Data Preparation

In [ ]:

```
import os
import cv2
import numpy as np
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Dataset paths
REAL_PATH = "/kaggle/input/faceforensics/FF++/real"
FAKE_PATH = "/kaggle/input/faceforensics/FF++/fake"
OUTPUT_FRAME_SIZE = (128, 128)  # Frame dimensions
FRAME_COUNT = 10  # Number of frames to extract per video
MAX_VIDEOS = 700  # Number of videos to process from each category

# Function to extract frames from a video
def extract_frames(video_path, output_size=(128, 128), frame_count=10):
    cap = cv2.VideoCapture(video_path)
    frames = []
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    step = max(total_frames // frame_count, 1)  # Uniform sampling

    for i in range(frame_count):
        cap.set(cv2.CAP_PROP_POS_FRAMES, i * step)
        ret, frame = cap.read()
        if not ret:
            break
        frame = cv2.resize(frame, output_size)
        frames.append(frame)
    cap.release()
    return np.array(frames)

# Prepare data and labels
data = []
labels = []

# Process real videos
print("Processing real videos...")
real_videos = os.listdir(REAL_PATH)[:MAX_VIDEOS]   # Limit to 300 videos
for video_file in tqdm(real_videos):
    video_path = os.path.join(REAL_PATH, video_file)
    frames = extract_frames(video_path, output_size=OUTPUT_FRAME_SIZE, frame_count=FRA
    if len(frames) == FRAME_COUNT:  # Ensure correct frame count
        data.append(frames)
        labels.append(0)  # Label 0 for real

# Process fake videos
print("Processing fake videos...")
fake_videos = os.listdir(FAKE_PATH)[:MAX_VIDEOS]  # Limit to 300 videos
for video_file in tqdm(fake_videos):
    video_path = os.path.join(FAKE_PATH, video_file)
    frames = extract_frames(video_path, output_size=OUTPUT_FRAME_SIZE, frame_count=FRA
    if len(frames) == FRAME_COUNT:
        data.append(frames)
        labels.append(1)  # Label 1 for fake

# Convert to numpy arrays
data = np.array(data)  # Shape: (num_videos, num_frames, 128, 128, 3)
labels = np.array(labels)

# Split into train, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(data, labels, test_size=0.3, rando
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_

# Normalize data
X_train = X_train / 255.0
X_val = X_val / 255.0
X_test = X_test / 255.0

# Convert labels to categorical
y_train = to_categorical(y_train, num_classes=2)
y_val = to_categorical(y_val, num_classes=2)
y_test = to_categorical(y_test, num_classes=2)

print(f"Data shapes: Train - {X_train.shape}, Validation - {X_val.shape}, Test - {X_te
```

Data Augmentation

In [ ]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Augment frames
datagen = ImageDataGenerator(
    horizontal_flip=True,
    rotation_range=10,
    zoom_range=0.1,
    brightness_range=[0.8, 1.2]
)

# Function to augment extracted frames
def augment_frames(frames):
    augmented_frames = []
    for frame in frames:
        frame = datagen.random_transform(frame)
        augmented_frames.append(frame)
    return np.array(augmented_frames)

# Augment training data
augmented_data = []
augmented_labels = []

for i in range(len(X_train)):
    augmented_frames = augment_frames(X_train[i])
    augmented_data.append(augmented_frames)
    augmented_labels.append(y_train[i])

# Combine original and augmented data
X_train_augmented = np.concatenate((X_train, np.array(augmented_data)))
y_train_augmented = np.concatenate((y_train, np.array(augmented_labels)))

print(f"Augmented Train Data: {X_train_augmented.shape}")
```

Model Architecture

In [ ]:

```
import tensorflow as tf
from tensorflow.keras.applications import Xception
from tensorflow.keras.layers import Dense, Flatten, TimeDistributed, LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout

# Define model
def build_improved_model(input_shape=(FRAME_COUNT, 128, 128, 3)):
    model = Sequential([
        TimeDistributed(Xception(weights='imagenet', include_top=False, input_shape=(1
        TimeDistributed(Flatten()),
        Dropout(0.5),  # Add dropout for regularization
        LSTM(128, return_sequences=False),
        Dropout(0.5),  # Add dropout
        Dense(64, activation='relu'),
        Dense(2, activation='softmax')
    ])
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

model = build_improved_model()
model.summary()
```

Model Training

In [ ]:

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

# Model checkpoint to save the best model in .keras format
checkpoint = ModelCheckpoint(
    "deepfake_detection_model.keras",  # Change to .keras
    monitor="val_accuracy",
    save_best_only=True,
    verbose=1
)

# Reduce learning rate on plateau
lr_scheduler = ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.5,
    patience=3,
    verbose=1
)

# Train the model
history = model.fit(
    X_train_augmented, y_train_augmented,
    validation_data=(X_val, y_val),
    epochs=40,
    batch_size=10,
    callbacks=[checkpoint, lr_scheduler]
)
model.save("deepfake_detection_model.keras")
```

Model Testing

In [ ]:

```python
from sklearn.metrics import classification_report, accuracy_score

# Load the best saved model
from tensorflow.keras.models import load_model
model = load_model('deepfake_detection_model.keras')

# Evaluate on test set
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Metrics
accuracy = accuracy_score(y_true, y_pred_classes)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Precision, Recall, F1-Score
print("Classification Report:")
print(classification_report(y_true, y_pred_classes, target_names=['REAL', 'FAKE']))
```

In [ ]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import numpy as np

# Plot accuracy and loss graphs
def plot_training_history(history):
    # Accuracy
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Compute confusion matrix
def plot_confusion_matrix(model, X_test, y_test):
    # Get predictions
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true = np.argmax(y_test, axis=1)
    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred_classes)
    cm_labels = ['Real', 'Fake']

    # Plot confusion matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=cm_labels, yticklab
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()

    # Classification report
    print(classification_report(y_true, y_pred_classes, target_names=cm_labels))

# Plot training history
plot_training_history(history)

# Plot confusion matrix
plot_confusion_matrix(model, X_test, y_test)
```

Real Time Detection

In [ ]:

```
import os
import cv2
import numpy as np
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [ ]:

```
# Function to extract frames from a video
def extract_frames(video_path, output_size=(128, 128), frame_count=10):
    cap = cv2.VideoCapture(video_path)
    frames = []
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    step = max(total_frames // frame_count, 1)  # Uniform sampling

    for i in range(frame_count):
        cap.set(cv2.CAP_PROP_POS_FRAMES, i * step)
        ret, frame = cap.read()
        if not ret:
            break
        frame = cv2.resize(frame, output_size)
        frames.append(frame)
    cap.release()
    return np.array(frames)
```

In [ ]:

```
from tensorflow.keras.models import load_model

# Load the model for real-time detection
loaded_model = load_model('deepfake_detection_model.keras')

def predict_video(video_path, model, output_size=(128, 128), frame_count=10):
    frames = extract_frames(video_path, output_size, frame_count)
    frames = frames / 255.0  # Normalize
    frames = np.expand_dims(frames, axis=0)  # Add batch dimension
    prediction = model.predict(frames)
    label = "FAKE" if np.argmax(prediction) == 1 else "REAL"
    confidence = prediction[0][np.argmax(prediction)]
    print(f"Prediction: {label} (Confidence: {confidence:.2f})")

REAL_PATH = "/kaggle/input/deep-fake-detection-dfd-entire-original-dataset/DFD_origina
FAKE_PATH = "/kaggle/input/deep-fake-detection-dfd-entire-original-dataset/DFD_manipul
# Test prediction on a video
real_sample_path = os.path.join(REAL_PATH, "/kaggle/input/deepfake-testing-videos/mode
fake_sample_path = os.path.join(FAKE_PATH, "/kaggle/input/deepfake-testing-videos/mode

print("Real Video Prediction:")
predict_video(real_sample_path,loaded_model)

print("Fake Video Prediction:")
predict_video(fake_sample_path,loaded_model)
```

In [ ]: