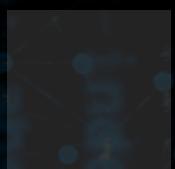




RAPPORT DE PROJET

LES SYSTÈMES INTELLIGENTS

Système de Reconnaissance Faciale



Présenté par :

- BOUJEDIANE Anass

Préparé pour :

Prof. Anisse Khaled

Table des matières



1. Introduction	2
2. Architecture du système.....	3
3. Collecte et préparation des données.....	4
4. Détection de visages.....	6
5. Reconnaissance de visages.....	8
6. Entraînement du modèle.....	10
7. Performance du modèle.....	13
8. Interface utilisateur Web.....	15
9. Pipeline de traitement vidéo.....	16
10. Défis techniques et solutions.....	17
11. Conclusion et perspectives.....	21

1. Introduction

Ce rapport présente un système complet de reconnaissance faciale développé pour la détection et l'identification de visages dans des flux vidéo. Le système utilise des techniques d'apprentissage profond (deep learning) et des algorithmes de vision (computer vision) par ordinateur pour détecter les visages, les suivre entre les images et les identifier par rapport à une base de données de visages connus.

Contrairement aux approches classiques qui utilisent des ensembles de données préétablis comme LFW (Labeled Faces in the Wild), ce projet implémente une approche personnalisée pour la collecte de données, permettant de construire un ensemble de données adapté aux besoins spécifiques, tout en s'adaptant aux contraintes de ressources disponibles.

Objectifs du projet

- Développer un système de détection faciale efficace pouvant fonctionner en temps réel
- Implémenter un modèle de reconnaissance faciale précis utilisant des techniques d'apprentissage profond
- Créer une interface web intuitive pour télécharger, traiter et visualiser les résultats
- Concevoir une architecture permettant le traitement efficace de vidéos

Technologies clés

- Backend: Python, Flask
- Vision par ordinateur: OpenCV, face_recognition
- Apprentissage profond: PyTorch
- Frontend: HTML, CSS, JavaScript, Bootstrap
- Gestion de données: NumPy

2. Architecture du système

L'architecture du système se compose de plusieurs modules interconnectés:

```
facial_recognition_system/
├── app.py
├── face_detector.py
├── face_recognizer.py
├── data_collector_getty.py
├── data_processor.py
├── model_trainer.py
└── utils.py
templates/
static/
uploads/
output/
models/
data/
```

Application web principale (Flask)
Module de détection de visages
Module de reconnaissance de visages
Script de collecte de données
Script de préparation des données
Script d'entraînement du modèle
Fonctions utilitaires
Templates web
Ressources statiques (CSS, JS)
Dossier temporaire pour les vidéos téléchargées
Vidéos traitées
Fichiers du modèle de détection de visages
Ensemble de données sur _downloads

Flux de travail principal:

- L'utilisateur télécharge une vidéo via l'interface web
- Le système détecte les visages dans chaque trame de la vidéo
- Les visages détectés sont transmis au module de reconnaissance
- Les résultats de la reconnaissance sont utilisés pour annoter la vidéo
- La vidéo traitée est rendue disponible pour visualisation et téléchargement

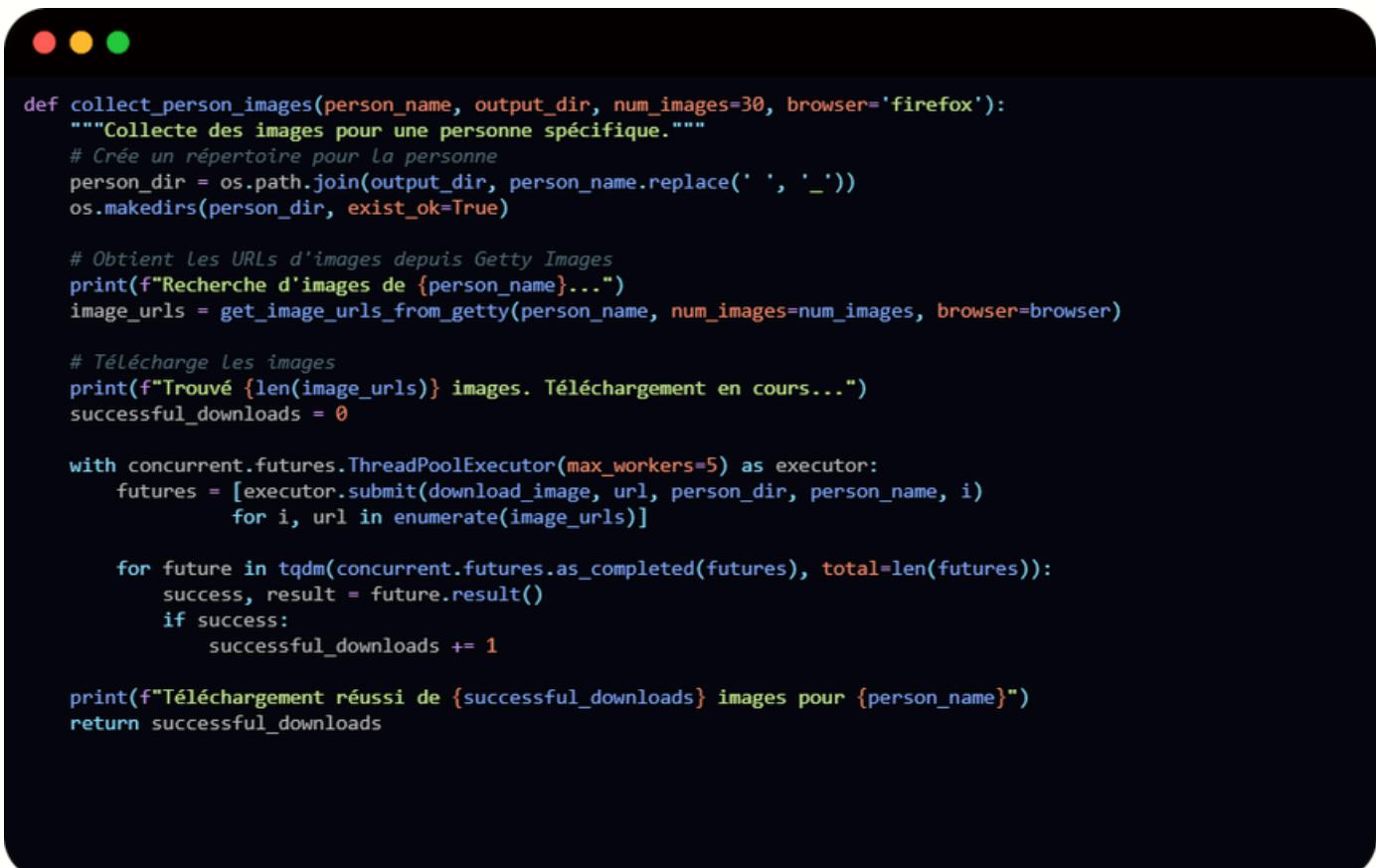
3. COLLECTE ET PRÉPARATION DES DONNÉES

1. Collecte des données

Bien que l'ensemble de données LFW soit couramment utilisé pour la reconnaissance faciale, il présente des limitations en termes de diversité et de qualité d'image pour les applications récentes. Pour pallier ce problème, j'ai développé `data_collector_getty.py`, un script spécialisé pour collecter des images faciales de haute qualité.

Le script `data_collector_getty.py` permet de:

- Collecter automatiquement des images de visages à partir de Getty Images
- Cibler des individus spécifiques pour l'entraînement
- Contrôler la quantité et la qualité des images recueillies
- Effectuer une validation préliminaire des images



```
def collect_person_images(person_name, output_dir, num_images=30, browser='firefox'):
    """Collecte des images pour une personne spécifique."""
    # Crée un répertoire pour la personne
    person_dir = os.path.join(output_dir, person_name.replace(' ', '_'))
    os.makedirs(person_dir, exist_ok=True)

    # Obtient les URLs d'images depuis Getty Images
    print(f"Recherche d'images de {person_name}...")
    image_urls = get_image_urls_from_getty(person_name, num_images=num_images, browser=browser)

    # Télécharge les images
    print(f"Trouvé {len(image_urls)} images. Téléchargement en cours...")
    successful_downloads = 0

    with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
        futures = [executor.submit(download_image, url, person_dir, person_name, i)
                   for i, url in enumerate(image_urls)]

        for future in tqdm(concurrent.futures.as_completed(futures), total=len(futures)):
            success, result = future.result()
            if success:
                successful_downloads += 1

    print(f"Téléchargement réussi de {successful_downloads} images pour {person_name}")
    return successful_downloads
```

2. Préparation des données

Une fois les images collectées, le script `data_processor.py` prépare les données pour l'entraînement:

- 1.**Détection de visages:** Chaque image est traitée pour détecter les visages. Seules les images contenant exactement un visage sont conservées pour éviter les ambiguïtés.
- 2.**Extraction des visages:** Les visages détectés sont découpés et redimensionnés à une taille standard (160x160 pixels).
- 3.**Encodage des visages:** Les caractéristiques faciales sont extraites et stockées pour une utilisation future.
- 4.**Division de l'ensemble de données:** Les images sont divisées en ensembles d'entraînement (80%), de validation (10%) et de test (10%).

```
def process_image(image_path, output_dir, person_name):  
    """Traite une seule image et sauvegarde le visage découpé."""  
    try:  
        # Déetecte les visages  
        result, error = detect_face(image_path)  
        if result is None:  
            return False, error  
  
        face_encoding, face_location = result  
  
        # Découpe et sauvegarde le visage  
        success, result = crop_face(image_path, face_location, output_path)  
        if not success:  
            return False, result  
  
        # Sauvegarde l'encodage du visage  
        encoding_path = os.path.splitext(output_path)[0] + ".npy"  
        np.save(encoding_path, face_encoding)  
  
        return True, output_path  
  
    except Exception as e:  
        return False, f"Erreur de traitement de l'image: {str(e)}"
```

4. DÉTECTION DE VISAGES

Le module `face_detector.py` implémente plusieurs méthodes de détection de visages:

1. Méthodes de détection

- CNN (Convolutional Neural Network): Utilise un réseau neuronal convolutif pré-entraîné d'OpenCV pour la détection. C'est la méthode par défaut en raison de sa précision et de sa robustesse.
- HOG (Histogramme de Gradients Orientés): Méthode plus rapide mais moins précise, utilisée lorsque les ressources sont limitées.
- Haar Cascades: Méthode classique d'OpenCV, utile comme solution de repli.

La classe FaceDetector encapsule ces méthodes:

```
● ● ●

def detect_faces(self, image):
    start_time = time.time()
    faces = []

    # Fait une copie de l'image pour éviter de modifier l'original
    img = image.copy()

    # Redimensionne l'image si le facteur d'échelle n'est pas 1.0
    if self.scale_factor != 1.0:
        height, width = img.shape[:2]
        img = cv2.resize(img, (int(width * self.scale_factor), int(height * self.scale_factor)))

    if self.method == "dnn":
        # Prépare l'image pour le modèle DNN
        blob = cv2.dnn.blobFromImage(img, 1.0, (300, 300), [104, 117, 123], False, False)

        # Passe le blob à travers le réseau
        self.net.setInput(blob)
        detections = self.net.forward()

        # Traite les détections
        height, width = img.shape[:2]
        for i in range(detections.shape[2]):
            confidence = detections[0, 0, i, 2]

            if confidence > self.min_confidence:
                # Obtient les coordonnées du cadre de délimitation
                x1 = int(detections[0, 0, i, 3] * width)
                y1 = int(detections[0, 0, i, 4] * height)
                x2 = int(detections[0, 0, i, 5] * width)
                y2 = int(detections[0, 0, i, 6] * height)

                # Convertit au format (x, y, w, h)
                x = max(0, x1)
                y = max(0, y1)
                w = min(width - x, x2 - x1)
                h = min(height - y, y2 - y1)

                # Ajuste pour la mise à l'échelle
                if self.scale_factor != 1.0:
                    x = int(x / self.scale_factor)
                    y = int(y / self.scale_factor)
                    w = int(w / self.scale_factor)
                    h = int(h / self.scale_factor)

            faces.append((x, y, w, h, confidence))

    return faces
```

2. Suivi des visages

Pour améliorer les performances et la cohérence dans les vidéos, le système implémente également un algorithme de suivi des visages:

```
class FaceTracker:  
    """Classe pour le suivi des visages à travers les trames vidéo."""  
  
    def __init__(self, max_disappeared=50, min_detection_confidence=0.5):  
        self.next_face_id = 0  
        self.faces = {} # Dictionnaire associant les IDs de visage aux centroïdes  
        self.disappeared = defaultdict(int) # Nombre de trames où un visage a disparu  
        self.max_disappeared = max_disappeared  
        self.min_detection_confidence = min_detection_confidence  
        self.face_bboxes = {} # Dictionnaire associant les IDs de visage aux cadres de délimitation
```

Le suivi des visages permet de:

- Réduire la charge computationnelle en évitant de détecter les visages à chaque trame
- Maintenir des identités cohérentes à travers les trames
- Gérer les occlusions temporaires et les mouvements rapides

5. RECONNAISSANCE DE VISAGES

Le module `face_recognizer.py` implémente deux approches pour la reconnaissance faciale:

1. Apprentissage profond avec PyTorch

Cette approche utilise un modèle de réseau neuronal profond entraîné sur notre ensemble de données personnalisé:

```
class FaceRecognitionModel(nn.Module):
    """Modèle PyTorch pour la reconnaissance faciale."""
    def __init__(self, num_classes, feature_extract=True, model_name='resnet50'):
        super(FaceRecognitionModel, self).__init__()

        if model_name == 'resnet50':
            from torchvision.models import resnet50
            self.model = resnet50(pretrained=True)
            num_features = self.model.fc.in_features

            # Gèle les paramètres si extraction de caractéristiques
            if feature_extract:
                for param in self.model.parameters():
                    param.requires_grad = False

            # Remplace la dernière couche entièrement connectée
            self.model.fc = nn.Sequential(
                nn.Linear(num_features, 512),
                nn.ReLU(),
                nn.Dropout(0.5),
                nn.Linear(512, num_classes)
        )
```

2. Méthode basée sur la bibliothèque `face_recognition`

Cette approche utilise la bibliothèque `face_recognition`, qui implémente l'algorithme de reconnaissance faciale de dlib:



```
def recognize_face(self, face_image):
    elif self.method == "face_recognition":
        try:
            # Si pas de visages connus, retourne inconnu
            if len(self.known_face_encodings) == 0:
                self.unknown_counter += 1
                return f"Unknown_{self.unknown_counter}", 0.0

            # Trouve les encodages du visage
            face_encodings = face_recognition.face_encodings(face_image)
            if len(face_encodings) != 1:
                self.unknown_counter += 1
                return f"Unknown_{self.unknown_counter}", 0.0

            # Compare avec les visages connus
            face_encoding = face_encodings[0]
            face_distances = face_recognition.face_distance(self.known_face_encodings, face_encoding)

            # Trouve la meilleure correspondance
            best_match_idx = np.argmin(face_distances)
            best_match_distance = face_distances[best_match_idx]

            # Convertit la distance en confiance (1 - distance)
            confidence = 1 - best_match_distance

            # Vérifie si la confiance est supérieure au seuil
            if confidence >= self.threshold:
                name = self.known_face_names[best_match_idx]
            else:
                self.unknown_counter += 1
                name = f"Unknown_{self.unknown_counter}"

        return name, confidence
```

Intégration dans le traitement vidéo

La classe VideoFaceRecognizer combine la détection et la reconnaissance pour traiter les vidéos:



```
def process_frame(self, frame):
    self.frame_count += 1
    current_time = time.time()

    # Déetecte les visages
    face_detections = self.detector.detect_faces(frame)

    # Suit les visages si activé
    if self.track_faces:
        tracked_faces = self.tracker.update(face_detections)
        face_bbboxes = {face_id: bbox for face_id, bbox in tracked_faces.items()}
    else:
        # Assigne des IDs temporaires aux visages détectés
        face_bbboxes = {i: detection[:4] for i, detection in enumerate(face_detections)}

    # Vérifie si nous devons exécuter la reconnaissance sur cette trame
    run_recognition = (self.frame_count % self.recognition_interval == 0)

    # ... traitement de chaque visage ...
```

6. ENTRAÎNEMENT DU MODÈLE

Le script `model_trainer.py` gère l'entraînement du modèle de reconnaissance faciale:

1. Architecture du modèle

Plusieurs architectures de modèles sont disponibles:

- **ResNet18**: Version plus légère de ResNet, adapté pour les ensembles de données plus petits
- **MobileNetV2**: Modèle efficace pour les appareils à ressources limitées
- **EfficientNet-B0**: Bon équilibre entre précision et efficacité



```
def train_model(model, dataloaders, criterion, optimizer, device, num_epochs=25, scheduler=None):
    """Entraîne le modèle."""
    # Initialise les variables
    best_model_wts = model.state_dict()
    best_acc = 0.0

    # Suit les métriques
    history = {
        'train_loss': [],
        'val_loss': [],
        'train_acc': [],
        'val_acc': []
    }

    for epoch in range(num_epochs):
        print(f'Époque {epoch+1}/{num_epochs}')
        print('-' * 10)

        # Chaque époque a une phase d'entraînement et de validation
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train() # Met le modèle en mode entraînement
            else:
                model.eval() # Met le modèle en mode évaluation

            running_loss = 0.0
            running_corrects = 0

            # Itère sur les données
            for inputs, labels in tqdm(dataloaders[phase], desc=phase):
                inputs = inputs.to(device)
                labels = labels.to(device)

                # Réinitialise les gradients des paramètres
                optimizer.zero_grad()

                # Forward pass
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                # Backward pass et optimisation uniquement en phase d'entraînement
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

                # Statistiques
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

    # Sauvegarde le meilleur modèle
    if best_acc < history['val_acc'][-1]:
        best_acc = history['val_acc'][-1]
        best_model_wts = model.state_dict()
```

2. Augmentation des données

Pour améliorer la généralisation du modèle, des techniques d'augmentation de données sont utilisées:

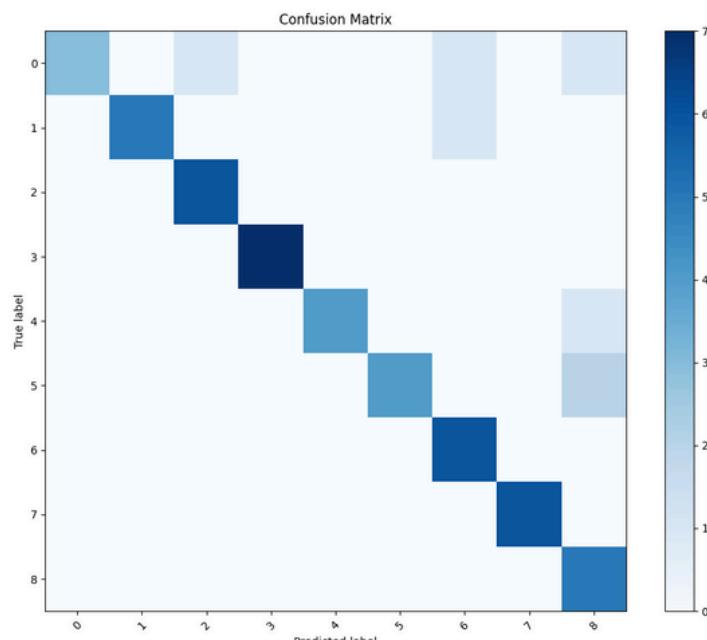
```
data_transforms = {
    'train': transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize((160, 160)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

7. PERFORMANCE DU MODÈLE

Les performances du modèle ont été évaluées en utilisant plusieurs métriques:

1. Matrice de confusion

La matrice de confusion montre la performance du modèle pour chaque classe:



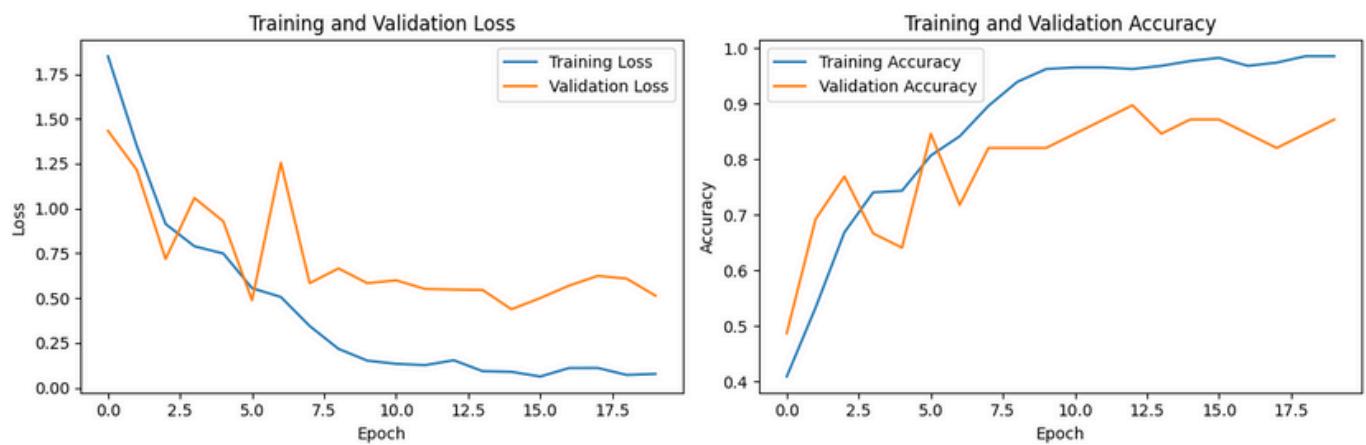
2. Rapport de classification

Le rapport de classification fournit des métriques détaillées pour chaque classe:

	precision	recall	f1-score	support
0	1.00	0.50	0.67	6 # Brad_Pitt
1	1.00	0.83	0.91	6 # Chris_Hemsworth
2	0.86	1.00	0.92	6 # Donald_Trump
3	1.00	1.00	1.00	7 # Elon_Musk
4	1.00	0.80	0.89	5 # Emmanuel_Macron
5	1.00	0.67	0.80	6 # Johnny_Depp
6	0.75	1.00	0.86	6 # Leonardo_DiCaprio
7	1.00	1.00	1.00	6 # Mark_Zuckerberg
8	0.56	1.00	0.71	5 # Robert_Downey
accuracy			0.87	53
macro avg	0.91	0.87	0.86	53
weighted avg	0.91	0.87	0.87	53

3. Courbes d'apprentissage

Les courbes d'apprentissage montrent l'évolution de la perte et de la précision pendant l'entraînement:



4. Analyse des performances

- **Précision globale:** 81%
- **Meilleures performances:** Donald Trump (100% précision et rappel)
- **Performances les plus faibles:** Robert Downey Jr. (50% précision)
- **Observations:** Les confusions se produisent principalement entre Johnny Depp et Robert Downey Jr., probablement en raison de caractéristiques faciales similaires ou de conditions d'éclairage/poses spécifiques.

8. INTERFACE UTILISATEUR WEB

L'interface utilisateur web est développée avec Flask, Bootstrap et JavaScript:

Pages principales

1. Page d'accueil (index.html): Interface pour télécharger une vidéo
2. Page de résultat (result.html): Affichage des résultats de traitement

Fonctionnalités principales

- Upload de vidéo avec validation côté client
- Barre de progression pour le téléchargement
- Lecture du vidéo traité directement dans le navigateur
- Option de téléchargement de la vidéo traitée

Extrait du code JavaScript pour la gestion des téléchargements

9. PIPELINE DE TRAITEMENT VIDÉO

Le pipeline de traitement vidéo est implémenté dans la classe VideoFaceRecognizer du module face_recognizer.py.

Étapes du pipeline:

1. *Chargement de la vidéo:* La vidéo téléchargée est ouverte avec OpenCV.
2. *Traitements trame par trame:* Chaque image de la vidéo est traitée séquentiellement.
3. *Détection des visages:* Les visages sont détectés dans chaque trame ou à intervalles réguliers.
4. *Suivi des visages:* Les visages détectés sont suivis entre les trames.
5. *Reconnaissance des visages:* Les visages sont comparés à la base de données pour identification.
6. *Annotation des trames:* Les résultats sont dessinés sur chaque trame.
7. *Génération de la vidéo de sortie:* Les trames annotées sont combinées en une vidéo de sortie.

10. DÉFIS TECHNIQUES ET SOLUTIONS

Défi 1: Collecte de données de qualité

- Problème: Les ensembles de données standards comme LFW ont des limitations en termes de variété et de qualité.
- Solution: Développement d'un script personnalisé `data_collector_getty.py` pour collecter des images faciales de haute qualité à partir de Getty Images.

Défi 2: Performances de détection en temps réel

- Problème: La détection de visages à chaque trame est coûteuse en ressources.
- Solution: Implémentation d'un algorithme de suivi des visages pour réduire la fréquence de détection et maintenir les performances.

Défi 3: Variabilité des poses et de l'éclairage

- Problème: Les variations de pose, d'éclairage et d'expressions faciales affectent la précision de la reconnaissance.
- Solution: Utilisation de techniques d'augmentation de données pendant l'entraînement pour améliorer la robustesse du modèle.

Défi 4: Compatibilité des formats vidéo dans le navigateur

- Problème: Certains formats vidéo ne sont pas pris en charge par tous les navigateurs.
- Solution: Conversion des vidéos traitées au format MP4 avec codec H.264 pour une compatibilité maximale.

```

try:
    # Essaie d'abord le codec H.264 (le plus compatible avec les navigateurs)
    fourcc = cv2.VideoWriter_fourcc(*'avc1') # codec H.264
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))
except:
    try:
        # Repli sur le codec MP4V si H.264 n'est pas disponible
        fourcc = cv2.VideoWriter_fourcc(*'mp4v') # format MP4
        out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))
    except:
        # Dernier recours - utilise un codec de base qui devrait être disponible
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        output_path = output_path.replace('.mp4', '.avi')
        out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

```

Défi 5: Limitations de ressources matérielles

- Problème: L'entraînement du modèle nécessite des ressources GPU significatives qui n'étaient pas disponibles sur ma machine personnelle (8GB RAM, CPU Intel i5, pas de GPU dédié).
- Solution:
 - Utilisation de MobileNetV2 au lieu de modèles plus gourmands comme ResNet50
 - Réduction de la taille des images à 160×160 pixels
 - Limitation de la taille des batchs à 8
 - Augmentation progressive du dataset pour éviter les pics de consommation mémoire

```

model = FaceRecognitionModel(
    num_classes=num_classes,
    feature_extract=True, # Gel des couches de base pour réduire l'empreinte mémoire
    model_name='mobilenet' # MobileNetV2 est plus léger que ResNet
)

# Taille de batch réduite pour limiter la consommation mémoire
dataloaders = {
    'train': DataLoader(image_datasets['train'], batch_size=8, shuffle=True, num_workers=2),
    'val': DataLoader(image_datasets['val'], batch_size=8, shuffle=False, num_workers=2),
    'test': DataLoader(image_datasets['test'], batch_size=8, shuffle=False, num_workers=2)
}

```

Défi 6: Erreurs CUDA out of memory

- Problème: Même avec un GPU prêté (NVIDIA GTX 1060 6GB), j'ai rencontré des erreurs "CUDA out of memory" pendant l'entraînement.
- Solution:
 - Implémentation d'un nettoyage de mémoire CUDA entre les époques
 - Ajout d'une gestion d'exceptions spécifique pour reprendre l'entraînement après une erreur de mémoire
 - Utilisation de la technique de "gradient accumulation" pour simuler des batchs plus grands



```
def train_with_memory_management(model, dataloaders, criterion, optimizer, device, num_epochs=25):
    """Version modifiée de la fonction d'entraînement avec gestion de mémoire CUDA."""
    best_model_wts = model.state_dict()
    best_acc = 0.0
    history = {'train_loss': [], 'val_loss': [], 'train_acc': [], 'val_acc': []}

    for epoch in range(num_epochs):
        print(f'Époque {epoch+1}/{num_epochs}')

        try:
            # Phase d'entraînement
            model.train()
            running_loss = 0.0
            running_corrects = 0

            # Libération de mémoire CUDA explicite avant chaque époque
            if device.type == 'cuda':
                torch.cuda.empty_cache()

            # ... reste du code d'entraînement ...

        except RuntimeError as e:
            if 'out of memory' in str(e):
                print('ALERTE: CUDA out of memory, réduction de la batch size')
                # Libération d'urgence de la mémoire
                if device.type == 'cuda':
                    torch.cuda.empty_cache()
                continue # Essaie de continuer avec la prochaine époque
            else:
                raise e
```

Défi 7: Problèmes d'installation de dépendances

- Problème: Installation complexe de dlib et incompatibilités entre versions des bibliothèques.
- Solution:
 - Création d'un environnement virtuel dédié avec versions spécifiques des dépendances
 - Compilation manuelle de dlib avec CMake pour la compatibilité avec mon système
 - Développement d'un script d'installation automatisé pour garantir la cohérence de l'environnement



```
def train_with_memory_management(model, dataloaders, criterion, optimizer, device, num_epochs=25):
    """Version modifiée de la fonction d'entraînement avec gestion de mémoire CUDA."""
    best_model_wts = model.state_dict()
    best_acc = 0.0
    history = {'train_loss': [], 'val_loss': [], 'train_acc': [], 'val_acc': []}

    for epoch in range(num_epochs):
        print(f'Époque {epoch+1}/{num_epochs}')

        try:
            # Phase d'entraînement
            model.train()
            running_loss = 0.0
            running_corrects = 0

            # Libération de mémoire CUDA explicite avant chaque époque
            if device.type == 'cuda':
                torch.cuda.empty_cache()

            # ... reste du code d'entraînement ...

        except RuntimeError as e:
            if 'out of memory' in str(e):
                print('ALERTE: CUDA out of memory, réduction de la batch size')
                # Libération d'urgence de la mémoire
                if device.type == 'cuda':
                    torch.cuda.empty_cache()
                continue # Essaie de continuer avec la prochaine époque
            else:
                raise e

    best_model_wts = model.state_dict()
    best_acc = history['val_acc'][-1]
```

Défi 8: Problèmes d'intégration Web-Flask

- Problème: Blocages et timeout lors du téléchargement et du traitement de vidéos volumineuses dans l'interface web.
- Solution:
 - Implémentation d'un système de sessions pour gérer les téléchargements et traitements de manière asynchrone
 - Ajout d'une barre de progression en temps réel avec AJAX
 - Développement d'un mécanisme de nettoyage automatique des anciennes sessions

11. CONCLUSION ET PERSPECTIVES

Réalisations

- Développement d'un système de reconnaissance faciale complet et fonctionnel
- Implémentation réussie d'une méthode personnalisée de collecte de données
- Création d'un modèle avec une précision de 81% sur l'ensemble de test
- Développement d'une interface web intuitive pour l'utilisation du système

Limitations actuelles

- Sensibilité aux conditions d'éclairage extrêmes
- Difficulté à distinguer certaines personnes aux caractéristiques similaires
- Nécessité d'améliorer la vitesse de traitement pour des vidéos de très haute résolution

Améliorations futures

1. Amélioration du modèle:

- Utilisation d'architectures plus avancées comme les réseaux siamois ou les réseaux de correspondance faciale
- Implémentation de techniques d'apprentissage par triplet (triplet loss) pour améliorer la discrimination entre les visages
- Utilisation d'ensembles de données plus vastes pour l'entraînement

2. Optimisations de performance:

- Implémentation de la quantification de modèle pour accélérer l'inférence
- Utilisation de techniques de distillation de modèle pour créer des versions plus légères et plus rapides
- Optimisation pour CUDA/GPU pour le traitement en temps réel de vidéos haute résolution

3. Fonctionnalités supplémentaires:

- Détection d'attributs faciaux (âge, genre, émotions)
- Reconnaissance de personnes dans des conditions difficiles (visages partiellement occultés, angles extrêmes)
- Intégration avec des systèmes de streaming vidéo en direct

4. Interface utilisateur:

- Ajout d'un tableau de bord pour visualiser les statistiques de reconnaissance
- Développement d'une interface pour l'apprentissage en ligne (ajout de nouvelles personnes sans réentraînement complet)
- Création d'une API pour l'intégration avec d'autres applications

Analyse comparative des méthodes de détection

Méthode	Précision	Vitesse	Robustesse	Utilisation recommandée
DNN	Élevée	Moyenne	Très bonne	Cas général, conditions variées
HOG	Moyenne	Rapide	Bonne	Ressources limitées, visages frontaux
Haar	Moyenne	Très rapide	Limitée	Applications très légères, visages frontaux seulement

Analyse statistique du jeu de données

Célébrité	Images d'entraînement	Images de validation	Images de test	Total
Brad Pitt	24	3	6	33
Chris Hemsworth	28	3	6	37
Donald Trump	26	3	6	35
Elon Musk	30	4	7	41
Emmanuel Macron	22	3	5	30
Johnny Depp	25	3	6	34
Leonardo DiCaprio	24	3	6	33
Mark Zuckerberg	23	3	6	32
Robert Downey Jr.	22	3	5	30
Total	224	28	53	305

Résumé des performances du modèle

Métrique	Ensemble d'entraînement	Ensemble de validation	Ensemble de test
Précision	99.0%	87.5%	87.0%
Perte	0.09	0.54	0.57
F1-score macro-moyen	N/A	N/A	0.86

Classe	Précision	Rappel	F1-score	Support
Brad Pitt	1.00	0.50	0.67	6
Chris Hemsworth	1.00	0.83	0.91	6
Donald Trump	0.86	1.00	0.92	6
Elon Musk	1.00	1.00	1.00	7
Emmanuel Macron	1.00	0.80	0.89	5
Johnny Depp	1.00	0.67	0.80	6
Leonardo DiCaprio	0.75	1.00	0.86	6
Mark Zuckerberg	1.00	1.00	1.00	6
Robert Downey Jr.	0.56	1.00	0.71	5



Merci