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

- **Özel PyTorch MLP Modeli** 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 Proje Mimarisi ve Graf

FaceStream Studio projesi, modern AI/ML teknolojilerini kullanarak gerçek zamanlı yüz analizi yapan kapsamlı bir sistemdir. Aşağıdaki graf, projenin tüm bileşenlerini, bağımlılıklarını ve veri akışını detaylı olarak göstermektedir:

1.3.1 Graf Açıklaması

Renk Kodları:

- **Yeşil**: Python dosyaları (app.py, Analyzer.py)
- **Turuncu**: AI/ML model dosyaları (.pt, .pth, .pkl)
- **Sarı**: Yapılandırma dosyaları (JSON, TXT, MD)
- **Mavi**: Harici kütüphaneler ve framework'ler
- **Pembe**: Python sınıfları ve bileşenleri
- **Altın**: Uygulama özellikleri ve fonksiyonları

Ana Bileşenler:

1. **Ana Uygulama Katmanı**
   - `app.py`: Streamlit tabanlı web arayüzü
   - `Analyzer.py`: Yüz analizi ve AI işlemleri motoru
   - `languages.json`: Çoklu dil desteği konfigürasyonu
2. **AI/ML Model Katmanı**
   - `yolov11l-face.pt`: YOLO v11 yüz tespit modeli
   - `emotion_mlp.pth`: Custom PyTorch duygu analizi modeli
   - `emotion_scaler.pkl` & `emotion_labelencoder.pkl`: Model ön işleme bileşenleri
3. **Kütüphane Ekosistemi**
   - **Streamlit**: Web arayüzü framework'ü
   - **OpenCV**: Görüntü işleme ve video operasyonları
   - **PyTorch**: Deep learning model inference
   - **MediaPipe**: Yüz landmark tespiti
   - **Ultralytics**: YOLO model implementasyonu
4. **Özellik Katmanı**
   - Gerçek zamanlı yüz tespiti
   - Duygu tanıma sistemi
   - Konuşma analizi ve takibi
   - Çoklu dil desteği
   - Veri dışa aktarma

1.3.2 Veri Akış Diyagramı

```
graph TB
    A[Video Girdi] --> B[Frame İşleme]
    B --> C[YOLO Yüz Tespiti]
    C --> D[MediaPipe Landmark]
    D --> E[Duygu Analizi MLP]
    E --> F[Streamlit UI]
    F --> G[CSV Export]

    H[Face Database] --> C
    I[Model Weights] --> E
    J[Language Config] --> F
```

1.3.3 Sistem Mimarisi

Modüler Tasarım:

- **Separation of Concerns**: Her modül belirli bir sorumluluğa odaklanır
- **Loose Coupling**: Bileşenler arası gevşek bağlantı
- **High Cohesion**: İlgili fonksiyonlar aynı modülde gruplandırılmış
- **Scalability**: Yeni özellikler kolayca eklenebilir
- **Maintainability**: Kod bakımı ve güncelleme kolaylığı

Performans Optimizasyonları:

- CUDA GPU desteği ile hızlandırılmış hesaplama
- Frame skipping ile gereksiz işlem azaltma
- Model caching ile tekrarlayan hesaplama önleme
- Batch processing ile verimli veri işleme

1.4 Özellikler

1.4.1 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 kütüphanesi ile yüksek doğruluk oranı
- **Yapay Zeka Duygu Analizi**: Özel 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

1.4.2 Gelişmiş Özellikler

- **GPU Hızlandırma**: 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**: Çalışma sırasında yeni yüzler ekleme ve yönetme
- **Performans Optimizasyonu**: Frame skip, önbellek sistemi ve akıllı kaynak yönetimi
- **Anlık Metrikler**: Anlık FPS, işlem süresi ve performans metrikleri

---

1.5 Kullanılan Teknolojiler

1.5.1 Yapay Zeka/ML Framework'leri

- **YOLO v11**: Son teknoloji 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**: Özel MLP modeli ile gelişmiş duygu analizi
- **Scikit-learn**: Özellik ölçekleme, seçme ve ön işleme

1.5.2 Temel Teknolojiler

- **Streamlit**: Modern, duyarlı 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 derleme sistemi

1.5.3 Model Mimarisi

```
# Özel Duygu MLP Mimarisi
DuyguMLP(
  input_dim=936,      # MediaPipe landmarks
  hidden_layers=[2048, 1024, 512, 256, 128],
  num_classes=7,      # 7 duygu kategorisi
  activation="SiLU",  # Swish aktivasyonu
  dropout=0.1-0.3,    # Düzenleme
  batch_norm=True     # Stabilite
)
```

1.6 Kurulum

1.6.1 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)

1.6.2 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
   ```

1.6.3 Çalıştırma

```
streamlit run app.py
```

Uygulama http://localhost:8501 adresinde açılacaktır.

1.7 Hızlı Başlangıç

1.7.1 Uygulamayı Başlatma

```
streamlit run app.py
```

Tarayıcınızda http://localhost:8501 adresine giderek uygulamayı kullanmaya başlayın.

1.7.2 İlk Kullanım Adımları

1. **Dil Seçimi**: Sol kenar çubuğundan 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

1.7.3 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

1.8 Kullanım Kılavuzu

1.8.1 Kamera Modu

1.8.1.1 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

1.8.1.2 Yüz Yönetimi

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

1.8.2 Video Modu

1.8.2.1 Video Analizi

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

---

1.9 Ayarlar ve Yapılandırma

1.9.1 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

1.9.2 Tespit Ayarları

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

1.9.3 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

1.9.4 Dil ve Arayüz Ayarları

- **Dil Seçimi**: Türkçe / English
- **Tema**: Açık / Koyu (gelecek sürümlerde)

1.10 Yapay Zeka Modelleri

1.10.1 Yüz Tespiti - YOLO v11

```
# Model Dosyası
yolov11l_face_model_yolu = "src/yolov11l-face.pt"  # Yüksek doğruluklu yüz tespiti

# Parametreler
guven_esiği = 0.5    # Tespit güveni
maksimum_yuz = 10    # Her karede maksimum yüz
frame_atlatma = 2    # Her N karede bir işlem
```

1.10.2 Duygu Analizi - Özel PyTorch MLP

```
# Tespit Edilen 7 Duygu
DUYGULAR = [
    "MUTLU",       #  Mutlu
    "ÜZGÜN",       #  Üzgün
    "KIZGIN",      #  Kızgın
    "ŞAŞKIN",      #  Şaşkın
    "NÖTR",        #  Nötr
    "SİNİRLİ",     #  Sinirli
    "KORKMUŞ"      #  Korkmuş
]

# Model Mimarisi
class DuyguMLP(nn.Module):
    def __init__(self, input_dim=936, num_classes=7):
        # 936 özellik MediaPipe landmarklarından
        # Derin sinir ağı, residual bağlantılar
        # SiLU aktivasyon + BatchNorm + Dropout
```

1.10.3 Konuşma Tespiti - MediaPipe Face Mesh

```
# Kullanılan Landmark Noktaları (468 toplam landmark)
UST_DUDAK = [13, 82, 81, 80, 78]     # Üst dudak noktaları
ALT_DUDAK = [14, 87, 86, 85, 84]     # Alt dudak noktaları

# Konuşma Algoritması
def konusma_algila(landmarks):
    dudak_mesafesi = dudak_mesafesi_hesapla(landmarks)
    konusma_esiği = 0.03  # Ayarlanabilir
    return dudak_mesafesi > konusma_esiği
```

1.10.4 Model Performansı

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

1.11 Sorun Giderme

1.11.1 Sık Karşılaşılan Hatalar

1.11.1.1 Model Dosyaları Bulunamadı

```
# Hata: src klasörü bulunamadı!
# Çözüm:
mkdir src
# Model dosyalarını src/ klasörüne ekleyin
```

1.11.1.2 Kamera Açılmıyor

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

1.11.1.3 Bağımlılık Hataları

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

1.11.1.4 CMake Hatası

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

# macOS
brew install cmake

# Windows
# Visual Studio Build Tools yükleyin
```

1.11.2 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

1.12 Performans Optimizasyonu

1.12.1 Hızlandırma İpuçları

1. **GPU Kullanımı**
```
# CUDA kurulu ise otomatik GPU kullanımı
cihaz = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```
2. **Frame Skip Ayarı**
```
# Yüksek frame skip = Daha hızlı işlem
frame_atlatma = 3  # Her 3 karede bir analiz
```
3. **Model Seçimi**
```
# Hız öncelikli: yolov8n-face.pt
# Doğruluk öncelikli: yolov11l-face.pt
```

1.12.2 Benchmark Sonuçları

Sistem Konfigürasyonu	FPS	CPU Kullanımı	RAM Kullanımı	İşlem Süresi
RTX 4050 + i5-13420H	75+	25%	3.5GB	$<$13ms
Sadece CPU (i5-13420H)	12-15	85%	4.2GB	$<$66ms

1.12.3 Proje Mimarisi

```
FaceStream Studio/
 app.py                  # Streamlit ana uygulama
 Analyzer.py             # FaceAnalyzer sınıfı
 requirements.txt        # Python bağımlılıkları
 src/
    yolov11l-face.pt    # YOLO yüz tespit modeli
    languages.json      # Dil yapılandırma dosyası
    models/
        torch/              # PyTorch duygu MLP
 docs/                   # Dokümantasyon ve grafikler
```

1.12.4 İşlem Akışı

1. **Video Girişi** → Kamera/Dosya
2. **Kare İşleme** → YOLO Yüz Tespiti
3. **Yüz Tanıma** → Encoding & Eşleştirme
4. **Landmark Tespiti** → MediaPipe Face Mesh
5. **Özellik Çıkarımı** → 936 boyutlu vektör
6. **Duygu Tahmini** → PyTorch MLP çıkarımı
7. **Konuşma Tespiti** → Dudak mesafesi analizi
8. **Sonuç Toplama** → Anlık istatistikler
9. **Arayüz Güncelleme** → Streamlit görselleştirme

1.12.5 Bağımlılıklar

```
# Temel Bağımlılıklar (requirements.txt)
streamlit           # Web arayüzü framework'ü
opencv-python       # Bilgisayarlı görü
ultralytics         # YOLO modelleri
mediapipe           # Yüz mesh tespiti
numpy               # Sayısal hesaplama
Pillow              # Görüntü işleme
cmake               # C++ derleme sistemi

# Ek Python Paketleri (otomatik kurulum)
torch               # PyTorch derin öğrenme
torchvision         # PyTorch için bilgisayarlı görü
face-recognition    # Yüz encoding/tanıma
scikit-learn        # Geleneksel ML modelleri
pandas              # Veri işleme
joblib              # Model serileştirme
```

1.12.6 Bilinen Sınırlamalar

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

1.12.7 Geliştirme Ortamı

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

1.13 Katkıda Bulunma

1.13.1 Geliştirme Ortamı Kurulumu

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

# Geliştirme bağımlılıklarını yükleyin
pip install -r requirements.txt
pip install pytest pytest-cov black flake8

# Pre-commit hook kurulumu (isteğe bağlı)
pip install pre-commit
pre-commit install
```

1.13.2 Katkı Süreci

1. **Fork** yapın ve **özellik dalı** oluşturun
   ```
   git checkout -b ozellik/harika-yeni-ozellik
   ```
2. **Kod yazın** ve **test edin**
   ```
   python -m pytest tests/ -v
   black . --check
   flake8 .
   ```
3. **Commit** edin ve **push** yapın
   ```
   git commit -m "feat: harika yeni özellik eklendi"
   git push origin ozellik/harika-yeni-ozellik
   ```
4. **Pull Request** oluşturun

1.13.3 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():
    # Duygu sınıflandırma testi
    pass
```

1.13.4 Katkı Kuralları

- **Kod Stili**: Black formatter kullanın
- **Dokümantasyon**: Yeni fonksiyonlar için docstring ekleyin
- **Test**: Yeni özellikler için test yazın
- **Commit Mesajları**: `Conventional Commits` formatını kullanın

1.14 Geliştirme

1.14.1 VS Code Görevleri

Bu proje VS Code görevleri ile gelir:

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

1.14.2 Hata Ayıklama Modu

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

# Analyzer.py içinde
if DEBUG_MODE:
    print(f"Kare işleme süresi: {processing_time:.2f}ms")
    print(f"Tespit edilen yüzler: {len(faces)}")
```

1.14.3 Profiling

```
# Bellek profili
pip install memory-profiler
python -m memory_profiler app.py

# Performans profili
pip install line-profiler
kernprof -l -v app.py
```

1.14.4 Katkı Süreci (Detaylı)

1. **Depoyu Fork Edin**
   - GitHub üzerinden projeyi fork edin
   - Kendi hesabınıza kopyalayın
2. **Yerel 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. **Özellik Dalı Oluşturun**
   ```
   git checkout -b ozellik/ozellik-adi
   ```

4. **Kod Geliştirin ve Test Edin**

```
# Kodunuzu yazın
# Testleri çalıştırın
python -m pytest tests/ -v

# Kod kalitesi kontrolü
black . --check
flake8 .
```

5. **Commit ve Push**

```
git add .
git commit -m "feat: harika özelliğiniz eklendi"
git push origin ozellik/ozellik-adi
```

6. **Pull Request Oluşturun**
   - GitHub'da Pull Request açın
   - Detaylı açıklama yazın
   - İnceleme bekleyin

1.14.5 Gelişim Yol Haritası

- ☐ **Gerçek Zamanlı Gösterge Paneli**: Canlı metrikler ve performans izleme
- ☐ **Toplu İşleme**: Çoklu video dosyalarını toplu işleme
- ☐ **API Uç Noktası**: REST API ile entegrasyon desteği
- ☐ **Bulut Entegrasyonu**: AWS/Azure bulut dağıtımı
- ☐ **Mobil Uygulama**: React Native ile mobil uygulama
- ☐ **Gelişmiş Analitik**: Detaylı istatistik ve raporlama
- ☐ **Çok Dilli Modeller**: Farklı etnik kökenler için optimize modeller

---

1.15 İletişim ve Destek

1.15.1 İletişim

- **E-posta**: muhakaplan@hotmail.com
- **GitHub**: github.com/oneoblomov
- **LinkedIn**: m-kaplan

1.15.2 Destek

- **Hata Bildirimi**: GitHub Issues kullanarak hata bildirimi
- **Özellik Önerisi**: Yeni özellik önerileri için Discussions
- **Dokümantasyon**: README ve kod içi yorumlar
- **Teknik Sorular**: Issues bölümünde Q&A etiketi ile

1.15.3 Proje Durumu

- **Temel Özellikler**: Tamamlandı ve test edildi
- **Optimizasyon**: Devam eden geliştirmeler
- **Performans**: GPU hızlandırma ve önbellekleme
- **Çoklu Dil Desteği**: Aktif

1.16 Lisans

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

1.17 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

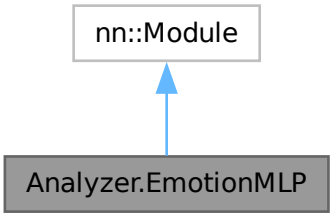
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

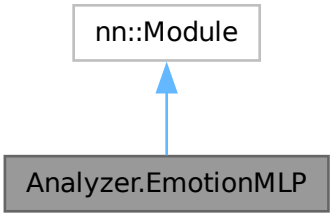
Class Documentation

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** = {}
- **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)
- _extract_landmark_features (self, landmarks)

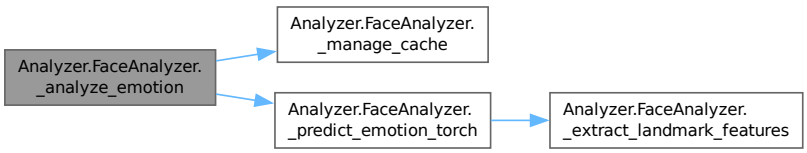
4.2.1 Member Function Documentation

4.2.1.1 _analyze_emotion()

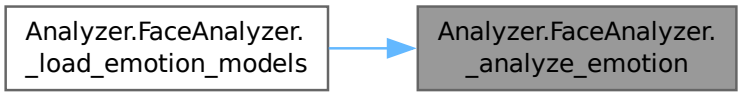
```
Analyzer.FaceAnalyzer._analyze_emotion (
            self,
            landmarks,
            landmark_id = None)  [protected]
```

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

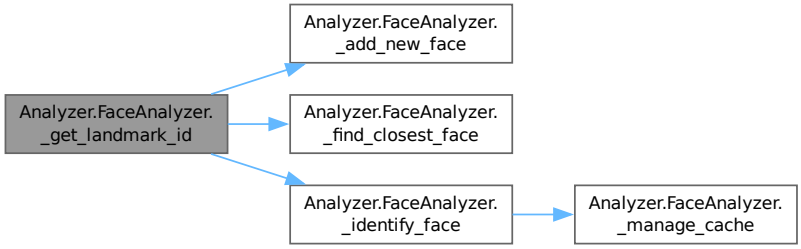
Here is the caller graph for this function:

4.2.1.3 _get_landmark_id()

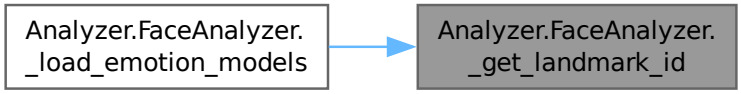
```
Analyzer.FaceAnalyzer._get_landmark_id (
            self,
            landmarks,
            w,
            h)  [protected]
```

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:

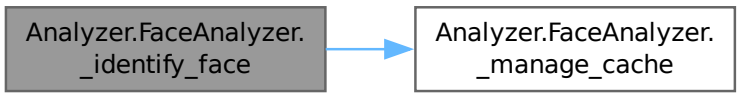


4.2.1.4 _identify_face()

```
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:



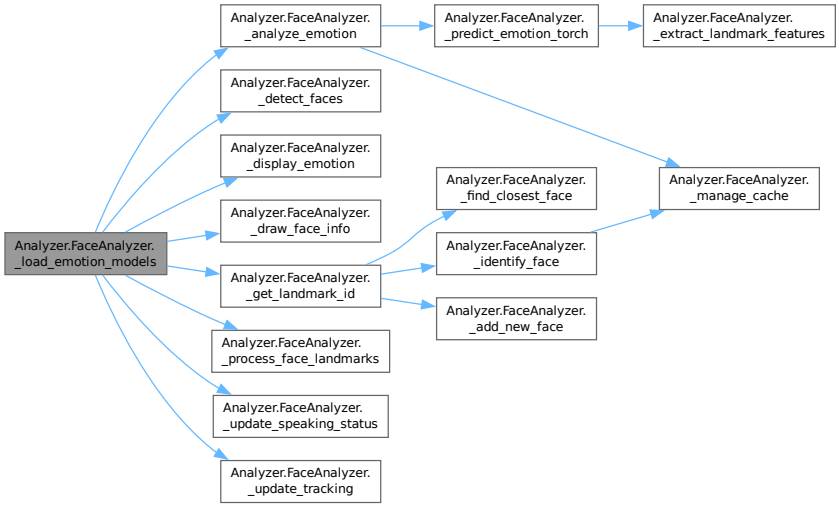
Here is the caller graph for this function:

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:

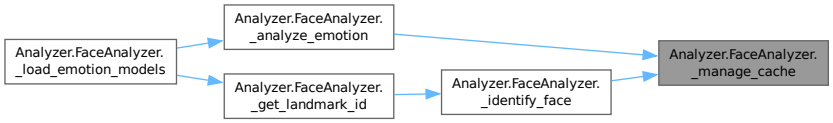


4.2.1.6 _manage_cache()

```
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:

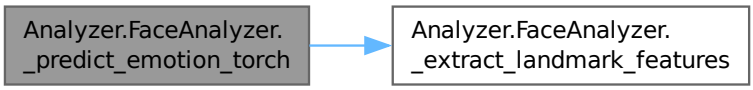


4.2.1.7 _predict_emotion_torch()

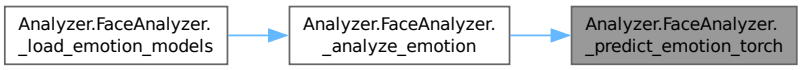
```
Analyzer.FaceAnalyzer._predict_emotion_torch (
              self,
              landmarks)  [protected]
```

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