# DeepGuard AI: Technical Model Architecture Documentation

## Table of Contents

## Project Overview

**DeepGuard AI** is an advanced deepfake detection system that combines computer vision, deep learning, and web technologies to identify manipulated media content. The system can analyze both images and videos to determine the authenticity of facial content with high accuracy.

### Key Features

- **Dual Detection Capability**: Supports both image and video deepfake detection
- **Real-time Processing**: Optimized for fast inference on CPU and GPU
- **Web Interface**: User-friendly Flask-based web application
- **Robust Architecture**: Built on proven deep learning frameworks

### Technology Stack

- **Framework**: TensorFlow 2.20.0 with Keras 3.11.3
- **Backend**: Flask 3.0.3 (Python Web Framework)
- **Computer Vision**: OpenCV 4.10.0 + MTCNN for face detection
- **Frontend**: HTML5, CSS3, JavaScript
- **Model Format**: Keras (.keras) format for TensorFlow 2.20.0 compatibility

## Model Architecture

### Base Architecture: MobileNet with Transfer Learning

The DeepGuard AI model is built upon **MobileNet** architecture, specifically chosen for its:

- **Efficiency**: Optimized for mobile and edge deployment
- **Speed**: Fast inference time suitable for real-time applications

- **Accuracy**: Proven performance on image classification tasks
- **Lightweight**: Reduced parameter count compared to traditional CNNs

## Detailed Architecture Breakdown

### 1. Base Model: MobileNet (Pre-trained on ImageNet)

```
Input Shape: (224, 224, 3)
Base Model Parameters: 3,228,864 (pre-trained)
Architecture: Depthwise Separable Convolutions
```

### 2. Feature Extraction Layers (86 layers total)

The MobileNet base consists of:

- **1 Standard Convolution Layer** (conv1)
- **13 Depthwise Separable Convolution Blocks**
- **Batch Normalization** after each convolution
- **ReLU Activation Functions**
- **Strategic Downsampling** through strided convolutions

**Layer Progression:**

```
1. Input Layer: (224, 224, 3)
2. conv1: (112, 112, 32) - Initial feature extraction
3. Depthwise Blocks 1-2: (112, 112, 64) → (56, 56, 128)
4. Depthwise Blocks 3-4: (56, 56, 128) → (28, 28, 256)
5. Depthwise Blocks 5-6: (28, 28, 256) → (14, 14, 512)
6. Depthwise Blocks 7-12: (14, 14, 512) - Deep feature learning
7. Depthwise Block 13: (7, 7, 1024) - High-level features
```

### 3. Custom Classification Head (9 layers)

**Added layers for deepfake detection:**

```
# Global Average Pooling
GlobalAveragePooling2D() → (1024,)

# Batch Normalization
BatchNormalization() → (1024,)

# Dense Layer 1
Dense(512, activation='relu') → (512,)
Dropout(0.5)

# Dense Layer 2
```

```
Dense(256, activation='relu') → (256,)
Dropout(0.3)

# Dense Layer 3
Dense(128, activation='relu') → (128,)
Dropout(0.3)

# Output Layer
Dense(1, activation='sigmoid') → (1,)  # Binary classification
```

### Model Statistics

- **Total Parameters**: 9,029,829 (34.45 MB)
- **Trainable Parameters**: 2,553,857 (9.74 MB)
- **Non-trainable Parameters**: 1,368,256 (5.22 MB)
- **Optimizer Parameters**: 5,107,716 (19.48 MB)

---

## Pre-trained Models Used

### 1. MobileNet (ImageNet Pre-trained)

- **Source**: TensorFlow/Keras Applications
- **Training Dataset**: ImageNet (1.2M+ images, 1000 classes)
- **Purpose**: Feature extraction backbone
- **Weights**: `imagenet` weights (frozen during initial training)
- **Alpha Parameter**: 1.0 (full model width)

### 2. MTCNN (Multi-task CNN)

- **Source**: `mtcnn` Python package
- **Purpose**: Face detection in videos
- **Components**:
  - **P-Net**: Proposal Network for candidate windows
  - **R-Net**: Refine Network for face/non-face classification
  - **O-Net**: Output Network for facial landmarks
- **Input**: RGB images (variable size)
- **Output**: Bounding boxes with confidence scores

---

## Training Process

### Two-Phase Training Strategy

#### Phase 1: Frozen Base Model Training

```
# Hyperparameters
EPOCHS_PHASE1 = 3
BATCH_SIZE = 8
```

```
LEARNING_RATE = 1e-4
IMG_SIZE = (224, 224)

# Optimizer
optimizer = Adam(learning_rate=1e-4)
loss = 'binary_crossentropy'
metrics = ['accuracy']
```

**Process:**

1. Freeze all MobileNet layers (base_model.trainable = False)
2. Train only custom classification head
3. Conservative learning rate for stability
4. Early stopping based on validation accuracy

**Phase 2: Fine-tuning**

```
# Fine-tuning parameters
EPOCHS_PHASE2 = 2
LEARNING_RATE_FINE = 1e-5  # 10x lower than Phase 1
FINE_TUNE_FROM = -10  # Last 10 layers
```

**Process:**

1. Unfreeze last 10 layers of MobileNet
2. Reduce learning rate by 10x
3. Fine-tune with smaller learning rate
4. Prevent overfitting with careful monitoring

### Training Callbacks

```
callbacks = [
    EarlyStopping(
        monitor='val_accuracy',
        patience=2,
        restore_best_weights=True
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=1,
        min_lr=1e-7
    ),
    ModelCheckpoint(
        monitor='val_accuracy',
        save_best_only=True
    )
]
```

### Data Augmentation Strategy

```
train_datagen = ImageDataGenerator(
    rescale=1./255,            # Normalize to [0,1]
    rotation_range=10,         # ±10° rotation
    zoom_range=0.1,           # ±10% zoom
    horizontal_flip=True      # Random horizontal flip
)
```

---

## Data Pipeline

### Expected Dataset Structure

```
Dataset/
├── train/
│   ├── real/          # Authentic images
│   └── fake/          # Deepfake/manipulated images
├── validation/
│   ├── real/
│   └── fake/
└── test/
    ├── real/
    └── fake/
```

### Data Preprocessing Pipeline

**For Images:**

1. **Load Image**: Using `tf.keras.preprocessing.image.load_img()`
2. **Resize**: Target size (224, 224) pixels
3. **Normalize**: Pixel values to [0, 1] range
4. **Batch Formation**: Expand dimensions for model input

**For Videos:**

1. **Video Loading**: OpenCV `cv2.VideoCapture()`
2. **Frame Sampling**: Every 2 frames per second (configurable)
3. **Face Detection**: MTCNN for face region extraction
4. **Face Preprocessing**:
   - Resize to (224, 224)
   - Normalize to [0, 1]
   - Batch processing
5. **Voting System**: Aggregate predictions across frames

---

## Feature Engineering

### Image Feature Extraction

- **Low-level Features**: Edges, textures, colors (handled by early MobileNet layers)
- **Mid-level Features**: Object parts, shapes (middle layers)
- **High-level Features**: Complex patterns, semantic information (deeper layers)
- **Custom Features**: Deepfake-specific patterns learned in classification head

### Video Feature Processing

- **Temporal Sampling**: Strategic frame selection to capture temporal inconsistencies
- **Face-focused Analysis**: Only analyze facial regions to reduce noise
- **Confidence Aggregation**: Combine multiple frame predictions for final decision

### Preprocessing Parameters

```
# Image preprocessing
IMG_SIZE = (224, 224)          # MobileNet standard input
NORMALIZATION = [0, 1]         # Pixel value range
COLOR_CHANNELS = 3             # RGB format

# Video preprocessing
FRAME_SAMPLING_RATE = 2        # Frames per second to analyze
MIN_FACE_CONFIDENCE = 0.9      # MTCNN detection threshold
FACE_PADDING = 0.1             # Extra margin around detected face
```

---

## Model Performance

### Architecture Efficiency

- **Model Size**: 34.45 MB (optimized for deployment)
- **Inference Time**:
  - Images: ~100-200ms per image (CPU)
  - Videos: ~50-100ms per frame (CPU)
- **Memory Usage**: ~1-2 GB RAM during inference

### Training Metrics

- **Training Time**: ~3 hours total (CPU optimized)
  - Phase 1: ~2 hours (frozen base)
  - Phase 2: ~1 hour (fine-tuning)
- **Batch Size**: 8 (optimized for CPU training)
- **Total Training Steps**: Limited for CPU efficiency

### Performance Characteristics

```
# Expected performance ranges (dataset dependent)
Training Accuracy: 85-95%
Validation Accuracy: 80-90%
Test Accuracy: 75-85%

# Inference metrics
Images/second (CPU): 5-10
Frames/second (Video): 2-5
```

---

## Deployment Architecture

### Web Application Stack

#### Backend (Flask)

```
# Core components
app = Flask(__name__)
model = tf.keras.models.load_model('models/deepfake_keras3_compatible.keras')
detector = MTCNN()  # Face detection

# Configuration
UPLOAD_FOLDER = 'static/uploads'
TEMP_FOLDER = 'temp'
MAX_FILE_SIZE = 100MB  # Configurable
```

#### File Handling

- **Supported Image Formats**: PNG, JPG, JPEG, GIF, BMP
- **Supported Video Formats**: MP4, AVI, MOV, MKV
- **Security**: Secure filename handling with Werkzeug
- **Storage**: Temporary file management with cleanup

#### API Endpoints

```
# Route definitions
@app.route('/')                          # Landing page
@app.route('/image', methods=['GET', 'POST'])   # Image detection
@app.route('/video', methods=['GET', 'POST'])   # Video detection
@app.route('/reports')                   # Analysis reports
```

### Prediction Pipeline

#### Image Prediction Flow

```
def predict_image(image_path):
    # 1. Load and preprocess
    img = load_img(image_path, target_size=(224, 224))
    x = img_to_array(img) / 255.0
    x = np.expand_dims(x, axis=0)

    # 2. Model prediction
    pred = model.predict(x)[0][0]

    # 3. Classification
    label = 'Real' if pred > 0.5 else 'Deepfake'
    confidence = pred if pred > 0.5 else 1 - pred

    return label, confidence
```

**Video Prediction Flow**

```
def predict_video(video_path):
    # 1. Initialize counters
    fake_votes = 0
    real_votes = 0

    # 2. Process frames
    for frame in video_frames:
        faces = detector.detect_faces(frame)
        for face in faces:
            # Extract and preprocess face
            face_region = extract_face(frame, face['box'])
            face_input = preprocess_face(face_region)

            # Predict
            pred = model.predict(face_input)[0][0]

            # Vote
            if pred > 0.5:
                real_votes += 1
            else:
                fake_votes += 1

    # 3. Final decision
    result = "Real" if real_votes > fake_votes else "Fake"
    confidence = max(real_votes, fake_votes) / (real_votes + fake_votes)

    return result, confidence, real_votes, fake_votes
```

---

## API Specifications

### Image Detection API

**Request**

```
POST /image
Content-Type: multipart/form-data

file: [image_file]
```

**Response**

```
{
    "result": "Real" | "Deepfake",
    "confidence": "95.67%",
    "filename": "uploaded_image.jpg",
    "processing_time": "0.15s"
}
```

### Video Detection API

**Request**

```
POST /video
Content-Type: multipart/form-data

video: [video_file]
```

**Response**

```
{
    "result": "Real" | "Fake",
    "confidence": "87.23%",
    "real_votes": 45,
    "fake_votes": 12,
    "total_frames_analyzed": 57,
    "processing_time": "12.34s"
}
```

### Error Handling

```
{
    "error": "Error message description",
    "code": "ERROR_CODE",
    "details": "Additional error information"
}
```

---

## Technical Implementation Details

### Dependencies and Versions

```
# Core ML Framework
tensorflow==2.20.0              # Main deep learning framework
keras==3.11.3                   # High-level neural networks API

# Computer Vision
opencv-python==4.10.0.84        # Video processing and face detection
Pillow==11.3.0                  # Image processing
mtcnn==0.1.1                    # Face detection (auto-installed)

# Web Framework
Flask==3.0.3                    # Web application framework
Werkzeug==3.1.3                 # WSGI utilities

# Scientific Computing
numpy==2.3.3                    # Numerical computations
scipy==1.16.2                   # Scientific computing utilities

# Data Analysis & Visualization
matplotlib==3.10.6              # Plotting and visualization
seaborn==0.13.2                 # Statistical data visualization
pandas==2.3.2                   # Data manipulation and analysis
scikit-learn==1.7.2             # Machine learning utilities
```

### System Requirements

#### Minimum Requirements

```
CPU: 4+ cores, 2.5+ GHz
RAM: 8 GB
Storage: 10 GB free space
Python: 3.11+ (tested on 3.13.7)
OS: Windows 10+, Linux, macOS
```

#### Recommended Requirements

```
CPU: 8+ cores, 3.0+ GHz (Intel i7/AMD Ryzen 7+)
RAM: 16 GB
GPU: NVIDIA GTX 1060+ (optional, for faster inference)
```

```
Storage: 20 GB SSD
Python: 3.13.7
OS: Windows 11, Ubuntu 20.04+, macOS 12+
```

## File Structure

```
DeepGuard-AI-feature/
├── merged_app.py                 # Main Flask application
├── train_mobilenet_transfer.py   # Model training script
├── eda_analysis.py              # Exploratory data analysis
├── check.py                     # GPU availability check
├── requirements.txt             # Python dependencies
├── analyze_model.py             # Model architecture analysis
├── README.md                    # Project documentation
├── models/
│   └── deepfake_keras3_compatible.keras  # Trained model
├── static/
│   ├── css/
│   │   └── styles.css           # Web styling
│   ├── uploads/                 # User uploaded files
│   └── captured_faces/          # Processed face images
├── templates/
│   ├── base.html               # Base template
│   ├── index.html              # Home page
│   ├── image.html              # Image detection page
│   ├── video.html              # Video detection page
│   └── reports.html            # Reports page
├── temp/                       # Temporary file storage
└── Testing samples/            # Sample test data
    ├── Deep fake video/
    ├── Fake images (trained over ANN)/
    └── Real images (trained over ANN)/
```

## Configuration Options

### Model Configuration

```python
# Model parameters (configurable in merged_app.py)
MODEL_PATH = 'models/deepfake_keras3_compatible.keras'
IMG_SIZE = (224, 224)           # Input image dimensions
CONFIDENCE_THRESHOLD = 0.5      # Classification threshold

# File upload limits
MAX_CONTENT_LENGTH = 100 * 1024 * 1024  # 100MB
ALLOWED_IMAGE_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif', 'bmp'}
ALLOWED_VIDEO_EXTENSIONS = {'mp4', 'avi', 'mov', 'mkv'}
```

### Training Configuration

```
# Training parameters (train_mobilenet_transfer.py)
BATCH_SIZE = 8                  # Batch size for training
EPOCHS_PHASE1 = 3              # Frozen base training epochs
EPOCHS_PHASE2 = 2              # Fine-tuning epochs
LEARNING_RATE = 1e-4           # Initial learning rate
IMG_SIZE = (224, 224)          # Training image size

# Data augmentation
ROTATION_RANGE = 10            # Random rotation degrees
ZOOM_RANGE = 0.1              # Random zoom factor
HORIZONTAL_FLIP = True         # Enable horizontal flipping
```

## Security Considerations

### File Upload Security

- **Filename Sanitization**: Using `secure_filename()` from Werkzeug
- **File Type Validation**: Extension and MIME type checking
- **File Size Limits**: Configurable maximum upload sizes
- **Temporary File Cleanup**: Automatic cleanup after processing

### Model Security

- **Input Validation**: Image/video format verification
- **Error Handling**: Graceful error handling and logging
- **Resource Management**: Memory and CPU usage monitoring

## Performance Optimization

### CPU Optimization

```
# TensorFlow CPU optimizations
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '1'  # Enable oneDNN optimizations
tf.config.threading.set_inter_op_parallelism_threads(0)  # Use all cores
```

### Memory Management

```
# Efficient memory usage
tf.config.experimental.enable_memory_growth = True  # GPU memory growth
# Automatic garbage collection after processing
import gc; gc.collect()
```

**Batch Processing**

- Small batch sizes (8) for CPU training stability
- Frame sampling in videos to reduce processing time
- Efficient face detection with confidence thresholds

---

## Deployment Instructions

### Local Development Setup

```
# 1. Clone repository
git clone <repository-url>
cd DeepGuard-AI-feature

# 2. Create virtual environment
python -m venv .venv
.venv\Scripts\activate  # Windows
source .venv/bin/activate  # Linux/macOS

# 3. Install dependencies
pip install -r requirements.txt

# 4. Run application
python merged_app.py
```

### Production Deployment

```
# Using Gunicorn (Linux/macOS)
gunicorn -w 4 -b 0.0.0.0:5000 merged_app:app

# Using Waitress (Windows)
waitress-serve --host=0.0.0.0 --port=5000 merged_app:app
```

### Docker Deployment

```
FROM python:3.13-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .
EXPOSE 5000

CMD ["python", "merged_app.py"]
```

# Model Training Guide

## Preparing Training Data

1. **Organize Dataset**: Follow the expected directory structure
2. **Data Quality**: Ensure high-quality, diverse images
3. **Balance**: Maintain balanced real/fake ratios
4. **Size**: Minimum 1000 images per class recommended

## Training Process

```
# 1. Prepare dataset in correct structure
# 2. Adjust hyperparameters in train_mobilenet_transfer.py
# 3. Run training
python train_mobilenet_transfer.py

# 4. Monitor training progress
# Check console output for metrics and loss values

# 5. Evaluate trained model
python analyze_model.py
```

## Model Validation

```
# Test model performance
from sklearn.metrics import classification_report, confusion_matrix

# Load test data and generate predictions
# Calculate metrics: accuracy, precision, recall, F1-score
```

# Troubleshooting Guide

## Common Issues

### 1. TensorFlow Version Conflicts

```
# Solution: Use exact versions from requirements.txt
pip install tensorflow==2.20.0
```

### 2. Memory Issues

```
# Reduce batch size in training
BATCH_SIZE = 4  # Instead of 8

# Enable memory growth for GPU
tf.config.experimental.enable_memory_growth(gpu)
```

#### 3. MTCNN Installation Issues

```
# Install MTCNN separately
pip install mtcnn

# Or use alternative face detection
# Replace MTCNN with OpenCV Haar Cascades
```

#### 4. Model Loading Errors

```
# Ensure correct model format
model = tf.keras.models.load_model('path/to/model.keras')

# Check TensorFlow/Keras compatibility
print(tf.__version__)  # Should be 2.20.0
```

### Performance Optimization Tips

1. **CPU Training**: Use small batch sizes (4-8)
2. **GPU Training**: Increase batch size (16-32) if memory allows
3. **Inference Speed**: Use model quantization for faster prediction
4. **Memory Usage**: Enable mixed precision training

---

## Future Enhancements

### Model Improvements

- **Advanced Architectures**: EfficientNet, Vision Transformer (ViT)
- **Ensemble Methods**: Combine multiple models for better accuracy
- **Self-supervised Learning**: Leverage unlabeled data
- **Attention Mechanisms**: Focus on critical facial regions

### Technical Enhancements

- **Real-time Processing**: WebRTC integration for live video analysis
- **Mobile Deployment**: TensorFlow Lite conversion for mobile apps
- **Cloud Integration**: AWS/Azure deployment with auto-scaling
- **API Enhancement**: RESTful API with authentication and rate limiting

### Feature Additions

- **Batch Processing**: Multiple file upload and processing
- **Report Generation**: Detailed analysis reports with visualizations
- **Model Versioning**: A/B testing different model versions
- **User Management**: Authentication and user-specific history

---

## License and Citations

### Model Attribution

- **MobileNet**: Originally developed by Google Research
- **MTCNN**: Joint Face Detection and Alignment using Multi-task CNN
- **TensorFlow**: Open-source machine learning framework by Google

### Research References

```
@article{howard2017mobilenets,
  title={MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications},
  author={Howard, Andrew G and Zhu, Menglong and Chen, Bo and Kalenichenko, Dmitry and Wang, Weijun and Weyand, Tobias and Andreetto, Marco and Adam, Hartwig},
  journal={arXiv preprint arXiv:1704.04861},
  year={2017}
}

@article{zhang2016joint,
  title={Joint face detection and alignment using multitask cascaded convolutional networks},
  author={Zhang, Kaipeng and Zhang, Zhifeng and Li, Zhongfei and Qiao, Yu},
  journal={IEEE Signal Processing Letters},
  year={2016}
}
```

---

## Contact and Support

### Technical Documentation

- **Created**: October 2025
- **Version**: 1.0.0
- **Framework**: TensorFlow 2.20.0
- **Python**: 3.13.7

### Model Statistics Summary

```
Architecture: MobileNet + Custom Classification Head
Total Parameters: 9,029,829
Model Size: 34.45 MB
Input Shape: (224, 224, 3)
Output: Binary Classification (Real/Deepfake)
Training Strategy: Transfer Learning + Fine-tuning
Deployment: Flask Web Application
```

---

*This technical documentation provides comprehensive information about the DeepGuard AI deepfake detection system. For additional details or support, please refer to the source code and comments within the individual Python files.*