🤖 Guía de Modelos de Machine Learning

📋 Modelos Requeridos

El sistema requiere tres modelos principales de machine learning:

1. YOLOv8-face (Detección Facial)
2. MobileFaceNet (Embeddings Faciales)
3. Liveness Detector (Anti-spoofing)

🎯 1. YOLOv8-face - Detección de Rostros

Descripción

Modelo especializado de YOLO para detección de rostros con landmarks faciales.

Obtención del Modelo

Opción A: Modelo Pre-entrenado (Recomendado)

```
# Clonar repositorio con modelos
git clone https://github.com/akanametov/yolov8-face.git

# Los pesos pre-entrenados están disponibles en:
# https://github.com/akanametov/yolov8-face/releases

# Descargar yolov8n-face.pt
wget https://github.com/akanametov/yolov8-face/releases/download/v1.0/yolov8n-face.pt

# Mover al directorio correcto
mv yolov8n-face.pt /home/ubuntu/attendance_facial_recognition/data/models/
face_detection/
```

Opción B: YOLOv8 Base (Para desarrollo/pruebas)

```
# El script download_models.py descarga YOLOv8 base
python scripts/download_models.py

# Funciona para detección básica pero no está optimizado para rostros
```

Configuración

En `.env`:

```
FACE_DETECTION_CONFIDENCE=0.5  # Ajustar según precisión deseada
```

Características

- **Velocidad**: ~50-100 FPS (GPU), ~10-20 FPS (CPU)

- **Precisión**: Alta para rostros frontales
- **Landmarks**: 5 puntos (ojos, nariz, comisuras labiales)
- **Tamaño**: ~6MB

🧠 2. MobileFaceNet - Reconocimiento Facial

Descripción

Red neuronal ligera para generar embeddings faciales de 128 dimensiones.

Obtención del Modelo

Opción A: InsightFace (Recomendado)

```
# Instalar insightface
pip install insightface

# Descargar modelo
import insightface
app = insightface.app.FaceAnalysis()
app.prepare(ctx_id=0, det_size=(640, 640))

# El modelo se descarga automáticamente
# Ubicación: ~/.insightface/models/

# Copiar modelo al proyecto
cp -r ~/.insightface/models/mobilefacenet.pth \
   /home/ubuntu/attendance_facial_recognition/data/models/face_recognition/
```

Opción B: Entrenar Propio Modelo

Si tienes un dataset propio:

```
# Estructura del modelo (implementada en app/utils/face_recognition.py)
# Ver: https://github.com/deepinsight/insightface

# Dataset recomendado para entrenamiento:
# - MS-Celeb-1M
# - VGGFace2
# - Minimum 10,000 identidades
```

Opción C: Usar Modelo Simple (Solo desarrollo)

El código ya incluye una implementación simple de MobileFaceNet que funciona con inicialización aleatoria para desarrollo. **NO usar en producción**.

Configuración

En `.env`:

```
FACE_SIMILARITY_THRESHOLD=0.6  # Umbral de coincidencia (0-1)
EMBEDDING_DIMENSION=128  # Dimensión de embeddings
```

Características

- **Dimensión de embedding**: 128D
- **Velocidad**: ~20ms por rostro
- **Tamaño**: ~4MB
- **Precisión**: >99% con modelo entrenado

🛡️ 3. Liveness Detector - Anti-Spoofing

Descripción

Detector de vivacidad para prevenir ataques con fotos, videos o máscaras.

Obtención del Modelo

Opción A: Silent-Face-Anti-Spoofing (Recomendado)

```
# Clonar repositorio
git clone https://github.com/minivision-ai/Silent-Face-Anti-Spoofing.git

# Descargar modelo pre-entrenado
cd Silent-Face-Anti-Spoofing
# Seguir instrucciones del repositorio

# Copiar modelo
cp model/anti_spoof_model.onnx \
   /home/ubuntu/attendance_facial_recognition/data/models/liveness/liveness_model.pth
```

Opción B: Entrenar Modelo Propio

Datasets recomendados:
- **CASIA-SURF**: http://www.cbsr.ia.ac.cn/users/jjyan/CASIA-SURF/
- **Replay-Attack**: https://www.idiap.ch/dataset/replayattack
- **OULU-NPU**: https://sites.google.com/site/oulunpudatabase/

```
# Arquitectura simple CNN (implementada en app/utils/liveness_detector.py)
# Para producción, entrenar con uno de los datasets mencionados

# Estructura típica:
# - Input: Imagen 112x112
# - Conv layers con MaxPooling
# - FC layers
# - Output: [fake_prob, real_prob]
```

Opción C: Método Heurístico (Solo desarrollo)

El código incluye un método heurístico simple basado en:
- Varianza de Laplaciano (detección de blur)
- Análisis de textura
- Rango de colores

Limitaciones: Precisión ~70-80%, no recomendado para producción.

Configuración

En `.env`:

```
LIVENESS_ENABLED=True
LIVENESS_THRESHOLD=0.5  # Umbral de clasificación (0-1)
```

Características

- **Velocidad**: ~50ms por rostro
- **Precisión**: >95% con modelo entrenado
- **Tamaño**: ~2-5MB

📦 Instalación Completa de Modelos

Script Automatizado

```
#!/bin/bash
# install_models.sh

MODELS_DIR="/home/ubuntu/attendance_facial_recognition/data/models"

# Crear directorios
mkdir -p $MODELS_DIR/face_detection
mkdir -p $MODELS_DIR/face_recognition
mkdir -p $MODELS_DIR/liveness

# 1. YOLOv8-face
echo "Descargando YOLOv8-face..."
wget -O $MODELS_DIR/face_detection/yolov8n-face.pt \
    https://github.com/akanametov/yolov8-face/releases/download/v1.0/yolov8n-face.pt

# 2. MobileFaceNet (desde InsightFace)
echo "Descargando MobileFaceNet..."
pip install insightface
python -c "
import insightface
app = insightface.app.FaceAnalysis()
app.prepare(ctx_id=-1)
"

cp ~/.insightface/models/buffalo_l/w600k_r50.onnx \
    $MODELS_DIR/face_recognition/mobilefacenet.pth

# 3. Liveness (Silent-Face-Anti-Spoofing)
echo "Descargando Liveness Detector..."
git clone https://github.com/minivision-ai/Silent-Face-Anti-Spoofing.git /tmp/liveness
cp /tmp/liveness/resources/anti_spoof_models/2.7_80x80_MiniFASNetV2.pth \
    $MODELS_DIR/liveness/liveness_model.pth

echo "✓ Modelos instalados correctamente"
```

Ejecutar:

```
chmod +x install_models.sh
./install_models.sh
```

🔧 Verificación de Modelos

Script de Prueba

```
python scripts/test_system.py
```

Este script verifica:
- ✅ Modelos existen en las rutas correctas
- ✅ Modelos se cargan correctamente
- ✅ Inferencia funciona

Prueba Manual

```python
from app.utils import FaceDetector, FaceRecognizer, LivenessDetector
from config import Config
import cv2

# Cargar modelos
detector = FaceDetector(Config.FACE_DETECTION_MODEL)
recognizer = FaceRecognizer(Config.FACE_RECOGNITION_MODEL)
liveness = LivenessDetector(Config.LIVENESS_MODEL)

# Probar con imagen
image = cv2.imread('test_face.jpg')

# Detectar rostro
detections = detector.detect_faces(image)
print(f"Rostros detectados: {len(detections)}")

# Generar embedding
if detections:
    face = detections[0]
    # ... procesar rostro
    embedding = recognizer.get_embedding(aligned_face)
    print(f"Embedding shape: {embedding.shape}")

    # Verificar liveness
    is_real, confidence = liveness.detect_liveness(aligned_face)
    print(f"Liveness: {is_real}, Confidence: {confidence}")
```

📊 Comparación de Modelos

YOLOv8-face vs Alternativas

Modelo	Velocidad (CPU)	Precisión	Tamaño
YOLOv8n-face	~20 FPS	Alta	6MB
MTCNN	~5 FPS	Media	2MB
RetinaFace	~10 FPS	Muy Alta	27MB
Dlib HOG	~30 FPS	Media	<1MB

MobileFaceNet vs Alternativas

Modelo	Velocidad	Precisión	Tamaño
MobileFaceNet	Rápido	99%+	4MB
FaceNet	Medio	99%+	90MB
ArcFace	Medio	99.5%+	250MB
DeepFace	Lento	97%	152MB

Recomendación: MobileFaceNet es óptimo para edge computing.

🎓 Recursos Adicionales

Papers Importantes

1. **YOLOv8**:
   - "YOLOv8: Ultralytics"
   - https://github.com/ultralytics/ultralytics

2. **MobileFaceNet**:
   - "MobileFaceNets: Efficient CNNs for Accurate Real-Time Face Verification on Mobile Devices"
   - https://arxiv.org/abs/1804.07573

3. **Face Anti-Spoofing**:
   - "Learning Deep Models for Face Anti-Spoofing: Binary or Auxiliary Supervision"
   - https://arxiv.org/abs/1803.11097

Datasets

1. **Training Face Recognition**:
   - MS-Celeb-1M
   - VGGFace2
   - CASIA-WebFace

2. **Training Liveness**:
   - CASIA-SURF
   - OULU-NPU
   - Replay-Attack

Repositorios Útiles

- InsightFace: https://github.com/deepinsight/insightface
- Ultralytics: https://github.com/ultralytics/ultralytics
- Silent-Face-Anti-Spoofing: https://github.com/minivision-ai/Silent-Face-Anti-Spoofing

---

⚡ Optimización para Edge Computing

Quantización de Modelos

Para mejorar velocidad en Raspberry Pi:

```
import torch

# Cargar modelo
model = torch.load('model.pth')

# Quantizar a INT8
model_quantized = torch.quantization.quantize_dynamic(
    model, {torch.nn.Linear}, dtype=torch.qint8
)

# Guardar modelo quantizado
torch.save(model_quantized, 'model_quantized.pth')
```

Conversión a ONNX

Para inferencia más rápida:

```
import torch.onnx

# Exportar a ONNX
dummy_input = torch.randn(1, 3, 112, 112)
torch.onnx.export(model, dummy_input, "model.onnx")

# Usar ONNX Runtime
import onnxruntime
session = onnxruntime.InferenceSession("model.onnx")
```

❓ FAQ

¿Puedo usar otros modelos?

Sí, el código es modular. Solo necesitas:
1. Implementar la interfaz correspondiente
2. Actualizar las rutas en `config.py`
3. Asegurar que las dimensiones de entrada/salida coincidan

¿Qué hacer si no tengo GPU?

Los modelos están optimizados para CPU. En Raspberry Pi:
- YOLOv8n funciona a ~10-15 FPS
- MobileFaceNet a ~20ms por rostro
- Es suficiente para control de asistencia

¿Necesito entrenar los modelos?

No necesariamente. Los modelos pre-entrenados funcionan bien para:
- Detección facial general
- Reconocimiento facial
- Liveness básico

Entrena tu propio modelo si:
- Tienes casos de uso muy específicos
- Necesitas mayor precisión en tu entorno
- Tienes un dataset grande y etiquetado

Guía actualizada: Diciembre 2024