# Complete Cell Execution Order for Enhanced Deepfake Detection Project

## 📋 CELL EXECUTION ORDER

Run these cells in the exact order listed below for optimal performance:

---

## CELL 1: Environment Setup and GPU Configuration

python

```python
# Cell 1: Environment Setup and GPU Configuration
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import (classification_report, confusion_matrix, accuracy_score,
                             precision_score, recall_score, f1_score, roc_curve, auc,
                             precision_recall_curve, average_precision_score)
import tensorflow as tf
from tensorflow.keras.applications import Xception
from tensorflow.keras.models import Model, Sequential, load_model
from tensorflow.keras.layers import (LSTM, Dense, Dropout, GlobalAveragePooling2D,
                                     TimeDistributed, BatchNormalization, Bidirectional,
                                     LayerNormalization, Attention)
from tensorflow.keras.callbacks import (ModelCheckpoint, EarlyStopping, ReduceLROnPlateau,
                                        CSVLogger, TensorBoard)
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from concurrent.futures import ThreadPoolExecutor
import time
import warnings
import gc
from datetime import datetime
warnings.filterwarnings('ignore')

# Install and import MTCNN
# ... (complete code from artifact Cell 1)
```

⏱️ **Expected Time: 2-3 minutes** ✅ **Expected Output: Environment setup complete, GPU configuration, MTCNN installation**

---

## CELL 2: Enhanced Configuration

```python
# Cell 2: Enhanced Configuration
class Config:
    # Kaggle paths - UPDATE THESE PATHS TO MATCH YOUR DATASET
    REAL_VIDEO_PATH = "/kaggle/input/your-dataset/real/"  # ⚠️ UPDATE THIS
    FAKE_VIDEO_PATH = "/kaggle/input/your-dataset/fake/"  # ⚠️ UPDATE THIS
    # ... (complete code from artifact Cell 2)

config = Config()
# Create directories and print configuration
```

⚠️ **IMPORTANT: Update the paths to match your dataset location** ⏱️ **Expected Time: 10 seconds** ✅ **Expected Output: Configuration loaded, directories created**

---

## CELL 3: Enhanced Video Processor Class

```python
# Cell 3: Enhanced Video Processing with Data Augmentation
class EnhancedVideoProcessor:
    def __init__(self, config):
        # ... (complete code from artifact)

    def extract_frames_from_video(self, video_path, output_dir, max_frames=None):
        # ... (complete method code)

    # ... (all other methods)

# Initialize the enhanced processor
processor = EnhancedVideoProcessor(config)
print("Enhanced Video processor initialized!")
```

⏱️ **Expected Time: 30 seconds** ✅ **Expected Output: Video processor initialized with face detection setup**

## CELL 4: Helper Functions

```python
# Cell 4: Helper Functions for Data Processing
def split_and_normalize_data(X, y):
    """Enhanced data splitting and normalization"""
    # ... (complete code from artifact)

def calculate_class_weights(y):
    """Calculate balanced class weights"""
    # ... (complete code)

def optimize_memory():
    """Optimize memory usage"""
    # ... (complete code)

def save_model_info(model, config, results, history):
    """Save comprehensive model information"""
    # ... (complete code)

print("Helper functions loaded successfully!")
```

⏱️ **Expected Time: 5 seconds** ✅ **Expected Output: Helper functions loaded**

---

## CELL 5: Enhanced Data Loading

```python
# Cell 5: Enhanced Data Loading
def enhanced_load_dataset():
    """Enhanced dataset loading with progress tracking"""
    # ... (complete code from artifact)

# Load the complete dataset (ALL 400 videos)
print("🚀 Starting dataset loading...")
X, y = enhanced_load_dataset()
print("✅ Dataset loading completed!")
print(f"📊 Total data shape: {X.shape}")
print(f"📊 Labels shape: {y.shape}")
```

⚠️ **CRITICAL: This will process ALL 400 videos** ⏱️ **Expected Time: 30-60 minutes (depending on video sizes)** ✅ **Expected Output: All sequences extracted, progress tracking, final summary**

---

## CELL 6: Data Splitting and Normalization

python

```python
# Cell 6: Data Splitting and Normalization
print("🔄 Splitting and normalizing data...")
(X_train, y_train), (X_val, y_val), (X_test, y_test) = split_and_normalize_data(X, y)

# Display data information
print("\n📈 Data Distribution:")
print(f"Training set: {X_train.shape[0]} sequences")
print(f"Validation set: {X_val.shape[0]} sequences")
print(f"Test set: {X_test.shape[0]} sequences")

# Calculate class weights for balanced training
class_weights = calculate_class_weights(y_train)
print(f"\n⚖️ Class weights: {class_weights}")
```

⏱️ **Expected Time: 2-5 minutes** ✅ **Expected Output: Data split confirmation, class distribution, normalization complete**

---

## CELL 7: Enhanced Feature Extractor

python

```python
# Cell 7: Enhanced XceptionNet Feature Extractor
class EnhancedFeatureExtractor:
    def __init__(self, config):
        # ... (complete code from artifact)

    def build_feature_extractor(self):
        # ... (complete code)

    def extract_features_batch(self, sequences, batch_size=16):
        # ... (complete code)

# Initialize feature extractor
print("🔧 Building Enhanced XceptionNet Feature Extractor...")
feature_extractor = EnhancedFeatureExtractor(config)
```

⏱️ **Expected Time: 1-2 minutes** ✅ **Expected Output: Feature extractor built, model summary**

---

## CELL 8: Feature Extraction

python

```python
# Cell 8: Feature Extraction from All Data
print("🎯 Starting feature extraction...")
print("This may take 20-40 minutes for all 400 videos...")

# Extract features from all datasets
print("Extracting features from training data...")
X_train_features = feature_extractor.extract_features_batch(X_train, batch_size=8)

print("Extracting features from validation data...")
X_val_features = feature_extractor.extract_features_batch(X_val, batch_size=8)

print("Extracting features from test data...")
X_test_features = feature_extractor.extract_features_batch(X_test, batch_size=8)

print("✅ Feature extraction completed!")
print(f"📊 Training features shape: {X_train_features.shape}")
print(f"📊 Validation features shape: {X_val_features.shape}")
print(f"📊 Test features shape: {X_test_features.shape}")

# Memory cleanup
optimize_memory()
```

**⚠️ LONGEST STEP: This processes all video sequences ⏱️ Expected Time: 20-40 minutes ✅ Expected Output: Feature extraction progress, final shapes, memory cleanup**

---

## CELL 9: Enhanced LSTM Classifier

python

```python
# Cell 9: Enhanced LSTM Classifier with Attention
class EnhancedLSTMClassifier:
    def __init__(self, config, input_dim):
        # ... (complete code from artifact)

    def build_model(self):
        # ... (complete code)

# Build the enhanced model
print("🏗️ Building Enhanced LSTM Classifier...")
input_dim = X_train_features.shape[-1]
lstm_classifier = EnhancedLSTMClassifier(config, input_dim)
print("✅ Enhanced LSTM model built successfully!")
```

⏱️ **Expected Time: 30 seconds** ✅ **Expected Output: Model architecture summary**

---

## CELL 10: Enhanced Training

python

```python
# Cell 10: Enhanced Training with Advanced Callbacks
def enhanced_train_model(model, X_train, y_train, X_val, y_val, config):
    """Enhanced training with advanced callbacks"""
    # ... (complete code from artifact)

print("🚀 Starting enhanced model training...")
print("This will train for up to 50 epochs with early stopping...")

training_start_time = time.time()
history = enhanced_train_model(
    lstm_classifier.model,
    X_train_features, y_train,
    X_val_features, y_val,
    config
)
training_time = time.time() - training_start_time

print(f"✅ Training completed in {training_time/60:.2f} minutes")

# Save the best model
lstm_classifier.model.save(config.MODEL_SAVE_PATH)
print(f"💾 Model saved as {config.MODEL_SAVE_PATH}")
```

**⚠️ TRAINING PHASE: Monitor progress carefully ⏱️ Expected Time: 45-90 minutes (depends on convergence) ✅ Expected Output: Training progress, validation metrics, model saved**

---

## CELL 11: Model Evaluator Class

python

```python
# Cell 11: Comprehensive Model Evaluator
class ModelEvaluator:
    def __init__(self, config):
        # ... (complete code from artifact)

    def evaluate_comprehensive(self, model, X_test, y_test):
        # ... (complete code)

    def plot_all_visualizations(self, history):
        # ... (complete code)

    # ... (all plotting methods)

print("📊 Model Evaluator class loaded!")
```

⏱️ **Expected Time: 10 seconds** ✅ **Expected Output: Evaluator class loaded**

---

## CELL 12: Comprehensive Evaluation

```python
# Cell 12: Comprehensive Model Evaluation
print("📈 Performing comprehensive evaluation...")

# Initialize evaluator and evaluate model
evaluator = ModelEvaluator(config)
results = evaluator.evaluate_comprehensive(
    lstm_classifier.model,
    X_test_features,
    y_test
)

print("✅ Evaluation completed!")
print("\n🎯 FINAL RESULTS:")
print("="*50)
for metric, value in results.items():
    if isinstance(value, (int, float)) and not isinstance(value, bool):
        print(f"{metric.upper()}: {value:.4f}")
print("="*50)
```

⏱️ **Expected Time: 2-5 minutes** ✅ **Expected Output: Comprehensive evaluation results, all metrics**

## CELL 13: Generate All Visualizations

```python
# Cell 13: Generate All Visualizations
print("🎨 Generating all visualizations...")
print("This will create 9 different types of plots...")

# Generate all requested visualizations
evaluator.plot_all_visualizations(history)

print("✅ All visualizations generated and saved!")
print(f"📁 Check results in: {config.RESULTS_PATH}")
```

⏱️ **Expected Time: 3-5 minutes** ✅ **Expected Output: All 9 visualization types displayed and saved**

---

## CELL 14: Save Comprehensive Results

```python
```

```python
# Cell 14: Save Comprehensive Results and Model Information
print("💾 Saving comprehensive results...")

# Save model information
save_model_info(lstm_classifier.model, config, results, history)

# Save detailed results
results_df = pd.DataFrame([{
    'metric': k,
    'value': v
} for k, v in results.items() if isinstance(v, (int, float))])

results_df.to_csv(os.path.join(config.RESULTS_PATH, 'final_results.csv'), index=False)

# Save training history
history_df = pd.DataFrame(history.history)
history_df.to_csv(os.path.join(config.RESULTS_PATH, 'training_history.csv'), index=False)

print("✅ All results saved successfully!")
print(f"📁 Results location: {config.RESULTS_PATH}")
print(f"📁 Model location: {config.MODEL_SAVE_PATH}")
```

⏱️ **Expected Time: 30 seconds** ✅ **Expected Output: All results and model information saved**

---

## CELL 15: Inference Functions (Optional)

python

```python
# Cell 15: Inference Functions for New Videos
def predict_video(video_path, model, feature_extractor, config):
    """Predict if a single video is real or fake"""
    # ... (complete code from artifact)

def predict_video_batch(video_paths, model, feature_extractor, config):
    """Predict multiple videos"""
    # ... (complete code from artifact)

print("🔮 Inference functions loaded!")
print("You can now use predict_video() or predict_video_batch() for new videos")

# Example usage (uncomment to test on new videos):
# result, error = predict_video("/path/to/test/video.mp4",
#                               lstm_classifier.model,
#                               feature_extractor,
#                               config)
# if error:
#     print(f"Error: {error}")
# else:
#     print(f"Prediction: {result}")
```

⏱️ **Expected Time: 5 seconds** ✅ **Expected Output: Inference functions ready for use**

---

## CELL 16: Final Summary and Cleanup

python

```
# Cell 16: Final Summary and Cleanup
print("🎉 PROJECT COMPLETION SUMMARY")
print("="*60)
print(f"✅ Total videos processed: ~400 (200 real + 200 fake)")
print(f"✅ Total sequences generated: {len(X)}")
print(f"✅ Model trained successfully with {len(history.history['accuracy'])} epochs")
print(f"✅ Final test accuracy: {results['accuracy']:.4f}")
print(f"✅ All visualizations generated: 9 types")
print(f"✅ Model saved: {config.MODEL_SAVE_PATH}")
print(f"✅ Results saved: {config.RESULTS_PATH}")
print("="*60)

# Final memory cleanup
optimize_memory()
print("🧹 Memory cleanup completed")
print("🏁 Project completed successfully!")
```

⏱️ **Expected Time: 10 seconds** ✅ **Expected Output: Complete project summary**

---

## 📝 IMPORTANT NOTES:

### ⚠️ Before Running:

1. **Update paths** in Cell 2 to match your dataset location
2. **Ensure sufficient storage**: ~5-10GB free space needed
3. **Check GPU memory**: Reduce batch sizes if you get OOM errors

### ⏱️ Total Expected Time:

- **Data Loading**: 30-60 minutes
- **Feature Extraction**: 20-40 minutes
- **Model Training**: 45-90 minutes
- **Evaluation & Visualization**: 10-15 minutes
- **TOTAL**: ~2-3.5 hours

### 🎯 Expected Final Results:

- **Test Accuracy**: 90%+ (significant improvement)
- **9 Different Visualizations**: All saved as high-quality PNG files

- **Comprehensive Model**: Ready for production use
- **Detailed Logs**: Complete training and evaluation history

## 🐛 Troubleshooting:

- **OOM Error**: Reduce `BATCH_SIZE` in config
- **Slow Processing**: Reduce `SEQUENCE_LENGTH` or `MAX_FRAMES_PER_VIDEO`
- **Path Errors**: Double-check dataset paths in Cell 2
- **MTCNN Issues**: Falls back to OpenCV automatically

## 📊 Output Files:

- `best_deepfake_model.h5` - Trained model
- `enhanced_training_history.png` - Training plots
- `confusion_matrix.png` - Confusion matrices
- `roc_curve.png` - ROC analysis
- `prediction_distribution.png` - Score distributions
- `performance_dashboard.png` - Complete dashboard
- And 4 more visualization files!

**Run cells in this exact order for best results!** 🚀