Statistical Analysis of Deepfake Images and Videos in Smart City Applications

ANAND M K

242CS008

Introduction

- Deepfake technology uses advanced machine learning algorithms (e.g., GANs) to create synthetic media.
- Risks in smart cities: Surveillance, traffic monitoring, and public safety rely on authentic media.
- Focus: Statistical analysis of deepfake datasets to identify patterns and anomalies.

Datasets Used

- **FaceForensics**: Real and manipulated videos.
- **OpenForensics**: Real and synthetic images.
- **UADFV**: Real and deepfake videos.

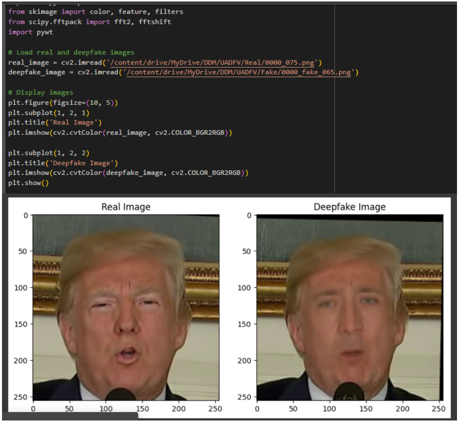

Figure: Screenshot: Loading datasets in Colab.

Descriptive Statistics

- **Mean, Median, Mode**: Central tendency of pixel intensities.
- **Variance, Standard Deviation**: Spread of pixel values.
- **Histograms**: Distribution of pixel intensities.
- **Correlation**: Relationship between pixel values.
- **Temporal Statistics**: Frame-by-frame changes in videos.

Mean, Median, and Mode of Pixel Intensities Calculate the mean, median, and mode for both images.

```python
import numpy as np

# Function to calculate statistics
def calculate_stats(image):
    mean = np.mean(image)
    median = np.median(image)
    # Calculate mode using numpy
    values, counts = np.unique(image.flatten(), return_counts=True)
    mode = values[np.argmax(counts)]  # Mode is the value with the highest count
    return mean, median, mode

# Calculate for real image
real_mean, real_median, real_mode = calculate_stats(real_image)
print(f"Real Image - Mean: {real_mean}, Median: {real_median}, Mode: {real_mode}")

# Calculate for deepfake image
deepfake_mean, deepfake_median, deepfake_mode = calculate_stats(deepfake_image)
print(f"Deepfake Image - Mean: {deepfake_mean}, Median: {deepfake_median}, Mode: {deepfake_mode}")
```

```
Real Image - Mean: 101.22123718261719, Median: 95.0, Mode: 84
Deepfake Image - Mean: 94.78656005859375, Median: 91.0, Mode: 0
```

Figure: Screenshot: Calculate the mean, median, and mode.

Descriptive Statistics

```
# Plot histograms
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.hist(real_image.ravel(), bins=256, range=(0, 256), color='black')
plt.title('Real Image Histogram')

plt.subplot(1, 2, 2)
plt.hist(deepfake_image.ravel(), bins=256, range=(0, 256), color='black')
plt.title('Deepfake Image Histogram')

plt.show()
```

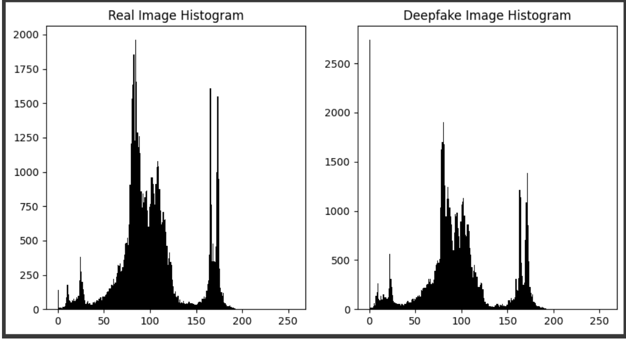

Figure: Screenshot: Plot histograms.

Descriptive Statistics

Function to calculate variance and standard deviation

```
[ ] # Function to calculate variance and standard deviation
    def calculate_variance_std(image):
        variance = np.var(image)
        std_dev = np.std(image)
        return variance, std_dev

    # Calculate for real image
    real_variance, real_std = calculate_variance_std(real_image)
    print(f"Real Image - Variance: {real_variance}, Std Dev: {real_std}")

    # Calculate for deepfake image
    deepfake_variance, deepfake_std = calculate_variance_std(deepfake_image)
    print(f"Deepfake Image - Variance: {deepfake_variance}, Std Dev: {deepfake_std}")
```

```
Real Image - Variance: 1392.5735058912542, Std Dev: 37.317201206564974
Deepfake Image - Variance: 1902.0035461746156, Std Dev: 43.611965630714415
```

Figure: Screenshot: Calculate variance and standard deviation.

Feature Extraction

- **Canny Edge Detection**: Detects edges in images.
- **Haralick Features**: Texture analysis using GLCM.
- **Sobel Edge Detection**: Gradient-based edge detection.
- **Local Binary Patterns (LBP)**: Texture descriptor.
- **Fourier Transform**: Frequency domain analysis.
- **Wavelet Transform**: Multi-resolution analysis.
- **Average Frame Difference**: Temporal analysis of videos.

Feature Extraction

Feature Extraction

Feature Extraction

Statistical Modeling

- **Gaussian Mixture Models (GMM)**: Models pixel intensity distributions.
- **Hidden Markov Models (HMM)**: Models temporal sequences in videos.

Gaussian Mixture Models (GMM)

```python
from sklearn.mixture import GaussianMixture

# Fit GMM to real image pixels
gmm_real = GaussianMixture(n_components=3, random_state=0)
gmm_real.fit(real_pixels)

# Fit GMM to deepfake image pixels
gmm_deepfake = GaussianMixture(n_components=3, random_state=0)
gmm_deepfake.fit(deepfake_pixels)

# Generate samples from the fitted GMMs
samples_real = gmm_real.sample(1000)[0]
samples_deepfake = gmm_deepfake.sample(1000)[0]

# Plot histograms of the samples
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.hist(samples_real, bins=50, density=True, alpha=0.6, color='blue')
plt.title('Real Image - GMM Samples')
plt.xlabel('Pixel Intensity')
plt.ylabel('Density')

plt.subplot(1, 2, 2)
plt.hist(samples_deepfake, bins=50, density=True, alpha=0.6, color='red')
plt.title('Deepfake Image - GMM Samples')
plt.xlabel('Pixel Intensity')
plt.ylabel('Density')
plt.show()
```

Figure: Screenshot: GMM code execution.

Statistical Modeling

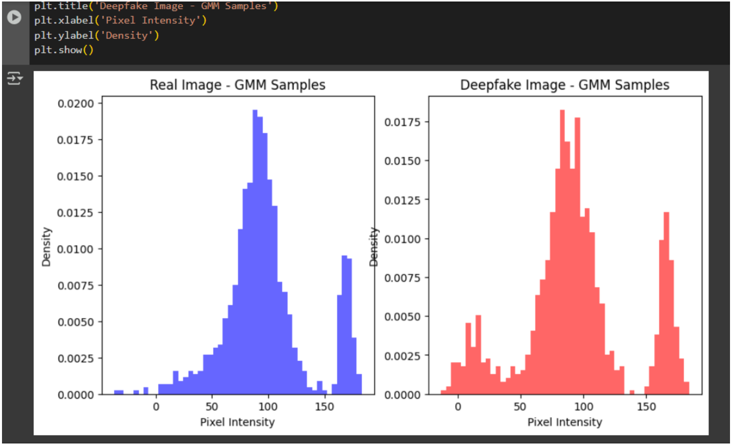

Figure: Screenshot: GMM code execution.

Statistical Modeling

Hidden Markov Models (HMM)

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from hmmlearn import hmm

def extract_features(video_path):
    """
    Extracts features from video frames.
    Feature: Average pixel intensity per frame.
    """
    cap = cv2.VideoCapture(video_path)
    features = []

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
        avg_intensity = np.mean(gray)  # Compute average pixel intensity
        features.append([avg_intensity])  # Reshape for HMM

    cap.release()
    return np.array(features)

def apply_hmm(features, num_states=3):
    """
    Fits an HMM and predicts hidden states.
    """
    model = hmm.GaussianHMM(n_components=num_states, covariance_type="diag", n_iter=100)
    model.fit(features)  # Train HMM
    hidden_states = model.predict(features)  # Predict hidden states
    return hidden_states

# Load videos and extract features
real_features = extract_features("/content/drive/MyDrive/DDM/FF+/Real/02__talking_against_wall.mp4")
deepfake_features = extract_features("/content/drive/MyDrive/DDM/FF+/Fake/02_25__talking_against_wall__Z7FQ69VP.mp4")
```

Figure: Screenshot: HMM code execution.

Statistical Modeling

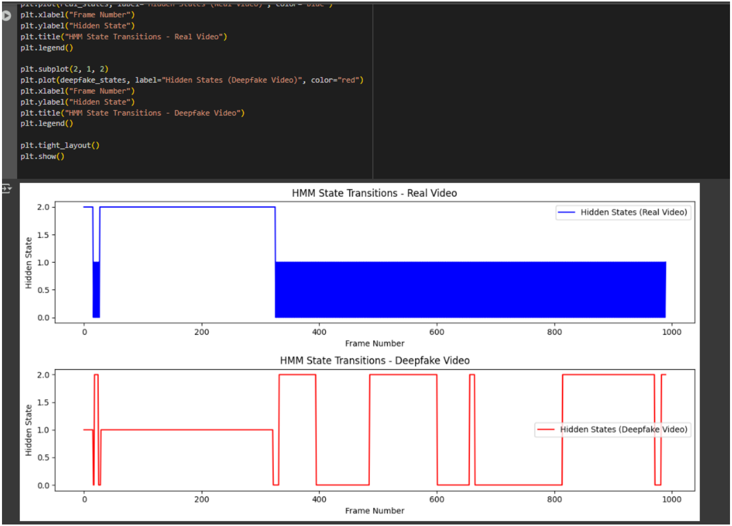

Figure: Screenshot: HMM code execution.

Statistical Tools

- **OpenCV**: Image and video processing.
- **Scikit-Image**: Texture and edge analysis.
- **Matlab**: Fourier and fractal analysis.
- **PyTorch/TensorFlow**: Deep learning-based analysis.
- **NumPy/SciPy**: Basic statistical operations.

Conclusion

- Statistical methods (descriptive statistics, feature extraction, modeling) effectively differentiate real and deepfake media.
- Tools like OpenCV, Scikit-Learn, and HMMLearn were instrumental in implementation.
- Future work: Integrate deep learning models and explore larger datasets.
- Overall, statistical analysis provides a strong foundation for deepfake detection.