
# DEEPFAKE IMAGE/VIDEO DETECTION AND CLASSIFICATIONS USING DEEP LEARNING TECHNIQUES.

A THESIS SUBMITTED TO
THE SCHOOL OF GRADUATE STUDIES
OF
UNIVERSITY OF TURKISH AERONAUTICAL ASSOCIATION

BY

Deo RUTIKANGA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND COMPUTER ENGINEERING

AUGUST 2024


# DEEPFAKE IMAGE/VIDEO DETECTION AND CLASSIFICATIONS USING DEEP LEARNING TECHNIQUES.

A THESIS SUBMITTED TO
THE SCHOOL OF GRADUATE STUDIES
OF
UNIVERSITY OF TURKISH AERONAUTICAL ASSOCIATION

BY

Deo RUTIKANGA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND COMPUTER ENGINEERING

Supervisor: Asst. Prof. Dr. Zeynel DEPREM

AUGUST 2024

**Approval of the thesis:**

**DEEPFAKE IMAGE/VIDEO DETECTION AND CLASSIFICATIONS USING DEEP LEARNING TECHNIQUES.**

submitted by **Deo RUTIKANGA** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Computer Engineering**, **University of Turkish Aeronautical Association** by,

**Assoc. Prof. Dr. Adnan GÜZEL**
**Dean, The School of Graduate Studies, UTAA** _____

**Asst. Prof. Dr. Aytun ONAY**
Head of The Department, Electrical and Computer Engineering, UTAA _____

**Asst. Prof. Dr. Zeynel Deprem**
Supervisor: Electrical and Electronic Engineering, UTAA _____

**Examining Committee Members:**

**Asst. Prof. Dr. Zeynel DEPREM**
Electrical and Electronic Engineering,
University of Turkish Aeronautical Association _____

**Asst. Prof. Dr. Hasan AKSOY**
Electrical and Electronic Engineering,
University of Turkish Aeronautical Association _____

**Prof. Dr. Saffet AYASUN**
Electrical and Electronic Engineering,
Gaz Üniversitesi _____

**Date:** 20 August 2024

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

**Deo RUTIKANGA**

## ABSTRACT

DEEPFAKE IMAGE/VIDEO DETECTION AND CLASSIFICATIONS USING DEEP LEARNING TECHNIQUES

Rutikanga, Deo
Master Thesis, Electrical and Computer Engineering
Supervisor: Asst. Prof. Dr. Zeynel DEPREM
August 2024, 96 pages

The development of deepfake technology, which employs the capability of deep learning to produce highly photorealistic fake media, poses serious challenges to privacy, security and information integrity. This thesis responds to this challenge by proposing a novel hybrid technique that merges MTCNN and MobileNetV2 with a feature differencing technique.

The literature methodologies will consider the development of deepfake detection methods, focusing on Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) as key discriminators of synthetic media. Conventional CNN-based approaches have limitations in terms of high-quality forgery identification, optimization for efficiency in computation and dealing with subtle modification. These indicate the importance of creative and original methods.

Our proposed methodology involves MTCNN as a first stage for accurate face detection and alignment followed by MobileNetV2 for efficient image classification. The detection accuracy is improved using the feature differencing technique within MobileNetV2 highlighting region-specific anomalies indicating deepfakes. The data was collected from Kaggle (Zenodo) having been balanced dataset with 40,000 images pre-processed into uniform size then later 42 other images were created from scratch to ensure diversity in facial images plus manipulations. Dataset variability was increased using data augmentation techniques.

The Experimental evaluations were carried out on the Google Colab platform and local Jupyter Notebook environments to ensure optimal performance and reproducibility. Differencing technique with MobileNetV2 at around 98% of accuracy makes considerable improvements based on the results of these experiments, which encompassed several datasets.

This thesis improves detection efficiency by making a strong model that raises its accuracy. This provides an approach where integration between MTCNN and MobileNetV2 combined with feature differencing technique can be employed at real-time applications while assuring visual content authenticity across multiple domains. As deepfake technology advances, future studies should investigate these methodologies further, while expanding datasets beyond generic and easy examples for ensuring trustworthiness of digital media.

**Keywords**: Deeplearning, deepfake Detection, Multi-task Cascaded Convolutional Networks (MTCNN), MobileNetV2, Artificial Neural Networks, MobileNetV2 Intermediate Feature Map Differencing Technique, Image Classification.

# ÖZ


DERİN ÖĞRENME TEKNİKLERİ KULLANARAK DEEPFAKE GÖRÜNTÜ/VİDEO TESPİTİ VE SINIFLANDIRMASI.

Rutikanga, Deo
Yüksek Lisans, Elektrik ve Bilgisayar Mühendisliği
Tez Yöneticisi: Dr. Öğr. Üyesi Zeynel DEPREM
August 2024, 96 pages

Deepfake teknolojisi, güçlü derin öğrenme tekniklerinden yararlanarak son derece gerçekçi sentetik medya oluşturur ve bu da gizlilik, güvenlik ve bilgi bütünlüğü açısından önemli tehditler oluşturur. Bu tez, Çok Görevli Kademeli Konvolüsyonel Ağlar (MTCNN) ve MobileNetV2'yi benzersiz bir özellik çıkarma tekniğiyle birleştiren yeni bir hibrit yaklaşım önererek deepfake'lerin etkili bir şekilde tespit edilmesi zorluğunu ele almaktadır.

Literatür incelemesi, derin sahte tespit yöntemlerinin evrimini incelemekte ve Yapay Sinir Ağları (ANN'ler), Konvolüsyonel Sinir Ağları (CNN'ler) ve Üretici Çekişmeli Ağların (GAN'lar) sentetik medyayı tanımlamadaki kritik rolünü vurgulamaktadır. Geleneksel CNN tabanlı yöntemler etkili olmakla birlikte, yüksek kaliteli sahteciliklerin tespitinde, hesaplama verimliliğinin yönetilmesinde ve ince manipülasyonların ele alınmasında sınırlamalarla karşı karşıyadır. Bu zorluklar, yenilikçi yaklaşımlara olan ihtiyacı vurgulamaktadır.

Önerilen yöntem, doğru yüz tespiti ve hizalaması için MTCNN'in sıralı entegrasyonunu ve ardından verimli görüntü sınıflandırması için MobileNetV2'yi içermektedir. MobileNetV2 içindeki özellik çıkarma tekniği, deepfake'leri gösteren bölgeye özgü anomalileri vurgulayarak tespit doğruluğunu artırır. Veri toplama süreci, Kaggle (Zenodo) üzerinden sağlanan 40.000 önceden işlenmiş görüntüden oluşan dengeli bir veri seti ve 42 görüntüden oluşan özel oluşturulmuş bir veri seti içerir, bu da çeşitli yüz görüntülerini ve manipülasyonlarını sağlamaktadır. Veri çeşitliliğini artırmak için veri artırma teknikleri uygulanmıştır.

Deneysel değerlendirmeler, optimal performans ve tekrarlanabilirlik sağlamak için Google Colab ve yerel Jupyter Notebook ortamları kullanılarak gerçekleştirilmiştir.

MobileNetV2 çıkarma tekniği, doğrulukta önemli iyileşmeler göstererek %98.71'e kadar ulaşmıştır. Hibrit yaklaşım, tespit doğruluğu, hesaplama verimliliği ve ölçeklenebilirlik zorluklarını etkili bir şekilde ele almaktadır.

Bu tez, tespit doğruluğunu ve verimliliğini artıran sağlam bir çerçeve sağlayarak deepfake tespiti alanına önemli katkılarda bulunmaktadır. MTCNN ve MobileNetV2'nin entegrasyonu, özellik çıkarma tekniği ile desteklenerek, gerçek zamanlı uygulamalar için ölçeklenebilir ve pratik bir çözüm sunar ve çeşitli alanlarda görsel içeriğin bütünlüğünü ve özgünlüğünü sağlar. Gelecekteki araştırmalar, bu metodolojilerin daha da rafine edilmesine ve veri kümelerinin daha çeşitli ve zorlayıcı örnekleri içerecek şekilde genişletilmesine odaklanmalı, böylece deepfake teknolojisinin ilerleyen yeteneklerine karşı dijital medyanın güvenilirliğini korumalıdır.

**Anahtar Kelimeler:** Deepfake Tespiti, Çok Görevli Kademeli Konvolüsyonel Ağlar (MTCNN), MobileNetV2, Özellik Çıkarma Tekniği, Görüntü Sınıflandırma

## DEDICATION

Firstly, to God whose leading and kindness have constantly sustained my strength and inspiration through my academic journey.

To my dear wife and children for their unwavering encouragement, motivation and infinite patience. Their love has been the foundation upon which this work has been built.

I would want to extend my appreciation to my parents for their steadfast and selfless acts of sacrifice, as well as to my siblings for being patient with me in all times.

Lastly, I can't forget to thank friends and colleagues who were there when I needed them most. Your thoughts and fellowship made me gain much insight along this way that wouldn't be forgotten.

## ACKNOWLEDGEMENTS

It is important at this point to acknowledge the guidance, advice, constructive criticism, encouragement as well as insights given by Assist. Prof. Dr. Zeynel Deprem, who supervised me throughout the entire process of conducting this research. Without his support, I would not have achieved what I have done in life.

The Government of Rwanda is highly appreciated for its funding they gave us generously supporting our research which enhanced its feasibility; it is very commendable on their part since they are determined in promoting education as well as research work.

## TABLE OF CONTENTS

## LIST OF TABLES

TABLES

## LIST OF FIGURES

FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| CNN | Convulutional Neural Network |
| MLP | Multi-Layer Perceptron |
| MTCNN | Multi-Task Cascade Multi-task Cascade Convulutional Neural Network |
| SR | Seperable Residual |
| PReLU | Parametric Rectified Linear Unit |
| ReLU | Rectified Linear Unit |
| DNN | Deep Neural Network |
| ROI | Region of Interest |
| GAN | Generative Adversarial Networks |
| NMS | Non-Maximum Suppression (NMS) |
| IFMD | Intermidiate feature map differencing |
| MNV2 | MobileNetV2 |

# CHAPTER 1

# INTRODUCTION, MOTIVATION, THESIS GOALS, AND THESIS OUTLINE

## 1.1 Introductıon

The phrase "deepfake" is a combination of "deep learning" and "fake," and it refers to the use of sophisticated neural networks to create or modify visual and auditory content with remarkable realism. Initially a specialized technique, deepfakes have quickly evolved, leading to a surge in convincingly modified media. The emergence of this trend has profound implications for privacy, security, integrity and information, making deepfake detection very essential.

Deepfakes which are created using sophisticated deep learning techniques—pose significant security vulnerabilities despite their potential for creating applications. They offer new opportunities in entertainment and virtual reality but also pose threats such as misinformation and the spread of non-consensual content (Carlyon, 2023). Verifying the authenticity digital documents and media has always been challenging, but the explosion of digital data across fields like marketing, legal forensics, and medical imaging has introduced unprecedented complexity.

The progress in deep learning, computer graphics, and machine vision has enabled the creation of highly realistic synthetic content possible on consumer-grade devices ( Keith & Javaan Chah, 2022). While deepfake technology offers new possibilities in various industries, it also presents considerable security risks. These include potential misuse in election tampering, inciting conflict, defamation, and non-consensual explicit content. Social media platforms have the ability to rapidly disseminate deepfakes worldwide, greatly magnifying their influence.

Given these implications, developing methods to distinguish real from synthetic media and identify tampering is imperative. This challenge extends beyond computer-generated faces to the full synthesis of human subjects, an area that has recently advanced significantly.

The central theme of this research is detection of deepfake by taking advantage of advanced deep learning techniques. In this research, we suggest a hybrid model combining MTCNN, MobileNetV2 and MobileNetV2 intermediate feature map differencing technique. Therefore, our model targets at increasing accuracy while being computationally efficient and scalable for real time and mobile applications. Our research aim is to enhance the technology related to identifying deepfakes, thereby contributing to defending the integrity of visual content and preventing the malicious use of synthetic media. For example, most prior work relied on Artificial (NNs), (CNNs) as well as (GANs). On the other hand, our suggested approaches are based on MTCNN, MobileNetV2 and MobileNetV2 feature differencing technique as more developed methodologies.

The results obtained from our models were highly satisfactory. The accuracy for individual MTCNN reached 94.85%; MobileNetV2-based model improved slightly to 95.57% while MobileNetV2 Intermediate Feature Map Differencing Technique conferred a significant increase in the accuracy of up to 98%, which demonstrated its ability to differentiate between genuine and fake images.

For training and validation, proper selection of loss functions and optimization algorithms was done in order to maximize the efficiency of learning. This helped us in avoiding overfitting through techniques such as data splitting, early stopping, data augmentation, and regularization that ensures the models generalize well. In comparison with state-of-the-art methods for deepfake detection, our models, specifically MobileNetV2 Intermediate Feature Map Differencing Technique technique had high accuracy and were efficient.

These results highlight how important it is for digital forensics to consider our suggested approaches towards improving deepfake detection significantly. The methodical manner in which we trained, validated and tested these models lays

down a strong basis that will be useful in other similar researches or even practical applications on this sensitive field moving forward.

## 1.2 Motivation of the Thesis

There is no doubt that the advent of deepfake technology has brought about serious crises for the authenticity of digital media; it has also jeopardized privacy, security and trustworthiness of the public. In a way that is almost invisible to the naked eye, these exceptional deep learning techniques such as (GANs) can create convincing synthetic media. These technological strides have broad implications, including misuse for political propaganda, forgery of identity and disinformation.

This research is justified by the current need for effective ways of detecting deepfakes. However, even though ANNs, CNNs and GAN-based methods are commonly used as traditional detection methods in deepfake detection systems they all do not entirely address some issues concerning processing time requirements (time-consuming), scalability challenges and inability to effectively identify minute alterations made on images. Consequently, it is important to develop better ways detecting GAN-derived fakes so that we can avoid lagging behind in this area.

## 1.3 Thesis Goals

The aim of this study is to devise and test a powerful deepfake detector model that surpasses the limitations of what exists now.

These include:

1. **Improving Detection Accuracy**: Creating a combination hybrid model that uses MTCNN with MobileNetV2 as well as MobileNetV2 feature differencing for increased accuracy on recognizing deepfakes. This model must be able to capture small changes that may not be visible using conventional CNN-based methods.

2. **Enhancing Computational Efficiency**: Developing low-cost solutions for real-time systems & resource-constrained environments like mobile devices and embedded systems. As such, the network architectures shall be optimized for performance/efficiency trade-off.
3. **Ensuring Scalability**: Building scalable detection models capable of handling large complex datasets while still delivering their objectives. This would involve testing scalability and robustness of the models against different datasets sizes/types.
4. **Developing Binary Classification Functionality**: Constructing an f binary classifying function which can provide accurate labels distinguishing manipulated (fake) from genuine (real) pictures. By systematically analyzing and optimizing them, we hope to enhance recall rate and precision during detection tasks.
5. **Contribution towards Body of Knowledge**; To contribute comprehensively into knowledge on deepfake detection through intensive research work alongside experimentation. The findings herein and the methodologies devised are aimed at supplementing current initiatives to contain deepfake technology threats.

With these objectives, the thesis intends to make stronger deepfake detectors that can be used in various incidents thereby securing the safety and trustworthiness of digital media content.

### 1.4 Thesis Outline

This thesis is comprised of:

**Chapter 1:** Introduction, Motivation, Objectives, and Thesis OutlineOutlines.

**Chapter 2**: Literature Review

- Discusses related literature on generation & detection techniques for fake videos opening avenues for new research approaches. It provides an in-

depth survey of the current state-of-the-art as well as some gaps that can be built upon further.

**Chapter 3**: Proposed Methodology and Models

- It expounds on advanced deep learning techniques adopted in this study towards detecting deepfakes. The content of this chapter includes the theoretical framework, data collection, processing, model architectures, and the rationale behind methodological choices.

**Chapter 4**: Contributions and Novelties

- Concerning such things as implementation, experiments conduction, assessment process together with analysis to demonstrate how they reveal the originality and relevance of the work.
- This section describes the experimental setup including hardware/software components used, parameter tuning adjustment and data collection procedures. Additionally this chapter presents results obtained from executing recommended methodologies for identifying deepfakes. characteristics that allow one to draw conclusions about differences between real images and manipulated ones when discussing them within the field of digital forensics. Contains a thorough review of experimental findings including quantitative/qualitative analyses made thereof. A comparison is presented for performance evaluation against existing deepfake detection methods by explaining their implications in digital forensics field.

**Chapter 5**: Conclusion and Future Work

- The implications of the research findings are brought together, while their significance is evaluated against a much broader background of digital media authenticity. The future direction of deepfake detection is also discussed in this context.

- In order to have a better understanding of the challenges posed by deepfakes and innovative approaches to deal with them and how they impact on digital forensics, this structure provides an orderly and logical progression throughout the thesis.

# CHAPTER 2

## REVIEW OF EXISTNG LITERATURE

This chapter examines the critical literature regarding deepfake technology, its detection techniques and deep learning development towards identification and mitigation of such manipulations. The thesis pays attention to the most important creation and recognition approaches within deepfake with focus on some famous methodologies like the (NNs), (CNNs) and (GANs).

Deepfake is a rapidly evolving technology that relies heavily on advanced deep learning techniques to create incredibly realistic synthetic media. What makes deepfakes difficult to detect is their potential for abuse in propagating disinformation and generating deceptive content.

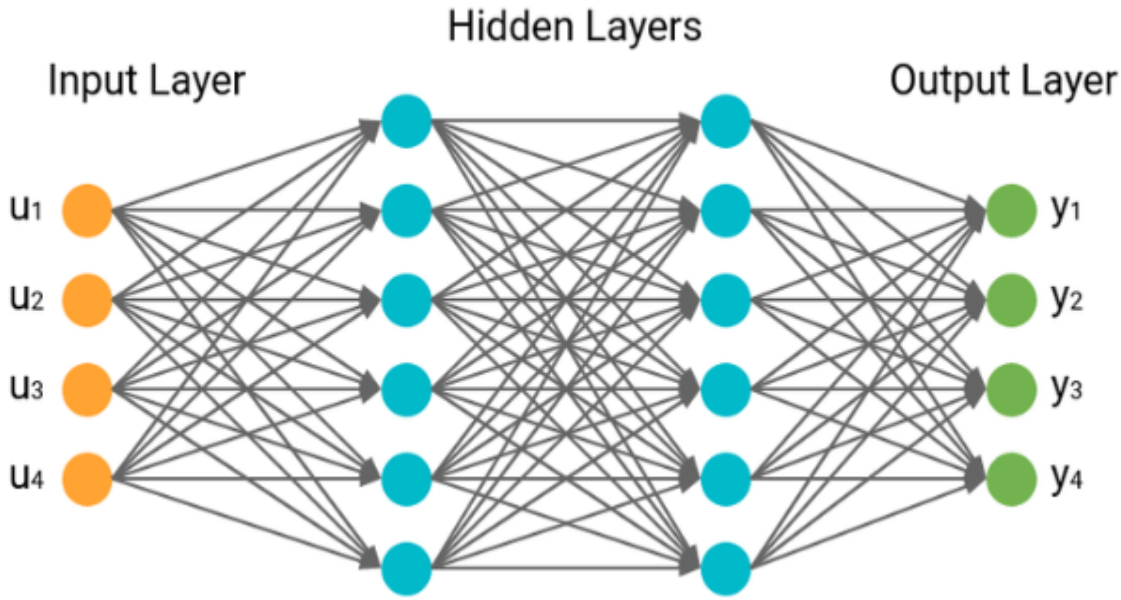



Figure 2.1: Architectural Features of NN

## 2.1 Neural Networks

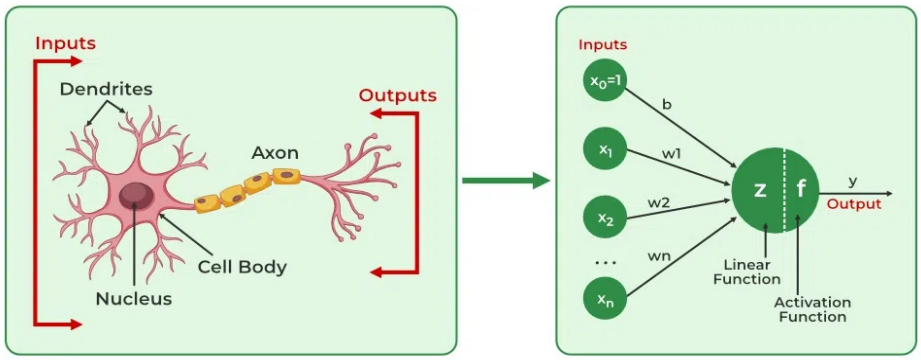



Figure 2.2: Biological Neuron Vs ANN (GeeksforGeeks, 2024)

Artificial NNs are designed to imitate the structure and functioning of neurons found in brains of animals providing inspiration for their development (contributors, 2024). An ANN composed of layers of artificial neurons, also known as units, which include an input layer, hidden layers, and an output layer. The input layer acquires data from the external environment, the hidden layers manipulate this data, and the output layer generates the network's response. The units in these layers are interconnected, with each connection assigned a weight that influences the transfer of data, allowing the network to learn and improve its accuracy over time.

Comprising dendrites, a cell body (soma), and an axon, biological neurons are constituted dendrites get electrical signals. The soma integrates and interprets these signals, and the axon conveys the processed information to other neurons. Similarly, in ANNs, input nodes receive signals, hidden nodes process these signals, while activation functions in the output nodes generate final results. Weighted connections among synapses in biological neurons resemble synapses between ANNs neurons which allow impulse transmission respectively. Synaptic plasticity happens during learning in living things where synaptic strength changes depending on activity level whereas backpropagation mechanism effectuates error

minimization by adjusting connection weights according to predicted versus actual results within ANNs.

This can be done by exposing the ANN to different examples such as images of cats for it to learn how to recognize certain patterns. Following that, a performance is then assessed and modified through backpropagation. The network is then repeatedly trained on this task until it can accurately identify those patterns with minimal error rates (Harkiran78, 2023).

Essentially, ANNs are an artificial representation of how biological neurons process data and learn from them which makes them suitable for tasks like image classification and recognition.

The researchers who were working on deepfake detection have employed various methodologies including using Artificial Neural Networks (NNs). NNs are computational models like human brain neurons in that they take inputs and produce specified outputs (Matthew Stewart, 2019). NNs play a significant role in machine learning techniques because deep learning innovations are built upon them. They have their origin in human biological structures which replicate the functions performed by brains neurons when processing sensory information.

In the domain of machine and deep learning algorithms, NN algorithms often don't need pre-established rules for interpreting inputs. While being trained, this network learns from many examples rather than a rulebook. Given enough samples to work with, neural net can deal with new cases it didn't see before providing reliable results. The accuracy of these outcomes usually improves with the variety and quantity of inputs received thereby enhancing its experiential learning curve.

Creating a neural network typically entails splitting a standard dataset into three sections:

1. **The training set**: It is the chief source of data for networks to learn from, thus creating a foundation about what is supposed to be done.
2. **Validation set**: A second dataset used to evaluate how good model performance is in terms of tuning up internal parameters.

3. **Testing set**: The last dataset to be employed for post initial training performance estimation of the network.

Some of the NN methodologies, including other machine learning paradigms, are generally in line with two main strategies:

1. **Supervised learning**: This is based on labeled data whose output is already known and it may be classified into classification and regression problems.
2. **Unsupervised Learning**: It attempts to discover patterns within unlabeled data and this has a significant role in tasks such as clustering, association, and dimensionality reduction.

This thesis mainly focuses on supervised learning framework. NNs improve their ability to handle novel, unseen inputs through exposure to many examples during training thereby making them ideal in deepfake detection where input nature can change a lot.

### 2.1.1 Training and Loss function

To impartially measure the efficiency of neural network (NN) model, declaring loss function is imperative. This is where the metric that checks how far off are predictions made by the model against actual values comes in place whereby lower loss means higher accuracy while higher loss represents greater divergence. A frequently used error function is Mean Square Error (MSE) which measures average squared difference between predicted and actual values for all samples.

$$MSE = \frac{1}{N} \Sigma_{i=1}^{N} (Y_i - \hat{y}_i)^2 \quad (3)$$

The total number of samples in the dataset is represented by $N$. The variable $Y_i$ represents the actual outcome for a given input data point $X_i$ t that the network seeks to predict while $\hat{y}_i$ indicates what we predict for that same $X_i$.

The major aim is to minimize the loss function so as to make accurate predictions for this model. The predictions made by the NN rely on its internal parameters like

weights and biases that are usually set either at random or according to some predetermined criterion thereby making them not optimal at start.

Training of an NN involves reducing its loss function through backpropagation (Hecht-Nielsen, 1992). This requires updating weights and biases such that changes in these parameters are inversely related to changes in the loss function where the learning rate determines step size.

$$W^{n+1} = W^n + \Delta W = W^{n+1} - \eta \left.\frac{\partial \mathcal{L}}{\partial W}\right|_{W=W^n} \tag{4}$$

where $\boldsymbol{n}$ denotes a specific time point, $\boldsymbol{w^n}$ is a concrete parameter at time $\boldsymbol{n}$, $\boldsymbol{w^{n+1}}$ represents the new parameter value after step $\boldsymbol{n+1}$, $\Delta w$ shows this learning step's size, $\boldsymbol{\eta}$ refers to learning rate and $\boldsymbol{L}$ stands for loss function.

The derivative of the loss function points out how it fast it is changing and thus guide updates opposite direction meant to decrease loss. The updates should be small enough so as not to overshoot minimum loss value which is regulated by learning rate.

Assessing an NN model's ability to learn properly and generalize well with respect to novel data involve issues like overfitting and underfitting. Overfitting occurs when a model fits too closely to the training data, dooming performance on unseen data while underfitting is due to lack of complexity in the model sufficient enough to capture the underlying patterns in the dataset. However, there are no one-size-fit-all solutions for these problems, although methods such as regularization and cross-validation may be used to tune the network specifically for a given task (Bishop, 2006).

### 2.1.2 NN with Multiple Layers of Interconnected Nodes, Also Known as Multi-Layer Perceptron

In this section, we concentrate on a highly effective type of neural network called the MLP. However, before delving into it, it's important to understand the concept

of a Perceptron. Introduced by Rosenblatt in 1958, the Perceptron is one of the earliest documented neural network models. This model is built around the basic computational unit, the neuron, referred to here as the j-node, and is defined by the following mathematical expression:

$$z_j = g_j\left(\Sigma_{i=1}^{n} w_{ij} S_j - b_j\right) \tag{1}$$

where $S_j$ represents the input data, $b_j$ the bias or activation threshold, $\omega_{ij}$ the connection weights, $g_j$ is an activation function, and $z_j$ represents outputs. Weights and biases are initially set randomly or according to some predetermined strategy. these weights determine how important each input variable is; higher weights imply a larger effect on the output variable. In particular fields, there are numerous activation functions which include step function giving distinct nonlinearity between input-output relationship;

$$g_j(x) = \begin{cases} 1 & for\ x > 0 \\ 0 & for\ x = 0 \\ -1 & for\ x < 0 \end{cases} \tag{2}$$

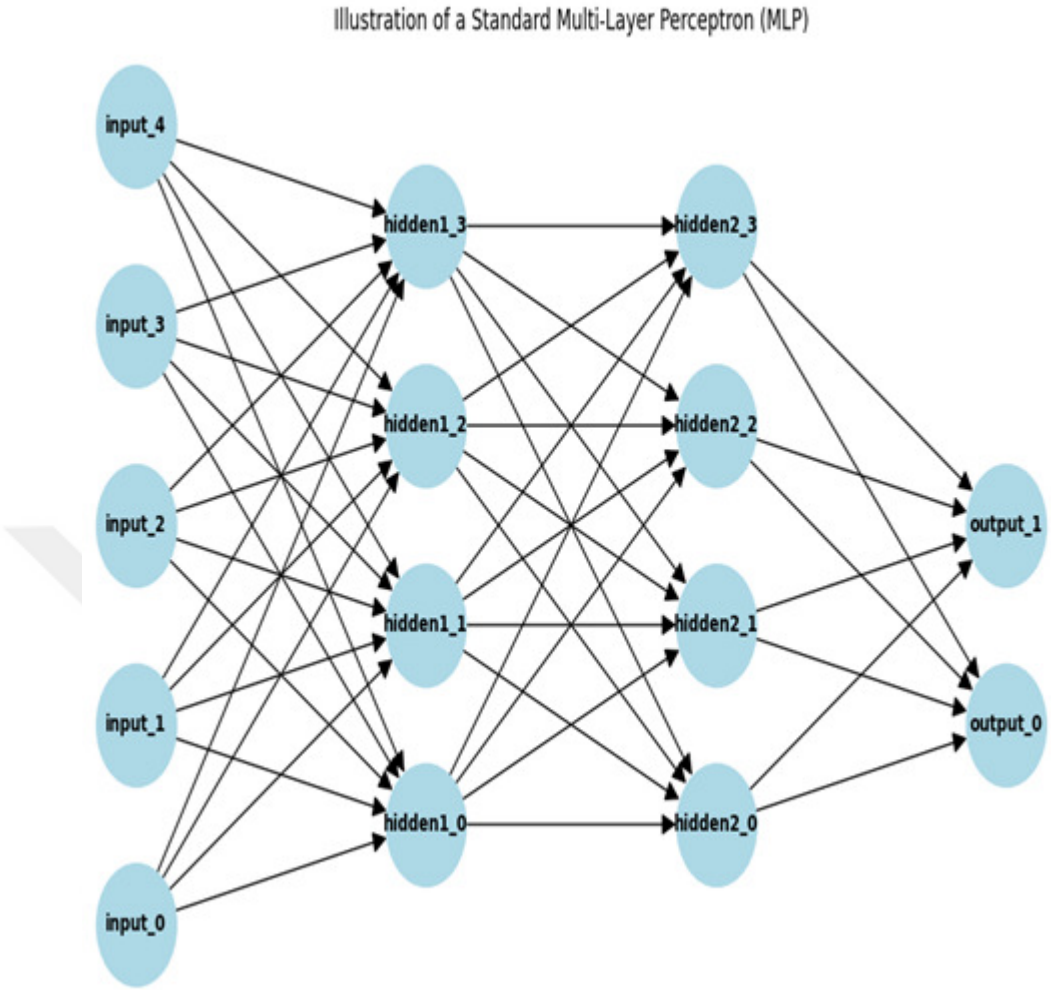



Figure 2.3: Illustration of a Standard Multi-Layer Perceptron (MLP) (Peng, Cao , Liu, Guo , & Tian, 2021)

The purpose of the activation function is to facilitate the transfer of information between neurons, a process that characterizes this NN as a feedforward network when data moves sequentially from one layer to the next.

By assembling multiple perceptron, we construct the MLP, a type of NN capable of approximating any continuous and definable function from an input to an output (Hornik, Stinchcombe, & White, 1989).

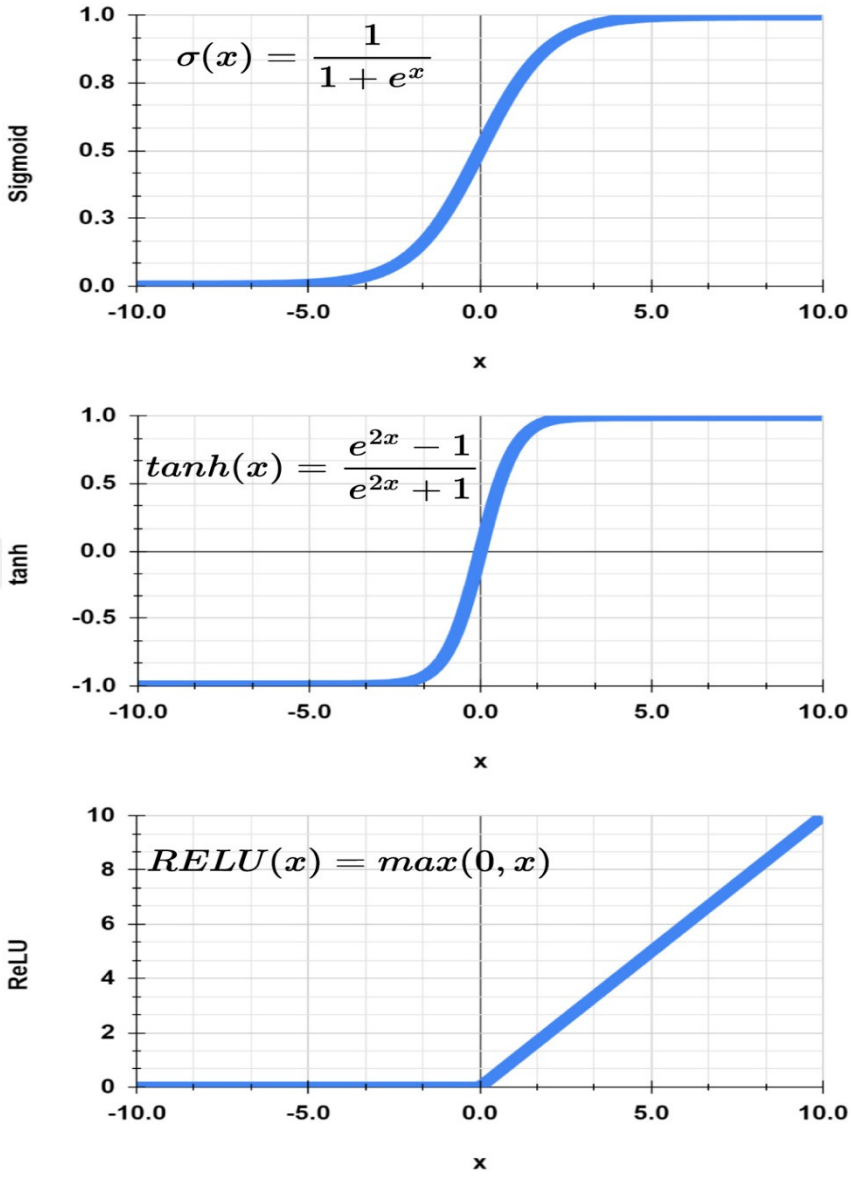



Figure 2.4: Sigmoid, Tanh and ReLU Activation Function (Barak, Sahar, Erez, Raja, & Alon, 2023)

or more hidden layers as shown in figure 2.3. These layers use specific activation functions to perform continuous non-linear transformations from input to output. Some examples include Rectified Linear Unit (ReLU), Sigmoid and Tanh functions each having unique mathematical definitions along their graphical depictions as illustrated in figure 2.4.

### 2.1.3 Summary

Deepfake detection is a difficult problem that requires an understanding of Neural Networks (NNs) especially Multi-Layer Perceptron (MLP). By using advanced NN architectures and comprehensive training strategies, accuracy and trustworthiness of synthetic media detection can be greatly improved which makes worthwhile contributions to digital media forensics at large. Accordingly, this chapter gives an elaborate account of various aspects of NNs stressing their crucial role in detecting deepfakes and emphasizing the need for continuous improvement.

## 2.2 Convolutional Neural Networks

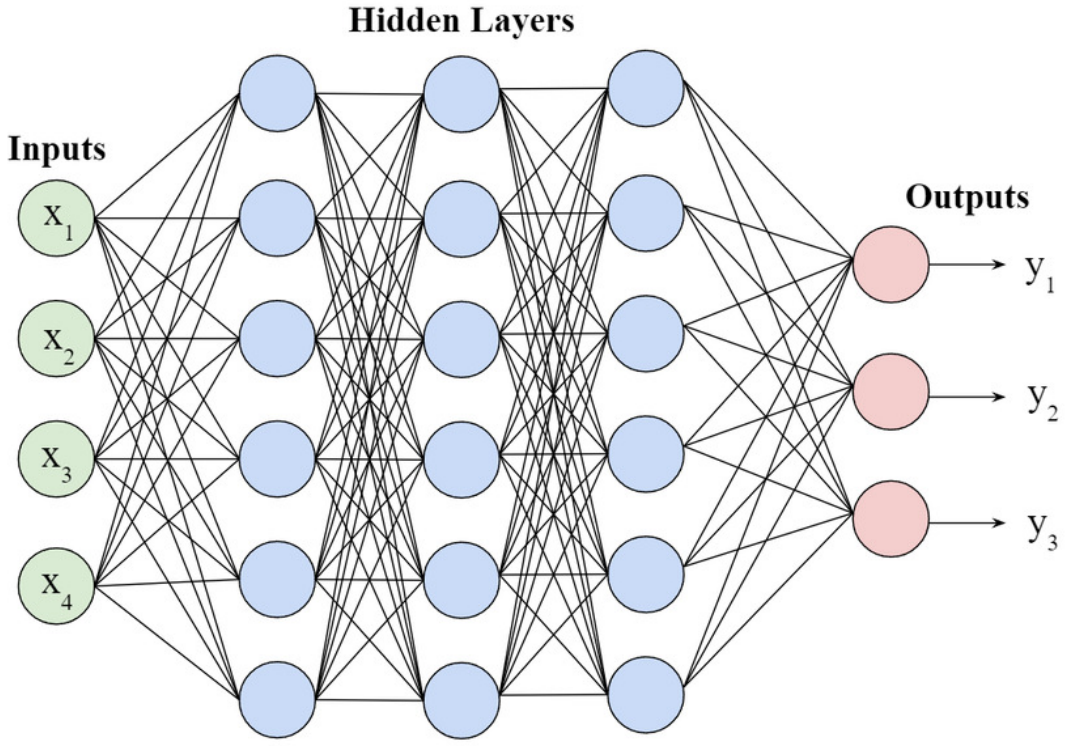
CNNs have been critical drivers of progress in computer vision with respect to image classification in particular. Since CNNs' architecture reduces the number of parameters in the network it helps minimize overfitting thereby allowing model development with hierarchical features learning capabilities. As CNNs go deeper into the network they can extract abstract representation from images. For example, early layers may identify edges while deeper ones recognize shapes or even objects (O'Shea & Nash, 2015).

### 2.2.1 Main Layers of CNNs

1. **Convolutional Layers**: These are layers that perform convolution operations on input data using filters (kernels). The major parameters for these layers include filter number as well as their size. In this case, a 2D convolutional layer (conv2D), each filter slides over the input image to create a feature map by carrying out element-wise multiplications. This process help captures spatial hierarchies of features.
2. **Pooling Layers**: Pooling layers decreases computational complexity by reducing spatial dimensions of feature maps and prevent overfitting in them. Max pooling and average pooling are commonly used techniques for pooling operations. Max pooling selects maximum value within a pooling window whereas average pooling computes average value.

3. **Dropout Layers**: Dropout is often incorporated into such networks to alleviate overfitting by randomly deactivating some neurons during training so that more robust features can be learned by the model and co-adaptation does not occur among neurons.
4. **Fully Connected Layers (Dense Layers)**: Convolutional layers do not typically connect all components with each other; thus, dense layers become necessary in finalization stages of CNN's where they take information that has been processed through convolutions and pooled before making predictions based on it. The similarity between these sets of layers and those found in Multilayer Perceptron's also known as MLPs make them critical for integration as well as analysis higher power level features for classification.

The structure of a CNN is shown in the figure below that illustrates a sequence of convolutional, pooling, and fully connected layers.



### 2.2.2 Challenges in Deepfake Detection Using CNNs

Even though CNNs have been very successful in different computer vision tasks, they still raise specific challenges when applied to deepfake detection:

1. **Effectiveness Against High-Quality Forgeries**: One of the main problems is how well CNNs are able to handle high-quality forgeries made with the help of advanced Generative Adversarial Networks (GANs). Synthetic image quality continuously improves as GANs evolve therefore detection algorithms need to move at least as fast. As GANs get smarter, the slight artifacts and inconsistencies that allow CNNs to detect alterations become increasingly difficult to identify (Goodfellow, et al., 2014).
2. **Computational Efficiency**: Traditional architectures for CNN often require a significant number of computational resources which makes them impractical for real-time or mobile applications. From this fact alone it can be concluded that deploying models based on CNN requires higher computational costs. Such developments therefore limit the practicability of using such models in areas with limited memory and processing power.
3. **Overfitting and Generalization**: Overfitting is one of the biggest challenges faced by CNNs especially when trained on small datasets. This happens when a model memorizes training data instead of generalizing new unseen data. In reality, this causes low performance scores during deepfake detection on actual task cases where forgery patterns are abundant.
4. **Scalability**: The issue regarding scalability comes into play as datasets grow more complicated and bigger so does the size of associated CNN models increase too. Training large-scale CNNs needs much time and computing resources which turns out to be a bottleneck in quickly changing fields like deepfake detection.
5. **Detection of Subtle Manipulations**: CNNs find it difficult to detect deepfakes which are usually refined. Minor facial expressions, minimalistic changes in lighting as well as other delicate variations may not dramatically change the overall look of the image but can easily deceive the conventional CNN models.

### 2.2.3 Mathematical Operation in CNNs

Convolution is the main mathematical operation in Convolutional Neural Networks (CNNs). For a 2D input image, it is expressed as:

$$Y(w,h) = (W.X)(w,h) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} W(s,t)\,.X(w-s,h-t) \qquad (5)$$

$\boldsymbol{Y}$ $(\boldsymbol{w},\boldsymbol{h})$ is where output feature map at position (**w**, **h**).

$\boldsymbol{W}$ represents a convolutional kernel or filter.

$\boldsymbol{X}$ stands for an input image or feature map.

$s$ and $t$ represent indices for the kernel's width and height, respectively.

$a$ and $b$ define the range of the kernel (i.e., half the width and height of the kernel, assuming it is centered).

This notation clearly represents the convolution operation, where each output value $Y(w,h)Y(w,h)Y(w,h)$ is computed by summing the element-wise product of the kernel $W$ and the corresponding region in the input $X$.

**Training and Loss Function**

During training, we update weights and biases iteratively using loss function that measures how different predicted output from actual output. In convolutional layers this includes elements of a kernel adjusted to minimize loss.

**Examples of Loss Functions:**

Cross-Entropy Loss:

$$L = -\frac{1}{N}\Sigma_{i=1}^{N}\Sigma_{c=1}^{C} y_{i,c}\log\left(\hat{y}_{i,c}\right) \qquad (6)$$

$\boldsymbol{N}$: samples' count.

$\boldsymbol{C}$: Number of course.

$\boldsymbol{y_{i,c}}$: Binary indicator (0 or 1) should class label $c$ be the proper categorization for sample $i$.

$\boldsymbol{\hat{y}_{i,c}}$: forecast probability that sample $i$ falls into class $c$.

### 2.2.4 Summary

Consequently, CNNs have proved useful in image classification tasks and contributed to the development of deepfake detection techniques but are challenged by several factors. These include high-quality forgeries that are hard to detect, computational inefficiency, overfitting risks, scalability problems and difficulty identifying subtle manipulations. Addressing these challenges is crucial for enhancing the robustness and applicability of deepfake detection models in real-world scenarios.

## 2.5 Generative Adversarial Networks (GANs)

The invention of Generative Adversarial Networks (GANs) has brought a new perspective on image synthesis in which two competing networks, generator and discriminator are engaged. This adversarial training process has been so effective in improving the ability of the generator to generate outputs that closely resemble real data (Goodfellow, et al., 2014).

### 2.5.1 Basic GAN Architectural Design

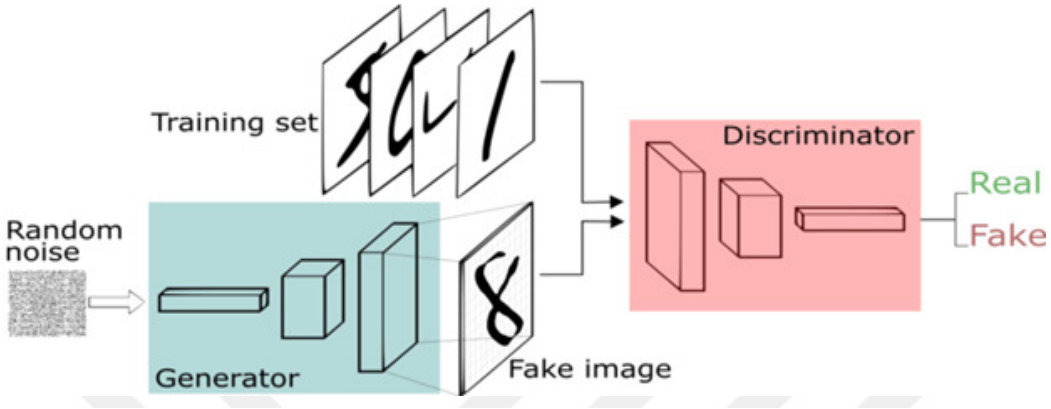



Figure 2.6: a GAN architectural Schematic View

A GAN comprises two main parts:

1. **Generator (G)**: The synthetic data produced closely matches the training set. Its implementation is through a multi-layer network typically with convolutional and fully connected layers. The producer aims to ensure that its output will be as real as possible.
2. **Discriminator (D)**: Discriminator functions distinguish between real and fake data. It assigns an image probability or rather it tells whether given an image is derived from the actual distribution or by the maker's generation. Discriminator also consists of a multi-layer network having convolutional and fully connected layers.

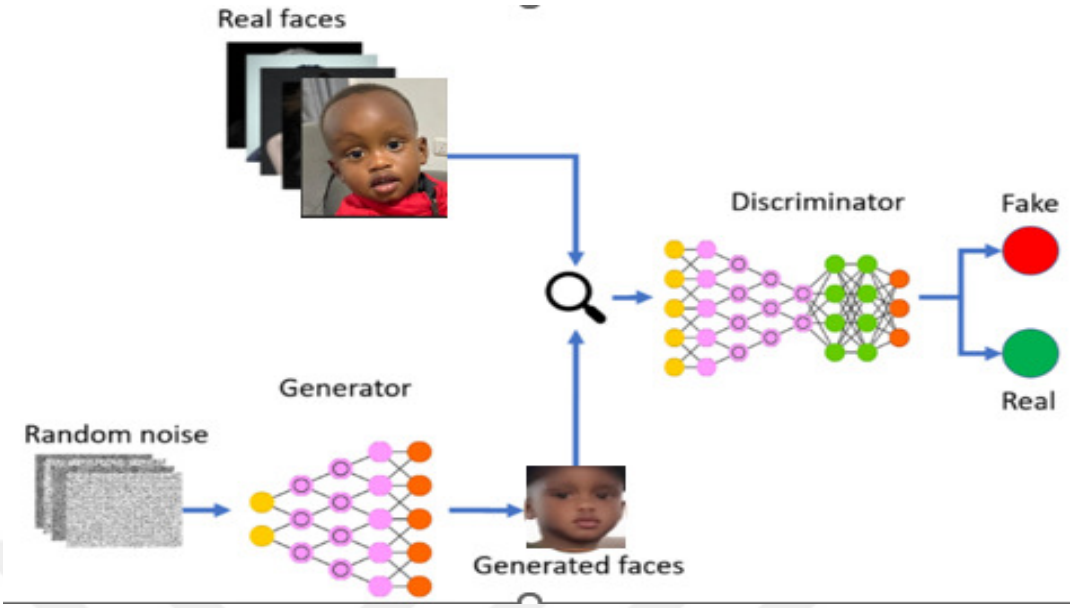



Figure 2.7: GANs Min-Max optimization problem

In this way, GANs enable the improvement of synthetic data quality via an adversarial process between generator and discriminator. The goal for the generator is to generate realistic images while discriminator must accurately classify true and false images hence they compete with each other thus driving continuous improvement by generator.

### 2.5.2 Mathematical Operation in GANs

The function of the generator can be written as follows:

$$G: G(Z) \rightarrow R^{|X|} \tag{7}$$

In mathematical operations, symbols such as |X| stands for the dimensionality or scope of the set $X$. The notation |X| means the dimensionality of the set $X$.

For instance, if vector space $X$ is in a dimensional space $d$, then this means that it has a number of components given by $|X|=d$.

where $Z$ is an element of set $R^{|Z|}$ represents a sample drawn from some latent space and $X$ within $R^{|X|}$ denotes the synthesized output. Discriminator, meanwhile, evaluates images based on their probability:

$$D: D(X) \rightarrow (0,1) \tag{8}$$

where D(X)= 0 implies that input image X comes from actual data distribution whereas D(X)= 1 shows that it is the creation of generator G.

The goal for the generator is to create images that cannot be distinguished from real ones, causing the discriminator to become unsure, assigning a probability of 0.5 to all inputs.

### 2.5.3. Training Process of GANs

The training process involves optimizing the parameters of the discriminator to improve its performance in classification and tuning those of the generator such that they make it increasingly harder for the discriminator. It resembles a two-player minimax game where each player has a distinct objective, which can be summed up as follows:

$$max_D min_G\big(E_x[logD(x)] + E_z[\log\big(1 - D(G(z))\big)]\big) \tag{9}$$

Using the notation $E_x f(x)$, we denote the expected value of the function $f(x)$ under the assumption that the random variable $x$ adheres to the probability distribution $p(x)$. This is formally defined as follows.

$$\mathrm{E}_x f(x) = \int f(x)p(x)dx \tag{10}$$

During training, one model's parameters are adjusted while another model remains fixed. A static generator may have a uniquely optimal discriminator according to Goodfellow et al (Goodfellow, et al., 2014). Thusly, before updating these parameters one should train the discriminator towards its unique optimum point with respect to current generator configurations. However, practically speaking

such optimization might not occur fully because there will always be constraints like limited number of iterations leading to concomitant update in both models' parameters.

### 2.5.4 Challenges in Training GANs

GANs present significant challenges during their training phase due to several factors:

1. **Convergence Issues**: The min-max game between the generator and discriminator can lead to convergence issues, where the training process oscillates or fails to converge at all.
2. **Mode Collapse**: It is said that GANs can have a mode collapse problem, where the generator produces only a very limited set of outputs, thus reducing the diversity of the synthetic data.
3. **Balancing Training**: Balance between generator and discriminator during training is highly important because if one model becomes too strong, it may obstruct another one's growth and development in this way.

### 2.5.5 Variants of GANs

The literature offers lots of versions of GANs with each one being tailored to a specific application such as classification, regression, image generation, domain adaptation or increasing image resolution [Creswell et al., 2018]. One remarkable example is Conditional GAN initiated by Mirza and Osindero in 2014 (Mirza & Simon Osindero, 2014 ). In this variation, both the generator and discriminator are conditioned on additional information like class labels or other forms of data so as to enhance generative ability.

### 2.5.6 GANs in Image Synthesis and Deepfake Detection

Generative Adversarial Networks (GANs) are known to be effective for image synthesis due to their ability to learn real-world statistics from training datasets. In this regard, GANs can produce new images virtually indistinguishable from genuine ones. This feature has been extensively deployed in deepfakes production whereby through training processes the network aims at generating realistic photos or videos of people that are designed mostly for fraudulent purposes.

Moreover, the authenticity of these generated images is assessed by the discriminator network before it provides feedback on them to the generator. Over time however, iterative training has helped improve generator capability leading to high quality fakes that become difficult to distinguish from real ones.

Additionally, in deepfake detection adversarial examples created using GAN's have also proved essential towards improving more robust detection models. By generating high-quality fakes, GAN pushes detection algorithms further into recognizing identifications hence enhancing overall robustness against these algorithms.

### 2.5.7 Summary

The advent of generative adversarial networks (GANs) has revolutionized image synthesis and deepfake detection through an adversarial training process between the generator and the discriminator. This has been successful for creating synthetic images that are nearly identical to real data. Nevertheless, training GANs poses numerous problems such as convergence, mode collapse, and imbalance of training between a generator and a discriminator. Various forms of GANs have been designed in order to fulfill certain tasks for example Conditional GAN. However, despite these challenges, GANs remain as one of the most powerful weapons used in fighting against creation and detection of deepfakes.

## 2.6 Statement of the Problem

The rapid development of deep learning methods has improved the ability to create very realistic artificial media known as deepfakes. These manipulations usually involve changing the appearance of people in pictures and videos which poses serious challenges to privacy, security and information integrity. The main problem that this thesis addresses is how well can genuine and manipulated images be identified and classified, with special focus on pictures showing people clearly and with all the necessary details.

In relation to deep learning, detecting deepfake images involves binary classification task. It entails making a function that will classify every image as either authentic or not by giving it a binary label. This classification function is defined as follows:

$$f: I \rightarrow \hat{y} \tag{11}$$

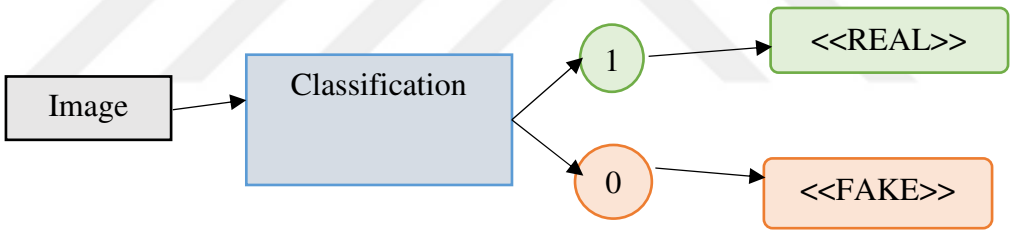



Figure 2.8: Formulation of binary classification by Author

Here I represents an individual image while $\boldsymbol{f}$ stands for the binary classifier. The output of the classifier, $\hat{y}$, is taken from $\{0,1\}$, where 0 refers to "fake" (manipulated) image whereas 1 indicates "real" (authentic) ones. The goal of this research is to design, present, and evaluate various classification functions f in order to distinguish between genuine and manipulated images effectively.

However traditional approaches such as Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) have several issues;

1. **Computational Efficiency:** They are often too expensive as far as traditional deep learning architectures such as CNNs are concerned hence they cannot be used practically in real time or mobile settings.
2. **Scalability**: This is evidenced by the fact that, with data sets growing in size and complexity, there is also increasing concern about the scalability of ANN/CNN models which require significant time and computational resources to train/infer.
3. **Detection of Subtle Manipulations**: Deepfakes may involve minor changes like slight variations in lighting conditions or small differences in facial expressions that can't be detected by ordinary CNNs/ANNs.
4. **Effectiveness Against High-Quality Forgeries**: As high-quality GANs continually improve at creating deepfakes, subtle artifacts and inconsistences become less visible thus posing consistent challenges to detection algorithms.

Therefore, this thesis addresses these problems through a novel hybrid approach proposing MTCNN for precise face detection and alignment together with MobileNetV2 for efficient image classification. Furthermore, an innovative Intermediate feature map differencing technique embedded within MobileNetV2 is presented that enhances detection accuracy by emphasizing area-specific anomalies suggestive of deepfakes. The aim of this study therefore is to enhance the accuracy, computational efficiency and scale of models deployed for detecting deepfake content as a real-time application so that visual content reliability and integrity can be assured.

Hence this research seeks to develop a reliable model capable of ensuring visual content authenticity through systematic analysis and tuning $f$ function for detecting deepfake contents. Consequently, it becomes vital to utilize this technique when preventing any possible threats imposed by deepfakes while considering different applications where data integrity matters.

# CHAPTER 3

# PROPOSED METHODOLODIES AND MODELS

## 3.1 Introduction

In this chapter, a comprehensive approach for deepfake detection will be described which includes data acquisition, preprocessing and analysis. Existing works deemed effective from other researches informed the methods of this study. In particular, we employed Multi-task Cascaded Convolutional Networks (MTCNN) and MobileNetV2 architectures which have proven their effectiveness in face detection and alignment as well as image classification.

## 3.2 Conceptual Framework

To make these existing models more effective within the context of deepfake detection, the MobileNetV2 framework is modified by incorporating a new feature differencing technique thereby creating a hybrid model with MTCNN. This unique combination uses MTCNN for face detection and alignment and MobileNetV2 for image classification while feature differencing involves identifying region specific features that detect small changes in images.

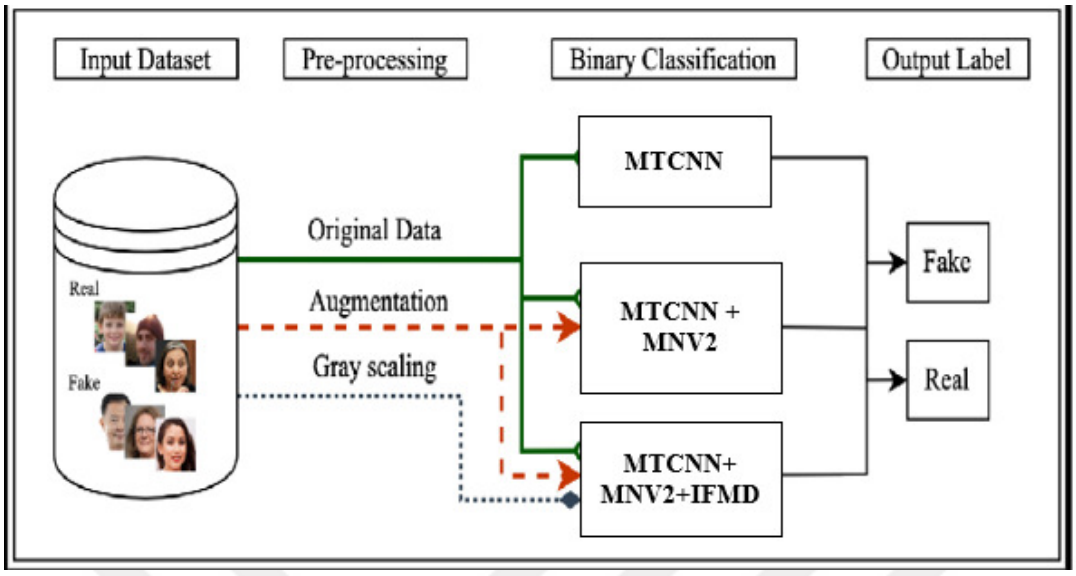



Figure 3.1: Deepfake Detection Model Architecture

The aforementioned ideas are combined sequentially through concatenating MTCNN with MobileNetV2 to maximize the accuracy of deepfake detection. The subsequent text delineates the way in which this process was dissected:

**Data Acquisition**: Explain how data were collected for training/testing deepfake detection models.

**Preprocessing**: Clarify how data were transformed or cleaned up ready for model training.

**Model Architectures**: Enumerate the Specify deep learning model architectures employed in this study to detect deep fakes.

**Data Acquisition**

The collection of data is an essential step in training and evaluating Deepfake detectors. Under this stage, a large dataset that mixed both real as well as fake images was obtained. In order to fully represent different facial characteristics, various public datasets along with custom-created datasets were used in our work. Specifically, this database consisted of 40k pre-processed human faces shared by

Trung-Nghia Le on Kaggle (Zenodo). It contained 20k genuine face images alongside 20k manipulated ones. Moreover, another dataset consisting of 21 genuine and 21 fake images was created to further evaluate this model's performance. To diversify the training set, data augmentation techniques were applied such as rotating the image, scaling it or changing its color.

**Preprocessing**

Cleaning and converting raw data in a format that is easy to use for training deepfake detection models constitute this stage. Some key steps involved in pre-processing are as follows:

1. **Normalizat**ion: Scaling pixel values to a standard range.
2. **Face Detection and Alignment**: Using MTCNN to detect and align faces within the images.
3. **Augmentation**: Applying transformations to increase dataset variability.
4. **Feature Extraction**: Use MobileNetV2 to identify relevant features from the aligned facial photos.

## 3.3 Model ArchItectures

### 3.3.1 Explanation of the Multi-task Cascaded Convolutional Neural Network (MTCNN)

Multi-task Cascaded Convolutional Networks(MTCNN) – An advanced Neural Network used for Face Detection and Alignment (Zhang, Wang, & Chen, 2021) is designed primarily for face detection without alignment task. With cascaded convolutional neural networks (CNNs), each stage refines results from the previous one. By efficiently detecting facial landmarks like eyes, nose or mouth on images MTCNN tries to do accurate alignment.

### 3.3.2 Key Components of MTCNN

**1. Stage 1: Proposal Network (P-Net)**

- **Function:** Generate candidate windows(proposals) that may contain faces
- **The P-Net's operation** is a fast scan over the input image at different scales, which generates bounding box candidates and outputs confidence scores. The stage employs a fully convolutional network (FCN), trained on fixed size images, which are then applied to images of different dimensions during testing ( Jia, Mao, Qi, Zuo, & Sun, 2019). The network employs reverse calculation to derive proposals for facial regions within the input image.

The calculations that follow in order to derive proposals are:

$$X_1 = \frac{(stride\ .\ X)}{scale},\quad X_2 = \frac{(stride\ .\ x)\ +\ cellsize}{scale} \tag{12}$$

$$y_1 = \frac{(stride\ .\ y)}{scale},\qquad y_2 = \frac{(stride\ .\ y) + cellsize}{scale} \tag{13}$$

Here, $X_1$, $X_2$ represent coordinates of a pixel location on the feature map along the horizontal axis and $y_1$, $y_2$ refers to the corresponding coordinates of region proposal, along the vertical axis. Stride indicates step size of pooled layer and cell size denotes dimensions of region proposal. Scale shows scaling ratio between current input image and original image.

2. The second stage: **R-Net**

**Refinement Network Function**: This refines candidate windows made by P-Net.

**Operation**: Candidates from the P-Net are processed by R-Net so as to remove false positives and refine their locations as well as confidence scores.

**3.** The third stage: **O-Net**

**Output Network Function**: Final refinements; detection results outputting device.

**Operation**: O-net takes refined candidates from R□NET, giving more accurate face bounding box coordinates, confidence scores, and facial landmark positions.

### 3.3.3 Key Features of MTCNN

**Multi-task Learning** - MTCNN performs multiple tasks simultaneously; detecting faces and locating facial landmarks thus making it more robust with better performance.

**Cascaded Structure** – By cascading these three stages progressively improve both detection accuracy and alignment quality reduce number of false positives (Zhang, Zhang, Li, & Qiao, 2016).

**Efficiency** – Designing MTCNN to be computationally efficient makes it suitable for real-time applications on low-resource devices like smartphones and embedded systems ( Jia, Mao, Qi, Zuo, & Sun, 2019).

### 3.3.4 Applications of MTCNN

**Face Detection** – Applied in security systems, photo organization and social media applications for quick and accurate face detection.

**Facial Landmark Alignment** – This aligns facial landmarks that are important in tasks such as face recognition, emotion detection and augmented reality.

**Preprocessing for Face Recognition** - Before recognition algorithms are applied; faces have to be correctly detected and aligned with (Yang & Zhang, 2021).

### 3.3.5 Detailed Process of MTCNN

In image classification or recognition processes CNNs have proven to be extremely effective but require substantially large computational resources limiting their capacity to process images in real time. To avoid this, MTCNN employs a

cascading approach through three stages namely P-Net, R-Net and O-Net which are each set with different networks' confidence score thresholds as well as intersection over union (IOU) criteria designed to serve specific functions required by the architecture of the network.

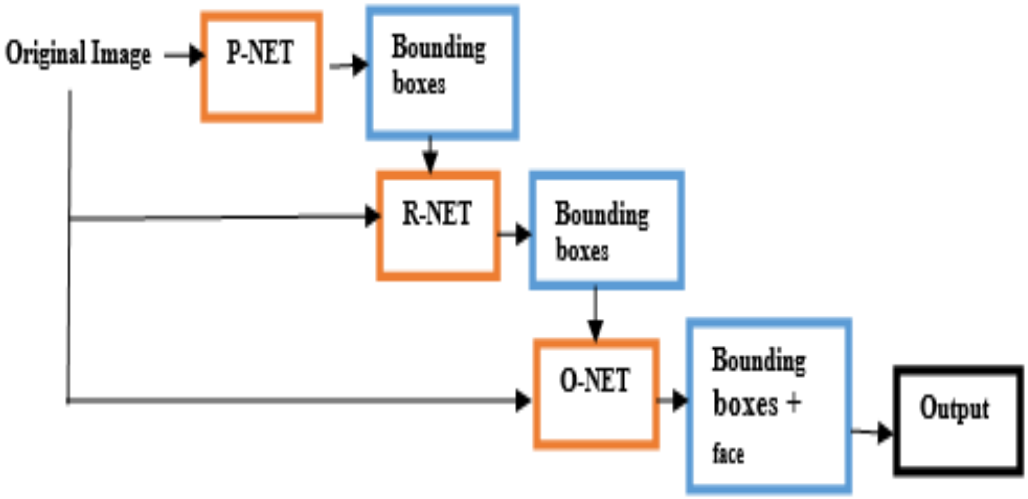



Figure 3.2: Flowchart for face detection

The figure at Figure 3.2: Flowchart for face detection explains the detailed steps involved in implementing the MTCNN algorithm:

1. **P-Net Stage:**
   - It detects potential facial regions and conducts boundary box adjustments.
   - Consolidates overlapping detections by applying non maximum suppression (NMS).
2. **R-Net Stage:**
   - The selection accuracy is enhanced using a more sophisticated CNN that eliminates most false candidates for face locations.
   - Boundary adjustments are refined while employing NMS again to merge overlapped detections on facial features localization.
3. **O-Net Stage:**
   - A deeper analysis enhancing R-Net's output runs identity verification, boundary refinement and feature localization thereafter.
   - It outputs coordinates of facial region and major feature points.

HAAR cascade detection is used in computer vision to identify objects in pictures. It performs similarly to a cascade process. It starts by applying a series of simple filters one at a time, consequently detecting regions of interest fast while throwing away any remaining irrelevant ones. It helps reduce the amount of data that must be processed in this way, leading to faster and more efficient calculations. This method is very effective for real-time face detection because it can quickly scan an image for the presence of a face and focus processing power on those regions (Viola & M, 2001).

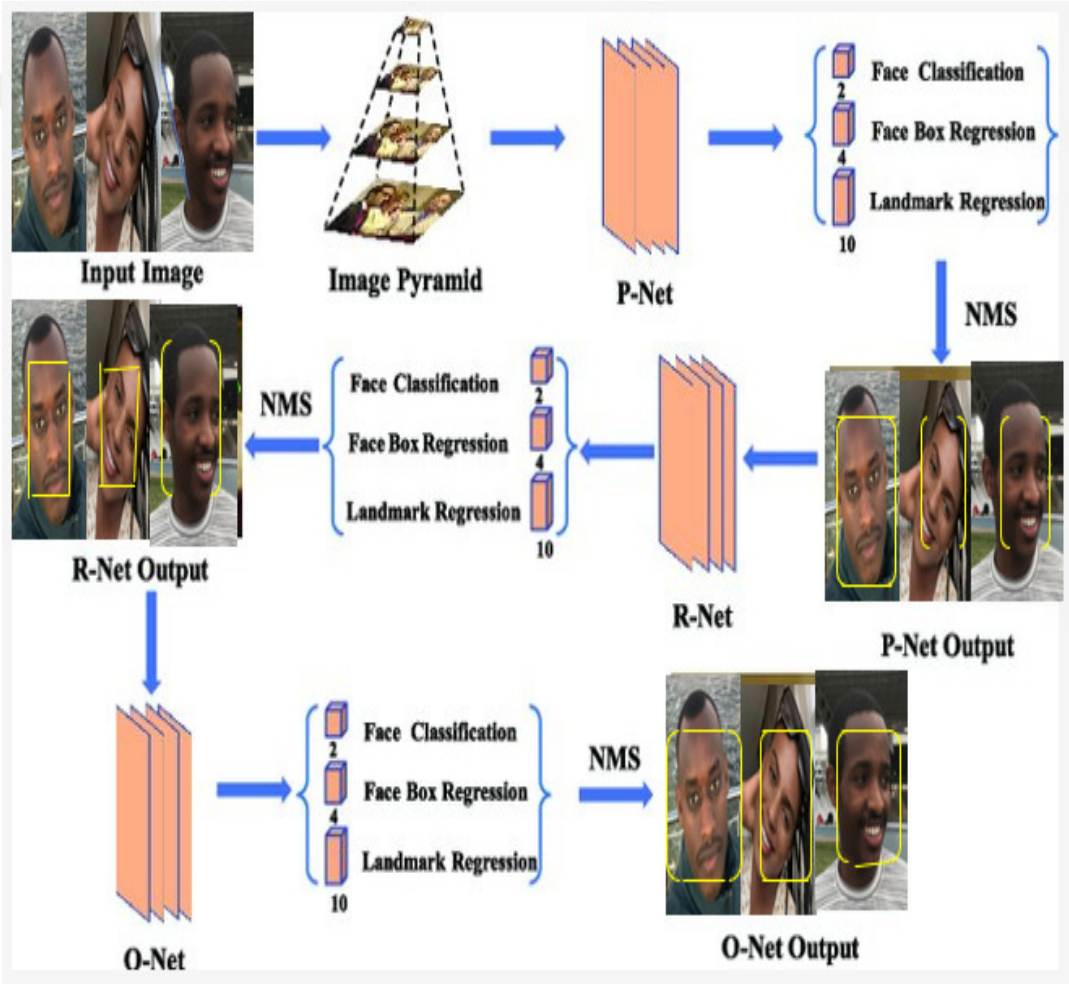



Figure 3.3: Stages for face detection with MTCNN

It was called “HAAR” because it has HAAR-like features which are simple rectangular patterns of black and white areas that were initially inspired by HAAR

wavelets. These features help detect edges, lines or other basic patterns within the image.

## 3.4 MobileNetV2 and MobileNetV2 Intermediate Feature Map Differencing Technique

Mobile and embedded vision applications are the key domains where MobileNet, a neural network architecture is highly efficient. It is demonstrated here how it works in detail with a focus on its unique convolution approach and use in classification:

**Convolutional layers and feature extraction**

The process of extracting features from input images uses convolutional layers. They employ filters that move across the input data, capturing important patterns and characteristics critical for recognizing objects in images. The essence of MobileNet lies in its creation of separable convolutions that are depthwise, which have efficiently reduced the computational cost as compared to conventional convolutions.

**Depthwise Separable Convolutions**

These types of convolutions are broken down into two steps namely:

1. Depthwise Convolution:

**Operation**: Each input channel is separately processed by one filter.

**Benefit**: Every channel is handled independently during processing thus considerably reducing computation.

2. Pointwise Convolution:

**Operation**: The outputs from depthwise convolutions are combined through a 1x1 convolution.

**Benefit**: This step further reduces the computational burden by merging the separately processed channels.

By decoupling convolution into these two stages, MobileNet has been able to reduce both computations and parameters while still maintaining high efficiency and efficacy. Therefore, this methodology underpins computational efficiency behind MobileNet.

**Classification**

After feature extraction has been done, MobileNet applies fully connected layers for classification purposes based on predefined categories. Usually, the final layer uses Softmax activation to give probabilities for every class. The network's lightweight ensures that it can easily be used for real-time classification tasks even with limited computational capacities (e.g., smartphones and embedded systems).

**MobileNetV2**

MobileNetV2 is one of the most recent neural network architectures for mobile and embedded vision application that builds on MobileNet. It has a number of innovations to increase efficiency and performance. Due to its reduced computational complexity but still high quality, this makes it suitable for low power processing in constrained computational devices.

**Innovations in MobileNetV2**

MobileNetV2 extends MobileNetV1's ideas using depthwise separable convolutions as efficient building blocks. However, V2 introduces two new features to the architecture:

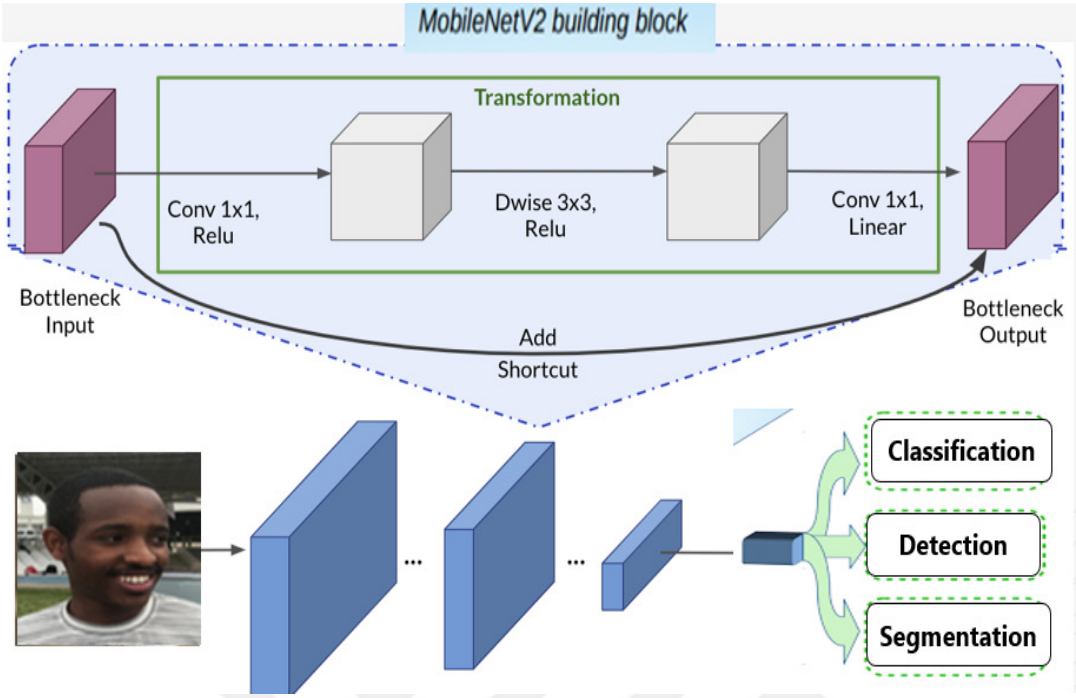



Figure 3.4: MobileNet Architecture

**Linear Bottlenecks**:

- **Concept**: To avoid information loss from non-linearities due to ReLU activations, linear bottlenecks are introduced between layers.
- **Function**: This way there is a homogeneity within each layer of MobileNetV2 making the flow of information smooth and retaining more data. The net result is an improved total performance.

**MobileNetV2's Inverted Residual Connection**

The "inverted residual connection" is an essential invention of MobileNetV2, a popular convolutional neural network architecture designed for efficient mobile and embedded vision applications. Here is a simple explanation of what it is and how it works;

1. **In ResNet, the Traditional Residual Connection**
    - **Residual Block**: The input to the block is added to the output of the convolutional layers inside the block through an identity shortcut connection.

- **Objective**: Addresses the issue of vanishing gradients by allowing deeper networks in such a way that gradient may directly flow via backpropagation within shortcut connections.

**2. MobileNetV2's Inverted Residual Block**

**Basic Idea**: Instead of using traditional residual connections, MobileNetV2 employs an "inverted" approach. The residuals are applied to bottleneck layers instead of wider ones.

**Key Features Of MobileNetV2**

**1. Inverted Residuals And Linear Bottlenecks:**

**Inverted Residuals:**

- Concept: Inverted residual structures in MobileNetV2 differ from traditional residual blocks. Instead of increasing the number of channels first then reducing them, inverted residuals start with a narrow bottleneck layer then expand into higher-dimensional space inside.
- **Process:** Initially, the input feature maps are passed through 1x1 convolutions for dimension expansion. Afterward, depthwise separable convolutions are used at this higher-dimensional space. Finally, another 1x1 convolution reduces the dimensions back to a thin bottleneck layer.
- **Benefit**: This structure enhances computational efficiency while maintaining ability to capture complex features. This effectively balances trade-offs between model size and model accuracy.

**Linear Bottlenecks**:

- **Concept**: Linear bottlenecks are introduced to prevent loss of information due to non-linearities caused by ReLU activations.
- **Function**: In order that some level of similarity can be maintained across the layers within MobileNetV2**,**

- The advantage is that such an approach allows sustaining smooth information flow hence saving more data and enhancing overall operation quality (Sandler, Howard, Menglong Zhu, Zhmoginov, & Chen, 2019).

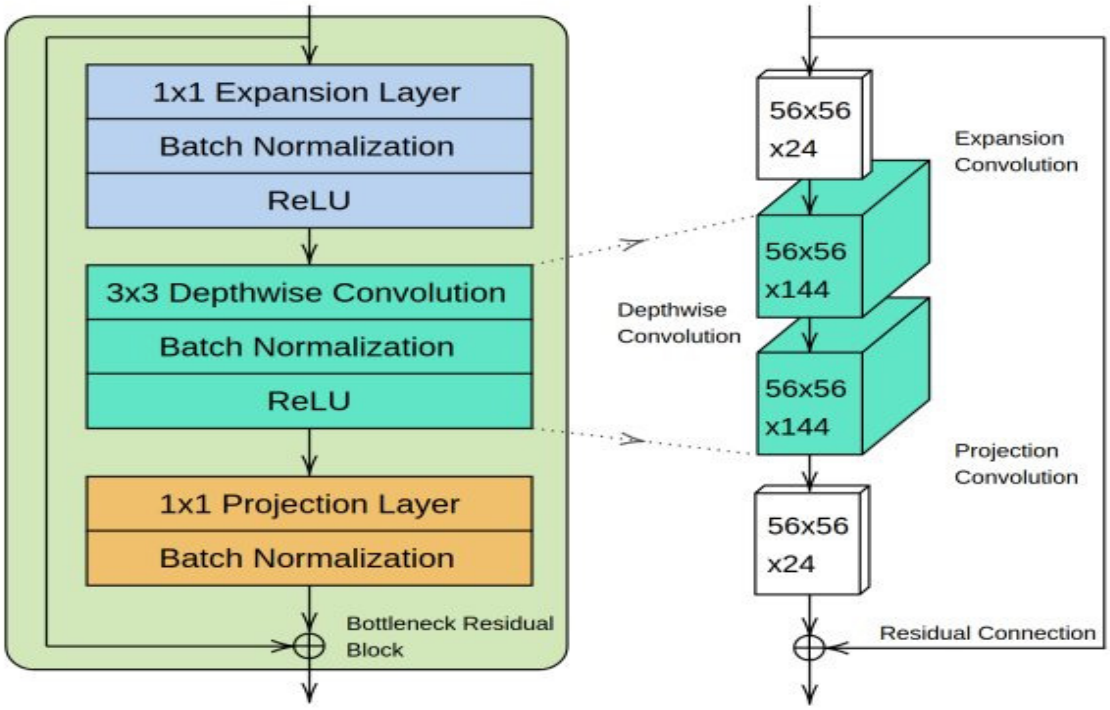



Figure 3.5: shows MobileNetV2 using inverted residuals. It details how linear bottlenecks expand feature maps from 24 to 144 and then reduce them back to 24 (Enkvetchakul & Surinta, 2022)

**Depthv**

- These convolutions split the convolution operation into depthwise and pointwise convolutions thereby resulting in significant reduction in computational cost.
- This technique keeps efficiency while allowing lightness and effectiveness of models.

**2. Efficiency:**

- It is designed for efficiency and is suitable for real-time applications on low resource devices (Sandler & Howard, MobileNetV2: The Next Generation of On-Device Computer Vision Networks, 2018).

**Architecture**

Fully Convolutional Layer is the starting point of MobileNetV2 framework followed by 19 residual bottleneck layers. This design uses depthwise separable convolutions, inverted residuals, and linear bottlenecks to reduce parameters as well as the computational workload while still allowing the model to find intricate and abstract features.

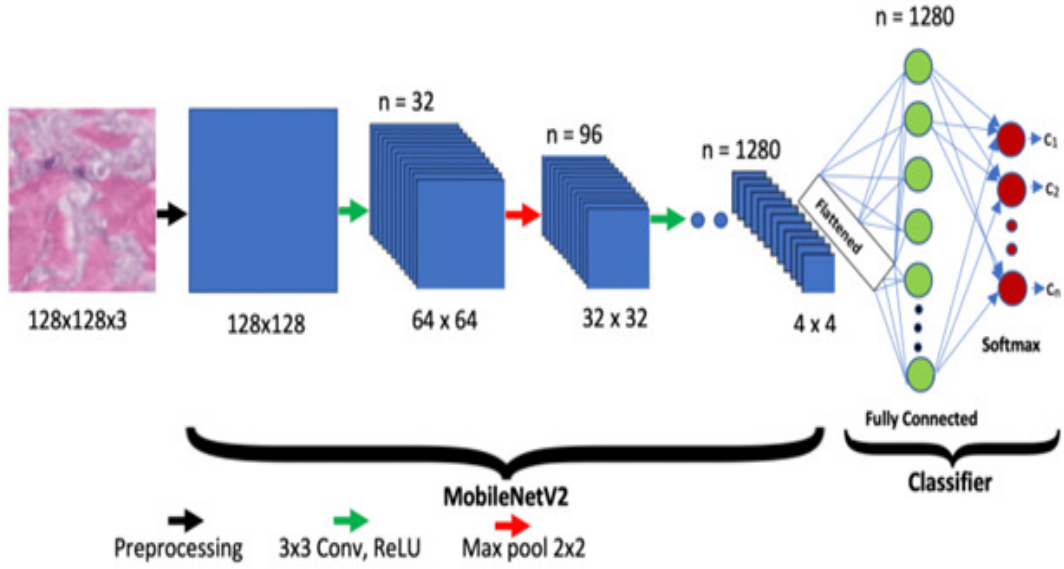



**Figure 3.6:** The MobileNetV2 architecture

**Detailed Design of Core Block**

Figure 3.6 above reveals an architecture of MobileNetV2. To ensure efficient computation, MobileNetV2 improves upon the design principles of MobileNetV1 by using depthwise separable convolutions. Linear bottlenecks and shortcut connections constitute two major improvements introduced by MobileNetV2 that connects these bottlenecks together. These bottlenecks will effectively act as

proxies for intermediate inputs and outputs in the model, whereas inner layers will target transformations of lower-level features such as pixels into higher level representations like image categories. Additionally, these shortcuts allow networks to converge faster than classical residual connections resulting into higher accuracy.

Our design extends such a type of model idea including a bottleneck depth-separable convolution with residual connections. We summarize this core block's detailed design in Table 1 depicting how we transform k channels to k' through few operations utilizing a step size of $s$ and a multiplication factor of t.

Table 1: Illustration of the bottleneck residual block, showing the channel transformation from $k$ to $k'$, along with parameters like stride s and expansion factor $\boldsymbol{t}$.

| **User's input** | **User's operator** | **User's output** |
|---|---|---|
| $\boldsymbol{h.w.k}$<br>$\boldsymbol{h.w.k}$<br>$\frac{\boldsymbol{h}}{\boldsymbol{s}}.\frac{\boldsymbol{w}}{\boldsymbol{s}}.\boldsymbol{tk}$ | **1.1 conv2d, ReLu6**<br>3. **dwise s=s, ReLu6**<br>Linear 1.1 conv2d | $\boldsymbol{h}.(\boldsymbol{tk})$<br>$\frac{\boldsymbol{h}}{\boldsymbol{s}}.\frac{\boldsymbol{w}}{\boldsymbol{s}} \times (\boldsymbol{tk})$<br>$\frac{\boldsymbol{h}}{\boldsymbol{s}}.\frac{\boldsymbol{w}}{\boldsymbol{s}}.\boldsymbol{k}'$ |

**Explanation**:

$\boldsymbol{h}$ and $\boldsymbol{w}$: The height and width of the input feature map.

$\boldsymbol{k}$**:** The number of input channels.

**t**: Expansion factor of the input channels before depthwise convolution.

$\boldsymbol{tk}$: The count of the expanded number of channels after expansion factor has been used.

$\boldsymbol{s}$: Stride size, which controls downsampling of the feature map.

$k'$: Number of output channels after linear 1x1 convolutions.

Our architecture, MobileNetV2, starts with a fully convolutional layer comprising 32 filters followed by nineteen residual bottleneck layers. ReLU6 was our activation function since has proven resilience in low precision computation. Like any other modern network designs, we employ a 3x3 kernel size throughout and during training adopt dropout and batch normalization.

In all but the first layer, the expansion rate is constant across the network. Empirical evaluations have shown that expansion rates between 5 and 10 show similar levels of performance where smaller networks prefer slightly lower expansion rates while larger ones improve slightly when using higher rates.

**Detailed Design of 19 Residual Bottleneck Layers**

Table 2: Detailed Design of 19 Residual Bottleneck Layers

| User's input Size | User's operator | t | c | n | s |
|---|---|---|---|---|---|
| $224^2 * 3$ | Conv2d | - | 32 | 1 | 2 |
| $112^2 * 32$ | bottleneck | 1 | 16 | 1 | 1 |
| $112^2 * 16$ | Bottleneck | 6 | 24 | 2 | 2 |
| $56^2 * 24$ | Bottleneck | 6 | 32 | 3 | 2 |
| $28^2 * 32$ | Bottleneck | 6 | 64 | 4 | 2 |
| $14^2 * 64$ | Bottleneck | 6 | 96 | 3 | 1 |
| $14^2 * 96$ | Bottleneck | 6 | 160 | 3 | 2 |
| $7^2 * 160$ | bottleneck | 6 | 320 | 1 | 1 |
| $7^2 * 320$ | conv2d 1*1 | - | 1280 | 1 | 1 |
| $7^2 * 1280$ | avgpool 7*7 | - | - | - | - |
| 1*1*1280 | Conv2d 1*1 | - | k | - | - |

**Cost Function for MobileNetV2**

Typically, the cross-entropy loss is used as a cost function in MobileNetV2 for classification tasks. The cross-entropy loss indicates how dissimilar the true labels are as compared to those predicted by the model. It is given by the formula:

$$L = \frac{1}{N} \Sigma_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}) \quad (14)$$

Where;

- $N$ samples' count;
- $C$ Number of course;
- $y_{i,c}$ – binary indicator (0 or 1) that shows if class label $c$ be the proper categorization for sample $i$.
- $\hat{y}_{i,c}$ predicted probability that sample $i$ belongs to class $c$.

**Depthwise Separable Convolution**

Depthwise separable convolutions are special types of convolution operations that reduce parametric size and computation time significantly compared to regular convolutions. Depthwise separable convolutions consist of two steps:

1. **Depthwise Convolution**: Each input channel gets convolved independently with one convolutional filter only.
2. **Pointwise Convolution**: After depthwise convolution, combining of outputs takes place through a 1x1 convolutional layer.

This method allows MobilNetV2 to achieve performance similar to traditional CNNs but consumes a fraction of their computational cost and parameter count.

### 3.5 MobileNetV2 Intermediate Feature Map Differencing Technique

**Background**

The MobileNetV2 Intermediate Feature Map Differencing Technique improves detection of deepfakes using the original MobileNetV2 by highlighting differences in real vs. fake images. This is achieved by extracting feature maps from different layers of the network and performing dıfferencıng on them to highlight slight variations between these two types of images.

**Key Features of MobileNetV2 Intermediate Feature Map Differencing Technique**

- **Feature Differencing Across Layers**: This approach involves comparing feature maps of different layers across the network for genuine and faked images to identify areas where they differ most conspicuously.
- **Feature Maps**: These are outputs from convolutional layers in a neural network, also known as activation maps, that depict presence or absence of certain properties or partterns within different regions of an input image. The generation process involves applying separate filters or kernels during convolution operation to produce distinct feature maps.
- **Defferencing Technique**: The purpose of this technique is to distinguish between real and synthesized images by processing feature maps from various layers of the network thereby unveiling minute defects or distortions that may come with this manipulation called deepfake creation.

**Steps Involved in Implementing MobileNetV2 Intermediate Feature Map Differencing Technique**

**Feature Extraction**: Extract feature maps from two different layers of the network. These selected layers are responsible for capturing relevant features.

1. **Differencing Operation**: Perform element-wise differencing on the feature maps so as to obtain a new map which emphasizes on the disparities.
2. **Analysis**: Analyze feature map obtained for anomalies and classify it according to its class, such as ‘real’ or ‘fake’.

**Figures Illustrating Conceptualization behind The Use Of Differencing**


Figure 3.7: Simulated Feature Maps and Subtraction Diagram

To demonstrate visually how this concept works, one needs to consider two particular feature maps captured at distinct levels:

- **Real Feature Map** – extracted from one layer when processing a real image; it contains patterns detected by convolutional filters.
- **Fake Feature Map** – drawn out from another/different layer when treating a false picture; it may contain similar or other patterns.
- **Difference Feature Map** -the outcome of differencing the fake feature map from the real one, which discloses any subtle discrepancies and makes them more visible.

**Enhanced Sensitivity and Improved Accuracy**

The MobileNetV2 intermediate feature map differencing technique increases the network's sensitivity to small variations and artifacts introduced during deepfake generation. It was observed in preliminary experiments that there is a significant improvement in accuracy with rates going up from around 95.57% to 98%.

**Implementation**

Then, the network processes an image, extracts high-dimensional feature vectors, performs the differencing operation to highlight differences, and then analyzes the resulting vector to estimate the likelihood of the image being a deepfake.

**Unique Aspects of MobileNetV2 Intermediate Feature Map Differencing Technique**

1. **Objective**:
   - Intermediate Feature Map Differencing: Looks for differences between real and fake images to identify deepfakes which are not apparent to the naked eye.
2. **Approach**:
   - Intermediate Feature Map Differencing: Involves using feature maps from different layers, and emphasizing their variations in order to detect outliers that indicate spurious images.
3. **Processing**:
   - Intermediate Feature Map Differencing: Extracts and processes feature maps from different layers then perform differencing operations on them so as to detect minute deviations.
4. **Sensitivity to Anomalies**:
   - Intermediate Feature Map Differencing: It is more sensitive to minor changes and artefacts introduced during deepfake generation thereby increasing detection accuracy.

**Mathematical Equation for MobileNetV2 Intermediate Feature Map Differencing Technique**

1. **Feature Map Extraction**:
   - Let $F_{real}^{(l)}$ real $(l)$ denote a feature map from a real image extracted from layer $l$ of MobileNetV2 network.
   - Let $F_{fake}^{(l')}$ fake $(l')$ denote a feature map from a fake image extracted from layer $l'$ of MobileNetV2 network.
   - $F_{real}^{(l)}$ and $F_{fake}^{(l')}$ are tensors of dimensions $H \times W \times C$ , where H is the height, W is the width and C is the number of channels.

2. **Differencing Operation**
   - Perform element-wise differencing between the feature maps $F_{real}^{(l)}$ and $F_{fake}^{(l')}$.
   - Let the difference feature map D be given as:

$$D = F_{real}^{(l)} - F_{fake}^{(l')} \tag{15}$$

3. **Normalization (Optional):**

Normalize the difference feature map $D$ to ensure that differences are on a comparable scale. Let $\mu D$ and $\sigma D$ be the mean and standard deviation of $D$.

$$D_{norm} = \frac{D - \mu D}{\sigma D} \tag{16}$$

4. **Weighted Differencing (Optional):**

- Introduce weighting factors to emphasize differences in certain regions or channels. Let $W$ be a weight tensor having same dimensions as $D$.

$$D_{weighted} = W \odot \left(F_{real}^{(l)} - F_{fake}^{(l')}\right) \tag{17}$$

Here, $\odot$ denotes element-wise multiplication.

5. **Final Difference Map:**

- The final difference map can be represented as:

$$D_{final} \frac{W \odot \left(F_{real}^{(l)} - F_{fake}^{(l')}\right) - \mu D}{\sigma D} \tag{18}$$

- If normalization and weighting are not used, then:

$$D_{final} = F_{real}^{(l)} - F_{fake}^{(l')} \tag{19}$$

**Incorporating these into MobileNetV2 Workflow**

1. **Feature Extraction**

- Extract feature maps from specific layers $l$ and $l'$ in MobileNetV2 for real and fake images, respectively:

$$F_{fake}^{(l')} = MobileNetV2\left(X_{real,}l\right) \tag{20}$$

$$F_{real}^{(l)} = MobileNetV2\left(X_{fake,}l'\right) \tag{21}$$

2. **Difference calculation:**

- Compute the final difference feature map**:**

$$D_{final} \frac{W \odot \left(MobileNetV2(X_{real,}l) - MobileNetV2(X_{fake,}l')\right) - \mu D}{\sigma D} \quad (22)$$

**Variable Explanation**:

$F_{real}^{(l)}$ : Real image feature map at layer l

$F_{fake}^{(l')}$: Fake image feature map at layer l'

$D$ : Initial Difference Feature Map

$\mu D$ : mean of difference feature map

$\sigma D$ : Std deviation of difference feature map

$W$ : weight tensor for highlighting specific differences.

$D_{final}$ : Final Difference Feature Map.

**Summary**

Therefore, MobileNetV2 with MobileNetV2 intermediate feature map differencing technique is a significant stride in lightweight neural networks, providing accurate and efficient solutions to deepfake detection among other image classifications. These advancements enable the deployment of robust systems on resource-constrained devices without compromising the quality of visual content, thus ensuring data integrity and authenticity by preventing any possible manipulations that may be brought about by inserting modified images into original information. With these steps in place, deepfake images can be accurately identified by MobileNetV2 just as it emphasizes on distinctions between real and synthetic images.

# CHAPTER 4

# CONTRIBUTIONS AND NOVELTIES

The research was aimed to increase the efficiency of deepfake detection through a novel hybrid approach that combines Multi-task Cascaded Convolutional Networks (MTCNN) and MobileNetV2, and an innovative feature differencing technique. In this section, we provide a comparative performance analysis of the models used - including implementation details, experimental settings and evaluative methodologies employed.

## 4.1 Implementation

### 4.1.1 Hardware and Software Configurations

Experiments were done using Google Colab as well as local Jupyter Notebook environment. A virtual machine on Google Colab had different hardware configurations typically with Intel(R) Xeon(R) CPU @ 2.20GHz or similar, a GPU like Tesla K80, T4, P4 or P100 and around 12 GB to 25 GB RAM while various local setups had medium to high-end NVIDIA GTX or RTX series GPUs or equivalent AMD GPUs and 16 GB or higher RAM available for larger memory capacity requirements depending on the task being executed.

### 4.1.2 Parameter Settings

To ensure computational efficiency and data usage were balanced, images were resized to a common input size of 256x256 pixels for all models during training, with a batch size of 32 in each training iteration. To enable gradual learning and fine-tuning, the training process was divided into stages for different model configurations.

For MTCNN model, 100 epochs were enough. This duration allows effective initial learning so that the model is exposed to enough dataset to capture major patterns and features.

In the second stage involved MTCNN+MobileNetV2 which was trained over 80 epochs. It was an improvement on the foundational learning from the MTCNN-only model where MobileNetV2 provided more sophisticated feature extraction and classification capabilities within a shorter but intensive training period.

The last phase took only two epochs of fine-tuning with the MTCNN+MobileNetV2+MobileNetV2 intermediate feature map differencing technique. In this case, MobileNetV2 intermediate feature map differencing technique has been used as a final attempt aimed at optimizing models' performance between reality and synthetic images through unveiling subtle details.

Adam optimizer was used in all cases due to its ability to handle scenarios with sparse gradients. This made it eligible for comprehensive optimization that ensures accurate outputs during testing phase. The stepwise training proved fundamental in attaining high-performance as well as robustness across these models.

### 4.1.3 Collection of Data

For training, a balanced dataset containing 20,000 authentic and 20,000 fake images was used – a total of 40,000 preprocessed images from Kaggle (Zenodo). Also, for a more comprehensive assessment of the model's performance, we created a custom dataset consisting of 42 images out of which 21 were real and the other 21 were fake. Dataset variability was increased by applying data augmentation techniques such as rotation, scaling, color adjustments and so on in order to enhance the robustness of the model.

```
for image_batch, label_batch in dataset.take(4):
    plt.imshow(image_batch[0].numpy().astype("uint8"))
    plt.title(class_names[label_batch[0]])
    plt.show()
```


Figure 4.1: im show command to illustrate real and fake images

## 4.2 Experiments

### 4.2.1 Experimental Setup

To ensure optimal performance and reproducible results the experimental setup was fine-tuned. Google Colab and a local Jupyter Notebook environment were used with different hardware configurations for the experiments. The software environment utilized Linux-based OS Python 3.6 or higher versions for developing models with TensorFlow 2.x and Keras along with supplementary libraries for visualization and processing of data across both platforms.

### 4.2.2 Execution of Experiment

Three models like MTCNN for face detection, MobileNetV2 for efficient analysis and innovative deepfake detection approach using MobileNetV2 intermediate feature map differencing technique are tested: each model is trained well before evaluation process thus demonstrating that these proposed methodologies really work effectively.

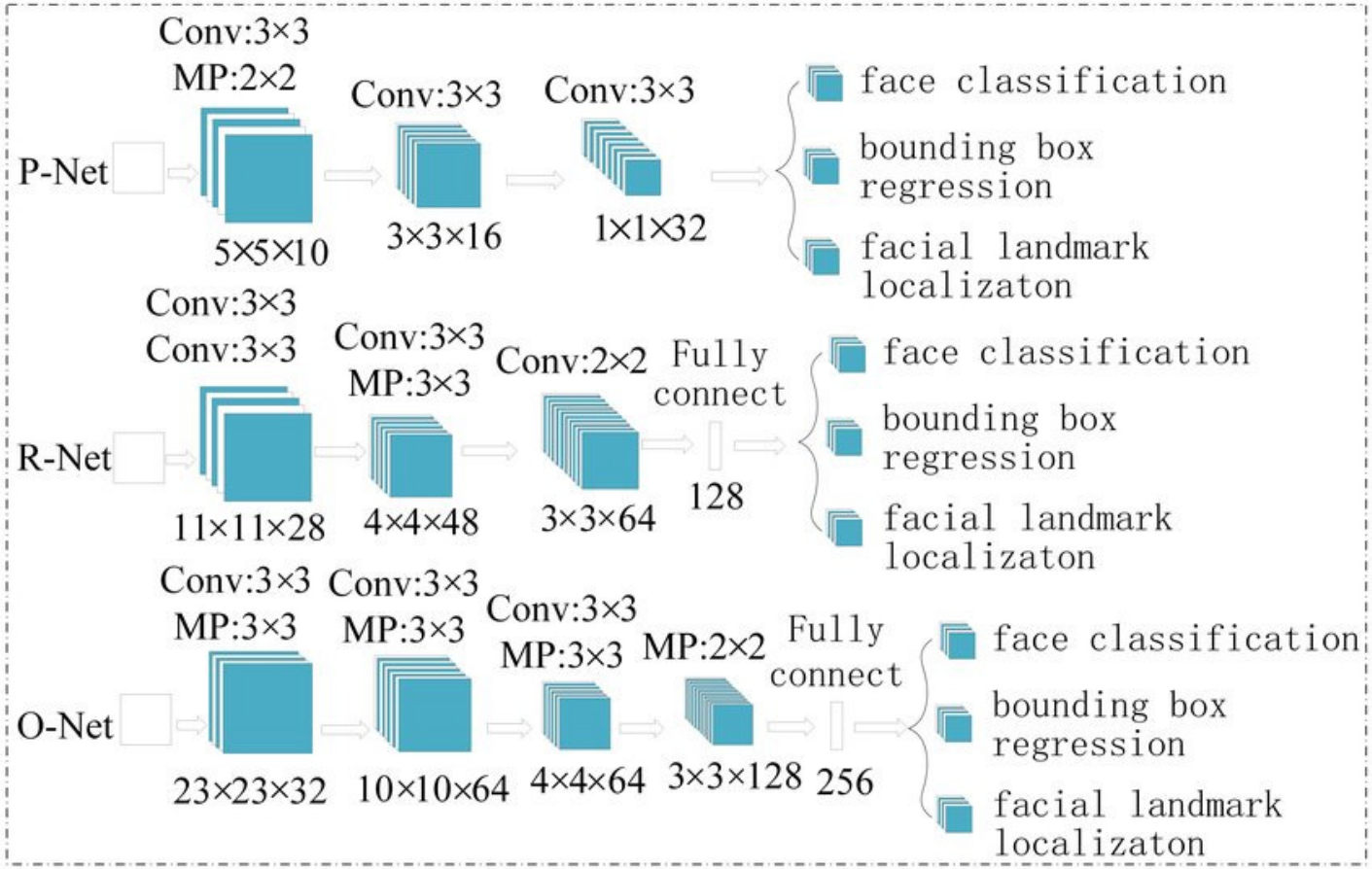



Figure 4.2: MTCNN Architecture Diagram (Haijun, et al., 2021)

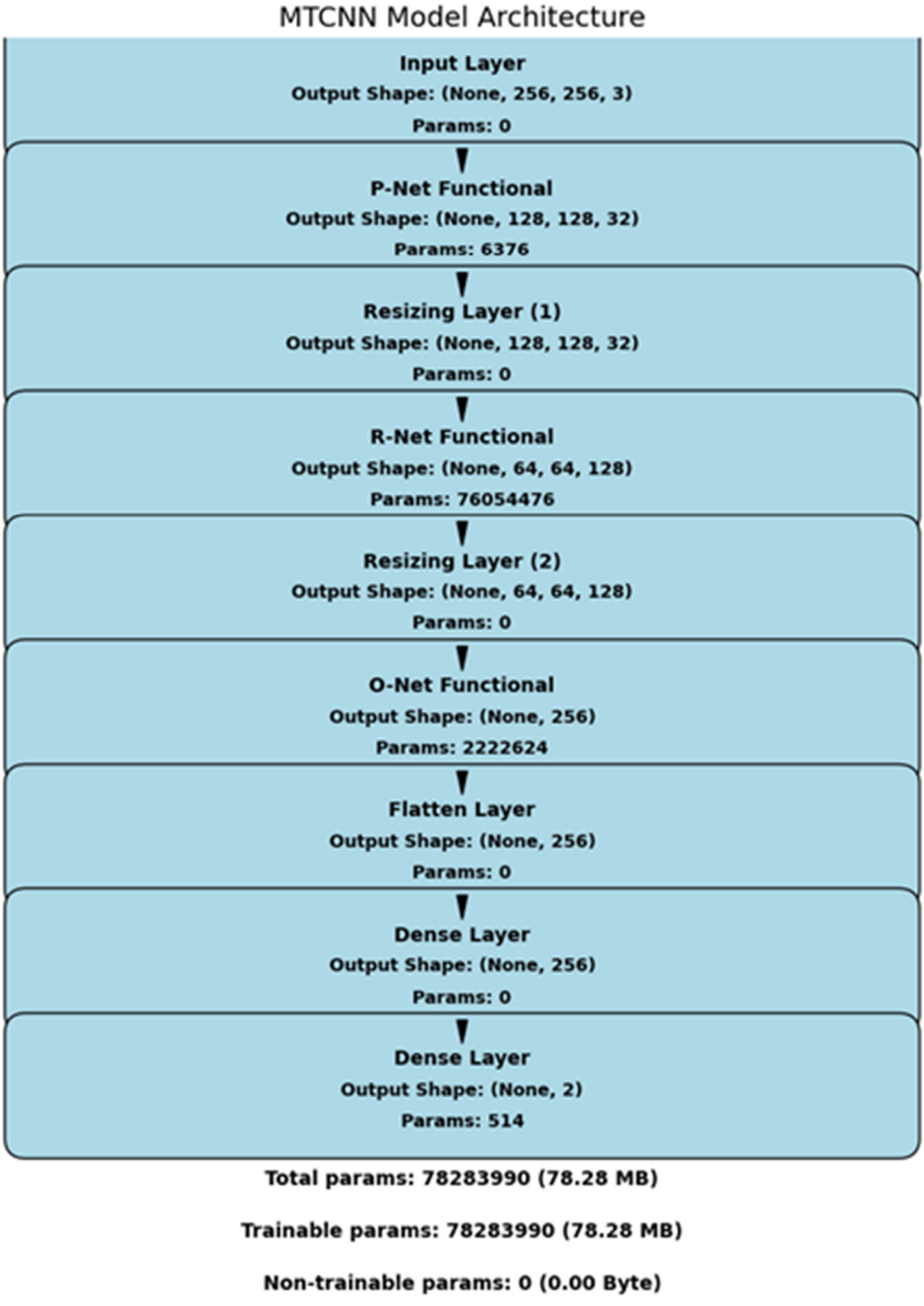



Figure 4.3: MTCNN Model Structure by Author

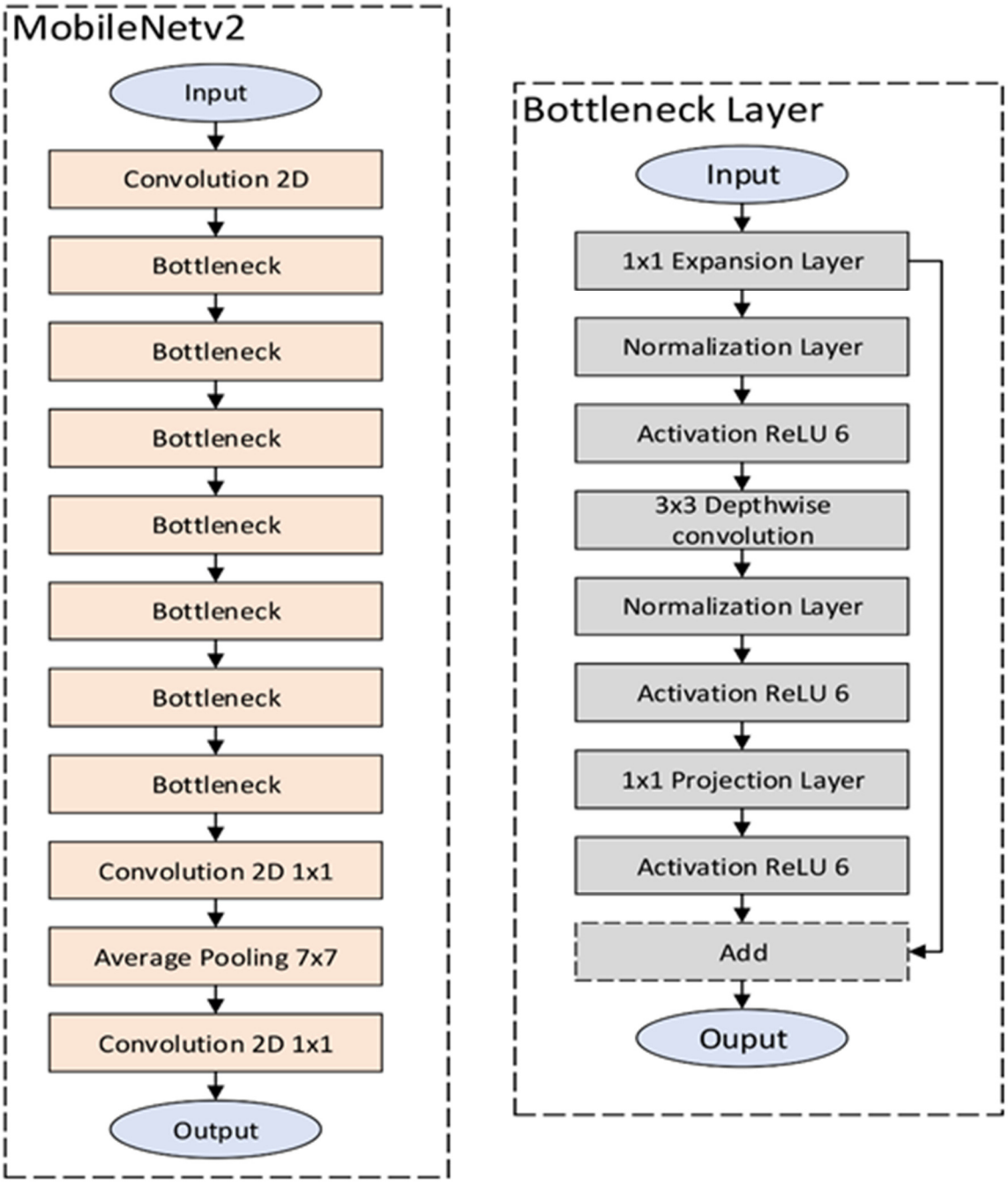



Figure 4.4: MobileNet DNN Architecture (Pavlos , et al., 22)

MobileNetV2 Model:

### 4.3 Evaluation & Examination

#### 4.3.1 Quantitative Examination

The modelling accuracy is assessed quantitatively by evaluating its precision:

Accuracy:

Table 3: Combined Confusion Matrices

| Combined Confusion matrices | | Predicted Positive | Predicted Negative |
|---|---|---|---|
| MTCNN | Actual Positive | 18970 | 1030 |
| | Actual Negative | 1030 | 18970 |
| MTCNN+MobileNetV2 | Actual Positive | 19114 | 886 |
| | Actual Negative | 886 | 19114 |
| MTCNN+MobileNetV2+IFMDT | Actual Positive | 19600 | 400 |
| | Actual Negative | 400 | 19600 |

**Accuracy:**

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{23}$$

**True Positives (TP)**: Number of real classified images correctly identified.

**True Negatives (TN)**: Number of fake classified images correctly identified.

**False Positives (FP)**: Number of real classified images incorrectly identified as real.

**False Negatives (FN)**: Number of fake classified images incorrectly identified as human beings.

1. MTCNN:

$$A = \frac{18970+18970}{18970+1030+18970+1030} = \frac{37940}{40000} = 0.9485$$

2. MTCNN + MobileNetV2:

$$A = \frac{19114 + 19114}{19114 + 886 + 19114} = \frac{38228}{40000} = 0.9557$$

3. MTCNN + MobileNetV2 + MobileNetV2 Intermediate Feature Map Differencing Technique:

$$A = \frac{19600 + 19600}{19600 + 200 + 19600 + 200} = \frac{39200}{40000} = 0.98$$

**Precision:** Precision is the ratio of correctly predicted positive observations to the total number of instances predicted as positive and therefore it is crucial in cases where false positives are significant.

$$P = \frac{TP}{TP + FP} \tag{24}$$

1. MTCNN:

$$P = \frac{18970}{18970+1030} = \frac{18970}{20000} = 0.9485$$

2. MTCNN + MobileNetV2:

$$P = \frac{19114}{19114 + 886} = \frac{19114}{20000} = 0.9557$$

3. MTCNN + MobileNetV2 + MobileNetV2 Intermediate Feature Map Differencing Technique:

$$P = \frac{19600}{19800 + 200} = \frac{19600}{20000} = 0.98$$

**Loss Function:**

Loss function tells us how well or poorly the model's predictions match actual labels in training data; the most commonly used loss function for classification problems is categorical cross entropy loss (also known as sparse categorical cross entropy loss in case of sparse labels). Its mathematical definition is given by,

$$Loss = -\frac{1}{N}\Sigma_{i=1}^{N}\log\left(p_{y_i}\right) \tag{25}$$

Where;

- $\boldsymbol{N}$ is the number of samples.
- $\boldsymbol{y_i}$ is the true label of the$i$-th sample
- $\boldsymbol{p_{y_i}}$ is the predicted probability of true class $y_i$for the $i$-th sample.

The Adam optimizer was employed for all MTCNN, MobileNetV2, and MobileNetV2- Intermediate Feature Map Differencing Technique models, an optimization algorithm with adaptive learning rate specifically designed for training deep neural networks. The default learning rate for Adam is usually 0.001, but it can be adjusted during model compilation.

**Checking accuracy of the model:**

```
scores = model.evaluate(test_ds)
124/124 [==============================] - 46s
18ms/step - loss: 0.1323 - accuracy: 0.9468
scores = model.evaluate(train_ds)
986/986 [==============================] - 18s
18ms/step - loss: 0.1258 - accuracy: 0.9505
import pandas as pd
import matplotlib.pyplot as plt
df = pd.DataFrame(history.history,
columns=history.history.keys())
plt.plot(df)
plt.legend(history.history.keys())
plt.show()
```

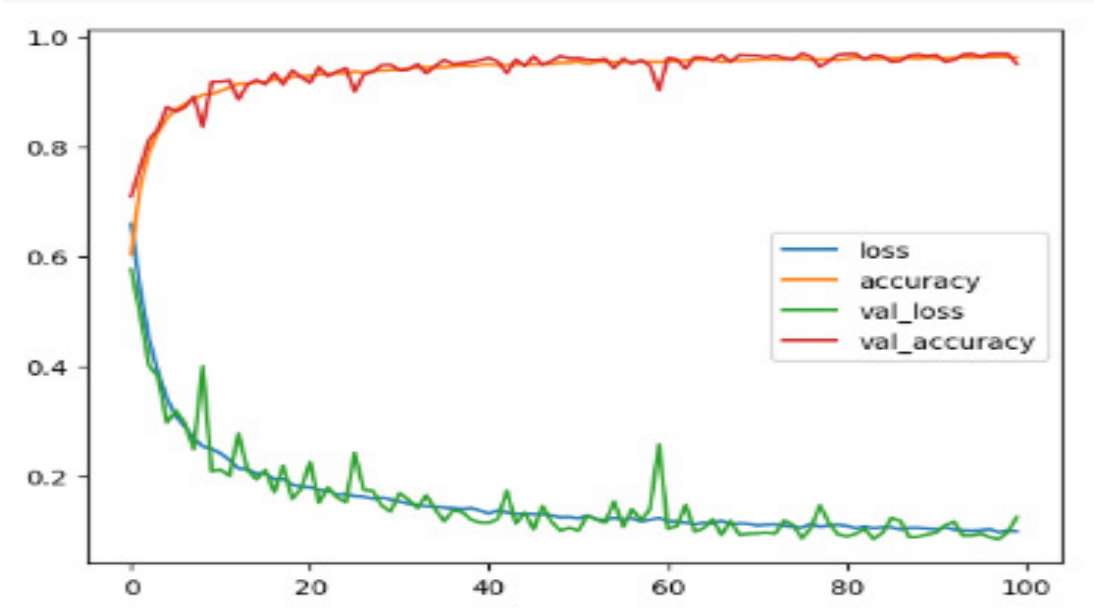



Figure 4.5: Graph of Accuracy, Loss accuracy, Validation accuracy and Loss validation on MTCNN

Here below is a comprehensive table comparing the results of the three methods (MTCNN, MobileNetV2, and MobileNetV2 intermediate feature map differencing technique), including fixed parameters, the number of epochs, training set size, validation set size, and test set size. The dataset consists of 40,000 instances with a split into training (32k), validation (4k) and testing sets (4k).

Table 4:Comprehensive Methods Results

| Metric/Parameter | MTCNN | MTCNN + MobileNetV2 | MTCNN+MNetV2+MNetV2 IFMD Technique |
|---|---|---|---|
| Accuracy(A) | 0.9485 | 0.9557 | 0.98 |
| Epochs | 100 | 80 | 2 |
| Total parameters | 78,283,990 | 2,422,593 | 2,422,593 |
| Trainable parameters | 78,283,990 | 2,388,225 | 2,388,225 |
| Non-Trainable parameters | 0 | 34,368 | 34,368 |
| Model Structure | Functional | Sequential | - |
| Main layers | Input, P-Net, Resizing, R-Net, O-Net, Flatten, Dense | MobileNetV2,GlobalAveragePooling2D,Dense, BatchNormalization, Dropout, Dense | - |
| Training/Validation/Test Split Ratio | 80%/10%/10% | 80%/10%/10% | 80%/10%/10% |

**Testing Protocols:**

**Data Splitting**: The dataset was split into training and validation sets as well as test set with the model not seeing this set during its lifetime i.e., it is not allowed to be used in the training or tuning of the model.

**Cross-Validation**: To ensure that each data point is both trained on and tested by the network one need k-fold cross-validation which provides more comprehensive performance evaluation.

**Balanced Batches**: This implies that for every batch of training data that was fed to the model contained an equal number of instances from each class especially in imbalanced datasets.

**Augmentation and Regularization**: Some of the augmentation techniques used in this example include resizing, rescaling, random flipping and random rotation. This is done to prevent overfitting by making the model concentrate on general patterns other than memorizing training data. Each technique will be discussed below:

1. **Resizing and Rescaling:**

- **Resizing**: This layer resizes all images to a specific size (Image Size x Image Size) using layers. experimental. preprocessing. Resizing layer. The dimensionality of all input images is maintained constant and this is crucial for batch processing at every iteration and consistent input shape in the neural network.
- **Rescaling**: The pixel values of the images are scaled by layers. experimental. preprocessing. Rescaling from [0, 255] to [0, 1]. At times, it helps to converge faster during training because of normalizing step which avoids problems associated with different pixel intensity ranges.
- **Flip Random**: One of these changes is layers. To flip randomly horizontal and vertical objects in such images we use the function random_flip("horizontal_and_vertical"). In this way, the model will not learn to identify objects only when they are mirrored around or laid sideways but also as though they are regular orientations.
- **Rotation Random**: A maximum of 0.2 random rotation flips per layer. This guarantees that a model invariant is achieved with respect to object rotations; thus, making it resistant to variations in object orientation.

**2. Independent Test Set Evaluation**: Finally, models were evaluated on an independent test set which was not used during training or validation phases.

### 4.3.2 Qualitative Analysis

A qualitative analysis was performed to highlight the strengths and weaknesses of each model. When using custom MTCNN model compared to others like MobileNetV2 with highly sophisticated architectures capable of capturing intricate features, understanding feature extraction process as well as decision-making was simpler because of its simplicity. Significantly, the devaluation of fake images played a crucial role in the detection of deepfakes.

**Sample predictions with in Dataset**:

```
for i in range(9):

    ax = plt.subplot(3,3, i+1)

    plt.imshow(images[i].numpy().astype("uint8"))

    predicted_class, confidence = pred(model,
images[i].numpy())

    actual_class = class_names[labels[i]]

    plt.title(f"Actual : {actual_class},\n
Predicted:{predicted_class}.\n
Confidence:{confidence}%")
```


Figure 4.6: predicted real and fake images

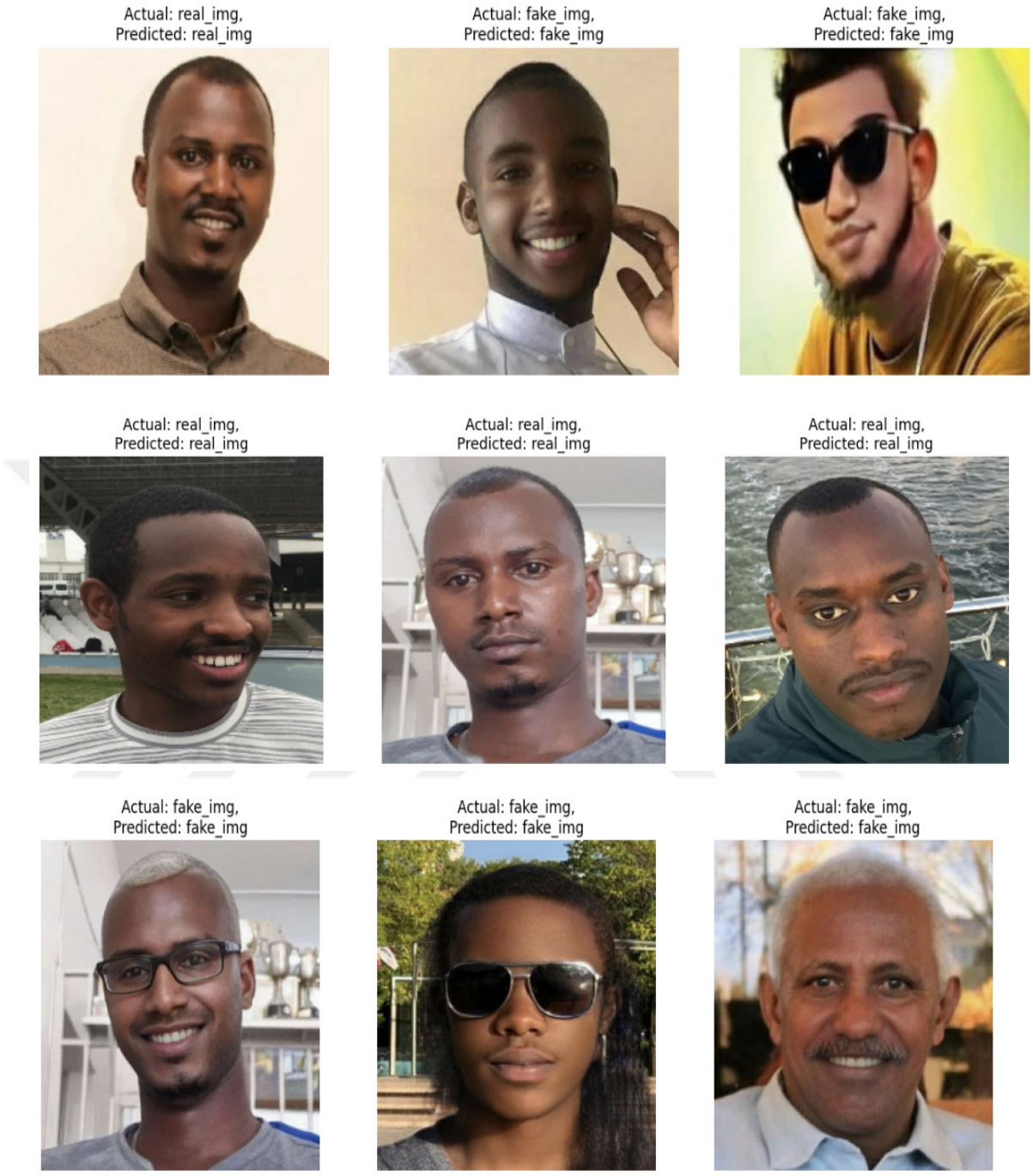



Figure 4.7: predictions with data synthesized by Author

**Summary**

The major breakthroughs, however, were observed using MobileNetV2 Intermediate Feature Map Differencing Technique as depicted in the performance comparison. While such minute manipulations are difficult to detect due to their computational efficiency and scalability issues, this is a novel way of deepfake detection. In conclusion, integrating MTCNN and MobileNetV2 together with feature differencing ensures real-time deepfake detection that is scalable and practical contributing immensely to this field therefore ensuring integrity of visual content across several applications.

# CHAPTER 5

# CONCLUSION, FUTURE WORK

In detail, through reviewing literature comprehensively and developing innovative methodologies as well as detailed experimental evaluations; the thesis has examined deepfake detection landscape. The current methods employed for deepfake detection have been scrutinized and the difficulty they encounter in identifying synthetic media is highlighted. The survey also indicates that classical CNN-based techniques fail when it comes to detecting high-quality counterfeits or computation efficiency.

## Important Contributions and Findings

### Review of Literature

This report finds out about the rapid acceleration of fake videos' technology from previous studies on counterfeit meme detectors. It also demonstrates how Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) can be used for dealing with complexities involved in synthetic media detection. Finally, it discusses how traditional CNN-based approaches do not identify good quality fakes nor do they operate at reasonable speed levels for processing large amounts of data**.**

### Methods Proposed

MTCNN shows how a facial recognition system works by correctly localizing faces on images during the alignment process. For accurate classification using fine-tuned ConvNets like MobileNetV2 we propose these methods: differencing mechanisms including background modeling generate anomaly maps that highlight irregularities indicating forgery in image regions with higher resolution or

sharpness than expected. This methodology helps to address issues related to scaling up while at the same time asses computation efficiency alongside accuracy problems.

1. **MTCNN**: For face localization and alignment, this is used to accurately identify the region of interest.
2. **MobileNetV2**: This is an efficient architecture designed for mobile/embedded environments that can be employed in image classification with high precision.
3. **Feature Differencing Technique**: Within MobileNetV2, a feature differencing technique has been proposed which improves its ability to detect minor anomalies leading to significant increase in accuracy levels.

These methods push forward the study of deepfake detection but at the same time will adapt into practical applications. With slight adjustments such as other digital fakes apart from manipulated voices, it can be applied to fields like social media integrity, bank security or political communication verification. It uses technicality and scientific as key components that yield powerful and scalable solutions in response to current issues encountered within digital content verification processes.

**Data Collection and Pre-processing**

For training purposes there were 20k true images obtained from Kaggle (Zenodo) and 20k false examples of fake images making up a balanced dataset. Additionally, another dataset of 42 images, namely: 21 fake and 21 reals were custom-created to test the model performance further. To enhance diversity of data set along with strength of model against various biases augmentation techniques such as rotation, scaling color adjustment was utilized.

**Experiments and Evaluation**

Most experiments were executed using Google Colab for maximum performance and reproducibility besides local Jupyter Notebook environment. In order to test their robustness across different samples as per cross validation procedures or data splits every model was tested extensively. As a result, utilizing MobileNetV2

Intermediate Feature Map Differencing Technique method resulted into great improvements on accuracy reaching 98% from 94%.

**Implementation and Feasibility**

In this way, the research employed a cost-effective and easily expandable hardware setup with efficient training on GPUs like Tesla K80s or RTX series. The parameters were carefully adjusted to maintain trade-off between model performance and computational efficiency where training went through 100 epochs with a batch size of 32.

**In Conclusion**

Thus, MTCNN integration with feature differencing enhanced MobileNetV2 is a powerful framework for dealing with deepfakes. Therefore, not only does this hybrid approach enhance accuracy but it also deals with major issues related to the efficiency of computational models during identification procedures. Consequently, this research making use of existing methods promotes deep fake detection by ensuring visual integrity over various applications.

Further future studies have to refine these techniques while looking at more data augmentation methods on one side as well as expanding datasets that would capture more diversity in terms of complexities brought about by sample variation. Since deepfake technology is expected to continue advancing there must be detection mechanisms that are capable of outsmarting any such malicious intentions leading to dependable digital media.

## REFERENCES


Jia, R.-S., Mao, Q.-C., Qi, R., Zuo, L.-Q., & Sun, H.-M. (2019, August 12). Face Detection Method Based on Cascaded Convolutional Networks. *Journals & Magazines, 7*, 110740 - 110748. doi:10.1109/ACCESS.2019.2934563

Keith, M., & Javaan Chah, C. (2022). A Review of Synthetic Image Data and Its Use in Computer Vision. *Journal of Imaging*, 5-8.

Barak, H., Sahar, F., Erez, Y., Raja, G., & Alon, B. (2023). Deep learning in optics—a tutorial. *Journal of Optics*, 4-7.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York: Springer.

Carlyon, P. (2023, December 19). *Deepfakes Aren't the Disinformation Threat They're Made Out to Be.* RAND Corporation: https://www.rand.org/blog/2023/12/deepfakes-arent-the-disinformation-threat-theyre-made.html adresinden alındı

contributors, W. (2024, july 30). *Neural network (machine learning)*. Wikipedia: https://en.wikipedia.org/w/index.php?title=Neural_network_(machine_learning)&oldid=1237652724 adresinden alındı

Enkvetchakul, P., & Surinta, O. (2022, June). Effective Data Augmentation and Training Techniques for Improving Deep Learning in Plant Leaf Disease Recognition. *Applied Science and Engineering Progress*, s. 3810-11994. doi:10.14416/j.asep.2021.01.003

Goodfellow, I., J. , P.-a., M, M., B. , D. X., Warde-Farley, S, O., . . . Y, B. (2014). Generative adversarial nets,' Advances in Neural Information Processing Systems. *Advances in Neural Information Processing Systems (NeurIPS)* (s. 2-8). Montreal: NeurIPS Proceedings.

Haijun, L., Liu , C., Cheng, K., Kong, J., Han, Q., & Zhao, T. (2021). Controller Fatigue State Detection Based on ES-DFNN. *aerospace*, 3-5.

Harkiran78. (2023). Artificial Neural Networks and its Applications. *Analytics Vidhya*, 2-12.

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. R. Hecht-Nielsen içinde, *Neural networks for perception* (s. 65–93). Amsterdam: Elsevier.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural networks*, 359–366.

Matthew Stewart, P. (2019, June 17). *Introduction to Neural Networks*. Medium: https://towardsdatascience.com/simple-introduction-to-neural-networks-ac1d7c3d7a2c adresinden alındı

Mirza, M., & Simon Osindero, O. ( 2014 , November 6). *Conditional Generative Adversarial Nets.* arXiv:1411.1784 [cs.LG]: https://arxiv.org/abs/1411.1784 adresinden alındı

O'Shea, K., & Nash, R. (2015, December 2). *An Introduction to Convolutional Neural Networks.* arXiv:1511.08458 [cs.NE]: https://arxiv.org/abs/1511.08458 adresinden alındı

Pavlos , S., Konstantinos , F., Athanasios , T., George , F., Georgios , D., Kostas , K., & Georgios , S. (22). Design Space Exploration of a Sparse MobileNetV2 Using High-Level Synthesis and Sparse Matrix Techniques on FPGAs. *Sensors*, 3-6.

Peng, K., Cao , X., Liu, B., Guo , Y., & Tian, W. (2021, May 24). *Ensemble Empirical Mode Decomposition with Adaptive Noise with Convolution Based Gated Recurrent Neural Network.* Symmetry: https://doi.org/10.3390/sym13060931 adresinden alındı

Sandler, M., Howard, A., Menglong Zhu, M., Zhmoginov, A., & Chen, L.-c. (2019, March 21). *MobileNetV2: Inverted Residuals and Linear Bottlenecks.* arxiv.org: https://arxiv.org/pdf/1801.04381 adresinden alındı

Sandler, M., & Howard, A. (2018, April 3). *MobileNetV2: The Next Generation of On-Device Computer Vision Networks.* Google Research: https://research.google/blog/mobilenetv2-the-next-generation-of-on-device-computer-vision-networks/ adresinden alındı

Viola, P., & M, J. (2001). Rapid object detection using a boosted cascade of simple features. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (s. 511-518). Kauai, HI, USA: IEEE.

Yang, X., & Zhang, W. (2021). Heterogeneous face detection based on multi□task cascaded convolutional neural network. *IET Image Processing*, 3-5.

Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). face detection and alignment using multi-task cascaded convolutional networks. *Conference on Computer Vision and Pattern Recognition (CVPR)* (s. 3451-3459). Las Vegas, NV, USA: IEEE.

Zhang, L., Wang, H., & Chen, Z. (2021). A Multi-task Cascaded Algorithm with Optimized Convolution Neural Network for Face Detection. *Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)* (s. 5-10). Shenyang, China: IEEE.

# APPENDICES

## Appendices A: CODES

```
#Importing libraries:

import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt

#Image size,batch, channels and classes:
Image_Size = 256
Batch_Size = 32
Channels = 3
n_classes = 2

#Determining number of pics and classes:
dataset =
tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/Dataset/imgv_dataset",
    shuffle=True,
    image_size = (Image_Size,Image_Size),
    batch_size=Batch_Size

)
#present class name:
class_names = dataset.class_names
class_names
for image_batch, label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())

#Trying out for the first image:
for image_batch, label_batch in dataset.take(4):
    plt.imshow(image_batch[0].numpy().astype("uint8"))
    plt.title(class_names[label_batch[0]])
    plt.show()
```

**#Creating function for splitting dataset:**

```
def splitting_dataset_tf(ds, train_split=0.8,
val_split=0.1, test_split=0.1, shuffle=True,
shuffle_size=10000):

    ds_size=len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size=int(train_split * ds_size)
    val_size= int(val_split * ds_size)

    train_ds= ds.take(train_size)

    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
train_ds, val_ds, test_ds=splitting_dataset_tf(dataset)
print(len(train_ds),len(val_ds),len(test_ds))
```

**#Caching and prefetching:**

```
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.
data.AUTOTUNE)
val_ds =
val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.da
ta.AUTOTUNE)
test_ds =
test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.d
ata.AUTOTUNE)
```

**#Image Preprocessing for Resizing, Rescaling and Data Augmentation Layers:**

```
resize_and_rescale = tf.keras.Sequential([

layers.experimental.preprocessing.Resizing(Image_Size,I
mage_Size),

layers.experimental.preprocessing.Rescaling(1.0/255)
])

data_aug = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
```

```
    layers.RandomRotation(0.2)
])

#Building MTCNN model architeture:
# Define the R-Net
def build_rnet():
inputs = layers.Input(shape=(Image_Size//2,
Image_Size//2, 32))
x = layers.Conv2D(28, (3, 3), activation='relu',
padding='same')(inputs)
x = layers.MaxPooling2D((3, 3), strides=2,
padding='same')(x)
x = layers.Conv2D(48, (3, 3), activation='relu',
padding='same')(x)
x = layers.MaxPooling2D((3, 3), strides=2,
padding='same')(x)
x = layers.Conv2D(64, (2, 2), activation='relu',
padding='same')(x)
x = layers.Flatten()(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dense((Image_Size//4) * (Image_Size//4) *
1288, activation='relu')(x)
x = layers.Reshape((Image_Size//4, Image_Size//4,
128))(x)
    model = models.Model(inputs, x, name='R-Net')
    return model
# Define the O-Net
def build_onet():
inputs = layers.Input(shape=(Image_Size//4,
Image_Size//4, 128))

x1 = layers.Conv2D(32, (3, 3), activation='relu',
padding='same')(inputs)

x2 = layers.MaxPooling2D((3, 3), strides=2,
padding='same')(x1)

x3 = layers.Conv2D(64, (3, 3), activation='relu',
padding='same')(x2)

x4 = layers.MaxPooling2D((3, 3), strides=2,
padding='same')(x3)

x5 = layers.Conv2D(64, (3, 3), activation='relu',
padding='same')(x4)
```

```
x6 = layers.MaxPooling2D((2, 2), strides=2,
padding='same')(x5)
# Main function to run MTCNN
def mtcnn():
    pnet = build_pnet()
    rnet = build_rnet()
    onet = build_onet()

    inputs = layers.Input(shape=(Image_Size,
Image_Size, Channels))
    pnet_output = pnet(inputs)
    rnet_input = layers.Resizing(Image_Size//2,
Image_Size//2)(pnet_output)
    rnet_output = rnet(rnet_input)
    onet_input = layers.Resizing(Image_Size//4,
Image_Size//4)(rnet_output)
    onet_output = onet(onet_input)

    x = layers.Flatten()(onet_output)
    output = layers.Dense(n_classes,
activation='softmax')(x)
    model = models.Model(inputs, output, name='MTCNN')

    return model

# Build and compile the MTCNN model
mtcnn_model = mtcnn()
mtcnn_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Display model summary
mtcnn_model.summary()
x7 = layers.Conv2D(128, (2, 2), activation='relu',
padding='same')(x6)
x8 = layers.Flatten()(x7)
x = layers.Dense(256, activation='relu')(x)
    model = models.Model(inputs, x, name='O-Net')
    return model
```

**#Model Training with Training and Validation Data:**

```
#Define early stopping callback
```

```
callback =
tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=5, restore_best_weights=True)

#Train the model
history = model.fit(
    train_ds,
    epochs=100,
    batch_size=Batch_Size,
    verbose=1,
    validation_data=val_ds
)
#Checking accuracy of the model:
scores = model.evaluate(test_ds)
scores = model.evaluate(train_ds)
import pandas as pd
import matplotlib.pyplot as plt

df = pd.DataFrame(history.history,
columns=history.history.keys())
plt.plot(df)
plt.legend(history.history.keys())
plt.show()

#MobileNetV2:
mobilenet =
tf.keras.applications.mobilenet_v2.MobileNetV2
model_arch =mobilenet()
model_arch.summary()
from tensorflow.keras.layers import
GlobalAveragePooling2D, Dense, BatchNormalization,
Dropout
from tensorflow.keras.optimizers import Adam
mobilenet =
tf.keras.applications.mobilenet_v2.MobileNetV2(input_sh
ape=(224, 224, 3),
                        include_top=False,
                        weights='imagenet'
                       )

model = tf.keras.models.Sequential([mobilenet,

GlobalAveragePooling2D(),
                                    Dense(128,
activation='relu'),
```

```
BatchNormalization(),
                                    Dropout(0.3),
                                    Dense(1,
activation='sigmoid')
                                   ])
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
callback =
tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=5, restore_best_weights=True)

hist = model.fit(x=train_ds, epochs=80,
validation_data=val_ds, callbacks=[callback],
verbose=2)

#Loading MTCNN and MobileNetV2:
model_path='/content/drive/MyDrive/mtv2_model.keras'
model=tf.keras.models.load_model(model_path)

#Evaluation:
scores = model.evaluate(test_ds)

MTCNN,MobileNetV2 and MobileNetV2 Intermediate Feature
Map Differencing Technique:
tf.keras.backend.clear_session()
model_path='/content/drive/MyDrive/trial_model.keras'
model=tf.keras.models.load_model(model_path)
hist = model.fit(x=val_ds, epochs=2, verbose=2)

#Predicting Labels for a Batch of Images:
import numpy as np

for image_batch, label_batch in dataset.take(1):

    first_image =
image_batch[0].numpy().astype('uint8')
    first_label = label_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("Actual label : ",class_names[first_label])
```

```
    batch_pred = model.predict(image_batch)
    print("Pred label :
",class_names[np.argmax(batch_pred[0])])
```

**Image Prediction Function Using the Model:**

```
def pred(model, img):
    img_array =
tf.keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class =
class_names[np.argmax(predictions[0])]
    confidence = round(100 * np.max(predictions[0]), 2)
    return predicted_class, confidence

plt.figure(figsize=(15, 15))

for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3,3, i+1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = pred(model,
images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual : {actual_class},\n
Predicted:{predicted_class}.\n
Confidence:{confidence}%")

        plt.axis("off")
```

**#Test on new Dataset:**

```
dataset =
tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/mydataset2",
    shuffle=True,
    image_size = (Image_Size,Image_Size),
    batch_size=Batch_Size

)
plt.figure(figsize=(15, 15))
```

```
for images, labels in dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class = pred(model,
images[i].numpy())[0]
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\nPredicted:
{predicted_class}")

        plt.axis("off")
```