm***, which was developed by Google AI in 2019, which is lightweight and can detect in real-time, suitable for use on mobile devices. It is an algorithm with a more compact feature extractor CNN inspired by MobileNetV1/V2 and adapted to the Single Shot Multibox Detector (SSD).

As shown in Figure 4.4, the BlazeFace architecture uses a 5x5 kernel size in the deep convolution layer. This approach makes the model shallower, enabling the model to process 200 - 1000 FPS images on mobile devices and minimizing the detection time [23].

Layer/block	Input size	Conv. kernel sizes
Convolution	128×128×3	5×5×3×24 (stride 2)
Single BlazeBlock	64×64×24	5×5×24×1 1×1×24×24
Single BlazeBlock	64×64×24	5×5×24×1 1×1×24×24
Single BlazeBlock	64×64×24	5×5×24×1 (stride 2) 1×1×24×48
Single BlazeBlock	32×32×48	5×5×48×1 1×1×48×48
Single BlazeBlock	32×32×48	5×5×48×1 1×1×48×48
Double BlazeBlock	32×32×48	5×5×48×1 (stride 2) 1×1×48×24 5×5×24×1 1×1×24×96
Double BlazeBlock	16×16×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	16×16×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	16×16×96	5×5×96×1 (stride 2) 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	8×8×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	8×8×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96

Figure 4.4 BlazeFace model architecture

As shown in Figure 4.5, face images were successfully extracted and saved using the Mediapipe.

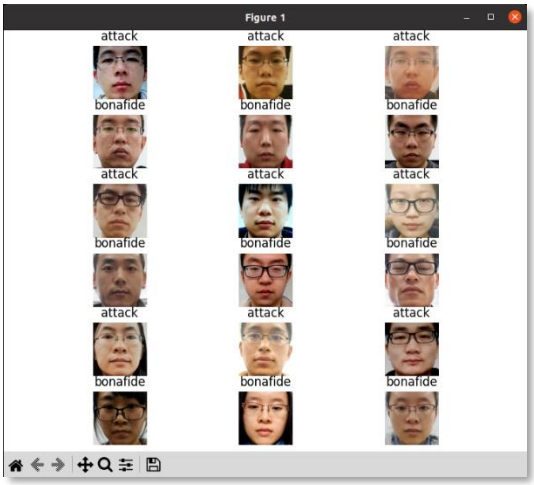


Figure 4.5 Our extracted face images from the original OULU-NPU Dataset

4.4. Data Augmentation

Although DL algorithms have achieved successful results in many areas today, these networks need a large and diverse dataset to successfully solve real-life problems and avoid overfitting problems. But often, collecting data in a variety and size to solve the problem is a difficult and laborious step in DL applications [24]. The larger the dataset to be used for the model and the more diverse the data, the more the model learns and gives accurate results. So the question is “If the dataset is limited, what should be done?”. To overcome the obstacle brought by this limitation, data augmentation is used. In order to enhance the performance of DL models, it is a technique to artificially enlarge the dataset by generating new data from existing data or altering current data. In this study, different data augmentation methods were applied to the dataset and augmented face images shown in Figure 4.6.

RandomCrop: The Crop method is a data augmentation method frequently used in image classification problems. It allows the model to recognize different positions and sizes of objects and improves the generalization ability of the model by increasing the variance in the dataset. However, as in this study, random crop can have a negative impact on solving some problems. While developing the model, random crop of the face images from which we extract features may cause loss of the features and prevent the model from learning properly. Therefore, the accuracy of the model can be compared in scenarios with and without the random crop method.

Horizontal Flip: Horizontal Flip method is another frequently used method in image classification problems. It allows the model to better recognize changes in the , helps the model become more resistant to the object's orientation, and helps the model become more resistant and resilient to rotation and symmetry. This method allows the pixels on the vertical axis to remain fixed while the pixels on the horizontal axis are changed.

RandomGamma: The images in the dataset we used for PAD were taken under different environmental conditions. When we consider the face presentation attacks, it is thought that the print or replay attacks shown to the camera will have different light levels and contrasts, so the random gamma method was applied to the dataset. In this way, the images taken in different environments and light conditions can be better recognized by the model.

RGBShift: When we examine the problem addressed in this study, face presentation attacks can be performed with different device screens (like in the mobile phone usage) or printing methods. With this method applied to the dataset, color channels are shifted to increase color variations in the image. Thus, the model can become more successful in distinguishing different lighting environments or image transmission differences. It is also thought that the model will perform better at different levels of colors with this method.

ColorJitter: With this method, different properties of the image such as brightness, contrast, saturation, and color balance can be changed. A random color change for each channel on the selected image can help the model perform better under different lighting conditions and image transmission method differences.

Normalization: The pixel values of the images used for the model range from 0 to 255. By normalizing these values across the dataset, the model can learn faster and more consistently. The mean and standard deviation of the pixel values of the images used in the dataset are calculated and normalization is performed with these values on the dataset.

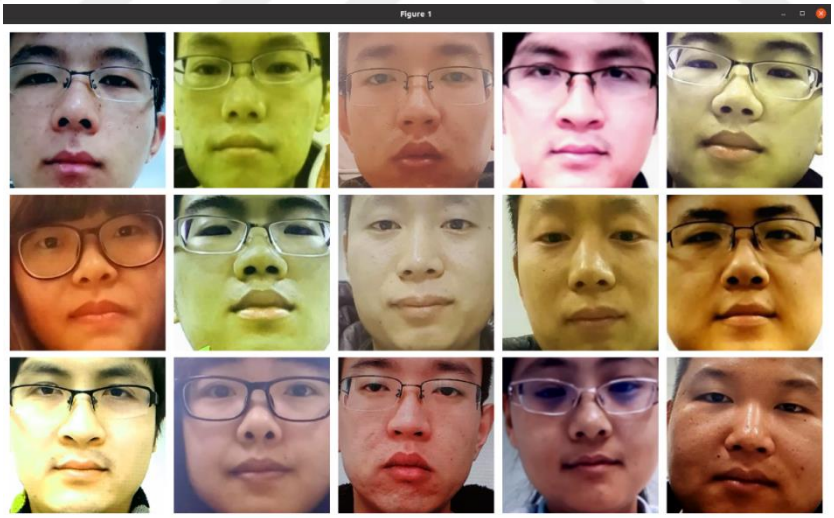


Figure 4.6 Augmented Images

4.5. Data Sampling

In 1993, Anand et al. studied how class imbalance affects backpropagation and optimization using a shallow NN [25]. They concluded that losses and gradients during

optimization are dominated by the majority class and the minority class has little effect on weight updates. Similarly, in this study, the face images extracted from the original dataset are imbalanced because the videos belonging to the attack class are more than the videos belonging to the bonafide class in the dataset. The face images prepared for use in the model are approximately 40000 attacks and 10000 bonafide. This unbalanced distribution may cause the model to memorize attack images and fail to learn bonafide images. To eliminate this situation and to enable the model to learn both classes correctly, as another contribution of our study, 3 different data sampling methods were used on the dataset and their effects on the model were examined.

4.5.1. Undersampling

The images to be used in the training phase were created by applying the sampling process after the data augmentation process. With the undersampling method, random samples were selected from the majority class and approximated to the minority class, and thus a new training dataset was created. After the undersampling method, the dataset was trained with approximately 10000 images in both classes and the results are analyzed in Table 5.3. With the undersampling method, the imbalance between the classes was eliminated and the effect of the minority class on the model was expected to increase. Since the size of the dataset is reduced with the undersampling method, the training speed of the model is expected to increase, but it is predicted that randomly selected and reduced samples from the majority class may cause the model to learn less and miss important details.

4.5.2. Oversampling

With the oversampling method, another data sampling method, random images from the minority class were selected and new samples were created from these images to bring them closer to the majority class. In this way, no data from the majority class was lost and it was planned to increase the impact of the minority class on the model thanks to the new images created by the minority class. After the oversampling method, the dataset was trained with approximately 40000 images in both classes. With the oversampling method, the imbalance between the classes was eliminated and the effect of the minority class on the model was expected to increase. The

oversampling method effect is shown in Table 5.3. With this method, the training speed of the model was expected to decrease due to the increase in the size of the dataset, but since there is no data loss in the majority class and the imbalance between the classes is eliminated, it is planned to prevent the model from memorizing.

4.5.3. Weighted Sampling

Another data sampling method is weighted random sampling. With this sampling method, depending on the weights of the classes in the dataset, each class is sampled equally in the desired size of the dataset. For example, the dataset size was set to 40000 and as a result, the training dataset was created with approximately 20000 images from each class. With this method, it is aimed to eliminate the class imbalance, and the ideal training data set is created by the flexible application of the dataset size. The model training speed can be optimized according to the sampling size and the model memorization can be prevented by weighted sampling. The weighted sampling method effect is shown in Table 5.3.

4.6. Base Models

4.6.1. ResNet

When solving a problem with CNN, it was predicted that a deeper model would solve complex problems better. Although the qualities learned for complex problems increase as the model gets deeper, the problem arises that as the model gets deeper, the backward propagating gradients become weaker and weaker. In other words, as we move toward the input layer, the gradients become very small or disappear, which negatively affects the training and performance of deep networks. With the ResNet architecture developed to solve this problem, gradients can flow directly through the connection points to the first layers.

In this study, the solution of the problem is tested on different ResNet architectures, and their performances are compared in Table 5.4. There are different models in ResNet architecture with 18, 34, 50, 101, and 152 layers. As shown in Figure 4.7, although the general structure is the same in these models, there are 4 different general layers in the ResNet architecture and the number of layers in the blocks within these layers increases.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 [3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.7 ResNet architectures created for ImageNet [11]

As can be seen in Figure 4.7, each ResNet architecture, regardless of the number of layers, has a kernel size of 7x7 and a feature map size of 64, with padding 3 and stride 2, and then the output size is reduced to 112x112. After the Conv1 output, stride 2 was applied with a maximum pooling of 3x3. The conv1 layer and the subsequent maximum pooling are also applied in other ResNet models with different numbers of layers, and then in each model, the number of repeated operations in the blocks within the layers is varied, so that the output sizes of the main layers are maintained, but the depth of the models is increased due to the increase of layers within the blocks. The output of each main layer (conv2_x, conv3_x, conv4_x, and conv5_x) was down-sampled with stride 2 and the output size was halved. Once the output size was halved, the number of filters in the next layer was doubled to maintain time complexity in each layer.

If we look at the operations within the blocks of the ResNet-34 architecture, there are convolution, batch normalization, and ReLu operations in each block. If we look at the operations in Block-1 in Layer-1 in Figure 4.8, first, a convolution process with

a 3x3 kernel size and 64 filters were applied. This process was then repeated once more as shown in Figure 4.8(b).

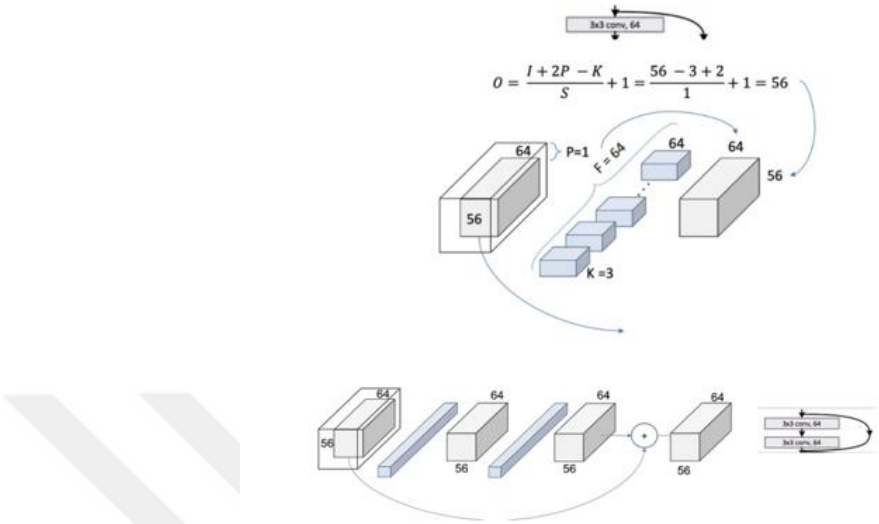


Figure 4.8 (a) ResNet Layer-1 Block-1 Operation (b) Repeating Block ops [26]

These two operations were then repeated 3 times to create Layer 1 as shown in Figure 4.9.

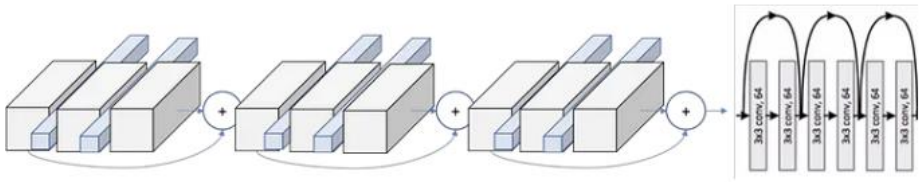


Figure 4.9 ResNet Layer-1 [26]

In this study, Python and Pytorch libraries were used to configure the ResNet model and solve the problem, and the necessary software was prepared and ResNet models were used.

4.6.2. InceptionV3

InceptionV3 is a CNN model developed by Christian Szegedy and his research team to win first place at ILSVRC in 2014. InceptionV3 builds on Google's previous Inception models and thanks to innovations, it achieves higher performance using fewer parameters.

InceptionV3 takes an RGB image of 299x299 pixels in the Input Layer. In the next few layers, the image is reduced to a smaller size, and the basic features of the image are extracted. The main layer of the Inception model consists of parallel convolution layers and pooling layers that use filters of different sizes (1x1, 3x3, 5x5) in the Inception modules. This allows the model to extract information from images at multiple scales. The Auxiliar Classifier is a classifier layer located in the deeper layers of the model, which helps the model to extract more information in the early stages. The output layer includes global average pooling and fully connected layers. These layers are used to classify features extracted from the image. In Figure 4.10, we can see the modified versions of original Inception Model v1 module structure. The architecture of the new InceptionV3 model constructed with the modified module structures is shown in Figure 4.11.

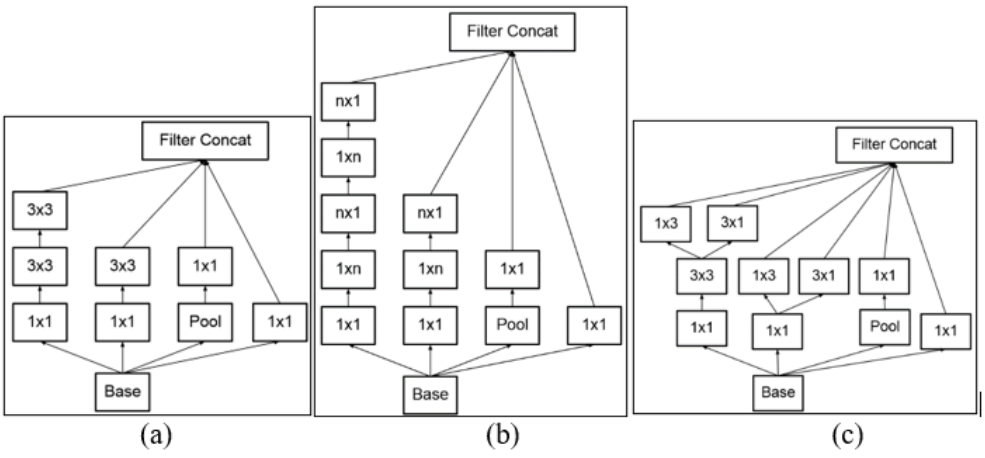


Figure 4.10 Inception.v3 Modules (a) Modified Inception Module (Original Inception modules where each 5×5 convolution is replaced by two 3×3 convolutions (b) Inception modules after the factorization of the n × n convolutions (generally n=7) (c) Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8×8) grids to promote high-dimensional representation [36]**.**

Type	Patch size/stride	Input size
conv	3x3/2	299x299x3
conv	3x3/1	149x149x32
conv padded	3x3/1	147x147x32
pool	3x3/2	147x147x64
conv	3x3/1	73x73x64
conv	3x3/2	71x71x80
conv	3x3/1	35x35x192
3xInception Mod1	As in Figure 4.9(a)	35x35x288
5xInception Mod2	As in Figure 4.9(b)	17x17x768
2xInception Mod3	As in Figure 4.9(c)	8x8x1280
pool	8x8	8x8x2048
linear	logits	1x1x2048
softmax	classifier	1x1x1000

Figure 4.11 The outline of Inception.v3 architecture [36]

4.7. MixStyle (MS) Method

When we consider the problem that this study aims to provide a solution to, considering that the data set on which the model is trained may be limited and the model may not be able to learn well enough, it is thought that the model should perform better learning by using various techniques on the data set. Domain Adaptation method can be applied to reduce the difference between the data set used in model training and real-world data. As another method, the Domain generalization method can be applied to make the model more successful in different or unknown domains. To find solutions to these problems, a new ANN architecture called MixStyle was developed in 2021 by Kaiyang Zhou et al [27]. Thanks to the MS module used in this study, which works as a plug-and-play and does not need any extra parameters, it is tried to increase the accuracy of the model without collecting more data.

Inspired in 2017 by the Adaptive Instance Normalization (AdaIN) method introduced by Xun Huang and Serge Belongie, MS is designed to edit CNN training by distorting the source domain style information instead of using a decoder to create a new image. In short, it can be easily run by placing it between the layers of the CNN model, as shown in Figure 4.12, without the need to create a new styled image [28]. In this figure we see two image instances with q and q`being their ground-truth labels. ϑℓ() represents the cross-entropy loss function. f and f' are feature maps extracted at

a certain layer for input images. μ and σ are the channel-wise mean and standard deviation. λ is random weighted sampled from beta distribution.

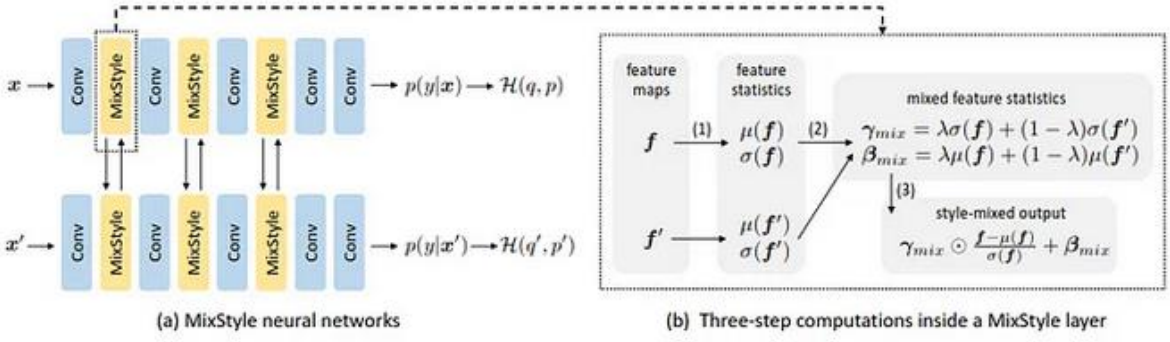


Figure 4.12 Schematic Overview MixStyle Neural Network [28]

The main idea of the MS module is to adapt the model to this diversity by mixing style features from different images during the training process. In this way, the model is improved to deal not only with the examples in the training dataset but also with new visual styles that it has not encountered before. MS simulates new styles by mixing the feature statistics between two samples using a random convex weight as shown in Figure 4.13. Given an input batch x, MS first creates a reference batch from this input batch. If domain labels are available, x is sampled based on samples from two different domains i and j. If domain labels are unknown, x is randomly sampled from the training data [27].

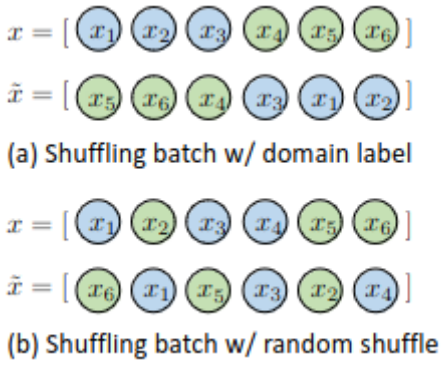


Figure 4.13 Generation of a Reference Batch [27].

4.8. Counterfactual Intervention (CI) Method

Considering the problem we are trying to solve in this study, a well-trained model should be independent of some features such as environmental conditions, sensor type and quality, and the identity of the person in front of the camera. Since these factors will vary in real-life conditions, we wanted to train the model in a way that minimizes them. Considering that for each input X, the model learns and outputs a Z vector with input features, this Z vector has various properties.

Cause factors are the class-dependent information encoded in Z. They are linked to universal and invariant patterns of bonafide or attack. The identity information in Z may include non-causal factors such as the environmental conditions in which the image was taken.

Sensor information is also very important in the prediction process of the model, as information from the capture sensors can be misleading for the model, and sensor information is often treated as a distortion factor in the literature.

In order to learn how all these causal factors affect the model, some causal factors are removed from Z and $\overline{Z}$ is generated. In this study, since it is impractical to generate counterfactual features directly and the causal factors in the high-level problem-specific features are unobservable and cannot be formulated, we tried to generate $\overline{Z}$ with 3 simple interventions (random zero, random shuffle and random replace).

Next, the difference between the initial forecast and its counterfactual alternate prediction is used to formulate the causal effect:

$$Y_{effect} = Y(Z) - Y(\overline{Z}) \tag{4.1}$$

The loss function is formulated as:

$$L = L(Y, y) + \lambda L(Y_{effect}, y) \tag{4.2}$$

After the Feature Encoder part of the model, the CI method was applied to the features obtained at a certain rate. Random zero, random shuffle and random

replace are applied to the Z vector and then the new vector is passed through the Classifier layer. The process is shown in detail in Figure 4.16.

4.9. Classifiers

In the two different CNN architectures considered in this study, the Z feature obtained from the Feature Encoder of the models and the $\bar{Z}$ feature obtained after being passed through the CI method, are passed through the Classifier layer of the model as shown Figure 4.14. Considering the problem to be solved in the thesis, the model has two classes to predict, attack and bonafide. Therefore, the output of the Classifier layers of the models was created to predict two separate classes. In the study, Linear and Convolution Classifiers were created and the effect of these classifiers on the model was compared in Table 5.2.

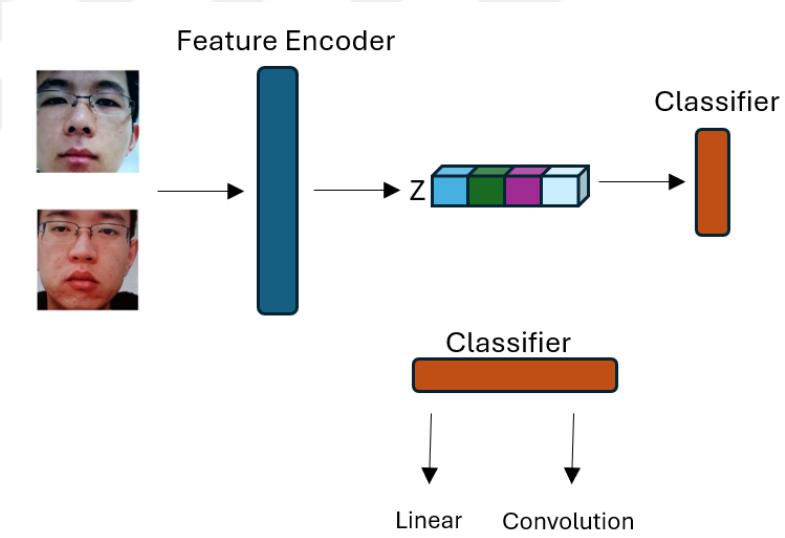


Figure 4.14 Z feature trough to Classifier Layer

4.9.1. Linear Classifier

A Classifier class was defined using Python libraries to be used in the classifier layer of the model. While developing the model for solving the problem in this study, the number of classes we want to classify is two, attack and bonafide. To predict two classifications, a linear classifier was created with the Pytorch library. In the classifier

layer created with the nn.Linear class, the number of input channels in the last layer of the ResNet structure before the classifier layer is given as 512, and the number of classes to be predicted is set to 2. In addition, the mean value of the linear layer is set to 0 and the standard deviation is set to 0.01 and it is set to be normally distributed. The bias values of the layer are set to 0. In the forward part of the class, the norm_flag value is set so that it can be optionally normalized. The Python code written in this way is set to be used in all ResNet architectures.

4.9.2. Convolutional Classifier

In the classifier part of the models we used in this study, a second classifier was created to compare with the Linear classifier and to understand the effect of the classifier layer on the model. After the Feature Encoder part of the models, the resulting vector is given to a simple CNN, and the output weights are obtained. The classifier architecture is shown in Figure 4.15.

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
├─Conv2d: 1-1                            1,179,904
├─BatchNorm2d: 1-2                       512
├─ReLU: 1-3                              --
├─Conv2d: 1-4                            295,040
├─BatchNorm2d: 1-5                       256
├─AdaptiveAvgPool2d: 1-6                 --
├─Linear: 1-7                            258
==========================================================================
Total params: 1,475,970
Trainable params: 1,475,970
Non-trainable params: 0
==========================================================================
```

Figure 4.15 Convolutional Classifier Summary

4.10. Our Proposed Models

In this study, two separate models have been developed based on ResNet and Inception models. The base architectures of these models were created using Pytorch and related Python libraries and MS and CI methods were integrated into the relevant parts of the models. For the ResNet architecture, the MS method was applied in layers 1 and 2 and the CI method was applied before entering the Classifier. Both methods were applied only in the training phase and thus the methods were implemented in a

plug-and-play manner without the need for any separate processing. Thanks to the dynamic software created the methods in the study were easily integrated into all Resnet Models and the metrics of the models were compared in Table 5.4. Likewise, to compare the effect of the methods on the models and the effect of different models on the solution of the problem, the InceptionV3 model was dynamically created with the relevant libraries, and the relevant methods were integrated into the model. Again, the comparison of each model is shown in Table 5.4.

The workflow of the model is as follows:

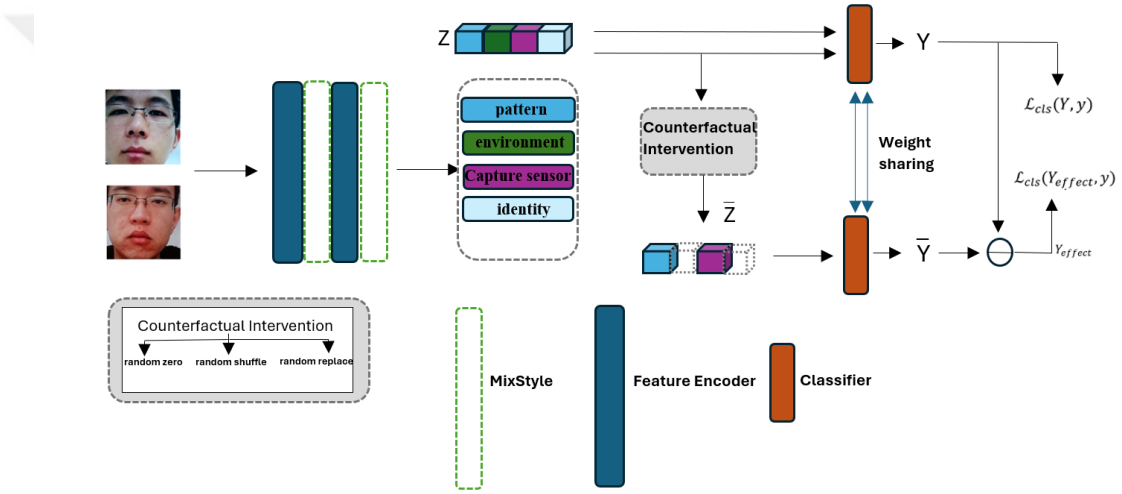


Figure 4.16 Our Proposed Models Workflow

4.10.1. Our Mixed ResNet Model

The first model used in the study is the ResNet Models, which include MS and CI modules integrated into the ResNet architecture. In this study, five ResNet architectures were studied, and the models were compared. Each model was run on RTX 3060 Laptop GPUs and since the GPUs memory was insufficient in the deeper models, the batch size value determined as 128 was reduced and the model was successfully run on GPUs. MS method is integrated between the convolutional layers (layer1 and layer2) as shown in Figures 4.12 and 4.16. The CI method was applied

after the feature encoder process and before the Classifier process as shown in Figure 4.16. These two methods were used only in the training phase. The summaries of the models are shown in the related Figures 4.17, 4.18, 4.19, 4.20, and 4.21.

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
├─MixStyleResCausalModel: 1-1           --
│    └─ResNet: 2-1                      --
│    │    └─Conv2d: 3-1                 9,408
│    │    └─BatchNorm2d: 3-2            128
│    │    └─ReLU: 3-3                   --
│    │    └─MaxPool2d: 3-4              --
│    │    └─Sequential: 3-5             147,968
│    │    └─Sequential: 3-6             525,568
│    │    └─Sequential: 3-7             2,099,712
│    │    └─Sequential: 3-8             8,393,728
│    │    └─MixStyle: 3-9               --
│    └─AdaptiveAvgPool2d: 2-2           --
│    └─Classifier: 2-3                  --
│    │    └─Linear: 3-10                1,026
│    └─RandomReplace: 2-4               --
│    └─RandomZero: 2-5                  --
│    └─ChannelShuffleCustom: 2-6        --
==========================================================================
Total params: 11,177,538
Trainable params: 11,177,538
Non-trainable params: 0
==========================================================================
```

Figure 4.17 Our Mixed ResNet18 Model Summary

```
Layer (type:depth-idx)                   Param #
==========================================================================
├─MixStyleResCausalModel: 1-1           --
│    └─ResNet: 2-1                      --
│    │    └─Conv2d: 3-1                 9,408
│    │    └─BatchNorm2d: 3-2            128
│    │    └─ReLU: 3-3                   --
│    │    └─MaxPool2d: 3-4              --
│    │    └─Sequential: 3-5             221,952
│    │    └─Sequential: 3-6             1,116,416
│    │    └─Sequential: 3-7             6,822,400
│    │    └─Sequential: 3-8             13,114,368
│    │    └─MixStyle: 3-9               --
│    └─AdaptiveAvgPool2d: 2-2           --
│    └─Classifier: 2-3                  --
│    │    └─Linear: 3-10                1,026
│    └─RandomReplace: 2-4               --
│    └─RandomZero: 2-5                  --
│    └─ChannelShuffleCustom: 2-6        --
==========================================================================
Total params: 21,285,698
Trainable params: 21,285,698
Non-trainable params: 0
==========================================================================
```

Figure 4.18 Our Mixed ResNet34 Model Summary

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
├─MixStyleResCausalModel: 1-1            --
|    └─ResNet: 2-1                       --
|    |    └─Conv2d: 3-1                  9,408
|    |    └─BatchNorm2d: 3-2             128
|    |    └─ReLU: 3-3                    --
|    |    └─MaxPool2d: 3-4               --
|    |    └─Sequential: 3-5              215,808
|    |    └─Sequential: 3-6              1,219,584
|    |    └─Sequential: 3-7              7,098,368
|    |    └─Sequential: 3-8              14,964,736
|    |    └─MixStyle: 3-9                --
|    └─AdaptiveAvgPool2d: 2-2            --
|    └─Classifier: 2-3                   --
|    |    └─Linear: 3-10                 4,098
|    └─RandomReplace: 2-4                --
|    └─RandomZero: 2-5                   --
|    └─ChannelShuffleCustom: 2-6         --
==========================================================================
Total params: 23,512,130
Trainable params: 23,512,130
Non-trainable params: 0
==========================================================================
```

Figure 4.19 Our Mixed ResNet50 Model Summary

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
├─MixStyleResCausalModel: 1-1            --
|    └─ResNet: 2-1                       --
|    |    └─Conv2d: 3-1                  9,408
|    |    └─BatchNorm2d: 3-2             128
|    |    └─ReLU: 3-3                    --
|    |    └─MaxPool2d: 3-4               --
|    |    └─Sequential: 3-5              215,808
|    |    └─Sequential: 3-6              1,219,584
|    |    └─Sequential: 3-7              26,090,496
|    |    └─Sequential: 3-8              14,964,736
|    |    └─MixStyle: 3-9                --
|    └─AdaptiveAvgPool2d: 2-2            --
|    └─Classifier: 2-3                   --
|    |    └─Linear: 3-10                 4,098
|    └─RandomReplace: 2-4                --
|    └─RandomZero: 2-5                   --
|    └─ChannelShuffleCustom: 2-6         --
==========================================================================
Total params: 42,504,258
Trainable params: 42,504,258
Non-trainable params: 0
==========================================================================
```

Figure 4.20 Our Mixed ResNet101 Model Summary

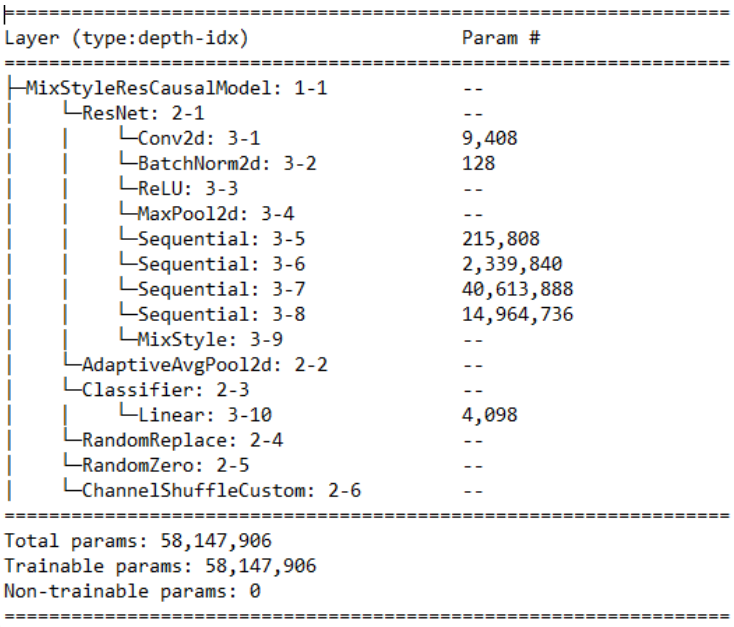
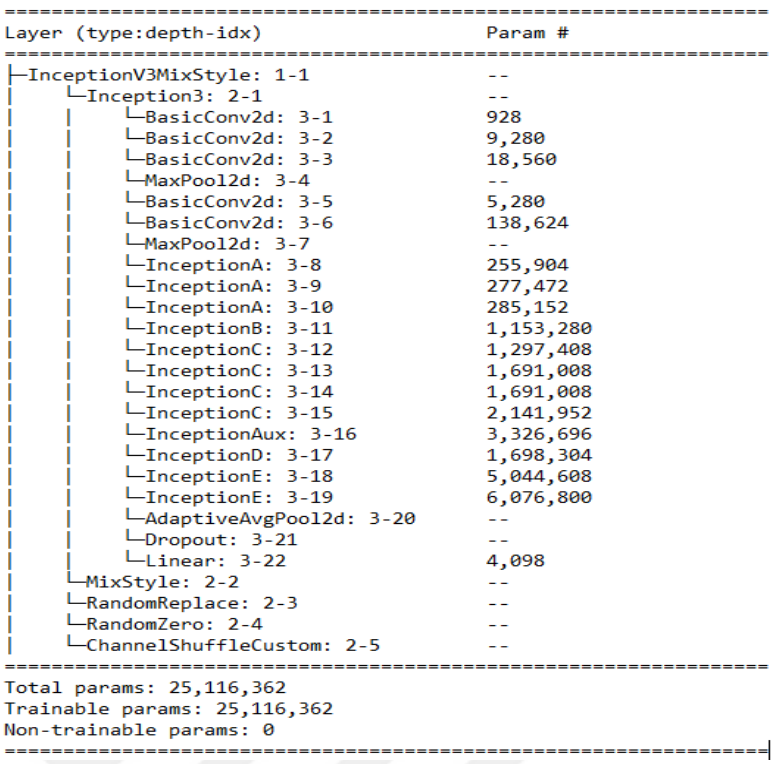


Figure 4.21 Our Mixed ResNet152 Model Summary

4.10.2. Our Mixed InceptionV3 Model

In our study, a second model was implemented and trained to understand how MS and CI methods applied to different CNNs affect the model. The architecture of this model was designed as an InceptionV3 model, and the MS method was applied after the selected layers. After the MS method, the Classifier layer was applied, and the CI method was applied to a certain percentage of the output. The CI method was applied at a rate of 0.2 in both models. After applying the CI method to the feature extracted from the feature extractor, the output was passed through the Classifier layer and the weights were calculated. The model summary is shown in Figure 4.22.



```
==========================================================================
Layer (type:depth-idx)                             Param #
==========================================================================
├─InceptionV3MixStyle: 1-1                        --
|    └─Inception3: 2-1                            --
|    |    └─BasicConv2d: 3-1                      928
|    |    └─BasicConv2d: 3-2                      9,280
|    |    └─BasicConv2d: 3-3                      18,560
|    |    └─MaxPool2d: 3-4                        --
|    |    └─BasicConv2d: 3-5                      5,280
|    |    └─BasicConv2d: 3-6                      138,624
|    |    └─MaxPool2d: 3-7                        --
|    |    └─InceptionA: 3-8                       255,904
|    |    └─InceptionA: 3-9                       277,472
|    |    └─InceptionA: 3-10                      285,152
|    |    └─InceptionB: 3-11                      1,153,280
|    |    └─InceptionC: 3-12                      1,297,408
|    |    └─InceptionC: 3-13                      1,691,008
|    |    └─InceptionC: 3-14                      1,691,008
|    |    └─InceptionC: 3-15                      2,141,952
|    |    └─InceptionAux: 3-16                    3,326,696
|    |    └─InceptionD: 3-17                      1,698,304
|    |    └─InceptionE: 3-18                      5,044,608
|    |    └─InceptionE: 3-19                      6,076,800
|    |    └─AdaptiveAvgPool2d: 3-20               --
|    |    └─Dropout: 3-21                         --
|    |    └─Linear: 3-22                          4,098
|    └─MixStyle: 2-2                              --
|    └─RandomReplace: 2-3                         --
|    └─RandomZero: 2-4                            --
|    └─ChannelShuffleCustom: 2-5                  --
==========================================================================
Total params: 25,116,362
Trainable params: 25,116,362
Non-trainable params: 0
==========================================================================
```

Figure 4.22 Our Mixed InceptionV3 Model Summary

5. EVALUATION

5.1. Evaluation Metrics

In this study, the critical parameters considered in PAD applications are calculated and recorded during the training of the model and evaluated afterwards. The equations from eq (5.1) to eq (5.4) are the well-known metrics of the confusion matrix. Besides them, for comparative purposes, we present the Half Total Error Rate (HTER) in eq (5.5), which is the average of the Attack Presentation Classification Error Rate (APCER or FPR) in eq (5.6), Normal Presentation Classification Error Rate (NPCER) in eq (5.7), the Bona fide Presentation Classification Error Rate (BPCER) in eq (5.8), Average Classification Error Rate (ACER) in eq (5.9), and the Area under the Receiver Operating Characteristic (ROC) Curve (AUC) value.

i. **False Positive Rate (FPR):** FPR indicates the rate at which the model falsely classified samples as positive when they are actually negative. The FPR expressed by the equation in (5.1) corresponds to the x-axis on the ROC curve and the smaller the FPR value of the model, the better the model performs.

$$\mathbf{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \tag{5.1}$$

ii. **True Positive Rate (TPR):** TPR indicates the proportion of truly positive samples that are correctly classified as positive. The TPR expressed by the equation in (5.2) corresponds to the y-axis on the ROC curve and the higher the TPR value, the better the model performs.

$$\mathbf{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{5.2}$$

iii. **Threshold:** Threshold specifies the threshold value used to separate positive and negative classes in classification. The outputs of the model are determined according to this threshold value. For example, if the threshold value is 0.5, the model accepts values greater than 0.5 as positive.

In the code section for the model we developed, these values were found using the "sklearn.metrics" library and used to calculate the other values reported in the study.

iv. **Right index:** Right index is the index value at the point where the absolute value of the difference between the TPR and FPR values obtained during the calculation of the ROC curve during the training of the model is minimum.

v. **Equal Error Rate:** ERR represents the FPR value at the point where the FPR and TPR values are equalized (their difference is the smallest).

$$\mathrm{ERR} = \mathrm{FPR}[\mathrm{right\_index}] \tag{5.3}$$

vi. **False Rejection Rate (FRR):** FRR indicates the false rejection rate of true positive samples in biometric recognition and classification applications. The equation of the FRR is (5.4):

$$\boldsymbol{FRR} = 1 - TPR \tag{5.4}$$

vii. **Half Total Error Rate (HTER):** HTER is a metric that shows the overall error rate in classification applications. As in Equation (5.5), it includes both false acceptance and false rejection errors.

$$\mathbf{HTER} = \frac{\mathrm{FPR} + \mathrm{FRR}}{2} \tag{5.5}$$

viii. **Attack Presentation Classification Error Rate (APCER):** APCER refers to the rate at which fake presentation examples are mistakenly accepted in PAD applications. The lower this value is, the better the model can detect presentation attacks. The equation of APCER shown blow:

$$\mathbf{APCER} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \tag{5.6}.a$$

$$= \frac{Number\ of\ attack\ presentations\ classified\ as\ genuine}{Total\ number\ of\ attack\ presentations} \qquad (5.6).b$$

ix. **Normal Presentation Classification Error Rate (NPCER):** APCER refers to the rate at which real presentation examples are mistakenly accepted in PAD applications.

$$\mathbf{NPCER} = \frac{\text{False Negatives}}{\text{False Negatives} + \text{True Positives}} \qquad (5.7.a)$$

$$= \frac{Number\ of\ normal\ presentations\ classified\ as\ attacks}{Total\ number\ of\ normal\ presentations} \qquad (5.7.b)$$

x. **Bonafide Presentation Classification Error Rate (BPCER):** BPCER and NPCER are conceptually similar, as both measure the rate of genuine presentations being misclassified as attacks. However, BPCER is the term specifically defined in the context of the ISO/IEC 30107 standard.

$$\mathbf{BPCER} = \frac{\text{Number of bona fide presentations classified as attacks(FN)}}{\text{Total number of bona fide presentations (FN + TP)}} \qquad (5.8)$$

xi. **Average Classification Error Rate (ACER):** provides a single value that represents the average error rate of the system, combining both APCER and NPCER. It offers a balanced view of the system's performance in distinguishing between genuine and fake samples.

$$\mathbf{ACER} = \frac{\text{APCER + NPCER (or BPCER)}}{2} \qquad (5.9)$$

5.2. Evaluation of the Models

The test results of the two different CNN models used in this study are given in the relevant figures. Each model was trained for 50 epochs, and the methods used in the study were integrated into the models in different combinations and compared. In order to understand the effect of different combinations on the models, ResNet18

architecture was used in the ResNet-based model. The outputs given in Figures 5.1 to 5.12 are the values obtained as a result of running the models once. These outputs are for reference to the metrics used in our study. The values given in Tables 5.1, 5.2, 5.3, 5.4 show the lowest average values obtained after 100 runs of the models.

MS and CI Integration

The two methods used in this study, MS and CI, are integrated into the ResNet18 model using weighted sampling and linear classifier and compared in Figures 5.1, 5.2, 5.3, and 5.4.

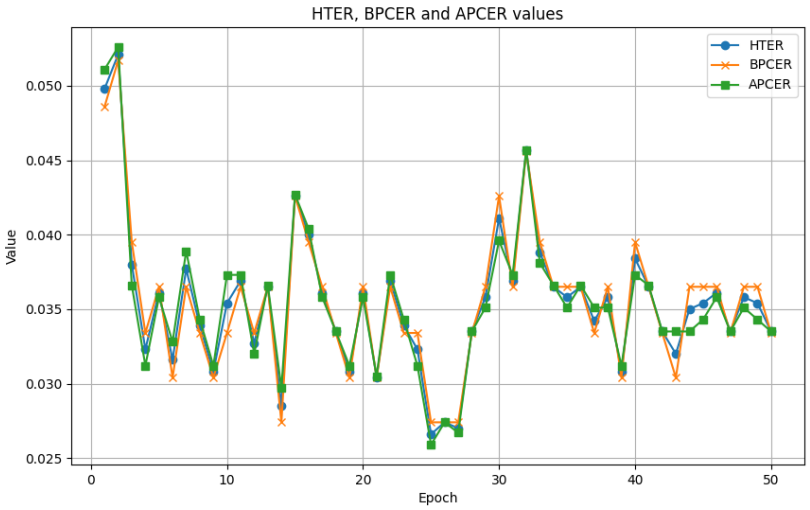


Figure 5.1 MS and CI methods are integrated into ResNet18 using weighted sampling and linear classifier is used.

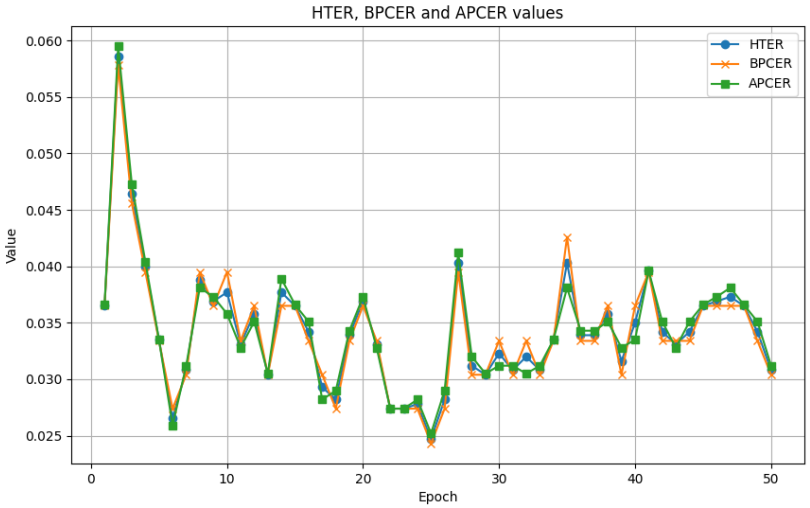


Figure 5.2 Only MS method is integrated into ResNet18 using weighted sampling and linear classifier is used.

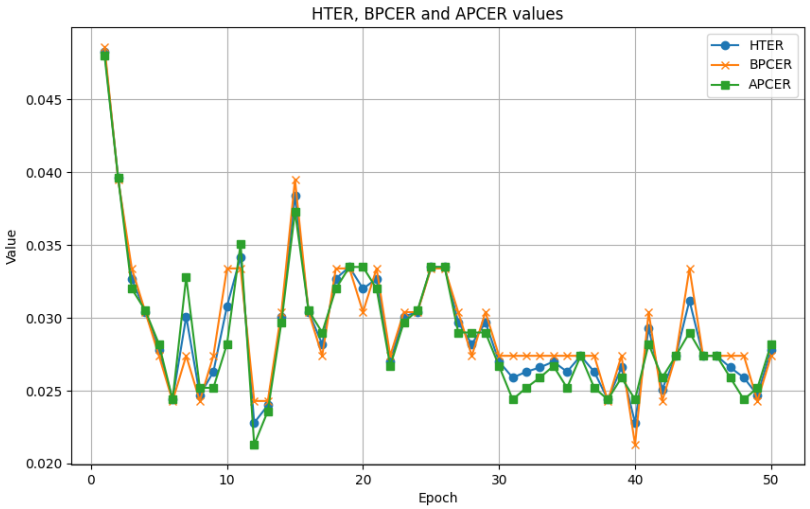


Figure 5.3 Only CI method is integrated into ResNet18 using weighted sampling and linear classifier is used.

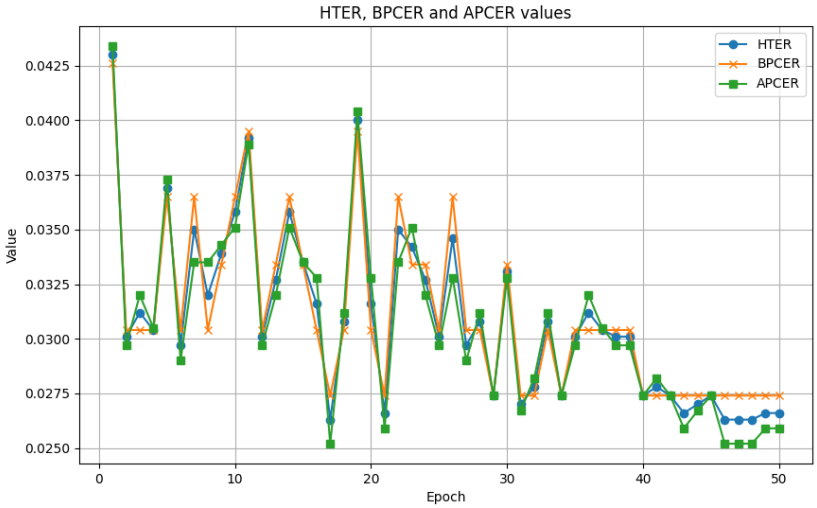


Figure 5.4 ResNet18 model with linear classifier is used without integration with MS or CI.

Three Sampling Evaluations: Over, Under, and Weighted

Three sampling methods Oversampling, Undersampling, and Weightedsampling(Figure 5.1) used in this study, integrated into the ResNet18 model using MS, CI, and Linear Classifier and compared in Figures 5.5, 5.6.

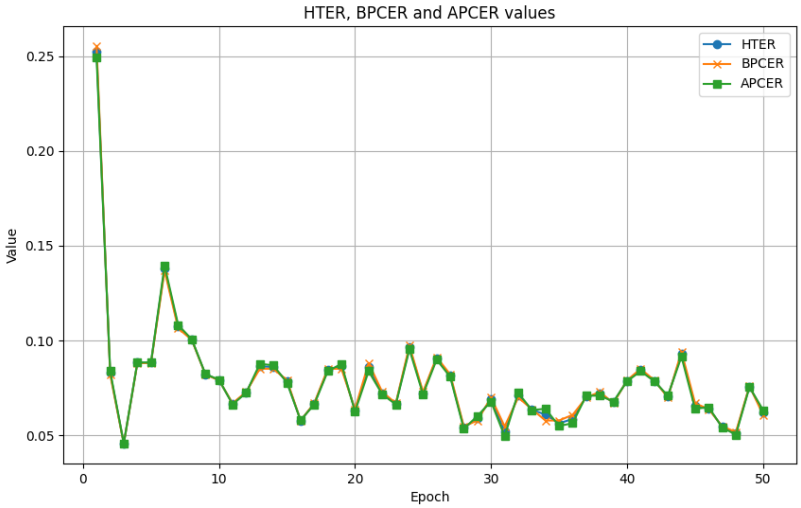


Figure 5.5 Oversampling used with ResNet18 model integrated with MS and CI with Linear Classifier.

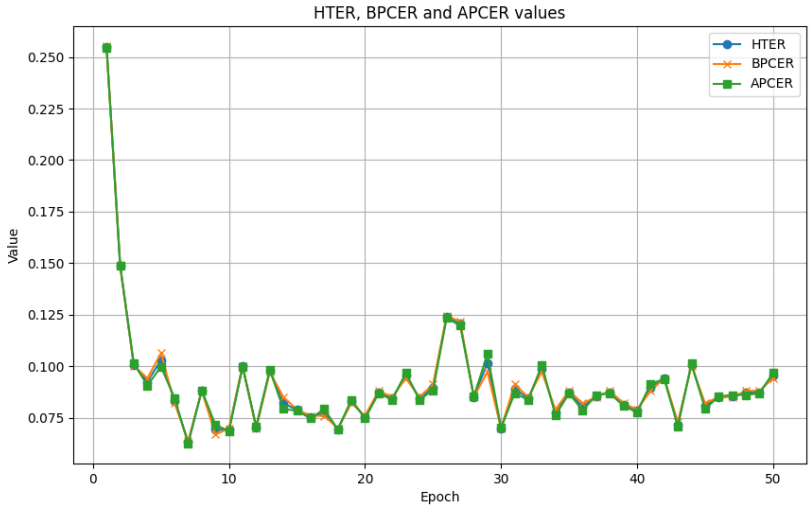


Figure 5.6 Undersampling used with ResNet18 model integrated with MS and CI with Linear Classifier.

Two different classifiers Linear and Convolutional used in this study, integrated into the ResNet18 model using MS and CI methods and compared in Figures 5.1 and 5.7.

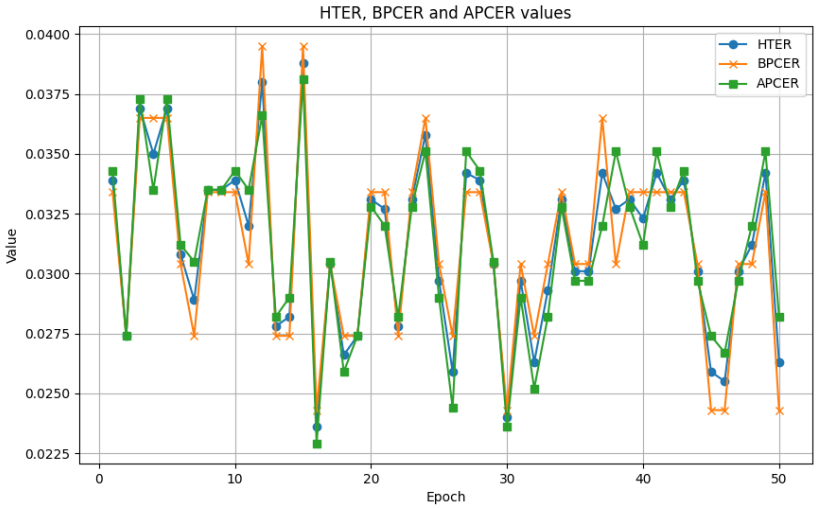


Figure 5.7 Convolution Classifier used with ResNet18 model integrated with MS and CI

Five different models ResNet18(Figure 5.1), 34, 50, 101, and 152 based and InceptionV3-based models using MS and CI methods and Linear Classification compared in Figures 5.1, 5.8, 5.9, 5.10, 5.11, 5.12.

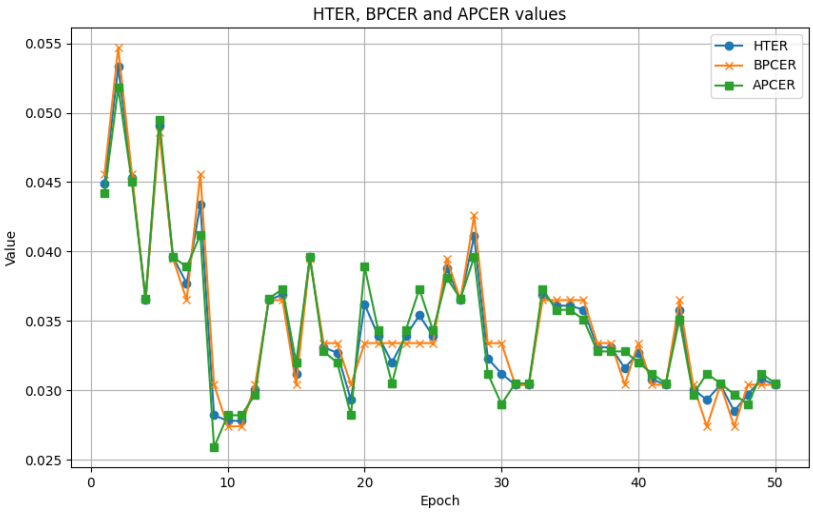


Figure 5.8 Our Mixed ResNet34 Results

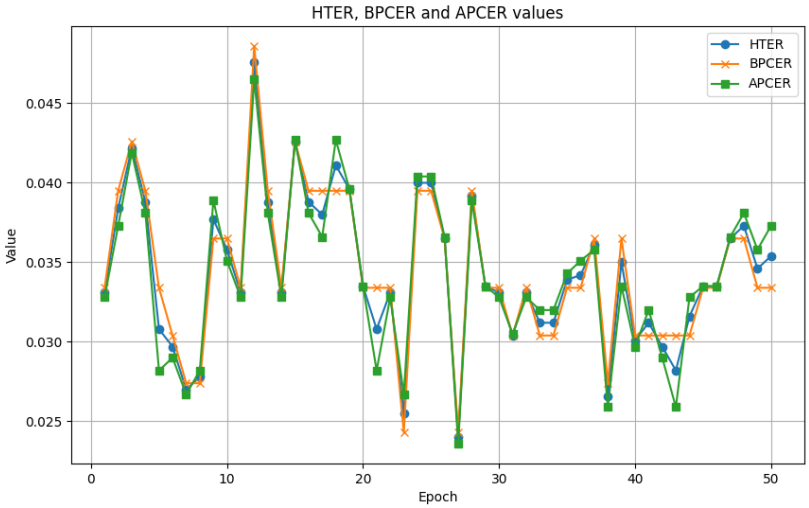


Figure 5.9 Our Mixed Resnet50 Results

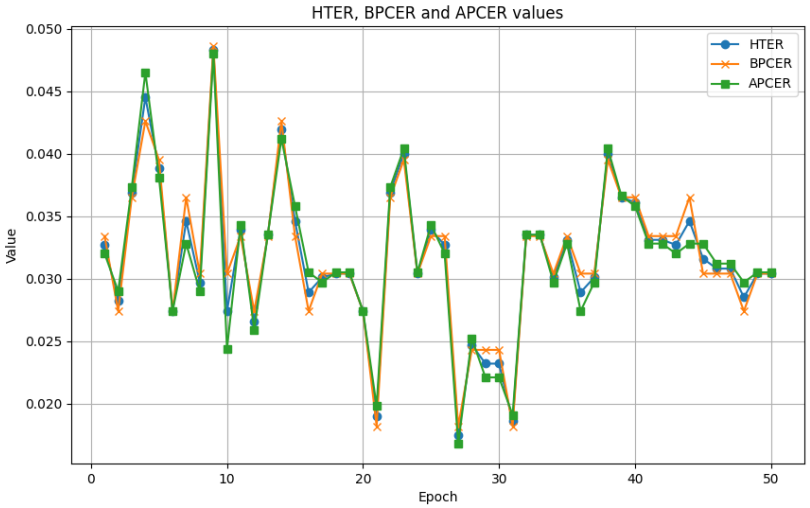


Figure 5.10 Our Mixed ResNet101 Results

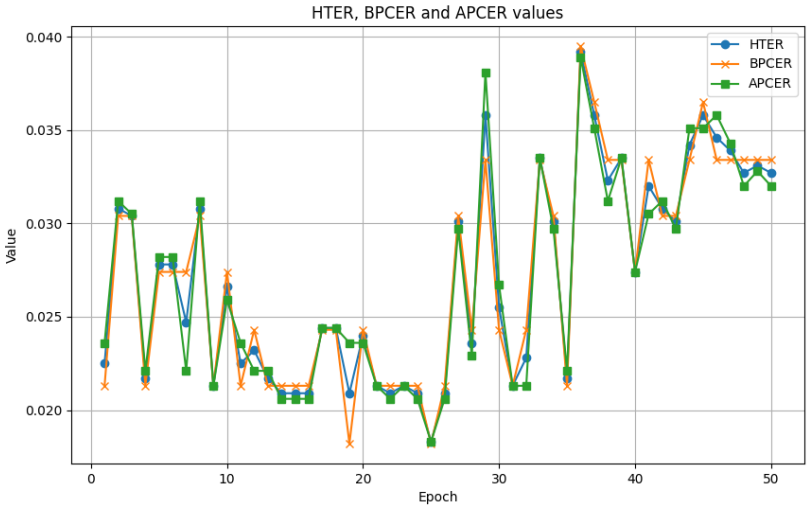


Figure 5.11 Our Mixed ResNet 152 Results

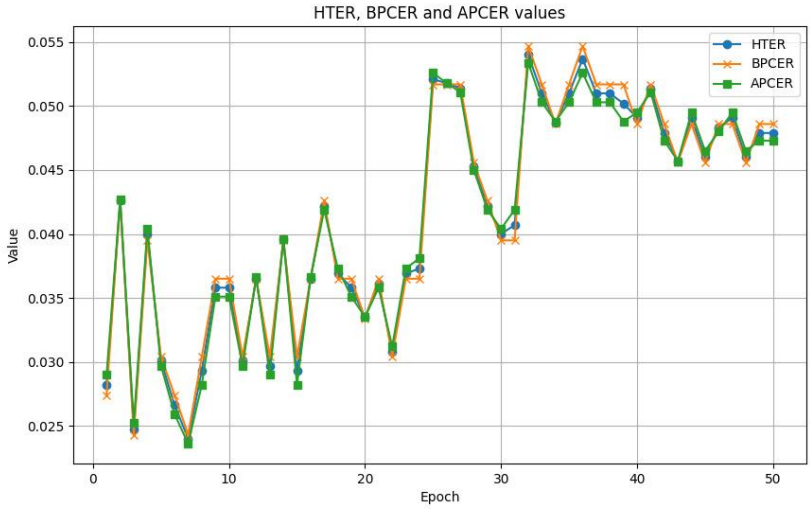


Figure 5.12 Our Mixed InceptionV3 Results

Table 5.1 Test Results with a Combination of MS and CI methods

ResNet18	**HTER**	**BPCER**	**APCER**
MixStyle, CI	0.0285	0.0274	0.0297
MixStyle	0.0278	0.0274	0.0282
Counterfactual Intervention	0.0293	0.0304	0.0282
None of them	0.0301	0.0304	0.0297

Table 5.2 Test Results with Two Different Classifiers

ResNet18	**HTER**	**BPCER**	**APCER**
Linear Classifier	0.0285	0.0274	0.0297
Convolutional Classifier	0.0240	0.0243	0.0236

Table 5.3 Test Results with Three Data Sampling Methods

ResNet18	**HTER**	**BPCER**	**APCER**
Weighted Sampling	0.0285	0.0274	0.0297
Oversampling	0.0521	0.0547	0.0495
Undersampling	0.0784	0.0790	0.0777

Table 5.4 Test Results of ResNet Models and InceptionV3 with MS and CI Methods

	HTER	**BPCER**	**APCER**
ResNet18	0.0285	0.0274	0.0297
ResNet34	0.0278	0.0274	0.0282
ResNet50	0.0266	0.0274	0.0259
ResNet101	0.0190	0.0182	0.0198
ResNet152	0.0183	0.0182	0.0183
InceptionV3	0.0240	0.0243	0.0236

When the models are trained and the average values are considered, the APCER, BPCER, and HTER results are generally very close to each other, except in exceptional cases. When considering these values, only the APCER value can be taken as a reference to understand the performance of the model. However, the average APCER, BPCER, and HTER results are shown in the tables.

In this study, as a contribution to the literature, three different data sampling methods were applied to the dataset and the results were compared. In order to better compare the results, the data sampling methods were compared with the ResNet18 architecture and the MS and CI methods integrated into the model. Considering the average results in Table 5.3, the Weighted Sampling method achieved much better results compared to the other two methods. In this direction, the Weighted sampling method was applied to the dataset for comparison with other models and the results were compared.

As another conclusion, two different Classifier layers were applied on the models and the results were compared. In order to better compare the results, two different Classifiers were compared with ResNet18 architecture and MS and CI methods integrated into the model. Table 5.2 shows that the Convolutional Classifier is more successful than the Linear Classifier when the average results are considered. Applying the convolutional classifier on the models can help to improve the accuracy of the model.

In this study, in order to understand the effect of MS and CI methods on the model, the methods were applied in different combinations on the ResNet18 architecture. Considering the average results in Table 5.1, it is observed that when MS and CI methods are applied to the model individually or together, the model obtains better results compared to the method without any method applied. According to these results, MS and CI methods were integrated into the models at the same time and the results were evaluated while comparing other models.

Finally, in order to understand the effect of MS and CI methods on different models, the methods were applied to different ResNet architectures and the InceptionV3 model. Considering the average values in Table 5.4, it is observed that ResNet architectures give better results as the depth of the model increases. Although the InceptionV3 model shows lower performance than ResNet 101 and 152, it is observed that it gives better results than ResNet 18, 34, and 50 architectures.

6. CONCLUSION

In this study, we have tried to find a solution to the problem of face presentation attack detection to make biometric face recognition systems, which are frequently used in various fields today, more secure. Various methods and algorithms have been developed in the literature to solve the problem. In this study, considering the areas and devices where biometric face recognition systems are used, software-based DL algorithms, which are thought to be more user and hardware-friendly, are used instead of expensive and laborious hardware solutions. Considering the proven success of CNN in image classification and the continuous improvement of algorithms in this field, two different CNN models were used in the study and the comparison of these models is shown in Table 5.4.

The first of the models used in the study is built on the ResNet architecture. Thanks to the success of ResNet in image classification problems and its architecture that prevents the memorization problem, it is predicted to be successful in solving the PAD problem. The ResNet architecture was created using Pytorch libraries and dynamically designed to be able to create all ResNet models. In this way, by changing only the model's name, the code can be written in such a way that it can be trained on all ResNet models. Another CNN model used in the study is InceptionV3. Again, the architecture of the model was created using Pytorch libraries, and the MS and CI methods used in the study were integrated into the relevant layers and the success of the model in face presentation attack detection was compared in table 5.4.

The OULU-NPU dataset was used in the training and testing phase of the models used in our study. Although this dataset is a large dataset created using different places, hardware, and people, different techniques were used to make the dataset suitable for the real-world problem of PAD. First of all, due to the unbalanced attack and bonafide images in the dataset, 3 different data sampling methods were used in our study and their comparisons are shown in Table 5.3.

In addition, in this study, two different methods were applied to increase the diversity of the dataset used and to provide better results for the model. The first of these methods, MS, was applied to different layers of both base models. The integration of the method into the model in a plug-and-play manner and running it only in the training phase without requiring any data duplication and registration process

like other dataset augmentation methods brings great convenience. The results of the MS integration are compared in Table 5.1. Another method, CI, was applied to both base models in the same way and the comparisons of the models are shown in Table 5.1. The outputs of the model as a result of integrating MS and CI methods in different combinations into the ResNet18-based model are also shown in Table 5.1.

The ResNet-based ResNet 152 architecture used in our study achieved remarkable success with 0.0183 HTER, 0.0182 BPCER, and 0.0183 APCER values. Similarly, the InceptionV3-based architecture was successful with 0.0240 HTER, 0.0243 BPCER and 0.0236 APCER, although it performed slightly below the first model.

The models were trained on a single dataset and this dataset was tried to be diversified with different methods. Although different methods are used, the success of the model can be increased by including a mask attack, which is one of the attack types that can increase the success of the model, and synthetic face data created using DL algorithms. Likewise, the models can be trained with other publicly available face datasets and the success of the models on different datasets can be compared. In the current study, parameters such as learning rate, optimizer, and epoch are used as fixed and the effect of these parameters on the model is not evaluated. In future studies, these parameters can be changed to find the most suitable parameters for the models.

REFERENCES

[1] McCulloch, W. S., & Pitts, W. (1943c). A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 5(4), 115–133. https://doi.org/10.1007/bf02478259

[2] Hebb, D. O. (1949). The organization of behavior; a neuropsychological theory. Wiley.

[3] Turing, A. M. (1950). Computing machinery and intelligence. Mind, 59, 433–460. https://doi.org/10.1093/mind/LIX.236.433

[4] Rosenblatt, F. (1963). PRINCIPLES OF NEURODYNAMICS. PERCEPTRONS AND THE THEORY OF BRAIN MECHANISMS. American Journal of Psychology, 76, 705.

[5] LeCun, Y., Boser, B.E., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.E., & Jackel, L.D. (1989). Handwritten Digit Recognition with a Back-Propagation Network. Neural Information Processing Systems.

[6] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[7] Aizenberg, I. N., Aizenberg, N. N., & Vandewalle, J. (2000). Multiple-Valued threshold logic and Multi-Valued neurons. In Springer eBooks (pp. 25–80). https://doi.org/10.1007/978-1-4757-3115-6_2

[8] Hinton G. E. (2007). Learning multiple layers of representation. Trends in cognitive sciences, 11(10), 428–434. https://doi.org/10.1016/j.tics.2007.09.004

[9] Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60, 84 - 90.

[10] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556.

[11] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.

[12] What is a neural network?. IBM. (2021, October 6). https://www.ibm.com/topics/neural-networks

[13] Lopez-Bernal, D., Silva, D.B., Ponce, P., & Molina, A. (2021). Education 4.0: Teaching the Basics of KNN, LDA and Simple Perceptron Algorithms for Binary Classification Problems. Future Internet, 13, 193.

[14] Werbos, P.J. (1990). Backpropagation Through Time: What It Does and How to Do It. Proc. IEEE, 78, 1550-1560.

[15] Bento, C. (2021, September 30). Multilayer Perceptron explained with a real-life example and python code: Sentiment Analysis. Medium. https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141

[16] Ian J. Goodfellow, Yoshua Bengio and Aaron Courville(2016) Deep Learning MIT Press.

[17] Washington University (2024). Chapter 18 Convolutional Neural Networks https://courses.cs.washington.edu/courses/cse416/22su/lectures/10/lecture_10.pdf

[18] Bouguezzi, S., Fredj, H. B., Belabed, T., Valderrama, C., Faiedh, H., & Souani, C. (2021). An efficient FPGA-Based convolutional neural network for classification: AD-MobileNet. Electronics, 10(18), 2272. https://doi.org/10.3390/electronics10182272

[19] Baheti, P. (2024, April 10). Activation Functions in Neural Networks [12 Types & Use Cases]. V7. https://www.v7labs.com/blog/neural-networks-activation-functions

[20] Saha, S. (2023, April 8). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[21] Boulkenafet, Z., Komulainen, J., Li, L., Feng, X., & Hadid, A. (2017). OULU-NPU: A Mobile Face Presentation Attack Database with Real-World Variations. 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), 612-618.

[22] Google. (n.d.). Google/mediapipe: Cross-platform, customizable ML solutions for live and streaming media. GitHub. https://github.com/google/mediapipe

[23] Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K., & Grundmann, M. (2019). Blazeface: Sub-millisecond neural face detection on mobile gpus. arXiv preprint arXiv:1907.05047.

[24] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of big data, 6(1), 1-48.

[25] Anand, R., Mehrotra, K.G., Mohan, C.K., & Ranka, S. (1993). An improved algorithm for neural network classification of imbalanced training sets. IEEE transactions on neural networks, 4 6, 962-9 .

[26] Ruiz, P. (2024, April 30). Understanding and visualizing ResNets - Towards Data Science. Medium. https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

[27] Zhou, K., Yang, Y., Qiao, Y., & Xiang, T. (2021). MixStyle Neural Networks for Domain Generalization and Adaptation. ArXiv, abs/2107.02053.

[28] Gaikwad, M. (2023, April 9). Review: Mix-style neural networks for domain generalization and adaptation. Medium. https://medium.com/@mohit_gaikwad/review-mix-style-neural-networks-for-domain-generalization-and-adaptation-2207ece76707

[29] Yu, T., Lu, J., Li, X., & Zhou, J. (2022). Salience-Aware Face Presentation Attack Detection via Deep Reinforcement Learning. IEEE Transactions on Information Forensics and Security, 17, 413-427. DOI: 10.1109/TIFS.2021.3135748

[30] Li, L., Feng, X., Boulkenafet, Z., Xia, Z., Li, M., & Hadid, A. (2016). An original face anti-spoofing approach using partial convolutional neural network. 2016 Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA), 1-6. DOI: 10.1109/IPTA.2016.7821013

[31] George, A., Mostaani, Z., Geissenbuhler, D., Nikisins, O., Anjos, A., & Marcel, S. (2019). Biometric Face Presentation Attack Detection With Multi-Channel Convolutional Neural Network. IEEE Transactions on Information Forensics and Security, 15, 42-55. DOI: 10.1109/TIFS.2019.2916652

[32] George, A., & Marcel, S. (2019). Deep Pixel-wise Binary Supervision for Face Presentation Attack Detection. 2019 International Conference on Biometrics (ICB), 1-8. arXiv: 1907.04047

[33] Raghavendra, R., Raja, K.B., & Busch, C. (2015). Presentation Attack Detection for Face Recognition Using Light Field Camera. IEEE Transactions on Image Processing, 24, 1060-1075. DOI: 10.1109/TIP.2015.2395951

[34] Liu, Y., Jourabloo, A., & Liu, X. (2018). Learning Deep Models for Face Anti-Spoofing: Binary or Auxiliary Supervision. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 389-398. arXiv: 1803.11097

[35] Shu, X., Tang, H., & Huang, S. (2020). Face spoofing detection based on chromatic ED-LBP texture feature. Multimedia Systems, 27, 161 - 176. DOI: 10.1007/s00530-020-00719-9

[36] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2818-2826.