My Project

Generated by Doxygen 1.13.2

Chapter 1

FaceStream Studio

1.1 Gerçek Zamanlı Yüz Analizi ve Konuşma Takibi Uygulaması

Yapay zeka destekli yüz tanıma, duygu analizi ve konuşma süresi ölçümü ile gelişmiş video analiz platformu

1.1.1 Yenilikler

- **Custom PyTorch MLP Model** ile gelişmiş duygu analizi
- **CUDA GPU desteği** ile hızlandırılmış işlem
- **MediaPipe Face Mesh** ile hassas yüz landmark tespiti
- **Gelişmiş konuşma analizi** ve süre takibi
- **Çoklu dil desteği** (Türkçe/İngilizce)

1.2 İçindekiler

1.3 Özellikler

1.3.1 Temel Özellikler

- **Gerçek Zamanlı Analiz**: Canlı kamera görüntüsü üzerinden anlık yüz analizi
- **Video Dosya Analizi**: MP4, AVI, MOV formatlarında video dosya desteği
- **Gelişmiş Yüz Tanıma**: Face Recognition library ile yüksek doğruluk oranı
- **AI Duygu Analizi**: Custom PyTorch MLP model ile 7 farklı duygu tespiti
- **Akıllı Konuşma Tespiti**: MediaPipe Face Mesh ile dudak hareketlerinden konuşma analizi
- **Detaylı Konuşma Takibi**: Kişi bazında milisaniye hassasiyetinde konuşma süresi ölçümü

1.3.2 Gelişmiş Özellikler

- **GPU Acceleration**: CUDA desteği ile hızlandırılmış işlem gücü
- **Çoklu Dil Desteği**: Türkçe ve İngilizce arayüz (languages.json)
- **Veri Analizi**: CSV formatında detaylı sonuç kaydetme ve analiz
- **Özelleştirilebilir Arayüz**: Streamlit tabanlı modern web arayüzü
- **Dinamik Yüz Veritabanı**: Runtime'da yeni yüzler ekleme ve yönetme
- **Performans Optimizasyonu**: Frame skip, cache sistemi ve akıllı kaynak yönetimi
- **Real-time Metrics**: Anlık FPS, işlem süresi ve performans metrikleri

---

1.4 Teknolojiler

1.4.1 AI/ML Framework'leri

- **YOLO v11**: State-of-the-art yüz tespiti (yolov11l-face.pt)
- **Face Recognition**: dlib tabanlı yüz encoding ve tanıma sistemi
- **MediaPipe Face Mesh**: Google'ın 468 noktalı yüz landmark tespiti
- **PyTorch**: Custom MLP modeli ile gelişmiş duygu analizi
- **Scikit-learn**: Feature scaling, selection ve preprocessing

1.4.2 Core Technologies

- **Streamlit**: Modern, responsive web uygulaması framework'ü
- **OpenCV**: Bilgisayarlı görü ve görüntü işleme
- **NumPy & Pandas**: Vektörel işlemler ve veri analizi
- **Pillow (PIL)**: Görüntü formatları ve işleme
- **CMake**: C++ bağımlılıkları için build sistem

1.4.3 Model Mimarisi

```
# Custom Emotion MLP Architecture
EmotionMLP(
  input_dim=936,      # MediaPipe landmarks
  hidden_layers=[2048, 1024, 512, 256, 128],
  num_classes=7,      # 7 emotion categories
  activation="SiLU",  # Swish activation
  dropout=0.1-0.3,    # Regularization
  batch_norm=True     # Stability
)
```

1.5 Kurulum

1.5.1 Sistem Gereksinimleri

- **Python**: 3.8+ (3.10+ önerilen)
- **İşletim Sistemi**: Windows 10+, macOS 10.14+, Ubuntu 18.04+
- **RAM**: En az 4GB (8GB+ önerilen)
- **GPU**: CUDA destekli GPU (opsiyonel, 3-5x hızlanma)
- **Kamera**: USB webcam veya dahili kamera
- **Depolama**: En az 2GB boş alan (model dosyaları için)

1.5.2 Hızlı Kurulum

1. **Repository'yi klonlayın**
```
git clone <repository-url>
cd "FaceStream Studio"
```
2. **Python sanal ortamı oluşturun** (Şiddetle önerilen)
```
# Windows
python -m venv venv
venv\Scripts\activate

# macOS/Linux
python3 -m venv venv
source venv/bin/activate
```
3. **Gerekli paketleri yükleyin**
```
pip install -r requirements.txt
```
4. **Model dosyalarını hazırlayın**
```
# src/ klasöründe şu dosyalar bulunmalı:
src/
 yolov11l-face.pt              # YOLO face detection model
 models/
    torch/                    # PyTorch emotion models
        emotion_mlp.pth
        emotion_scaler.pkl
        emotion_labelencoder.pkl
```

1.5.3 Çalıştırma

```
streamlit run app.py
```
Uygulama http://localhost:8501 adresinde açılacaktır.

1.6 Hızlı Başlangıç

1.6.1 Uygulamayı Başlatma

```
streamlit run app.py
```
Tarayıcınızda http://localhost:8501 adresine giderek uygulamayı kullanmaya başlayın.

1.6.2 İlk Kullanım Adımları

1. **Dil Seçimi**: Sol sidebar'dan Türkçe/English seçin
2. **Mod Seçimi**: Kamera veya Video analiz modunu belirleyin
3. **Model Ayarları**: Eşik değerleri ve performans parametrelerini düzenleyin
4. **Yüz Kaydetme**: Sağ panelden bilinen kişilerin yüzlerini ekleyin
5. **Analizi Başlatın**: "Başlat" butonuna tıklayarak gerçek zamanlı analizi başlatın

1.6.3 Temel İşlemler

- **Yüz Ekleme**: Fotoğraf yükleyin → İsim girin → "Yüz Ekle"
- **Konuşma Takibi**: Dudak hareketleri otomatik algılanır
- **Duygu Analizi**: Yüz ifadeleri gerçek zamanlı işlenir
- **Veri Kaydetme**: Sonuçlar CSV formatında indirilebilir

1.7 Kullanım Kılavuzu

1.7.1 Kamera Modu

1.7.1.1 Canlı Analiz

- **Başlatma**: "Başlat" düğmesine tıklayın
- **İzleme**: Gerçek zamanlı yüz tespiti ve duygu analizi
- **Durdurma**: "Durdur" düğmesi ile analizi sonlandırın
- **Sonuçlar**: Konuşma süreleri sağ panelde görüntülenir

1.7.1.2 Yüz Yönetimi

- **Yeni Yüz Ekleme**: Fotoğraf yükleyin ve isim girin
- **Yüz Silme**: simgesi ile kayıtlı yüzleri silin
- **Otomatik Tanıma**: Eklenen yüzler otomatik olarak tanınır

1.7.2 Video Modu

1.7.2.1 Video Analizi

- **Dosya Yükleme**: MP4, AVI, MOV formatında video seçin
- **Analiz Başlatma**: " Analiz Başlat" düğmesine tıklayın
- **İlerleme Takibi**: Progress bar ile analiz durumunu izleyin
- **Sonuç Görüntüleme**: Tamamlandığında detaylı sonuçlar görüntülenir

---

1.8 Ayarlar ve Yapılandırma

1.8.1 Model Ayarları

Parametre	Açıklama	Varsayılan	Aralık
Yüz Eşleşme Eşiği	Yüz tanıma hassasiyeti	0.6	0.↩ 3-0.8
Maksimum Yüz Sayısı	Aynı anda tespit edilecek yüz sayısı	5	1-20
Frame Atlatma	Performans için frame sayısı	2	1-10

1.8.2 Tespit Ayarları

Parametre	Açıklama	Varsayılan	Aralık
Konuşma Hassasiyeti	Dudak hareketi eşiği	0.03	0.01- 0.1

1.8.3 Görüntüleme Seçenekleri

- **İsimleri Göster**: Tanınan yüzlerin isimlerini gösterir
- **Konuşma Sürelerini Göster**: Anlık konuşma sürelerini gösterir
- **Duygu Analizini Göster**: Yüz ifadelerini gösterir
- **Sınırlayıcı Kutuları Göster**: Yüzlerin etrafında kutu çizer

1.8.4 Dil ve UI Ayarları

- **Dil Seçimi**: Türkçe / English
- **Tema**: Light / Dark (gelecek sürümlerde)

1.9 AI Modelleri

1.9.1 Yüz Tespiti - YOLO v11

```
# Model Dosyası
model_path = "src/yolov11l-face.pt"  # High accuracy face detection

# Parametreler
confidence_threshold = 0.5    # Detection confidence
max_faces = 10                # Maximum faces per frame
frame_skip = 2                # Process every N frames
```

1.9.2 Duygu Analizi - Custom PyTorch MLP

```
# Tespit Edilen 7 Duygu
EMOTIONS = [
    "HAPPY",       #  Mutlu
    "SAD",         #  Üzgün
    "ANGRY",       #  Kızgın
    "SURPRISED",   #  Şaşkın
    "NEUTRAL",     #  Nötr
    "ANNOYED",     #  Sinirli
    "FEAR"         #  Korkmuş
]

# Model Architecture
class EmotionMLP(nn.Module):
    def __init__(self, input_dim=936, num_classes=7):
        # 936 features from MediaPipe landmarks
        # Deep neural network with residual connections
        # SiLU activation + BatchNorm + Dropout
```

1.9.3 Konuşma Tespiti - MediaPipe Face Mesh

```
# Kullanılan Landmark Points (468 total landmarks)
UPPER_LIP = [13, 82, 81, 80, 78]     # Üst dudak noktaları
LOWER_LIP = [14, 87, 86, 85, 84]     # Alt dudak noktaları

# Konuşma Algoritması
def detect_speaking(landmarks):
    lip_distance = calculate_lip_distance(landmarks)
    speaking_threshold = 0.03  # Configurable
    return lip_distance > speaking_threshold
```

1.9.4 Model Performansı

Model	Doğruluk	Hız (FPS)	GPU Bellek
YOLO v11 Face	95%+	30-60	2GB
Emotion MLP	87%	100+	1GB
MediaPipe Mesh	99%	60+	0.5GB

1.10 Sorun Giderme

1.10.1 Sık Karşılaşılan Hatalar

1.10.1.1 Model Dosyaları Bulunamadı

```
# Hata: src folder not found!
# Çözüm:
mkdir src
# Model dosyalarını src/ klasörüne ekleyin
```

1.10.1.2 Kamera Açılmıyor

```
# Linux için kamera izinleri
sudo usermod -a -G video $USER
# Oturum kapatıp açın
```

1.10.1.3 Bağımlılık Hataları

```
# Yeniden yükleme
pip uninstall -r requirements.txt -y
pip install -r requirements.txt --upgrade
```

1.10.1.4 CMake Hatası

```
# Ubuntu/Debian
sudo apt-get install cmake build-essential

# macOS
brew install cmake

# Windows
# Visual Studio Build Tools yükleyin
```

1.10.2 Hata Kodları

Kod	Açıklama	Çözüm
E001	Model dosyası yok	Model dosyalarını src/ klasörüne ekleyin
E002	Kamera erişim hatası	Kamera izinlerini kontrol edin
E003	Video dosyası bozuk	Başka bir video dosyası deneyin
E004	Bellek yetersiz	Frame skip değerini artırın

1.11 Performans Optimizasyonu

1.11.1 Hızlandırma İpuçları

1. **GPU Kullanımı**

   ```
   # CUDA kurulu ise otomatik GPU kullanımı
   device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
   ```

2. **Frame Skip Ayarı**

   ```
   # Yüksek frame skip = Daha hızlı işlem
   frame_skip = 3  # Her 3 frame'de bir analiz
   ```

3. **Model Seçimi**

   ```
   # Hız öncelikli: yolov8n-face.pt
   # Doğruluk öncelikli: yolov11l-face.pt
   ```

1.11.2 Benchmark Sonuçları

Sistem Konfigürasyonu	FPS	CPU Kullanımı	RAM Kullanımı	İşlem Süresi
RTX 4050 + i5-13420H	75+	25%	3.5GB	$<$13ms

CPU Only (i5-13420H)	12-15	85%	4.2GB	<66ms

1.11.3 Proje Mimarisi

```
FaceStream Studio/
 app.py                    # Streamlit ana uygulama
 Analyzer.py               # FaceAnalyzer sınıfı (core engine)
 languages.json            # Çoklu dil desteği
 requirements.txt          # Python dependencies
 src/
    yolov11-face.pt     # YOLO face detection model
    models/
        torch/              # PyTorch emotion MLP
 docs/                     # Documentation (optional)
```

1.11.4 İşlem Akışı

1. **Video Input** → Camera/File
2. **Frame Processing** → YOLO Face Detection
3. **Face Recognition** → Encoding & Matching
4. **Landmark Detection** → MediaPipe Face Mesh
5. **Feature Extraction** → 936D vector from landmarks
6. **Emotion Prediction** → PyTorch MLP inference
7. **Speaking Detection** → Lip distance analysis
8. **Results Aggregation** → Real-time statistics
9. **UI Update** → Streamlit visualization

1.11.5 Dependencies

```
# Core Dependencies (requirements.txt)
streamlit            # Web UI framework
opencv-python       # Computer vision
ultralytics         # YOLO models
mediapipe           # Face mesh detection
numpy               # Numerical computing
Pillow              # Image processing
cmake               # C++ build system

# Additional Python Packages (auto-installed)
torch               # PyTorch deep learning
torchvision         # Computer vision for PyTorch
face-recognition    # Face encoding/recognition
scikit-learn        # Traditional ML models
pandas              # Data manipulation
joblib              # Model serialization
```

1.11.6 Bilinen Limitasyonlar

- **İşlem Gücü**: Yüksek çözünürlük videolarda GPU gereksinimi
- **Aydınlatma**: Düşük ışık koşullarında performans azalması
- **Çoklu Yüz**: 10+ yüz durumunda FPS düşüşü
- **Model Boyutu**: YOLO v11 modeli ∼50MB boyutunda
- **Platform**: Windows'ta CMake kurulumu karmaşık olabilir

1.11.7 Geliştirme Ortamı

```
# Development kurulumu
git clone https://github.com/kullanici/facestream-studio.git
cd facestream-studio
pip install -r requirements-dev.txt
```

1.12 Katkıda Bulunma

1.12.1 Geliştirme Ortamı Kurulumu

```
# Repository'yi fork edin ve klonlayın
git clone https://github.com/YOUR_USERNAME/facestream-studio.git
cd "FaceStream Studio"

# Development dependencies yükleyin
pip install -r requirements.txt
pip install pytest pytest-cov black flake8

# Pre-commit hooks kurulumu (opsiyonel)
pip install pre-commit
pre-commit install
```

1.12.2 Katkı Süreci

1. **Fork** yapın ve **feature branch** oluşturun
   ```
   git checkout -b feature/amazing-new-feature
   ```
2. **Kod yazın** ve **test edin**
   ```
   python -m pytest tests/ -v
   black . --check
   flake8 .
   ```
3. **Commit** edin ve **push** yapın
   ```
   git commit -m "feat: add amazing new feature"
   git push origin feature/amazing-new-feature
   ```
4. **Pull Request** oluşturun

1.12.3 Test Yazma

```
# tests/test_analyzer.py örneği
import pytest
from Analyzer import FaceAnalyzer

def test_face_analyzer_init():
    analyzer = FaceAnalyzer("src/yolov11l-face.pt")
    assert analyzer.device is not None
    assert analyzer.model is not None

def test_emotion_detection():
    # Test emotion classification
    pass
```

1.12.4 Katkı Kuralları

- **Code Style**: Black formatter kullanın
- **Documentation**: Yeni fonksiyonlar için docstring ekleyin
- **Testing**: Yeni özellikler için test yazın
- **Commit Messages**: `Conventional Commits` formatını kullanın

1.13 Geliştirme

1.13.1 VS Code Görevleri

Bu proje VS Code task'ları ile gelir:

```
# Streamlit uygulamasını çalıştır
Ctrl+Shift+P → "Tasks: Run Task" → "Run Streamlit App"
```

1.13.2 Debug Modu

```
# app.py içinde debug modu
DEBUG_MODE = True  # Ek loglar ve metrikler için

# Analyzer.py içinde
if DEBUG_MODE:
    print(f"Frame processing time: {processing_time:.2f}ms")
    print(f"Detected faces: {len(faces)}")
```

1.13.3 Profiling

```
# Memory profiling
pip install memory-profiler
python -m memory_profiler app.py

# Performance profiling
pip install line-profiler
kernprof -l -v app.py
```

1.13.4 Katkı Süreci (Detaylı)

1. **Repository'yi Fork Edin**
   - GitHub üzerinden projeyi fork edin
   - Kendi hesabınıza kopyalayın
2. **Local Kurulum**
   ```
   git clone https://github.com/YOUR_USERNAME/facestream-studio.git
   cd "FaceStream Studio"
   git remote add upstream https://github.com/ORIGINAL_OWNER/facestream-studio.git
   ```
3. **Feature Branch Oluşturun**
   ```
   git checkout -b feature/your-feature-name
   ```
4. **Kod Geliştirin ve Test Edin**
   ```
   # Kodunuzu yazın
   # Testleri çalıştırın
   python -m pytest tests/ -v

   # Code quality check
   black . --check
   flake8 .
   ```
5. **Commit ve Push**
   ```
   git add .
   git commit -m "feat: add your amazing feature"
   git push origin feature/your-feature-name
   ```
6. **Pull Request Oluşturun**
   - GitHub'da Pull Request açın
   - Detaylı açıklama yazın
   - Review bekleyin

1.13.5 Gelişme Roadmap

- ☐ **Real-time Dashboard**: Live metrics ve performance monitoring
- ☐ **Batch Processing**: Çoklu video dosyalarını toplu işleme
- ☐ **API Endpoint**: REST API ile entegrasyon desteği
- ☐ **Cloud Integration**: AWS/Azure cloud deployment
- ☐ **Mobile App**: React Native ile mobil uygulama
- ☐ **Advanced Analytics**: Detaylı istatistik ve raporlama
- ☐ **Multi-language Models**: Farklı etnik köken için optimize modeller

1.14 İletişim ve Destek

1.14.1 İletişim

- **E-posta**: muhakaplan@hotmail.com
- **GitHub**: Bu repository'deki Issues bölümü
- **LinkedIn**: [Profil bağlantısı]

1.14.2 Destek

- **Bug Reports**: GitHub Issues kullanarak hata bildirimi
- **Feature Requests**: Yeni özellik önerileri için Discussions
- **Dokümantasyon**: README ve kod içi yorumlar
- **Teknik Sorular**: Issues bölümünde Q&A etiketi ile

1.14.3 Proje Durumu

- **Temel Özellikler**: Tamamlandı ve test edildi
- **Optimizasyon**: Devam eden geliştirmeler
- **Performans**: GPU acceleration ve caching
- **Internationalization**: Çoklu dil desteği aktif

1.15 Lisans

Bu proje MIT Lisansı altında lisanslanmıştır. Detaylar için LICENSE dosyasına bakınız.

1.16 Teşekkürler

Bu proje aşağıdaki açık kaynak projelerden yararlanmaktadır:

- Ultralytics YOLO
- MediaPipe
- Face Recognition
- Streamlit
- OpenCV

Bu projeyi beğendiyseniz yıldız vermeyi unutmayın!

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

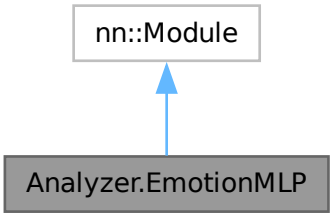
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

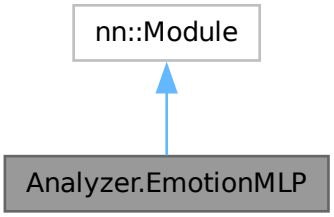
Class Documentation

4.1 Analyzer.EmotionMLP Class Reference

Inheritance diagram for Analyzer.EmotionMLP:



Collaboration diagram for Analyzer.EmotionMLP:



Public Member Functions

- **__init__** (self, input_dim=936, num_classes=7)
- **forward** (self, x)

Public Attributes

- initial

- middle_main
- middle_shortcut
- final

4.1.1 Member Data Documentation

4.1.1.1 final

Analyzer.EmotionMLP.final

Initial value:

```
=  nn.Sequential(
            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.SiLU(),
            nn.Dropout(0.15),

            nn.Linear(256, 128),
            nn.LayerNorm(128),
            nn.SiLU(),
            nn.Dropout(0.1),

            nn.Linear(128, num_classes)
        )
```

4.1.1.2 initial

Analyzer.EmotionMLP.initial

Initial value:

```
=  nn.Sequential(
            nn.Linear(input_dim, 2048),
            nn.BatchNorm1d(2048),
            nn.SiLU(),
            nn.Dropout(0.3),

            nn.Linear(2048, 1024),
            nn.BatchNorm1d(1024),
            nn.SiLU(),
            nn.Dropout(0.25)
        )
```

4.1.1.3 middle_main

Analyzer.EmotionMLP.middle_main

Initial value:

```
=  nn.Sequential(
            nn.Linear(1024, 512),
            nn.BatchNorm1d(512),
            nn.SiLU(),
            nn.Dropout(0.2)
        )
```

4.1.1.4 middle_shortcut

Analyzer.EmotionMLP.middle_shortcut

Initial value:

```
=  nn.Sequential(
            nn.Linear(1024, 512),
            nn.BatchNorm1d(512)
        )
```

The documentation for this class was generated from the following file:

- Analyzer.py

4.2 Analyzer.FaceAnalyzer Class Reference

Public Member Functions

- **__init__** (self, model_path, temp_faces=None, fps=30)
- **process_frame** (self, frame)

Public Attributes

- **device** = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- **model** = YOLO(model_path).to(self.device)
- int **FRAME_SKIP** = 2
- float **SPEAKING_LIP_DIST_THRESHOLD** = 0.07
- float **FACE_MATCH_THRESHOLD** = 0.6
- int **MAX_FACES** = 10
- face_mesh
- **known_faces** = self._load_temp_faces(temp_faces) if temp_faces else {'encodings': [ ], 'names': [ ]}
- int **frame_counter** = 0
- dict **tracked_faces** = {}
- int **next_id** = 1
- dict **speaking_times** = {}
- dict **active_speakers** = {}
- dict **emotion_cache** = {}
- bool **show_names** = True
- bool **show_times** = True
- bool **show_emotion** = True
- bool **show_bounding_boxes** = True
- **fps** = fps
- dict **face_id_cache** = {}
- **emotion_model** = None
- **emotion_scaler** = None
- **emotion_labelencoder** = None
- **emotion_selector** = None
- list **emotion_feature_names** = None
- **torch_model** = None
- **torch_scaler** = None
- **torch_le** = None
- bool **using_torch** = False

Static Public Attributes

- tuple **EMOTION_COLOR** = (255, 255, 255)
- int **MAX_FACE_ID_CACHE_SIZE** = 128
- int **MAX_EMOTION_CACHE_SIZE** = 100

Protected Member Functions

- _load_emotion_models (self)
- **_load_temp_faces** (self, temp_faces)
- **_detect_faces** (self, rgb_frame)
- **_process_face_landmarks** (self, frame, rgb_frame)
- _get_landmark_id (self, landmarks, w, h)
- **_display_emotion** (self, frame, landmarks, emotion, color, w, h)
- **_update_speaking_status** (self, landmarks, w, h)
- **_draw_face_info** (self, frame)
- **_update_tracking** (self, boxes, rgb_frame)
- **_find_closest_face** (self, center)
- **_add_new_face** (self, rgb_frame, box, center)
- _manage_cache (self, cache, key, value, max_size)
- _identify_face (self, encoding)
- _analyze_emotion (self, landmarks, landmark_id=None)
- _predict_emotion_torch (self, landmarks)
- _predict_emotion_model (self, landmarks)
- _extract_landmark_features (self, landmarks)

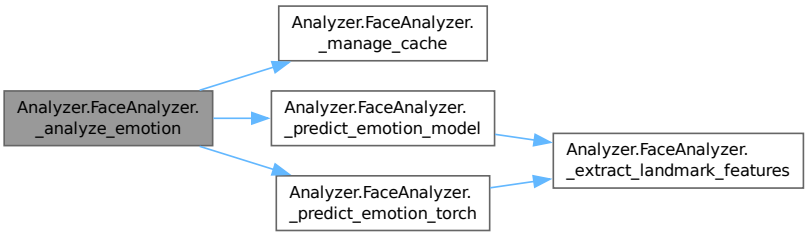
4.2.1 Member Function Documentation

4.2.1.1 _analyze_emotion()

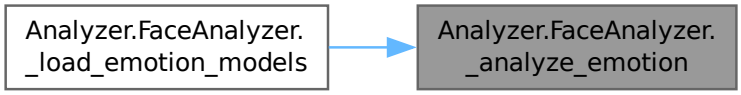
```
Analyzer.FaceAnalyzer._analyze_emotion (
            self,
            landmarks,
            landmark_id = None)  [protected]
```

Face expression based emotion recognition with caching

Here is the call graph for this function:



Here is the caller graph for this function:

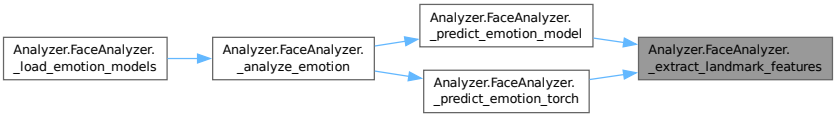


4.2.1.2 _extract_landmark_features()

```
Analyzer.FaceAnalyzer._extract_landmark_features (
            self,
            landmarks)  [protected]
```

Extract and normalize landmark features - simplified version

Here is the caller graph for this function:

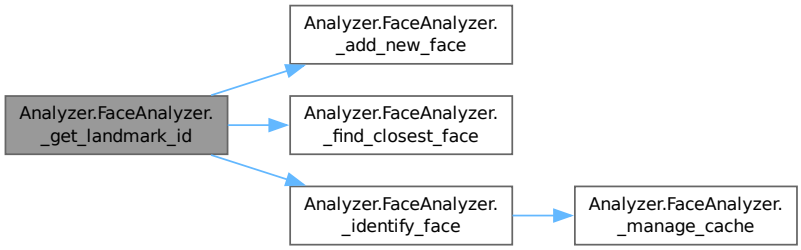


4.2.1.3 _get_landmark_id()

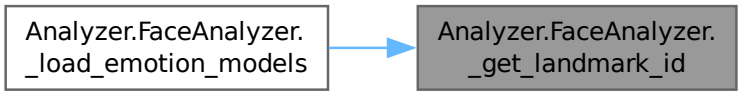
```
Analyzer.FaceAnalyzer._get_landmark_id (
            self,
            landmarks,
            w,
            h)  [protected]
```

Create a unique ID for this landmark to use in caching

Here is the call graph for this function:



Here is the caller graph for this function:

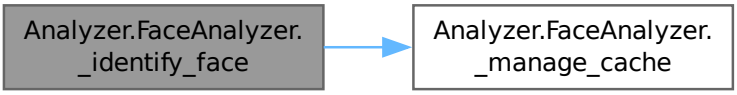


4.2.1.4 _identify_face()

```
Analyzer.FaceAnalyzer._identify_face (
            self,
            encoding)  [protected]
```

Identify a face using a manual cache for frequently seen faces

Here is the call graph for this function:



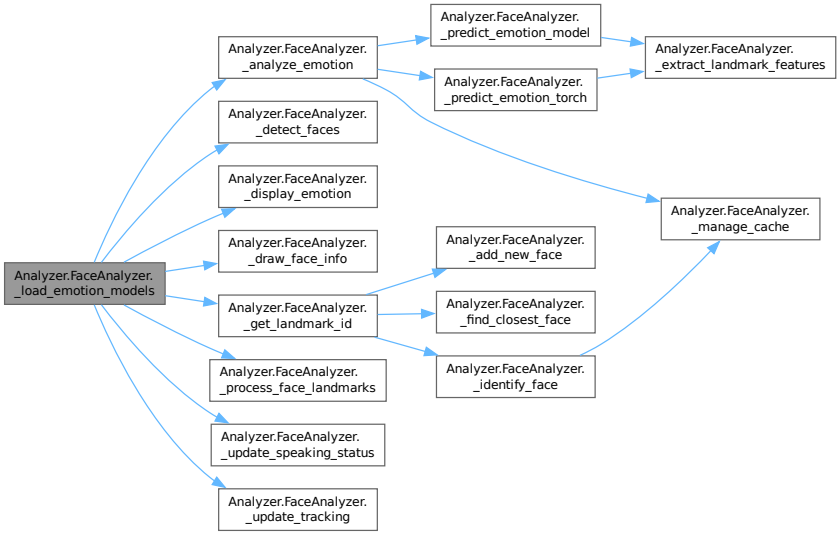
Here is the caller graph for this function:

4.2.1.5 _load_emotion_models()

```
Analyzer.FaceAnalyzer._load_emotion_models (
            self)  [protected]
```

Lazily load emotion models when needed

Here is the call graph for this function:

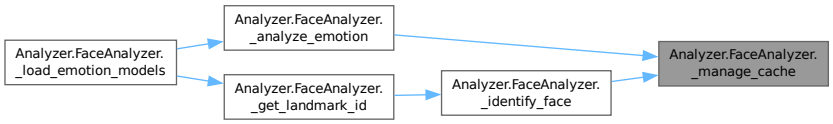


4.2.1.6 _manage_cache()

```
Analyzer.FaceAnalyzer._manage_cache (
            self,
            cache,
            key,
            value,
            max_size)  [protected]
```

Centralized cache management to avoid code duplication

Here is the caller graph for this function:

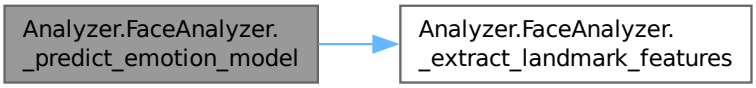


4.2.1.7 _predict_emotion_model()

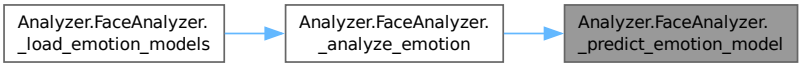
```
Analyzer.FaceAnalyzer._predict_emotion_model (
              self,
              landmarks)  [protected]
```

Predict emotion using sklearn model

Here is the call graph for this function:



Here is the caller graph for this function:

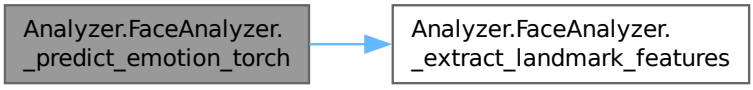


4.2.1.8 _predict_emotion_torch()

```
Analyzer.FaceAnalyzer._predict_emotion_torch (
              self,
              landmarks)  [protected]
```

Predict emotion using PyTorch model

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.2 Member Data Documentation

4.2.2.1 face_mesh

`Analyzer.FaceAnalyzer.face_mesh`

Initial value:

```
=  mp.solutions.face_mesh.FaceMesh( # type: ignore
            max_num_faces=self.MAX_FACES,
            refine_landmarks=True,
            min_detection_confidence=0.5,
        )
```

The documentation for this class was generated from the following file:

- Analyzer.py

4.3 app.FaceStreamStudioApp Class Reference

Public Member Functions

- **__init__** (self)
- **load_languages** (self)
- **get_lang** (self)
- **init_session** (self)
- **create_analyzer** (self, fps=30)
- **configure_analyzer** (self, analyzer)
- **process_frame** (self, analyzer, frame, frame_placeholder)
- **handle_media_stream** (self, input_source, is_camera=True)
- **model_settings** (self)
- **detection_settings** (self)
- **display_settings** (self)
- **ui_settings** (self)
- **export_results** (self)
- **settings_interface** (self)
- **display_saved_face** (self, face_name, in_sidebar=True)
- **display_temp_faces_panel** (self)
- **add_new_face_ui** (self)
- **display_speech_results** (self, analyzer)
- **camera_interface** (self)
- **video_interface** (self)
- **main** (self)

Public Attributes

- str **div** = " "
- **LANGUAGES** = self.load_languages()
- **DEFAULT_LANGUAGE** = next(iter(self.LANGUAGES), "English")

The documentation for this class was generated from the following file:

- app.py