# Deepfake Detection Project Guide

## Project Goal

To build a CNN-based model that can detect deepfake images or videos by distinguishing between real and altered media.

## Learning Objectives

1. Understand CNNs and their role in image classification.
2. Learn to preprocess image data for deepfake detection.
3. Build and train a CNN model to classify deepfake vs. real images.
4. Evaluate and document model performance.

---

## Project Breakdown

We'll provide detailed code snippets and explanations for each step.

---

## 1. Environment Setup

### Libraries

Ensure you have the necessary libraries installed.

```
pip install tensorflow keras opencv-python matplotlib numpy
```

### Imports

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Fla
tten, Dense, Dropout
```

```
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
```

## 2. Data Collection and Preprocessing

### 1. Collecting Data

- Use an existing dataset, like the **FaceForensics++** dataset or similar.
- Alternatively, collect a sample dataset with both real and fake images.

### 2. Preprocessing Images

Define an image processing function to resize images and normalize pixel values.

```
IMG_SIZE = 64

def preprocess_image(image_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
    image = image / 255.0  # Normalize pixel values to [0, 1]
    return image
```

### 3. Loading Dataset

Prepare lists of images and labels.

```
real_images = [preprocess_image(f'real/{img}') for img in os.
listdir('real')]
fake_images = [preprocess_image(f'fake/{img}') for img in os.
listdir('fake')]
X = np.array(real_images + fake_images)
```

```
y = np.array([0] * len(real_images) + [1] * len(fake_images))
# 0: Real, 1: Fake
```

## 4. Train-Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, tes
t_size=0.2, random_state=42)
```

## 3. Model Building

### CNN Architecture

Start with a basic CNN for image classification and gradually increase complexity based on your data and results.

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SI
ZE, IMG_SIZE, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Binary classification
])

model.compile(optimizer='adam', loss='binary_crossentropy', m
etrics=['accuracy'])
model.summary(
```

## 4. Model Training

Train the model on your dataset with a validation split.

```
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2, batch_size=32)
```

---

## 5. Model Evaluation

1. **Loss and Accuracy Plot**
   - Plot accuracy and loss over epochs to monitor training.

```
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

2. **Performance on Test Set**

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.2f}")
```

3. **Confusion Matrix**

   Generate a confusion matrix to analyze the classification performance.

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

y_pred = (model.predict(X_test) > 0.5).astype("int32")
cm = confusion_matrix(y_test, y_pred)
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xtickla
bels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print(classification_report(y_test, y_pred))
```

## 6. Model Improvement (Optional)

If initial results are unsatisfactory, consider:

- **Data Augmentation**: Increase dataset variability by flipping, rotating, or altering brightness of images.

  ```
  from tensorflow.keras.preprocessing.image import ImageData
  Generator

  datagen = ImageDataGenerator(rotation_range=20, width_shif
  t_range=0.1, height_shift_range=0.1, horizontal_flip=True)
  datagen.fit(X_train)
  ```

- **Hyperparameter Tuning**: Experiment with different kernel sizes, number of filters, or optimizers.
- **Regularization Techniques**: Use dropout or batch normalization to prevent overfitting.

## 7. Documentation and Reporting

1. **Document the Process**: Include the following sections:
   - Project overview and objectives.
   - Data sources and preprocessing steps.
   - Model architecture, rationale for choices, and tuning approaches.
   - Evaluation results and model improvements.

2. **Reflections**: Describe any challenges, what worked well, and ideas for further improvements.

---

## Deliverable

- **Notebook or Python Script**: Ensure all code is well-documented with comments explaining each step.
- **Project Report**: Summarize findings, challenges, and lessons learned. This can be in the form of a separate document or markdown cells within the notebook.