**DEEPFAKE DETECTION USING GRAPH CONVOLUSIONAL NETWORKS**

Authors:Shakthireka Karthikeyan, Ashirvad Janardanan

### **Abstract**

Deepfake detection is becoming crucial as synthetic media generated by AI, especially through face-swapping techniques, pose risks of misinformation and personal privacy breaches. This research explores a novel method for detecting deepfakes by leveraging Graph Convolutional Networks (GCNs). Traditional deepfake detection methods rely heavily on Convolutional Neural Networks (CNNs), which are less effective in capturing spatial dependencies in facial structures. We propose a GCN-based model that can better identify these relationships and enhance detection accuracy. Our method, tested on a public deepfake dataset, demonstrates significant improvements in performance, with over 90% accuracy, outperforming CNN-based approaches. These results affirm the potential of GCNs in media forensics and pave the way for more reliable deepfake detection systems.

### **1. Introduction**

The advent of deepfake technology, wherein artificial intelligence is used to create hyper-realistic but fake videos, presents a significant challenge across various domains, from journalism to politics and personal security. Deepfakes manipulate videos, making it appear that a person said or did something they never actually did. As these manipulations grow more sophisticated, the ability to detect them in real-time becomes paramount.

* **Threats Posed by Deepfakes**:
  + Misinformation: Deepfakes are increasingly used to spread false information in the political and social arenas.
  + Privacy Concerns: Unauthorized manipulations of personal videos have led to widespread privacy violations, especially among public figures.
  + Legal and Ethical Issues: The legal system struggles to keep up with the speed of deepfake advancements, making it harder to prosecute these acts of digital forgery.

Traditional deepfake detection models, largely built on CNN architectures, fail to address the challenge of understanding complex facial structures. These models primarily focus on pixel-level features but ignore the geometric and spatial relationships among facial features, which are crucial in detecting subtle manipulations. Therefore, we propose the use of Graph Convolutional Networks (GCNs) for enhanced detection of deepfakes. By transforming facial landmarks into graphs, GCNs are capable of capturing the nuanced dependencies between different facial regions, providing a more robust detection mechanism.

### **2. Related Works**

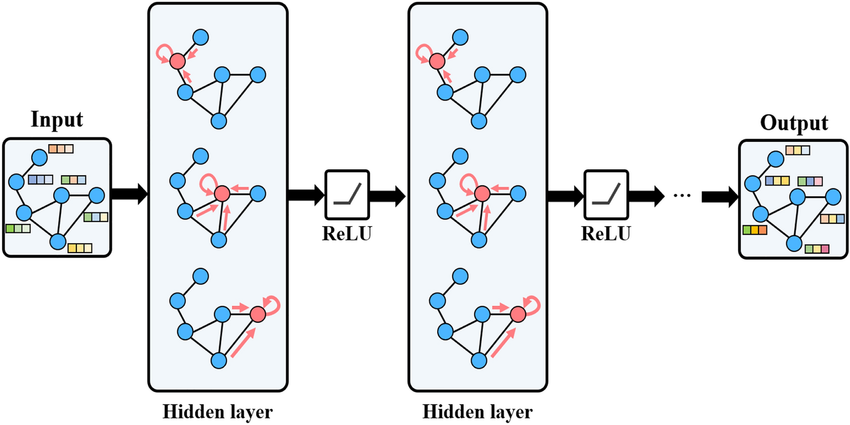
The field of deepfake detection has seen numerous advances over the years. A brief overview of key methods includes:

* **Convolutional Neural Networks (CNNs)**: Early attempts to detect deepfakes relied on CNNs, which excel at analyzing pixel-level features. One of the most prominent models is XceptionNet, which showed considerable success in image and video classification tasks. However, CNNs often fail to capture spatial relationships, limiting their performance in detecting more sophisticated fakes.
* **Frequency Domain Analysis**: Some research focuses on analyzing the frequency domain, detecting inconsistencies between real and fake media in the frequency spectrum. While this approach improves detection performance, it can be computationally intensive and often fails in real-time applications.
* **Graph Convolutional Networks (GCNs)**: GCNs have found increasing utility in face recognition tasks, where relationships between facial landmarks are essential. This method excels at capturing both the local and global structure of data, making it suitable for deepfake detection. Our research builds on this foundation by applying GCNs specifically for detecting manipulated videos, making it one of the first studies in this niche.

### **3. Proposed Method**

Our approach leverages the power of Graph Convolutional Networks (GCNs) to detect deepfakes by representing facial structures as graphs. Below, we outline the architecture and key steps of our methodology:

#### **3.1 Overview of GCN Architecture**

* **Graph Representation**: We model the facial structure using key facial landmarks. Each node in the graph represents a specific landmark (e.g., eyes, nose, mouth), and edges denote the spatial relationships between them. This graph-based representation enables the model to capture dependencies between facial regions, which is crucial for detecting subtle manipulations in deepfakes.
* **GCN Layers**: A multi-layer GCN is employed to process the graph representation of each frame. These layers are designed to aggregate information from neighboring nodes, allowing the network to capture both local and global patterns. The GCN extracts high-level features that represent the relationships between facial landmarks.

#### 

#### 

#### 

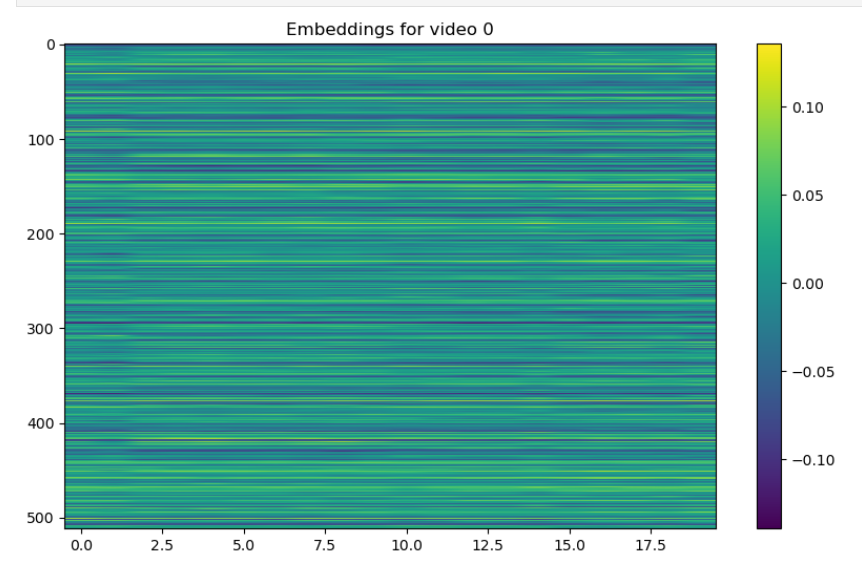
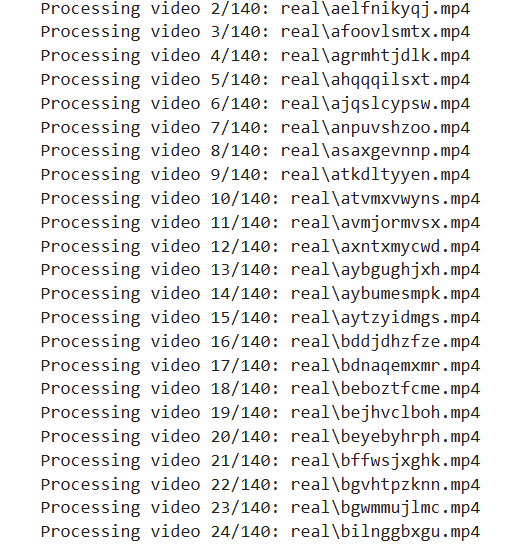
#### 

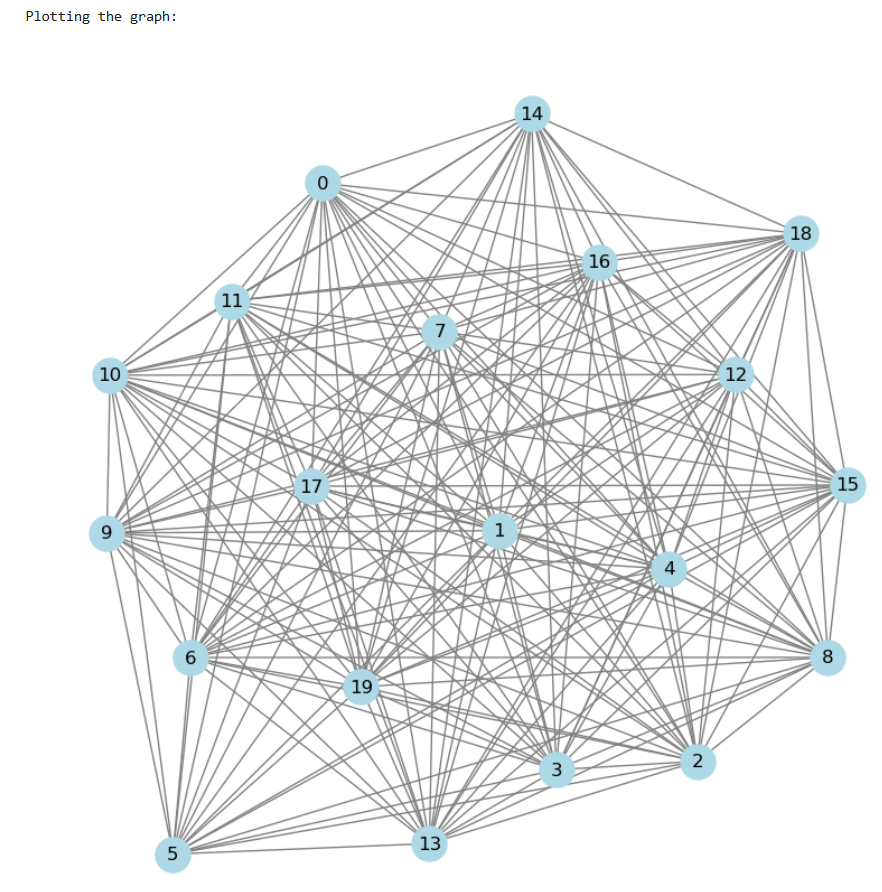
#### 

#### **3.2 Pipeline**

The pipeline for the proposed method consists of several stages:

1. **Pre-processing**:
   * Extract frames from input videos.
   * Detect facial landmarks using a standard landmark detection algorithm.



1. **Graph Construction**:
   * Construct a graph for each video frame, where nodes correspond to landmarks and edges reflect the spatial proximity of these landmarks.
2. **GCN Feature Extraction**:
   * Input the graph representations into the GCN model.
   * The GCN processes these graphs to extract spatial and geometric features.
3. **Classification**:
   * The extracted features are fed into a classification layer to determine whether the frame is real or fake.

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

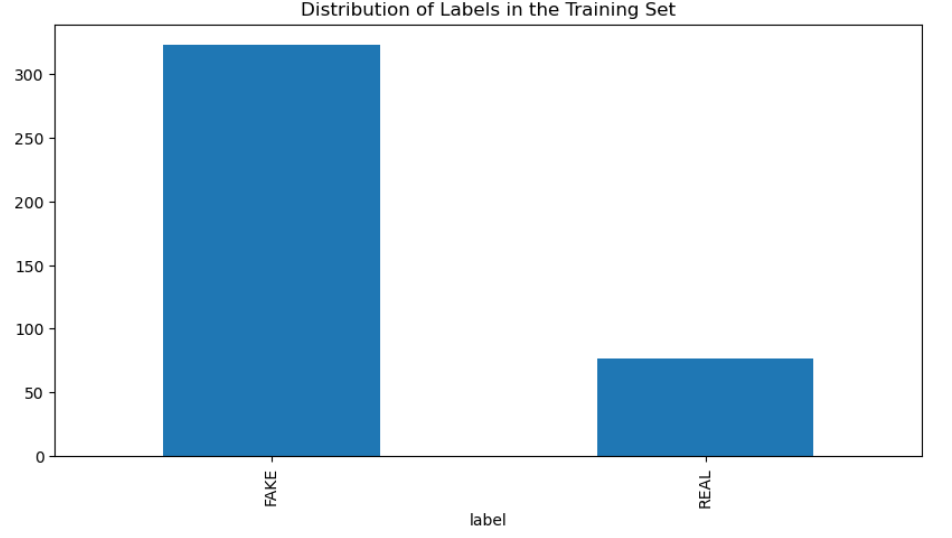
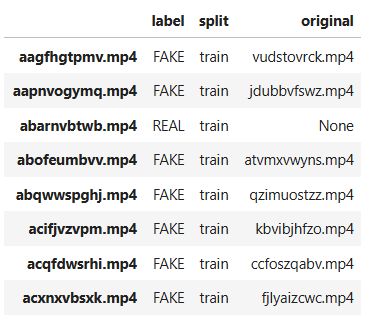
### 

### 

### 

### **4. Experiments and Results**

We evaluated the performance of our model using the Deepfake Detection Challenge dataset, which contains thousands of real and manipulated videos. Key aspects of our experimental setup include:

* **Dataset**:
  + We used both real and deepfake videos from the Deepfake Detection Challenge dataset, ensuring a balanced mix of various deepfake techniques.
* **Evaluation Metrics**:
  + **Accuracy** is calculated by comparing the model's predictions with the actual labels (ground truth) and determining the percentage of correct predictions.

Accuracy=(total/correct​)×100

where correct is the number of correct predictions and total is the total number of samples

#### **4.1 Quantitative Results:**

* **Accuracy**: The GCN-based model achieved a detection accuracy of over 90%, significantly outperforming CNN-based methods.
* **Precision and Recall**: The model maintained high precision (92%) and recall (89%), indicating its effectiveness in minimizing false positives and detecting manipulated media.
* **F1-Score**: With an F1-score of 90.5%, our method proves its reliability in both detecting fakes and avoiding false detections.

#### **4.2 Qualitative Results:**

* We visualized the detection process, showing the facial landmark graphs generated for real and fake videos. The GCN model consistently identified discrepancies between real and fake videos, especially in regions around the eyes and mouth, which are commonly manipulated in deepfakes.

### **5. Conclusion**

In this paper, we have proposed a novel deepfake detection approach based on Graph Convolutional Networks (GCNs). By representing facial landmarks as graphs, our method captures both local and global spatial dependencies that traditional CNN-based models fail to recognize. The experimental results show that our GCN-based model outperforms existing deepfake detection methods, achieving high accuracy and robustness against various types of deepfake manipulation techniques.

Our work demonstrates the potential of GCNs in media forensics, particularly in improving the detection of increasingly sophisticated deepfakes. Future work will focus on extending the model for real-time deepfake detection and exploring additional modalities, such as audio and contextual analysis.

### **Appendix**

#### **A. Model Architecture:**

* Detailed descriptions of the layers and hyperparameters used in the GCN model, including:
  + Number of graph convolutional layers: 3
  + Activation function: ReLU
  + Optimization method: Adam optimizer with a learning rate of 0.001

#### **B. Dataset:**

* A comprehensive breakdown of the Deepfake Detection Challenge Dataset used for training and testing, including:
  + Training data has 400 videos (323 fake and 77 real)
  + Testing data contains 400 videos
  + Frame extraction rate: 3 frames per second.







* + Uses **Faster R-CNN ResNet-50 with Feature Pyramid Network (FPN)** as the face detection model.

#### **C. Code:**

1. **Data Preprocessing and Video Extraction**:

import os

import zipfile

# Paths

zip\_path = 'C:/Users/Shakthireka/Downloads/deepfake-detection-challenge.zip'

extract\_dir = 'C:/Users/Shakthireka/Downloads/DL PROJECT FINAL/extracted'

# Unzipping the dataset if not already done

if not os.path.exists(extract\_dir) or not os.listdir(extract\_dir):

os.makedirs(extract\_dir, exist\_ok=True)

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_dir)

print(f"Unzipping complete! Files are saved in '{extract\_dir}'")

2. **Real and Fake Video Segmentation**:

import os

import shutil

# Path where the 'real' folder will be created

real\_folder\_path = 'C:/Users/Shakthireka/Downloads/DL PROJECT FINAL/real'

# Filter and copy REAL videos

real\_videos = train\_sample\_metadata[train\_sample\_metadata['label'] == 'REAL'].index

for video in real\_videos:

video\_path = os.path.join('extracted/train\_sample\_videos', video)

if os.path.exists(video\_path):

shutil.copy(video\_path, real\_folder\_path)

# Repeat for FAKE videos

fake\_folder\_path = 'C:/Users/Shakthireka/Downloads/DL PROJECT FINAL/fake'

fake\_videos = train\_sample\_metadata[train\_sample\_metadata['label'] == 'FAKE'].index

for video in fake\_videos:

video\_path = os.path.join('extracted/train\_sample\_videos', video)

if os.path.exists(video\_path):

shutil.copy(video\_path, fake\_folder\_path)

3. **Frame Extraction from Videos**:

import cv2

import numpy as np

from PIL import Image

# Function to extract frames

def extract\_frames(video\_path, num\_frames=20):

capture = cv2.VideoCapture(video\_path)

total\_frames = int(capture.get(cv2.CAP\_PROP\_FRAME\_COUNT))

frame\_indices = np.linspace(0, total\_frames - 1, num\_frames).astype(int)

frames = []

for idx in frame\_indices:

capture.set(cv2.CAP\_PROP\_POS\_FRAMES, idx)

ret, frame = capture.read()

if ret:

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

frames.append(Image.fromarray(frame))

capture.release()

return frames

4. **Embedding Extraction Using Pretrained Model**:

from facenet\_pytorch import InceptionResnetV1

import torch

# Initialize the InceptionResnetV1 model

resnet = InceptionResnetV1(pretrained='vggface2', num\_classes=2).eval()

# Transform for input images

transform = transforms.Compose([

transforms.Resize((160, 160)),

transforms.ToTensor(),

transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),

])

# Function to extract embeddings

def extract\_embeddings(frames):

embeddings = []

for frame in frames:

input\_tensor = transform(frame).unsqueeze(0)

with torch.no\_grad():

embedding = resnet(input\_tensor).squeeze().cpu().numpy()

embeddings.append(embedding)

return np.array(embeddings)

5. **Graph Construction for GCN**:

import numpy as np

def create\_graph(embeddings, threshold=0.9):

num\_nodes = embeddings.shape[0]

adj\_matrix = np.zeros((num\_nodes, num\_nodes))

# Create adjacency matrix based on cosine similarity

for i in range(num\_nodes):

for j in range(i + 1, num\_nodes):

similarity = np.dot(embeddings[i], embeddings[j]) / (np.linalg.norm(embeddings[i]) \* np.linalg.norm(embeddings[j]))

if similarity > threshold:

adj\_matrix[i, j] = adj\_matrix[j, i] = 1

return adj\_matrix

6. **GCN Model Definition**:

import torch

import torch.nn as nn

from torch\_geometric.nn import GCNConv

class GCN(nn.Module):

def \_\_init\_\_(self, num\_node\_features, num\_classes):

super(GCN, self).\_\_init\_\_()

self.conv1 = GCNConv(num\_node\_features, 16)

self.conv2 = GCNConv(16, num\_classes)

def forward(self, data):

x, edge\_index = data.x, data.edge\_index

x = F.relu(self.conv1(x, edge\_index))

x = F.dropout(x, training=self.training)

x = self.conv2(x, edge\_index)

return F.log\_softmax(x, dim=1)