DEEP-FAKE FORENSICS CHALLENGE

2023-2024

Presented By :

SFAR ADAM

MZOUGHI YASMINE

HORCHANI JIHED

KCHAOU ONS

FRINI FARES

HADDAD AHMED

TABLE OF CONTENT

I. INTRODUCTION

1.IDEOLOGY :

In our social media-driven lives, images hold a central role, but they are increasingly vulnerable to manipulation. The ease of swapping or altering identities, facilitated by software editing tools (shallowfakes) and AI-driven technology (deepfakes), poses a substantial threat.

This threat extends beyond mere deception, potentially harming reputations and influencing public opinion. Addressing this issue requires vigilance in recognizing manipulated content and fostering awareness about the potential consequences in our society.

Figure 1 : Face Recognition pattern

2. IMAGE MANIPULATION TOOLS :

Shallowfakes , refer to manipulated media content created through basic or surface-level editing techniques, such as pixel-level alterations, splicing, or straightforward modifications using traditional image or video editing tools**.**

Deepfake, involve the application of deep learning techniques, particularly generative neural networks, to create highly realistic and often deceptive media content. These algorithms analyze and synthesize visual or auditory data, producing sophisticated fabrications that can be challenging to distinguish from genuine content.

II . UNDERSTANDING PROPOSED RESEARCH PAPER :

1.USED DATASET : :

The Shallowfakes Dataset In our quest for a comprehensive training dataset, we turned to CASIAv2, which provided us with a trove of 7,490 genuine images and 4,948 manipulated images. These manipulated images exhibited slicing and copy-move manipulations, enriching our dataset with diverse alterations. Upon segmentation, the training, validation, and test sets emerged in an 8:1:1 ratio, culminating in a training set of 5,992 authentic images and 3,958 manipulated images. Our test dataset comprised images drawn from CASIAv1, Columbia, COVERAGE, and NIST16, encapsulating 1,832 genuine and 2,259 manipulated images, showcasing a spectrum of alterations encompassing slicing, copy-move, and inpainting.

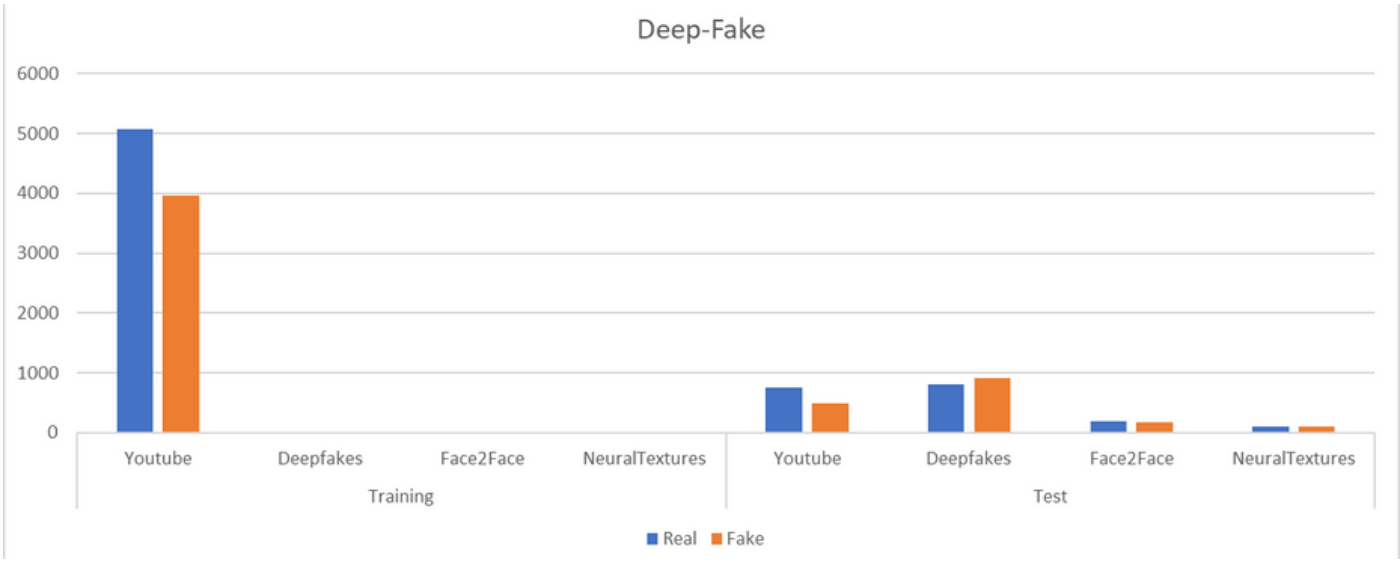

Figure2:Bar Graphof different deepfake datasets

The Deepfakes Dataset Our approach to the deepfakes dataset drew from the expansive FaceForensics++.Employing this resource, we integrated authentic videos from YouTube, Google's DeepFakeDetection dataset, and synthetic videos generated through automated face manipulation methodologies. From this integration, we precisely extracted image frames, carefully balancing the count of authentic and manipulated frames, resulting in 8,449 genuine and 7,330 manipulated frames, mirroring the characteristics of the shallowfakes dataset. These frames were then meticulously divided into training, validation, and test sets, allocating a ratio of 6:2:2 for authentic frames and 5:2:3 for manipulated frames.

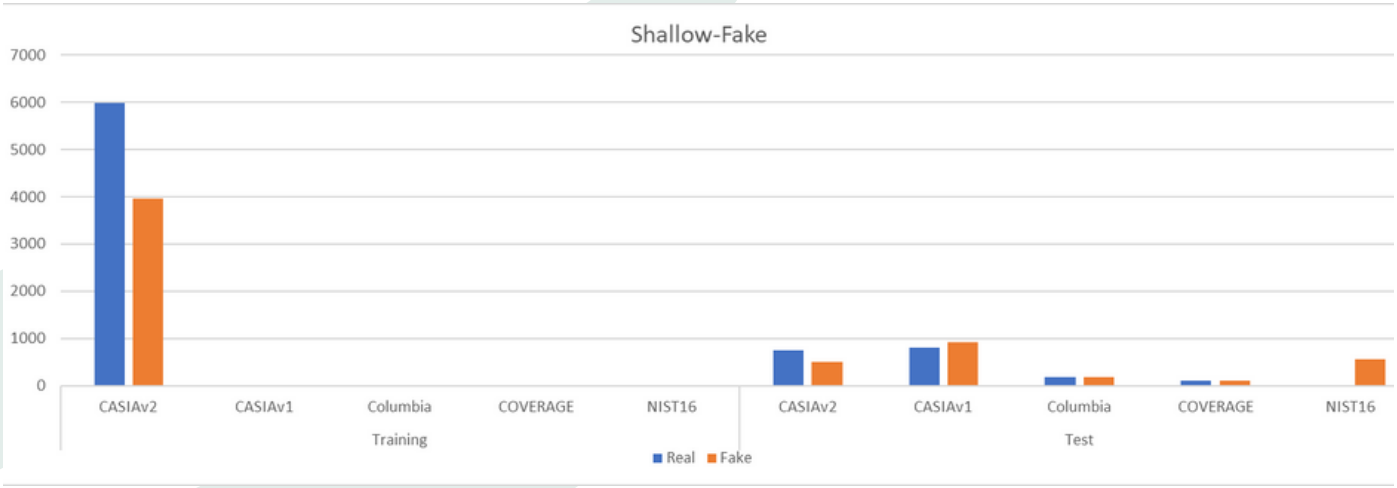

Figure2:Bar Graph of different shallowfake datasets

2.USED ARCHITECTURE :

Researchers utilized the UPerNet, a multi-task framework within CNN architecture for semantic segmentation. UPerNet optimizes segmentation accuracy by refining initial results with a boundary optimization module. The architecture combines encoder-decoder features with spatial paths and lateral connections for improved performance. Our understanding of their work began with a breakdown of each step

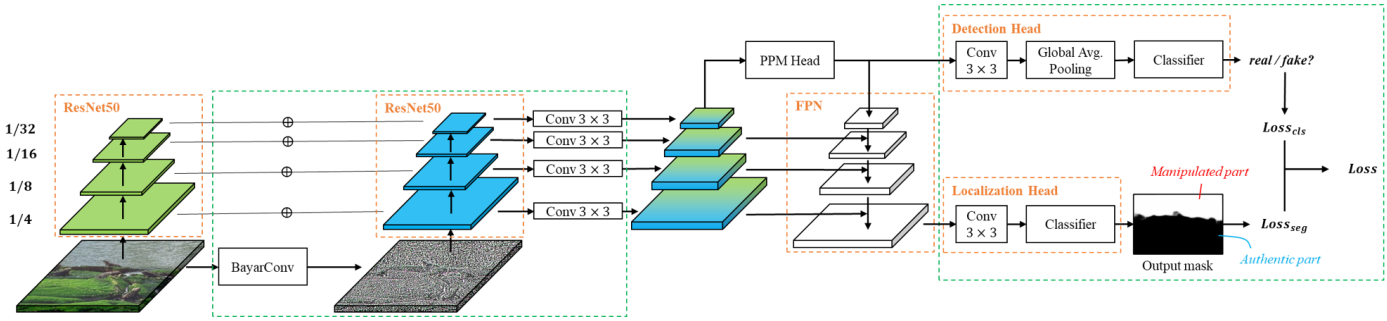

Figure3:The architecture of each model

ResNet-50 :

We snippet demonstrates a practical example of using a pre-trained ResNet-50 model for image classification. We provides a clear understanding of the workflow, from model loading and image preparation to inference and results interpretation.

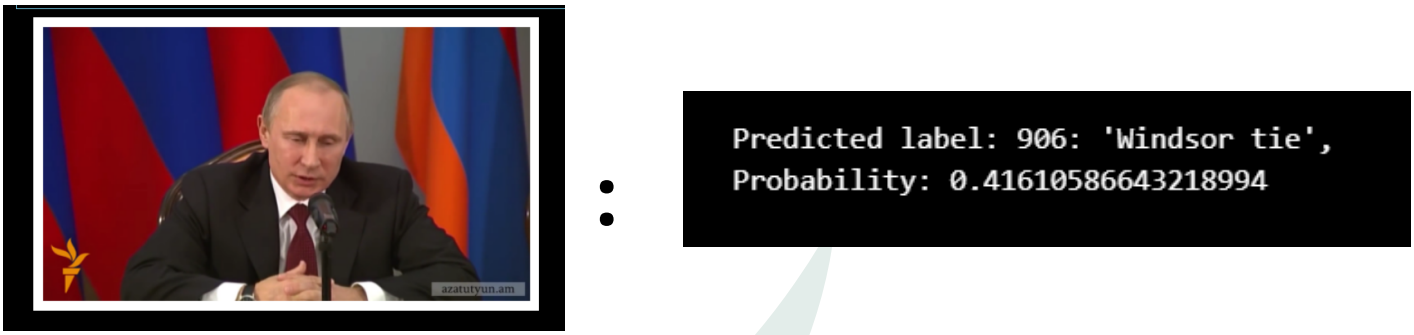

Figure 5:Resnet50 prediction

Color Capture with BayerConv:

Using BayerConv, the researchers efficiently extracted color details by incorporating Bayer filters and learnable high-pass filters. These filters not only analyze colors but also reveal noise distribution differences, aiding in the detection of potential forgeries

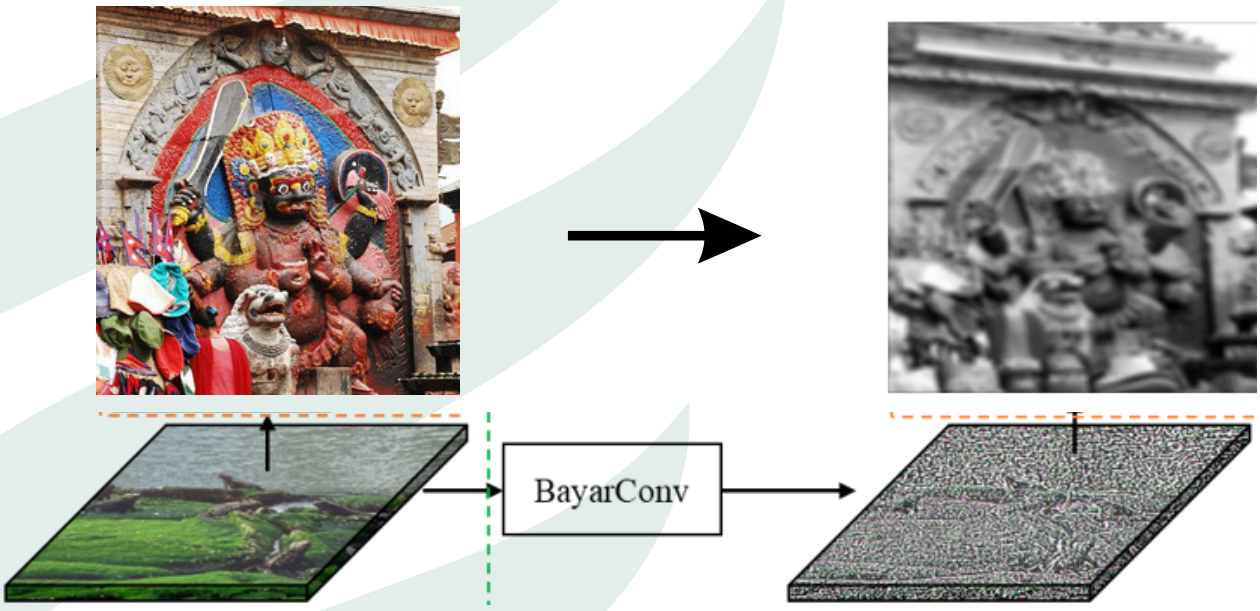

Figure6:Manipulated image before and after BayerConv Model

We load and preprocesses an image, define a BayarConv2d object, passes the image through the layer, converts and visualizes a specific channel of the output, and prints the output shape

Conv3x3 Function:

This function returns a 3x3 convolutional layer with specified input and output channels, stride, and padding. It's used for creating convolutional layers in the Bottleneck module.

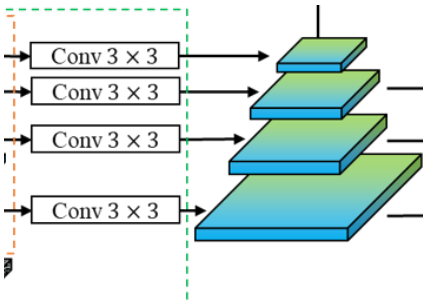

Figure7:Conv3x3 Architecture

The Bottleneck module :

consists of three convolutional layers with batch normalization and a residual connection, facilitating efficient model training. The script includes image preprocessing using PyTorch transforms, loading an image, and transforming it for input. An instance of the Bottleneck module is created, and a forward pass is performed on a preprocessed image. The shapes of input and output tensors are printed, and the input and output images are visualized using matplotlib.

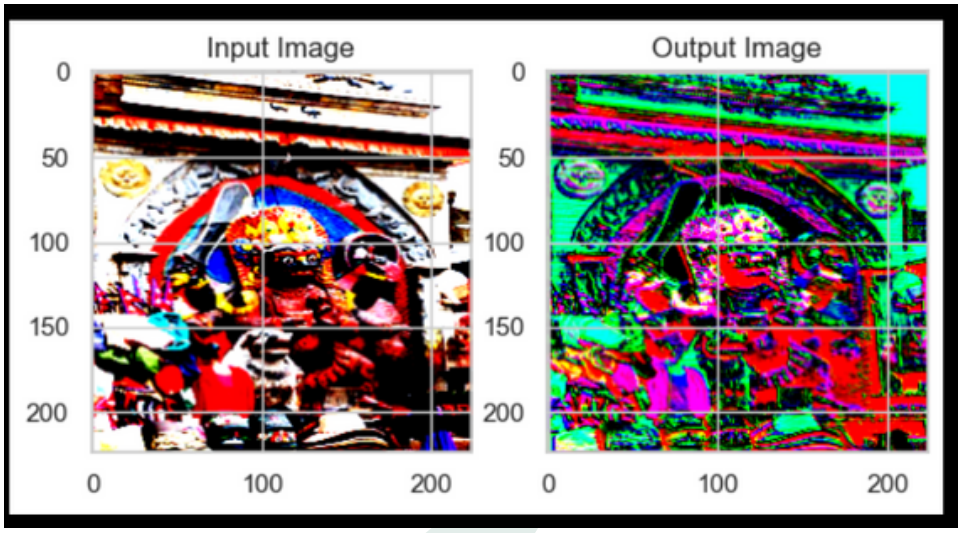

Figure8:Bottleneck module input and output image

PPM (Pyramid Pooling Module):

The Pyramid Pooling Module is a component used in convolutional neural networks (CNNs) for semantic segmentation tasks. It was introduced in the PSPNet (Pyramid Scene Parsing Network) architecture. The goal of PPM is to capture multi-scale information from different levels of a feature hierarchy. It does this by pooling features at different pyramid scales and then concatenating them to form a comprehensive representation.

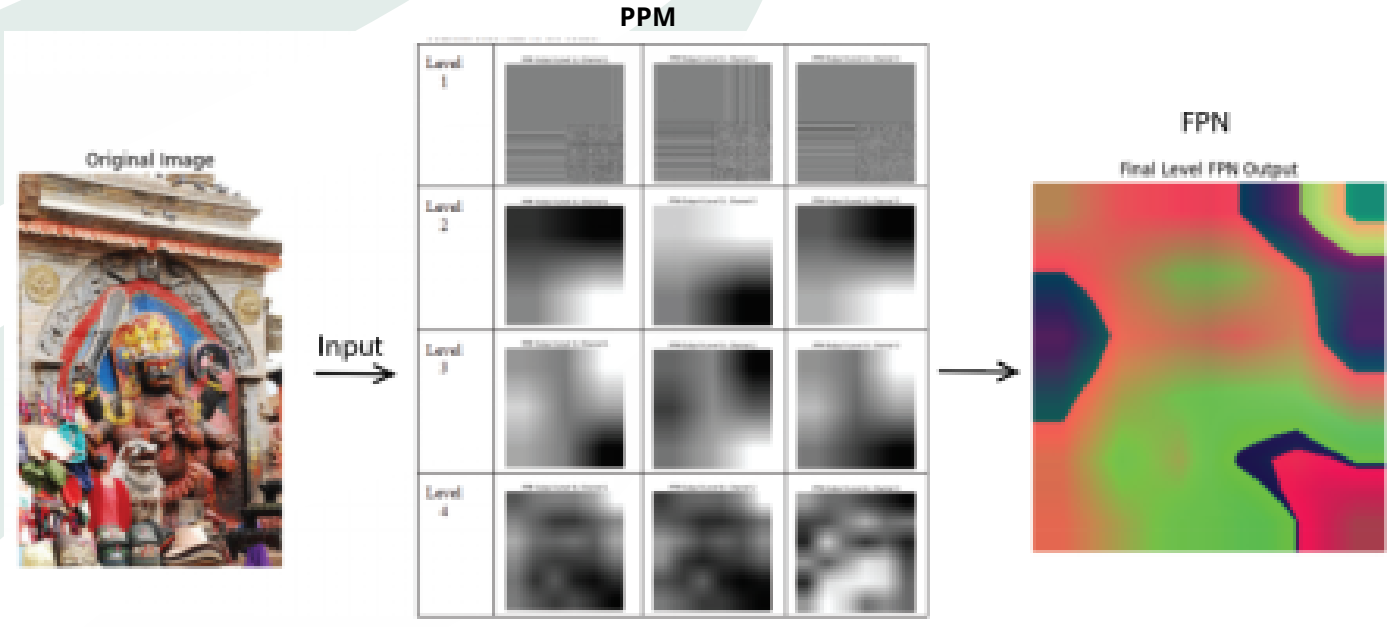

Figure9:PPM and FPN modules input and output

FPN (Feature Pyramid Network):

Feature Pyramid Network is another architecture designed to address challenges related to object detection and recognition at different scales. FPN was introduced to improve the ability of object detectors to handle objects of varying sizes in an image.

FPN enhances the feature hierarchy of a CNN by adding a top-down pathway and lateral connections. This allows the network to reuse features from different levels of abstraction, helping in detecting objects at multiple scales. FPN is commonly used in two-stage object detection frameworks, such as Faster R-CNN, to improve the handling of objects at different resolutions.

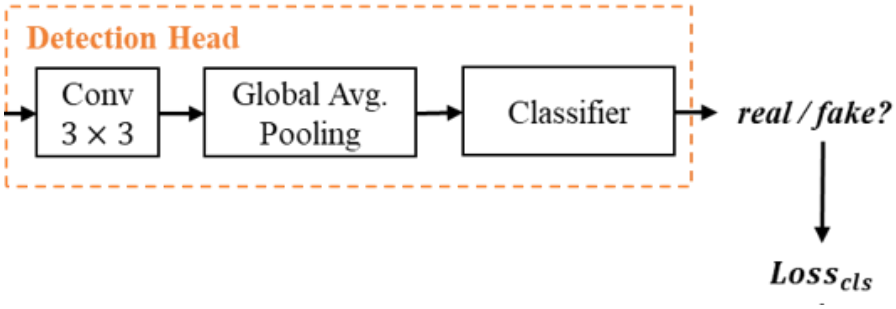

Figure10:Architecture of detection head

The detection head :

employs a Pyramid Pooling Module (PPM) to process high-resolution information, capturing global context and context-aware features. Operating between the encoder and decoder, the PPM aggregates details across varying scales. In the encoder phase, feature maps are condensed through maximum pooling, forming a single feature map that combines local details and broader context.

The decoder then progressively upsamples the map, enhancing pixel-level details. The decoder's output, either a "segmentation mask" or "Bounding Boxes," refines pixel-level information, contributing to the effectiveness of the Pyramid Pooling Module. The detection head outputs a binary result, indicating whether the entire image is "real" or "fake."

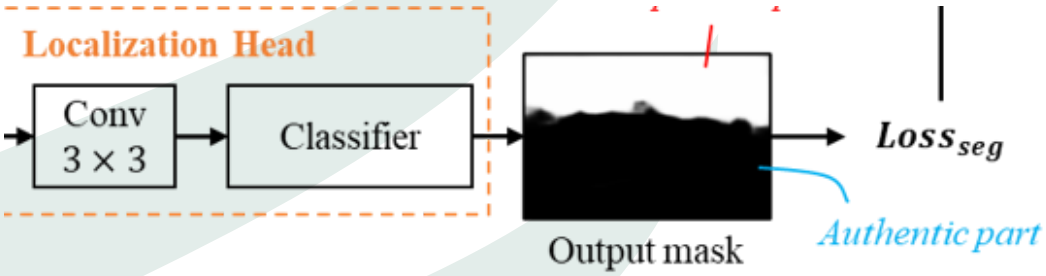

Figure11:Architecture of localization head

Localization Head :

is incorporated to identify manipulated areas. The Feature Pyramid Network (FPN) refines features at various scales, utilizing a pre-trained convolutional neural network (backbone), such as ResNet. The FPN creates a feature pyramid that preserves fine details and includes semantic context. The Localization Head analyzes these features to produce a mask highlighting forged regions, offering a visual representation of detected forgery. This collaborative approach ensures a comprehensive understanding of manipulated areas by capturing specific details at different scales in the feature pyramid.

Loss function :

$$L = \alpha \cdot Loss_{clf} + \beta \cdot Loss_{seg}$$

The final loss function of our network comprises two components: at the image level, we utilize binary cross-entropy (BCE) to ensure accurate binary detection results ($Losscl f$), while at the pixel level, Dice loss ($Losss$eg) is employed to measure the overlap between the generated mask and the ground truth, enhancing mask correctness
The network's final loss combines binary cross-entropy (BCE) for image-level detection and Dice loss ($Losss$eg) for pixel-level mask accuracy. It is a weighted sum ($\alpha \cdot Losscl f + \beta \cdot Losss$eg) with α = 0.04 and β = 0.16, striking a balance determined empirically. Edge loss from the original MVSS-Net is omitted as unnecessary.

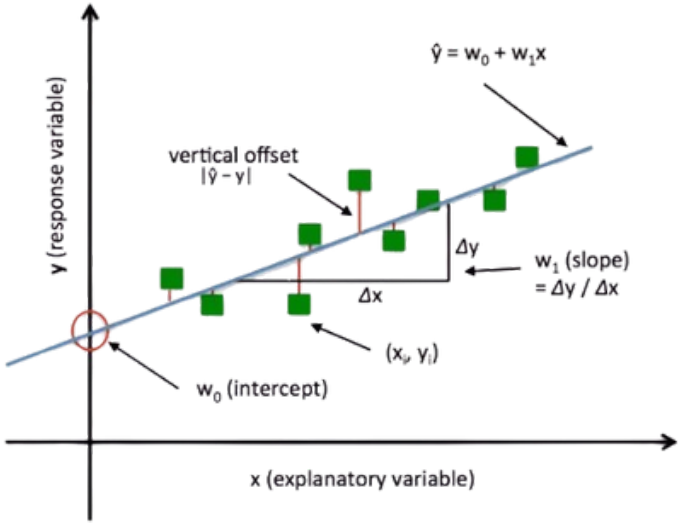

Figure13:The curve of the response variable in relation to the explanatory variable

III . METHODOLOGY :

1 - WORKING ENVIRONMENT

•Google Colaboratory:

The use of Google Colab brought several advantages, including access to powerful computing resources without the need for extensive local hardware setup. Colab's built-in features for paralle processing and dependency management streamlined the development process, accelerating the iteration phase.

Python:

-Chosen as the programming language, Python was a natural fit due to its clear syntax, flexibility and extensive library support. The seamless integration between Google Colab and popular Pythor libraries such as NumPy, Pandas.
.-It is crucial to emphasize that our decision to work on both a global and national (Tunisia) scale demonstrates our intention to gain a comprehensive perspective, encompassing influences from both global and country-specific contexts.

2 - DATA VISUALIZATION AND EXPLORATION :

We'll explore the realm of image Visualization and Manipulation Techniques to detect Shallowfakes and Deepfakes. For Shallowfakes, we'll focus on understanding and exploring the data, followed by employing various processing and manipulation techniques. Similarly, for Deepfakes, our journey involves delving into data understanding, exploring, and employing specific processing and manipulation techniques.

Importing libraries :

We import all the libraries needed to run our code successfully

We had to install others

```
import pandas as pd
import matplotlib as plt
from glob import glob
import numpy as np
import cv2
import os
import matplotlib.image as mpimg
import seaborn as sns
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torchvision.transforms as transforms
```

Figure14: Code of the imported Libraries

```
Shallow_data.shape

(9949, 4)
```

Figure15:Number of records and features used

The dataset contains a total of 9949 records and 4 features

```
    Shallow_data.info()
    Shallow_data.describe()
86]

<class 'pandas.core.frame.DataFrame'>
Index: 9949 entries, 0 to 9948
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   image_path  9949 non-null   object
 1   mask_path   3958 non-null   object
 2   edge_path   3958 non-null   object
 3   Real/Fake   9949 non-null   int64
dtypes: int64(1), object(3)
memory usage: 388.6+ KB
```

Figure16:Dataset's path

The dataset comprises four columns: "image_path," containing 9949 non-null entries indicating the file paths for images; "mask_path" and "edge_path," each with 3958 non-null entries, representing file paths for masks and edges, respectively; and "Real/Fake," a column with 9949 non-null entries consisting of integer values (0 or 1) indicating the authenticity of the corresponding images (0 for real, 1 for fake).

Figure17:Bar Graph comparing number of fake and real images within shallowfake dataset

--> We identified that our dataset contains more Real images than Fake ones

Figure18:Bar Graph comparing number of fake and real images within DeepFake dataset

--> We identified that our dataset contains more Real images than Fake ones

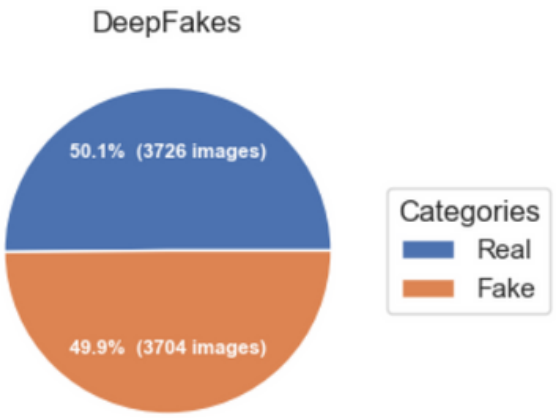

Figure20:Pie Chart comparing number of fake and real images within deepfake dataset

-->This emphasizes the proportion of real images compared to fake ones

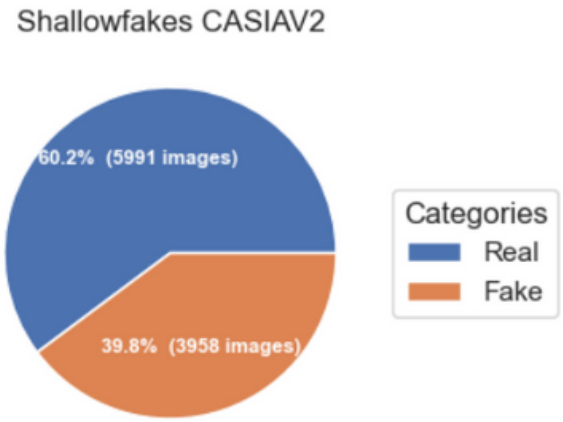

Figure19:Pie Chart comparing number of fake and real images within shallowfake dataset

-->We visualize the proportions of real and fake images using a “PIE Chart”. It combines the number of real and fake images into a single list, creates pie chart slices with percentages and absolute values displayed, adds a legend for the categories, and sets the title to “DeepFakes". This provides a clear and informative way to understand the distribution of the data

3 - DATA PREPARATION :

Mounting the drive represents a critical step in maximizing efficiency. This process empowers immediate access and utilization of existing data directly from the source, bypassing the need for repetitive downloads. This not only saves valuable time but also streamlines workflows, ensuring seamless and readily available access to required data whenever needed. By eliminating the download bottleneck, we unlock a world of enhanced productivity and data exploration.

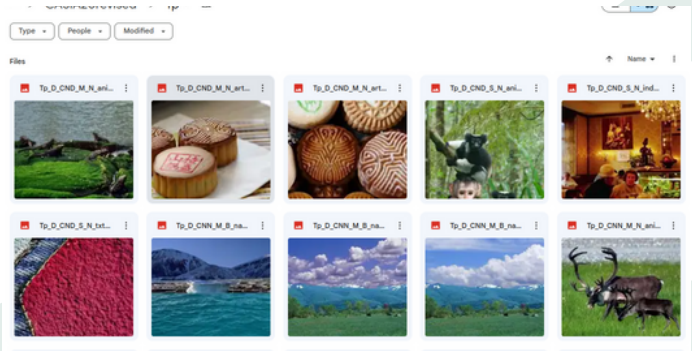

Figure21:Casiav1 revised dataset’s folder

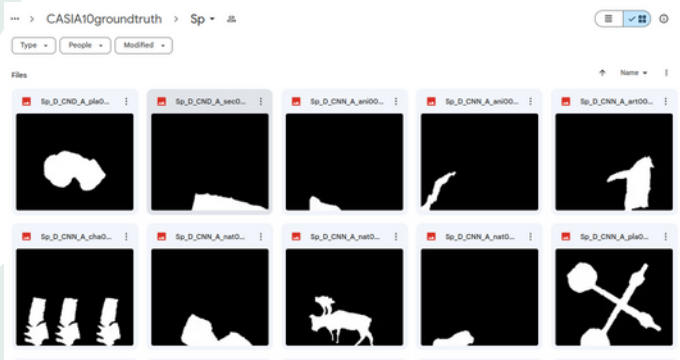

Figure22:masked dataset’s folder

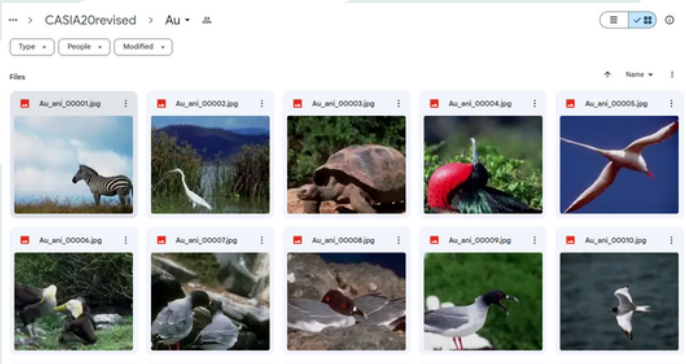

Figure23:Casia20 revised dataset’s folder

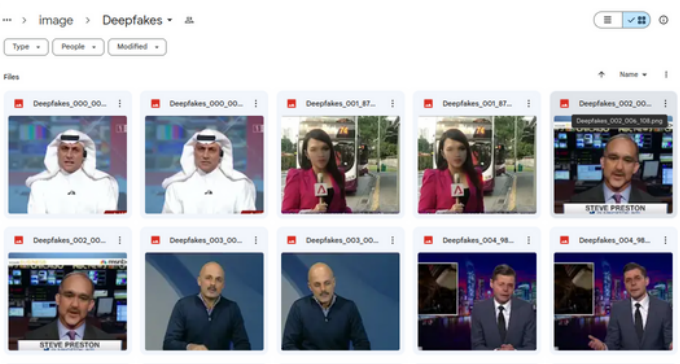

Figure24:Deepfake dataset’s folder

Following the successful mounting of the drive, we proceed to set the working directory. Subsequently, we commence the process of constructing and downloading the required data.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
%cd /content/drive/MyDrive
```

Figure25:Importing different paths to the code

We clone the main project from github and we create new directory called data to put on it the required dataset used for both training and evaluation in same time

```
!git clone https://github.com/zjbthomas/ShallowDeepFakesLocalization.git
```
Python

Figure26:Cloning the main project from github

II - Downloading Required Datasets

Deepfakes

We use wget (linux command-line) to rapidly and directly download data in our working directory

```
!wget https://www.dropbox.com/s/o5410tl5v4vxsth/ICNC2023-Deepfakes.tar.xz
```
Python

```
import tarfile
import lzma

file_path = 'ICNC2023-Deepfakes.tar.xz'

with lzma.open(file_path, 'rb') as file:
    with tarfile.open(fileobj=file, mode='r') as tar:
        tar.extractall()
```
Python

Figure27:Download the datasets (Casia v1 and Casia v2)

- CASIA V2 is hosted on Google Drive, so we utilize the gdown command to swiftly download the images and then use unzip to extract them
- CASIA V1 is accessible in a GitHub repository, and we clone the directory before extracting its contents into our designated drive directory.

For COLUMBIA , we used the rarfile library to extract our ".rar" file .

COLUMBIA

```
!wget https://www.dropbox.com/s/bo10et4p1zg08aj/ImSpliceDataset.rar
```
Python

```
import rarfile

# Specify the path to the RAR file
rar_file = "ImSpliceDataset.rar"

# Open the RAR file in read mode
with rarfile.RarFile(rar_file, "r") as rar:
    # Extract all files to the current directory
    rar.extractall()

# Print completion message
print("Files extracted successfully!")
```
Python

Figure28:Downloading Columbia dataset s via rarfile extraction

- For COVERAGE, We manually downloaded the data and then uploaded it to the designated drive directory .
- For NIST16 , it is only dedicated for model testing.
- Now after we downloaded all the required datasets, our Drive folder looks like this :

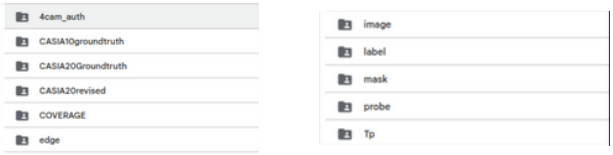

Figure29:Drive folder

Updating Paths :

The subsequent task involves updating the image paths in train_paths.txt, val_paths.txt, and test_paths.txt from the provided paths.zip file, enabling us to proceed with training the model.

Environments Preparation :

In this section, our focus is on setting up our working environments by installing the necessary libraries and packages. This step is crucial due to the absence of documentation and a requirements of .txt file outlining the specific environment configurations within the main project .

```
!pip install -q torch torchvision torchdata Ninja
```

Figure30:Installing torchdata Ninja

4 - REPRODUCED RESULTS : :

- Before training, a crucial difference was identified: the original study utilized a powerful GPU cluster comprising four 32 GB NVIDIA V100 Volta GPUs.
-
- This presents a challenge for replicating results with potentially less powerful hardware. We are actively exploring solutions, such as finding similar hardware, scaling down the model, or using alternative training methods, to ensure faithful replication of the original research.
-
- Therefore, we must devise a solution that neither involves altering or reducing the datasets nor modifying the architecture. Any modifications will not contribute to achieving the desired results, HOW !

-USING COLLAB PRO -

Training :

- -> We're harnessing the power of Colab Pro. With the acquisition of Colab Pro and access to a robust A100 instance, we are poised to initiate the training process. As part of this transition, we modify the code, shifting it from execution in a distributed system to running on a single GPU.

Start Training

we start training by executing this cell :

```
!python -u train_torch.py --paths_file Data_Preparation/dataset_paths/paths_train.txt --val_paths_file Data_Preparation/dataset_paths/paths_val.txt --model ours
```

Python

Figure31:Code execution

```
Epoch 0:   0%|                                        | 7/5527 [00:34<7:23:00,
```

Figure32:Remaining time for the 1st epoch running

- This will takes ages to complete training and Colab prob Ressources is not enough so we need to move on to GPUs Cluster
- In this section, we will proceed with training on a cluster instance equipped with 8 Tesla V100 16GB GPUs.

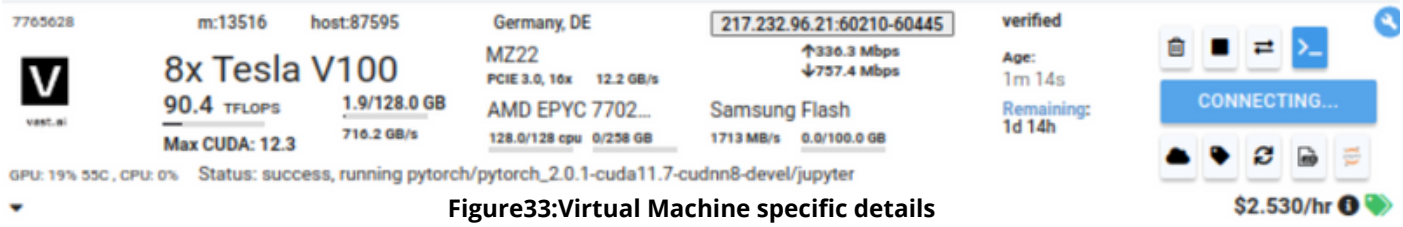

Figure33:Virtual Machine specific details

Failed Attempt !

GPUs Cluster :

We have leased this cluster from “vast.ai” to facilitate model training and enhancement. We connect drive to it to copy our ready to use Environement. This instance give us an access to Jupyter Server and Terminal As soon as we finished envirements setup we start training

Nvidia-smi :

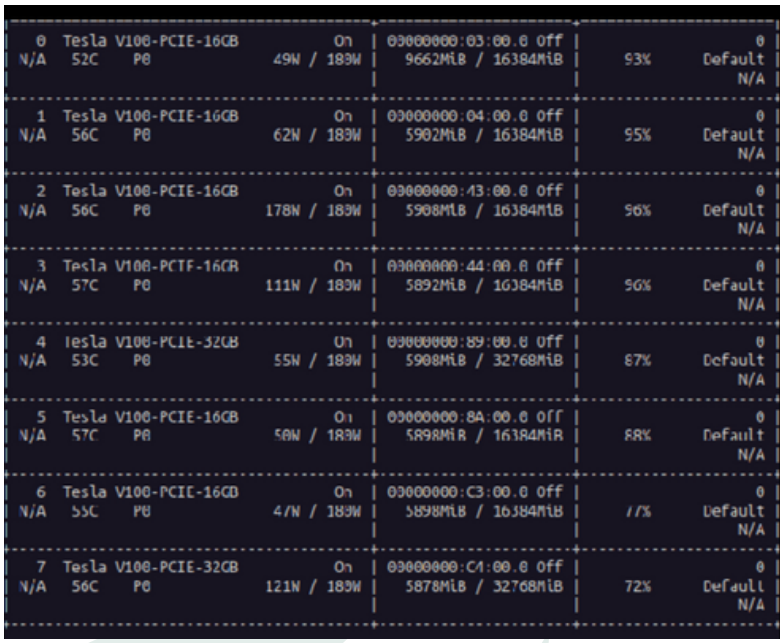

Figure34:Different GPUsin the machines

Starting :

After meticulously setting up the environment, including the Jupyter server and terminal access, we initiated the training process. While we successfully trained the model for a significant portion of the initial phase, reaching approximately 200 epochs, the process stopped prematurely due to resource limitations.

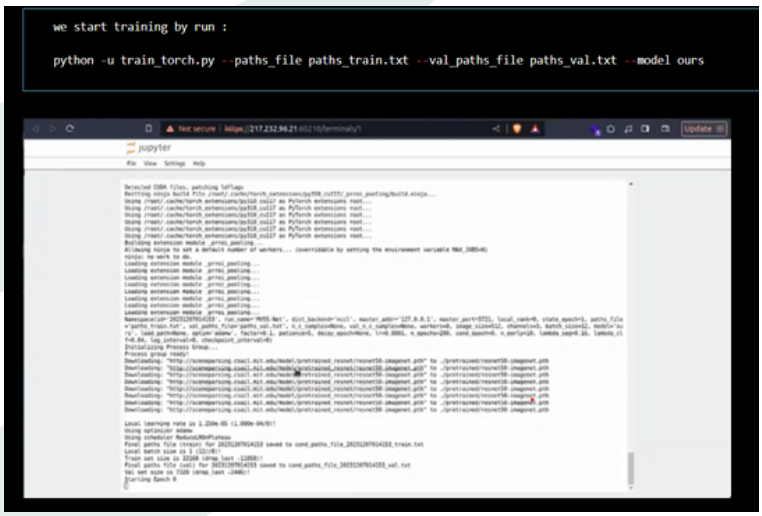

Figure35:Commands used to start trainingFigure34:Different GPUsin the machines

Now it takes only 15min per epoch

Here are snapshots capturing the training progress. For a comprehensive overview of the entire process, you can refer to a concise video covering all the discussed steps.

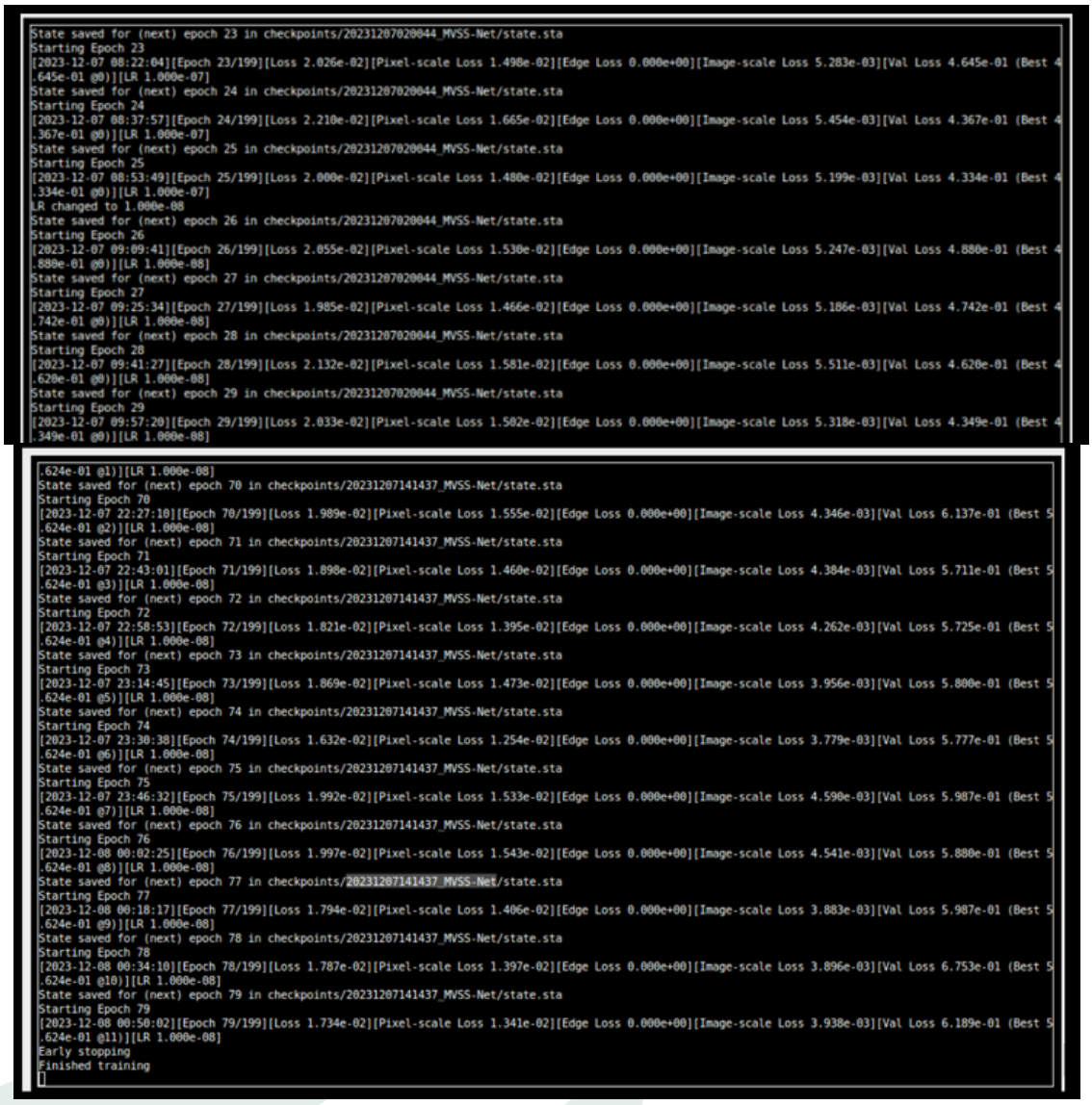

Figure36:Training process

Now, it's time for model evaluation. We retrieve the saved model from the checkpoints and proceed to test it on Colab. Concurrently, we initiate contemplation on an approach to enhance the obtained results.

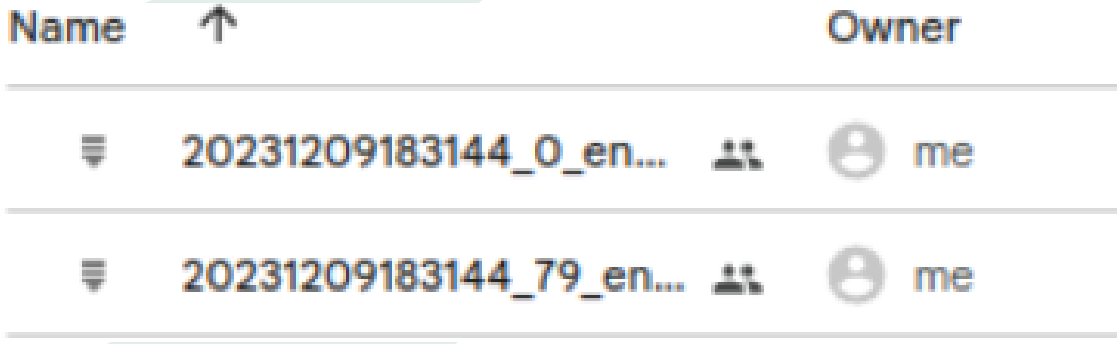

Figure37:Uploading the output on google drive

-->These are the checkpoints that been saving during the training.
-->Despite the early termination, we salvaged the saved model from the generated checkpoints. This preserved the model's state at its current progress, paving the way for future evaluation. We are currently preparing to assess the model's performance on Colab to identify areas for improvement.

Evaluation :

The script is building a PyTorch extension module, loading a pre-trained model, and making predictions on a dataset of 3526 images.

Figure38:Evaluating result on only shallowfake images

However, it encounters two issues during processing: A size mismatch for the image Tp_D_CRN_S_N_nat10130_pla00049_11524.jpg in the CASIA20revised dataset. Another size mismatch for the image 95t.tif in the COVERAGE dataset. Despite these issues, the script continues processing the remaining images.

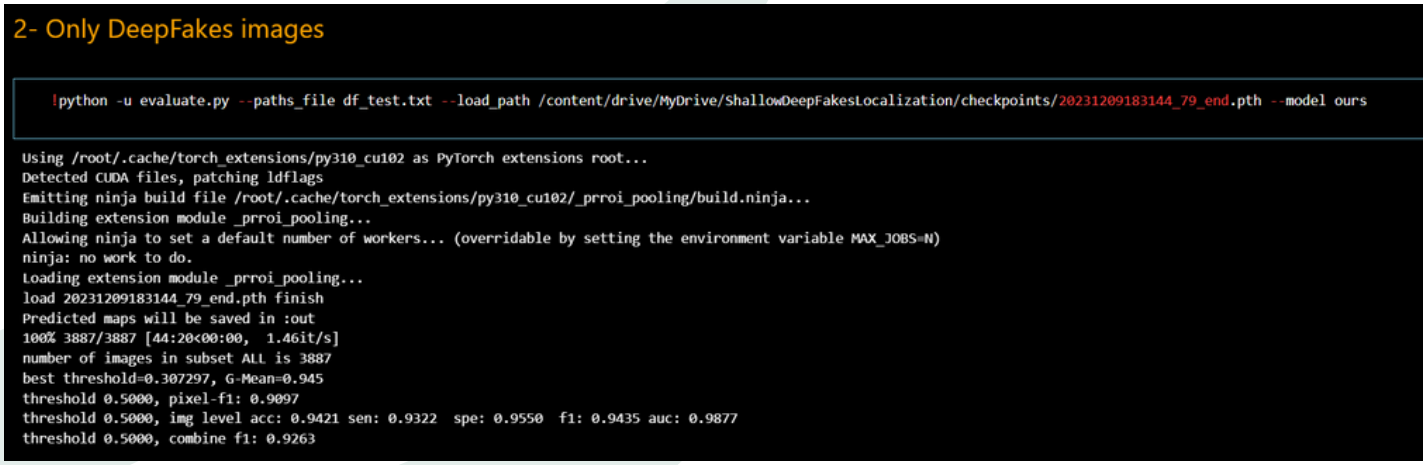

Figure39:Evaluation result on only deepfake images

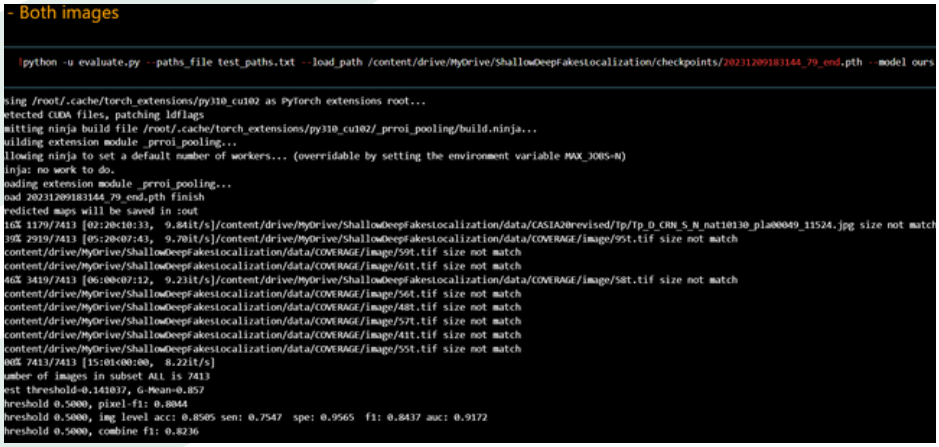

Figure40:Evaluation result on both shallow- and deep- fake images

The current results unveil valuable insights for the crucial final phase of model enhancement. A critical question emerges: which areas of the existing architecture necessitate refinement to bolster overall performance.
Additionally, identifying the most efficient and cost-effective approach for result improvement becomes a paramount concern. Moving forward, meticulously analyzing the results and implementing strategically targeted optimization techniques will propel the model towards achieving its full potential.

5 - OPTIMIZATION : :

We will delve into a detailed, step-by-step discussion of the three primary approaches proposed in the paper. The goal is to thoroughly explore and elucidate each approach, aiming to enhance the results reported in the research document.
This comprehensive examination will provide a nuanced understanding of the methodologies proposed, allowing for a more in-depth analysis of their potential impact and effectiveness in improving the outcomes as presented in the paper.

Now after finishing evalution and exploring in depth the results listed in the paper we choose **three primary** approaches which are :
1 - Architecture Modifying : In this approach we suggest to change Resnet50 backbonex
2 - Hyperparameter Tuning : In this approach we suggest re-train the model with new more complexe hyperparameters
3 - Dataset : In this section we suggest to finetune the model in new ShallowFakes datasets

Architecture Modifying

In the Architecture Modification approach, we suggest changing the foundational structure of our model by replacing the Resnet50 backbone. This means we'll look for a different, potentially more suitable structure that can better adapt to the specific needs of our task. By doing this, we hope to find a backbone architecture that improves the overall performance of our model, making it more effective in addressing the challenges of the problem we're tackling.

Backbone Benchmarking

```
from numpy.core.fromnumeric import size
import plotly.express as px
import pandas as pd
```

```
df = pd.read_csv("benchmark.csv")
```

```
fig = px.scatter(df, x="param_count", y="top1",
                 color="model",
                 hover_data=['fps'],
                 title='PyTorch Backbone Benchmark - Scatter Plot(Y: top1 acc, X: #param)'
                 )
fig.show()
```

Figure41:Comparing different backbone benchmarking

New models excel in certain situations, but their complexity might not translate to our specific task. Resnet50's simplicity and performance on our problem make it a good choice for now. Exploring new models is promising, but resource limitations require careful consideration

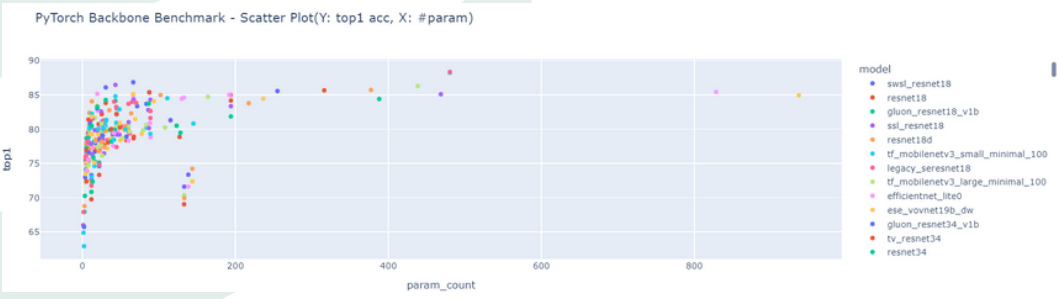

Figure42:PyTorch Backbone ranked by parameters

Hyperparameter Tuning :

We are transitioning to the Hyperparameter Tuning phase, where the approach entails retraining the model while adjusting existing parameters and introducing new ones

We **start** by changing :

- image_size = 256 , Batch_size = 32 to accelerate training process - lambda_seg = 0.1 , lambda_clf = 0.1 to balance between classification loss and segmentation loss - optim = "adam", then in another experience we introduce "RMSprop" because is converge rapidly and ensure learnings rates adpting

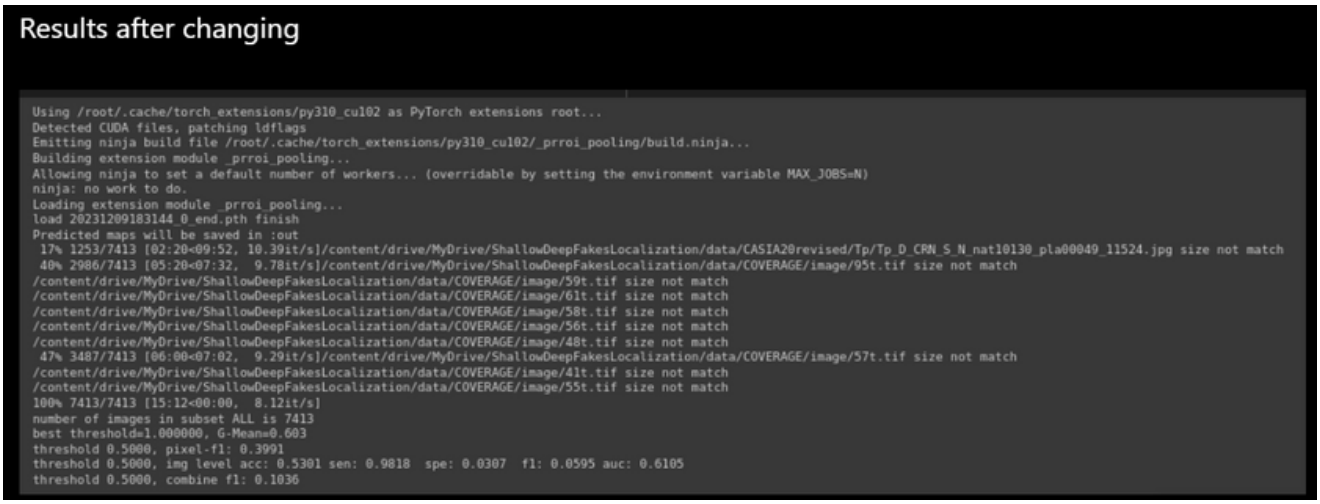

Figure43:Results after changing parameters

The second approach might be inefficient because the model in the research paper is likely optimized with specific hyperparameters. Experimenting with different settings could be expensive and yield worse results. We will therefore focus on the third approach.

Dataset :

- INSIGHTS

	Method	Training Set		Test Set											
				Shallowfakes				Deepfakes				Both			
				F1			AUC	F1			AUC	F1			AUC
		S	D	Pixel-level	Image-level	Combined		Pixel-level	Image-level	Combined		Pixel-level	Image-level	Combined	
1	MVSS-Net	√		**0.60**	**0.68**	**0.64**	0.79	0.24	0.51	0.33	0.53	0.42	0.62	0.50	0.68
2			√	0.45	0.00	0.01	0.50	0.90	0.93	0.91	0.98	0.67	0.62	0.64	0.71
3		√	√	0.59	0.64	0.62	0.78	0.90	0.94	0.92	0.97	**0.75**	0.81	**0.78**	0.87
4	Ours w/o BayerConv	√		**0.60**	0.62	0.61	0.83	0.30	0.29	0.29	0.55	0.45	0.47	0.46	0.72
5			√	0.44	0.27	0.33	0.43	**0.92**	**0.95**	**0.94**	**0.99**	0.68	0.69	0.68	0.78
6		√	√	0.59	0.65	0.62	**0.85**	0.87	0.93	0.90	0.98	0.72	0.81	0.76	0.91
7	Ours with BayerConv	√		**0.60**	**0.68**	**0.64**	**0.85**	0.27	0.29	0.28	0.47	0.44	0.51	0.47	0.70
8			√	0.45	0.12	0.19	0.45	**0.92**	**0.95**	**0.94**	**0.99**	0.68	0.66	0.67	0.81
9		√	√	**0.60**	0.66	0.63	0.84	0.91	0.94	0.93	**0.99**	**0.75**	**0.82**	**0.78**	**0.92**

Figure44:Table of final results after testing

Our model performs well on the Deepfakes dataset, but needs improvement on Shallowfakes. We propose fine-tuning the model on the Shallowfakes dataset to improve its performance and make it more versatile. This will help the model adapt and better detect shallowfakes, making it more effective in a wider range of scenarios.

- CHECKING FOR LABELED DATASET

To execute this approach, we are actively searching online for an open-source dataset that aligns with our specific requirements.

- DATASET DETAILS

The dataset contains two classes - REAL and FAKE. - For REAL, we collected the images from Krizhevsky & Hinton's CIFAR-10 dataset - For the FAKE images, we generated the equivalent of CIFAR-10 with Stable Diffusion version 1.4 - There are 100,000 images for training (50k per class) and 20,000 for testing (10k per class)

- **DATA PREPARATION**

In this step, we will undertake data preparation tasks such as consolidating all images into a single folder and generating a paths.txt file .. to the perform finetuning

- **TRAINING**

Now we can start Training after generating the paths file we perform training using this command in our machine

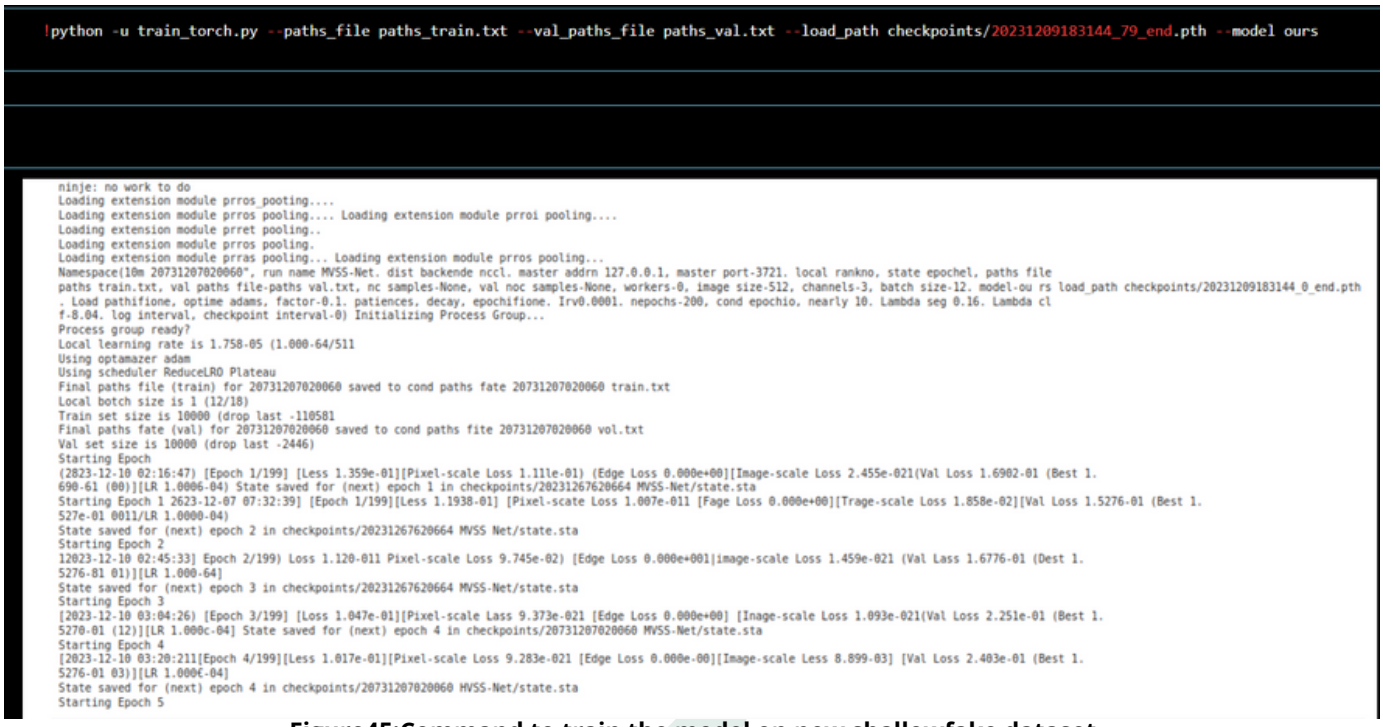

Figure45:Command to train the model on new shallowfake dataset

Model stop training after reaching epoch 34
--> After finishing training we will go through evaluation

- **EVALUATION**

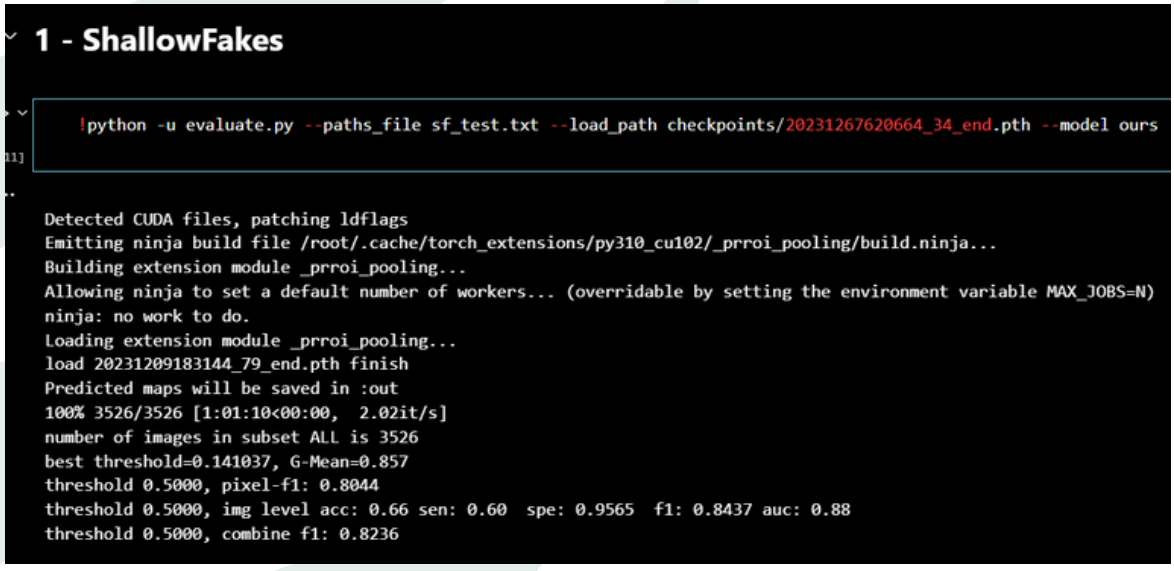

Figure46:Evaluating results on shallowfake dataset

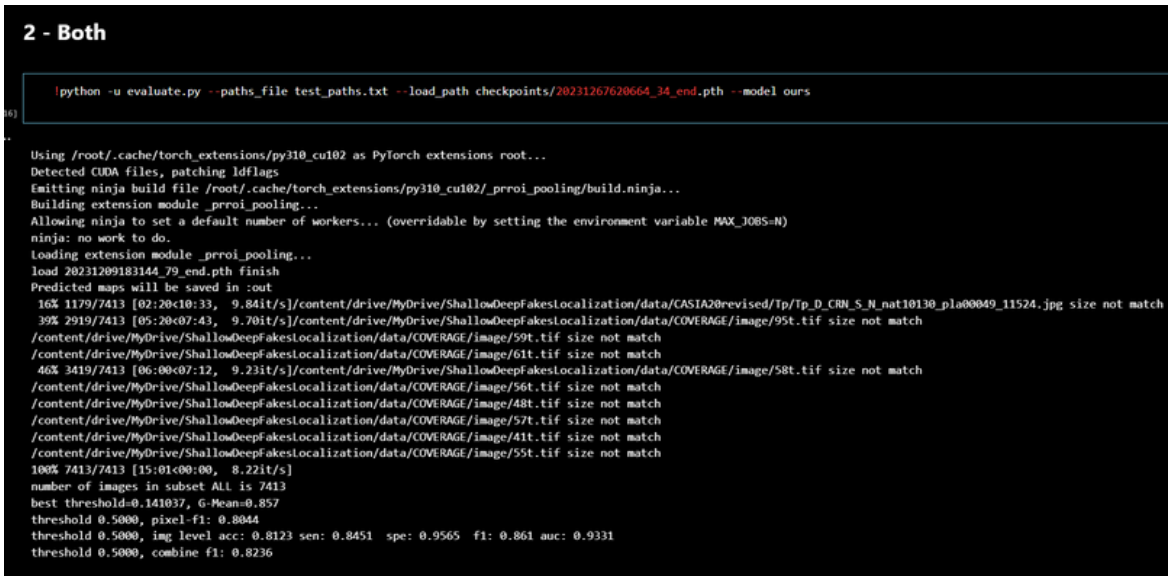

Figure46:Evaluating results on deepfake dataset

\DataSets Metrics\	ShallowFakes	Both
Pixels	0.8044	0.8044
Inf	0.60	0.8451
f1	0.8437	0.861
AUC	0.88	0.9331

figure47:Datasets Metrics Values

In summary, our methodology has exhibited outstanding performance, notably outperforming others in both AUC and F1 score metrics. This success can be attributed to our meticulous fine-tuning strategy, leveraging a pre-trained model and harnessing the extensive data from the source domain. This approach enables us to effectively capture nuanced features from shallowfake images, enhancing model efficacy without succumbing to overfitting. Moreover, our findings underscore the robustness of our approach across diverse datasets, validating its versatility and applicability.

IV. CONCLUSION : :

The successful attainment of desired results through the application of the Shallowfakes data upgrade method underscores its efficacy in enhancing model performance.

Furthermore, the subsequent implementation of two additional methodologies, namely "Hyperparameter Tuning" and "Modifying the architecture" using alternative models, has contributed to the overall optimization of our system.

As we look toward the future, it becomes evident that there exists a promising avenue for further improvement by synergizing these optimization methods. The convergence of Shallowfakes data upgrade, Hyperparameter Tuning, and Architectural Modification holds the potential to create a composite approach that leverages the strengths of each method. This amalgamation could serve as a catalyst for refining the Area under the Curve (AUC) and, by extension, elevating the overall performance metrics of our model.

In essence, the pursuit of an integrated and refined methodology reflects our dedication to pushing the boundaries of what is achievable in the realm of Deep learning.
As we navigate the landscape of optimization strategies, the synthesis of these approaches stands as a testament to our commitment to excellence and innovation in the pursuit of superior model performance.

V. REFERENCES: :

[1] [online] Available: https://globalnews.ca/news/7588121/facial-recognition-canada-rights/

[2] [online] Available: Shallow-_and_Deep-_fake_Image_Manipulation_Localization_Using_Deep_Learning.pdf

[3] [online] Available: https://drive.google.com/drive/folders/1-4Kts04Hvod4ZawTn2RnlWoev3ZpQSgZ?usp=drive_link

[4] [online] Available: https://github.com/jyhedHR/TSYP_CS_CHALLENGE_EspritSB_GRINTA