Deep Fake Detection using Metric Learning

Devadharshini Ayyappan
Electrical Engineering
NCSU
dayyapp@ncsu.edu

Kriti Singh
Electrical Engineering
NCSU
ksingh23@ncsu.edu

Rashmi Datta
Electrical Engineering
NCSU
rdatta2@ncsu.edu

I. Introduction

Deepfake videos have become a significant concern for society in recent years, especially with the increasing sophistication of Generative Adversarial Networks and Variational Auto-encoders. These manipulative videos have the power to alter the identity of individuals, potentially leading to serious consequences for their reputation and privacy. Detecting these videos is a challenging task, particularly when it comes to low-resolution formats that are widely circulated on social media platforms like WhatsApp. The CelebDF dataset provides an opportunity to evaluate and develop deepfake detection techniques in a controlled and standardized environment. In fact, previous studies using the CelebDF dataset have shown that current deepfake detection methods struggle to maintain their performance in low-resolution formats. Therefore, this report proposes a deep learning-based approach using metric learning to classify deepfake videos in low-resolution formats. This approach has the potential to significantly improve the accuracy of detecting deepfakes, particularly in low-resolution formats, and provide a valuable tool for combating the spread of manipulative content online.

II. Related Work

We reviewed several papers on face detection and deepfake image classification. We found that using the MTCNN for face detection and the XceptionNet architecture for video classification is a popular and effective approach.While deep CNN models can very effectively detect local artifacts, modern deepfakes generation techniques can create a wide range of artifacts, from ones that are local to those that span the entire image. Moreover, there is a large diversity in the types of artifacts produced owing to the different types of generation techniques available. Research [1] introducing the FaceForensics dataset, analyzes the performances of various detectors based on leared features i.e architectures like MesoNet, Inception and XceptionNet.Among the different architectures, XceptionNet performs best on various image corpus size compared to other architectures. In [2] Zhu et al. propose to utilize 3D facial details to detect deepfakes. Authors find that merging the 3D identity texture and direct light is significantly helpful in detecting deepfakes. They employ the XceptionNet CNN model for feature extraction. A face cropped image and its 3D detail is used to train the detection model. They also perform a detailed analysis of a number of different feature fusion strategies. The proposed technique was trained on FaceForensics++ dataset and evaluated on(1) FaceForensics++, (2) Google Deepfake Detection Dataset, and (3) DFDC datasets.

As analyzed in [3], XceptionNet was performing better on a huge corpus of data and many other methods like LSTM and RNN were used to improve the accuracy of the predictions.The accuracy didn't show much improvement.Metric learning was used on the image embeddings learned from the facenet model which showed huge improvement in the accuracy of predictions.

III. DATASET

The dataset that we used in this project is CelebDF and CelebDF-v2. The CelebDF is a publicly available dataset that contains over 590,000 videos of 1,100 celebrities. The dataset includes both real and manipulated videos, with the manipulated videos generated using a variety of deepfake techniques, including face swapping and reenactment. The videos are of varying resolutions, ranging from 360p to 1080p, and include different facial expressions, poses, and lighting conditions.The CelebDF-v2 is an updated version of the CelebDF dataset that was released in 2020.

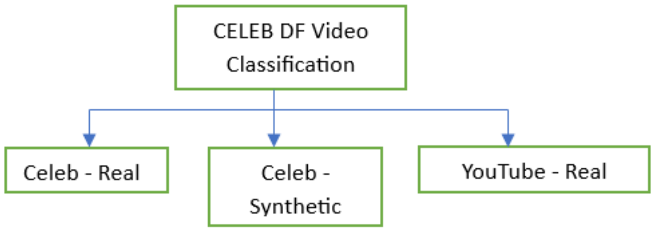

Fig. 1. CelebDF Dataset Split

The dataset contains over 5,639 real videos and 5,639 deepfake videos generated using a variety of techniques, including face swapping, reenactment, and neural rendering. The videos are of varying resolutions, ranging from 480p to 1080p, and include different facial expressions, poses, and lighting conditions. The dataset includes metadata for each video, such as the manipulation method used and the identity of the celebrity. There are other similar datasets such as The Unreal and Avatar DeepFake Videos (UADFV), FaceForensics++, DeepFake Detection Challenge (DFDC) and DeeperForensics-1.0. The Unreal and Avatar DeepFake Videos (UADFV) dataset is a synthetic dataset that includes both

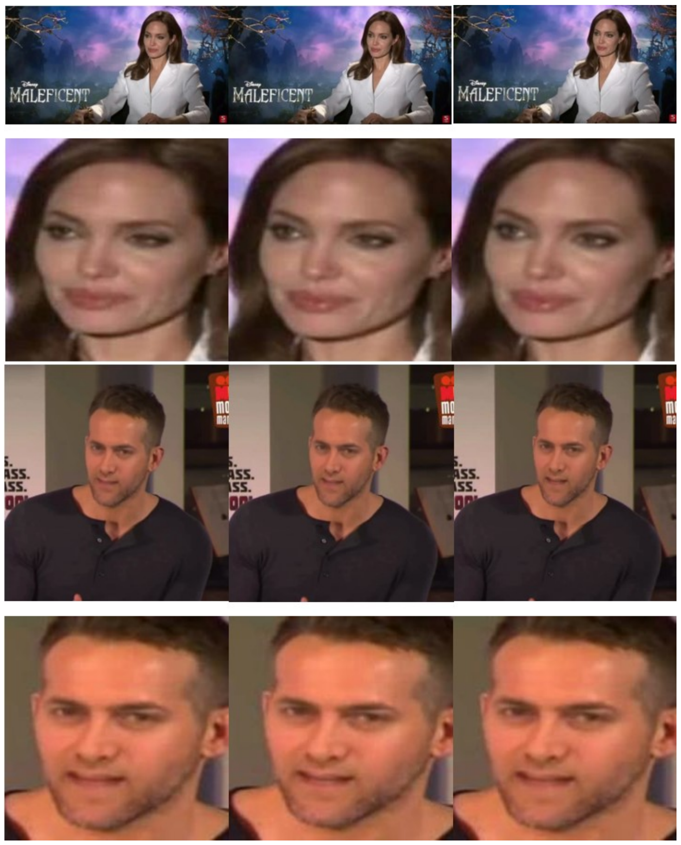
Fig. 2. The first two rows depict the real frames and its corresponding extracted faces using MTCNN. The last two rows depict the fake frames and its corresponding extracted faces using MTCNN

real and deepfake videos generated using CGI techniques. While the dataset contains a large number of videos, the synthetic nature of the videos may limit its applicability to real-world scenarios. The FaceForensics++ dataset is a benchmark dataset for deepfake detection that includes both real and manipulated videos. However, the dataset is limited to a smaller number of individuals and a smaller number of manipulation techniques compared to CelebDF. The DeepFake Detection Challenge (DFDC) dataset is a large-scale dataset that includes over 120,000 videos of real and manipulated footage. While the dataset is comprehensive, it is limited to a smaller set of individuals and scenarios compared to CelebDF. The DeeperForensics-1.0 dataset is a large-scale dataset that includes both real and manipulated videos generated using a variety of techniques. However, the dataset does not include information about the individuals in the videos, making it difficult to apply to real-world scenarios where the identity of the individuals may be important.

Compared to these datasets, CelebDF offers a larger and more diverse set of individuals and manipulation techniques, making it a more comprehensive dataset for deepfake detection.

IV. Dataset Preprocessing

To prepare the CelebDF dataset for deepfake detection, we first extracted frames from the video files using OpenCV's VideoCapture function. We then used the Multi-Task Cascaded Convolutional Neural Network (MTCNN) algorithm to detect and extract faces from each frame. MTCNN is a popular face detection and alignment algorithm that uses a deep neural network to detect faces with high accuracy and it consists of three stages, with each stage performing a specific task in the face detection and alignment process.

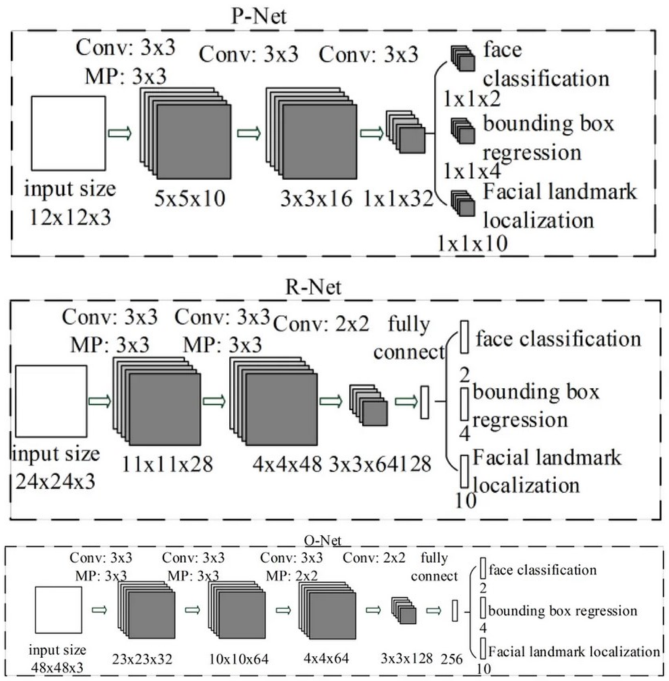

Fig. 3. MTCNN Architecture

MTCNN Architecture: The first stage of MTCNN is a fully convolutional network called Proposal Net that generates candidate regions of interest (RoIs) in an image. These candidate RoIs are then passed to the second stage, which is a convolutional neural network called Refine Net that refines the candidate RoIs and eliminates false positives. Finally, the third stage, Output network, uses a convolutional network and a non-maximum suppression algorithm to generate the final set of bounding boxes around each face in the image. The MTCNN architecture with three layers are depicted in Fig.1

One of the key advantages of MTCNN is that it is able to detect faces at different scales and orientations, which is important for real-world applications where faces may be of different sizes and orientations in different images. MTCNN is also able to handle occlusions, pose variations, and other challenges in face detection.

By applying MTCNN to each frame in a video, we can obtain a set of face images that can be used to train a deepfake detection model. By using MTCNN to extract faces from both real and manipulated videos, we can ensure that the resulting face images are consistent across all samples and can be used to accurately classify real and manipulated videos.

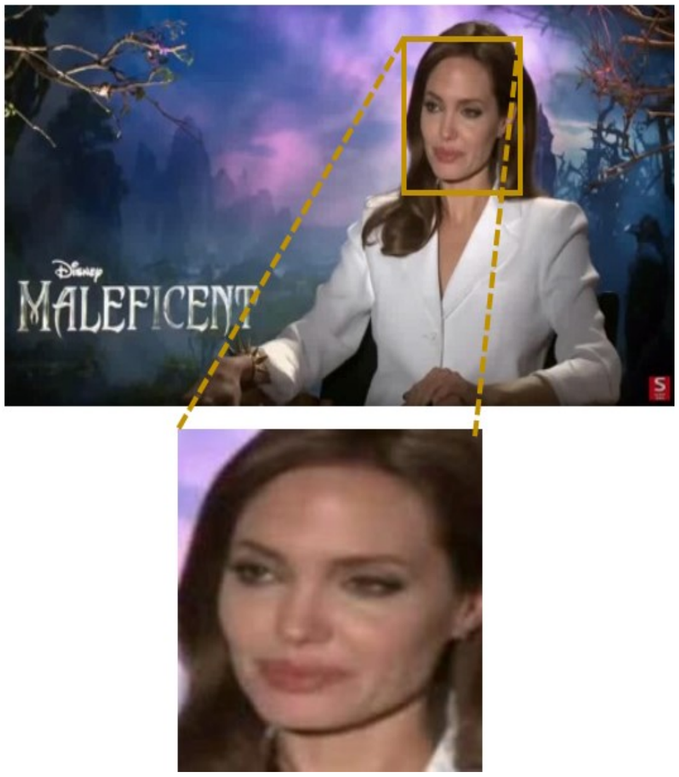

Fig. 4. Face extraction from a frame (real) using MTCNN algorithm

V. Our Approach

We started with training Xception Net architecture with pretrained weights and analyzed the performance of the network with the frames extracted from the videos from the dataset.Further to imporve the performance of classification we implemented feature extraction through state of the art FaceNet architecture which was trained using Triplet loss function and the embeddings obtained from this model was further used to train classical Machine Learning classifiers like Stochastic Gradient Descent and Random Forest.

VI. Xception Net

The Xception architecture has 36 convolutional layers forming the feature extraction base of the network.The 36 convolutional layers are structured into 14 modules, all of which have linear residual connections around them, except for the first and last modules. In short, the Xception architecture is a linear stack of depthwise separable convolution layers with residual connections. This makes the architecture very easy to define and modify; it takes only 30 to 40 lines of code using a high- level library such as Keras [2] or TensorFlow-Slim [17], not unlike an architecture such as VGG-16 [18], but rather unlike architectures such as Inception V2 or V3 which are far more complex to define. An open-source implementation of Xception using Keras and TensorFlow is provided as part of the Keras Applications module2, under the MIT license.

VII. Metric Learning with Triplet Loss

The triplet loss for face recognition has been introduced by the paper [6] from Google. They describe a new approach to train face embeddings using online triplet mining.The goal of the triplet loss is to make sure that two examples with the same label have their embeddings close together in the embedding space and two examples with different labels have their embeddings far away. However, we don't want to push the train embeddings of each label to collapse into very small clusters. The only requirement is that given two positive examples of the same class and one negative example, the negative should be farther away than the positive by some margin. This is very similar to the margin used in SVMs, and here we want the clusters of each class to be separated by the margin. To formalise this requirement, the loss will be defined over triplets of embeddings:an anchor, a positive of the same class as the anchor,a negative of a different class.

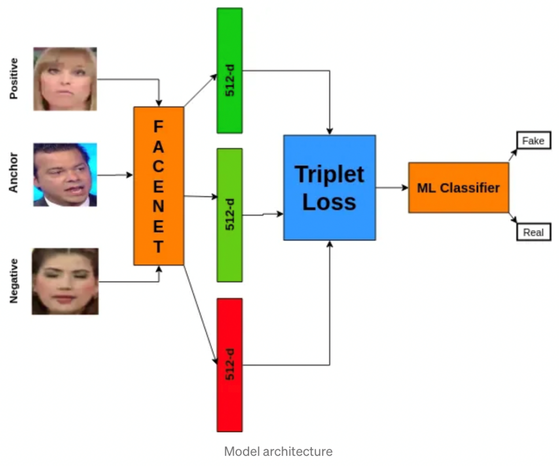

Fig. 5. Metric Learning Architecture

To formalise this requirement, the loss will be defined over triplets of embeddings:an anchor, a positive of the same class as the anchor,a negative of a different class as shown in Fig.4 equation. We minimize this loss, which pushes d(a,p) to 0 and d(a,n)to be greater than d(a,p)+margin. As soon as n becomes an "easy negative", the loss becomes zero.

$$\mathcal{L} = max(d(a,p) - d(a,n) + margin, 0)$$

Fig. 6. Equation for triplet loss

Based on the definition of the loss, there are three categories of triplets:

Easy Triplets: Triplets which have a loss of 0, because d(a,p)+ margin <d(a,n)

Hard Triplets: Triplets where the negative is closer to the anchor than the positive, i.e. d(a,n)<d(a,p)

Semi-Hard Triplets: Triplets where the negative is not closer to the anchor than the positive, but which still have positive loss i.e d(a,p)<d(a,n)<d(a,p)+ margin

Each of these definitions depend on where the negative is, relatively to the anchor and positive. We can therefore extend these three categories to the negatives: hard negatives, semi-hard negatives or easy negatives.

VIII. Implementation

We used pretrained weights of Xception Net for intermediate layers where features were extracted and at the end added a base model with a maxpooling layer followed by the a fully connected layer with 512 neurons followed by two dropout layers with 0.4 and 0.5 values respectively.The last layer was a softmax layer with 2 outputs.Nadam optimizer was used and the best learning rate that we found for the model was 0.05, beta1 was 0.9, beta2 was 0.999, epsilon was 1e-08 with schedule decay of 0.004.Categorical cross entropy loss function was used during training.

Data Augmentation was performed on the training and validation data before using it for training.We applied rotation, zoom, flip and width and height shift along with rescaling on the dataset.For training we used 15 per cent of our dataset for validation.

For implementing the triplet loss function there are two ways of mining triplets : online and offline.Online triplet mining was introduced in Facenet.The idea here is to compute useful triplets on the fly, for each batch of inputs. We compute embeddings for all examples in the batch using a pre-trained neural network.For each example in the batch,we find the closest positive and negative examples based on the distance metric between their embeddings.We filter out the triplets that do not have two positives and one negative example. This can be done by checking whether the closest positive examples for a given anchor example are the same or not, and whether the closest negative example is different from the anchor example.If there are more than B3 valid triplets, select the B3 triplets with the highest triplet loss, which is a function of the distance between the anchor and positive examples and the distance between the anchor and negative example.

This technique gives you more triplets for a single batch of inputs, and doesn't require any offline mining. It is therefore much more efficient.

We used semi-hard triplets as those showed the best results for face verification and clustering according to the reference literature.Once the semi-hard triplets are extracted we train them using the pretrained weights of FaceNet and the triplet loss function is used for training.We added a base network after the frozen layers of FaceNet with three fully connected layers of size 256, 256 and 128 respectively.We used two drop out layers with 0.1 as values of the drop out parameter.At last we added a dense layer which gives us an embedding vector of size 64.We trained this network for our dataset with batch size of 32 and 100 epochs. We used Adam optimizer and the learning rate that we used was 0.0003. After getting the optimzed embeddings for our dataset we gave these embeddings as input to ML classifiers SGD and Random Forest.

IX. Phase I Results

For the Phase I of our project, we used XceptionNet Architecture as our model for which we got an initial accuracy of 50%. The loss curves for this is depicted in Fig. 7. The dataset we used was CelebDF. To further improve the accuracy, we performed various experiments using the XceptionNet Architecture as the base. We tried LSTM along with XceptionNet architecture on the same dataset. In this case, the last layer of the XceptionNet architecture was removed and the embeddings were fed to the LSTM network and trained on this combination. But still, the accuracy of the combined model did not improve much and gave only 52%. This has tabulated in Table III. Another experiment involved training the Xception Network with a combination of CelebDF and CelebDF V2 dataset and the maximum accuracy that we got in this case was 70%. Finally, We decided to implement metric learning for the real and fake detection.

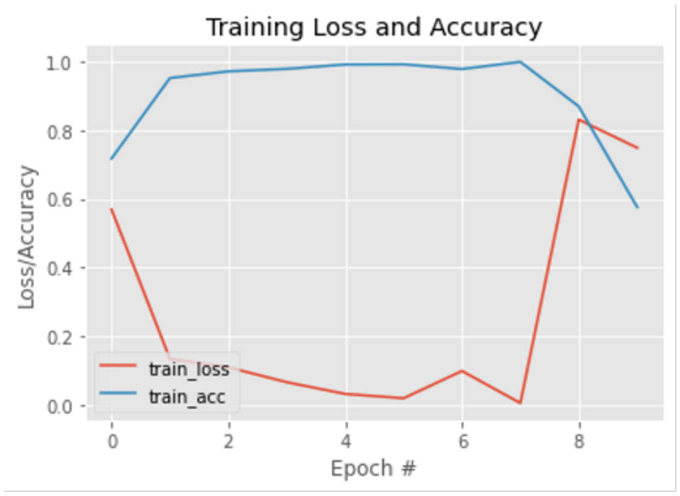

Fig. 7. Loss and accuracy plot in Phase I

TABLE I
Hyperparameter Tuning results from Phase I

Trials	Learning rate	Batch Size	Number of Epochs	Accuracy
1	0.1	64	10	50%
2	0.05	64	10	53%
3	0.01	64	10	52%

X. Phase II Results

In the Second Phase, we used metric learning with the embeddings from the XceptionNet Architecture initially. But it gave similar results as in phase I. So, we used FaceNet architecture to generate the embeddings and performed the metric learning using this. The results for the same has been tabulated in Table III where we achieved an accuracy of 87.5%.

For visualization of the embeddings, we used the TSNE plot which is depicted in Fig. 5. t-SNE (t-Distributed Stochastic

TABLE II
METRICS OBTAINED IN PHASE II

Measures	Triplets + SGD	Triplets + RF
AUC Score	0.95138	0.95833
Accuracy	0.875	0.875
Precision	0.8	0.8
Recall	1.0	1.0
F1 Score	0.888889	0.888889

Neighbor Embedding) is a widely used technique for dimensionality reduction and visualization of high-dimensional data. In our project, we used t-SNE to visualize the distribution of our data in a two-dimensional plot. The t-SNE plot shows the clustering of data points based on their similarity in a reduced two-dimensional space. The closer the points are in the plot, the more similar they are. This allows us to identify patterns and relationships between the data points that may not be easily visible in the original high-dimensional space. In our project, we used the TSNE plot to visualize the embeddings before and after the triplet loss is applied. In the figure, we can see that on the left-side plot, the real and fake embeddings are distributed throughout the space. But in the right-side plot, the real and fake embeddings are neatly clustered after applying the triplet loss.

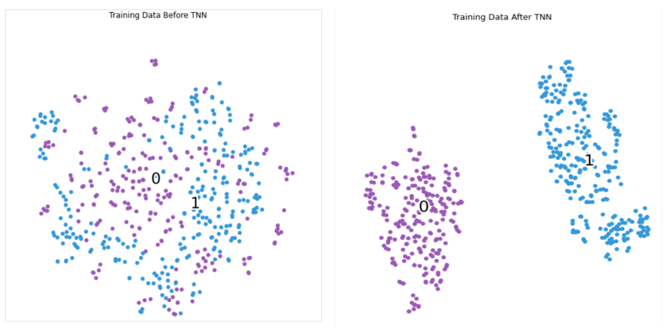

Fig. 8. TSNE plots depicting comparison of data before and after training the model with Triplet Network

We then used classical machine learning models to do the binary classification on the clustered embeddings. The classifiers that we used are Stochastic Gradient Descent (SGD) Classifier and Random Forest (RF) Classifier. The results have been tabulated for reference in Table II and we got high AUC scores of 95% which is depicted in Fig. 9.

XI. Conclusion

From the above experiments and the results, we found that Metric Learning with Facenet Architecture works better on CelebDF dataset. In our reference literature, the FaceForensics++ dataset was used and it had been shown to give better performance compared to other datasets. And the performance of CelebDF dataset was comparatively poorer in the base literature. XceptionNet showed better performance with a larger corpus of training dataset. Therefore, by implementing metric learning on the CelebDF dataset, we achieved higher performance metric with smaller data corpus.

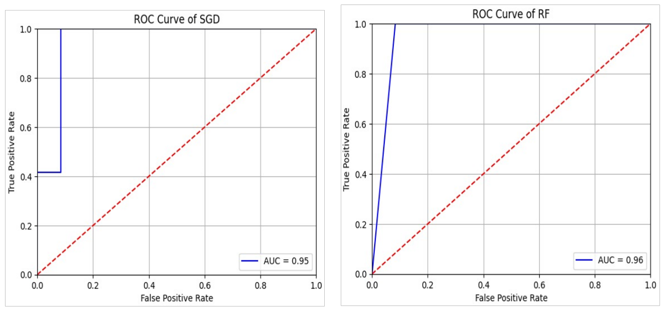

Fig. 9. ROC curve of the SGD and RF Classifiers respectively depicting False vs True positive rate

TABLE III
EXPERIMENTS PERFORMED AND SELECTED HYPERPARAMETERS

Architectures Used	Dataset	Best Learning Rate	No. of Epochs	Batch size	Accuracy
XceptionNet(10k frames)	CELEB DF	0.05	10	64	50%
LSTM+Frames	CELEB DF	0.01	30	64	52%
XceptionNet(30k frames)	CELEB DF+CELEB DF v2	0.05	30	64	70%
Facenet+Metric Learning	CELEB DF+CELEB DF v2	0.0003	100	32	87.5%

References

[1] FaceForensics++: Learning to detect manipulated facial images-https://arxiv.org/abs/1901.08971: :text=https%3A//doi.org/10.48550/arXiv.1901.08971
[2] Xiangyu Zhu, Hao Wang, Hongyan Fei, Zhen Lei, and S. Li. 2021. Face Forgery Detection by 3D Decomposition. In Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2928–2938
[3] Detecting deepfakes with metric learning https://arxiv.org/abs/2003.08645: :text=https%3A//doi.org/10.48550/arXiv.2003.08645
[4] Shreyan Ganguly, Aditya Ganguly, Sk Mohiuddin, Samir Malakar, Ram Sarkar/ViXNet: Vision Transformer with Xception Network for deepfakes based video and image forgery detection.
[5] Xception: Deep Learning with Depthwise Separable Convolutions https://doi.org/10.48550/arXiv.1610.02357
[6] FaceNet: A Unified Embedding for Face Recognition and Clustering https://arxiv.org/abs/1503.03832

XII. Source code

The source code for this project is available at : https://github.com/RashmiD25/DeepFake_Detection