UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH
FACULTÉ DES SCIENCES DHAR EL MAHRAZ

Face Recognition System with Adversarial Attack using Reface in Deep Learning

Réalisé par :

Youssef Eljaouhary
Hisham Bamouh

Encadrant :

Jamal Rifii

Master WISD

Contents

1 Introduction **2**

2 Démonstration **3**

3 Détails de l'Implémentation **4**

4 background Mathématiques de la méthode Reface **4**
4.1 Objectif d'Optimisation pour les Attaques Adversariales 4
4.2 Perturbation du Réseau de Transformation Adversarial (ATN) . . . 5
4.3 Procédure d'Entraînement de l'Attaque par Ensemble 5

5 Explication de code **6**
5.1 Attaque Reface (`reface_attack.py`) 6
5.2 Génération des Encodages (`generate_embeddings.py`) 7
5.3 Reconnaissance Faciale (`face_recognition.py`) 7
5.4 Détecteur d'Attaque (`adversarial_detector.py`) 8

6 Conclusion **8**

7 References **8**

1 Introduction

Cette section présente un aperçu du système de **reconnaissance faciale**, en mettant en avant son objectif, ses principales fonctionnalités, ainsi que l'importance de l'étude des **attaques adversarielles**. Le système s'appuie sur l'apprentissage profond pour effectuer une reconnaissance faciale en temps réel. Il intègre également des mécanismes permettant de simuler et de **détecter des attaques adversarielles**, renforçant ainsi la robustesse et la **sécurité du système.**

Une **attaque adversariale** désigne une technique visant à tromper un modèle d'intelligence artificielle en modifiant légèrement les données d'entrée, de manière souvent imperceptible pour l'œil humain, mais suffisante pour induire une mauvaise prédiction du modèle. **Reface** est une méthode de génération d'images qui permet de superposer artificiellement un visage sur un autre, afin de simuler des attaques de type « impersonation ».

2 Démonstration

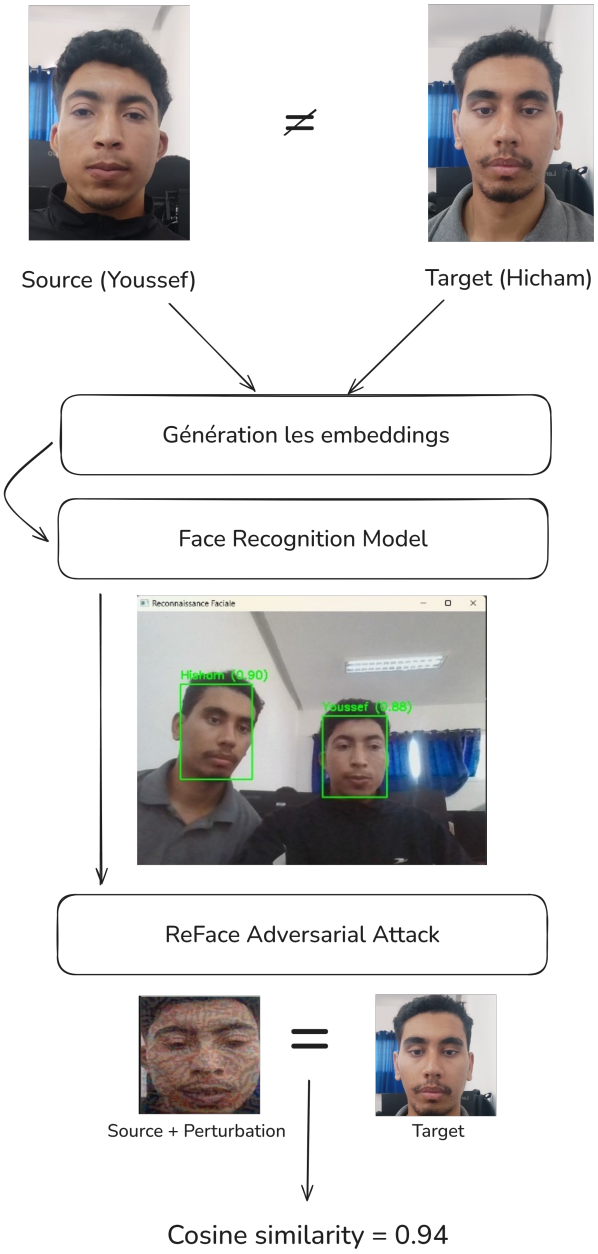

Figure 1: Démonstration du système de reconnaissance faciale et d'attaque.

Le système intègre la détection de visages, la génération d'**embeddings**, la reconnaissance faciale, ainsi que la simulation d'attaques adversarielles. Les composants clés comprennent **MTCNN** pour la détection des visages, **InceptionResnetV1** pour la génération des embeddings, ainsi que des modules personnalisés pour la simulation et la détection d'attaques adversarielles.

Cette section décrit le flux de travail du système ainsi que les dépendances logicielles nécessaires à son fonctionnement, telles que **PyTorch**, **OpenCV**, et d'autres bibliothèques liées à l'apprentissage profond et au traitement d'image.

3 Détails de l'Implémentation

Cette section décrit l'implémentation du système en Python.

- **`generate_embeddings.py`** : Génère les empreintes faciales (embeddings) pour les individus connus à l'aide du modèle `InceptionResnetV1`.
- **`face_recognition.py`** : Effectue la reconnaissance faciale en temps réel à partir d'une webcam, avec une prise en charge de la simulation d'attaques adversariales.
- **`reface_attack.py`** : Implémente une attaque adversariale visant à manipuler les embeddings faciaux en utilisant des techniques de substitution de visages.
- **`adversarial_detector.py`** : Détecte les perturbations adversariales en comparant les normes L2 des embeddings.

Les algorithmes clés utilisés incluent :

- La **similarité cosinus** pour la reconnaissance faciale, permettant de mesurer la proximité entre les embeddings.
- L'**optimisation basée sur le gradient** pour générer les attaques adversariales, en perturbant les images d'entrée de manière ciblée.

4 background Mathématiques de la méthode Reface

Cette section décrit l'attaque adversariale implémentée dans `reface_attack.py`, qui utilise une optimisation basée sur le gradient pour perturber les images d'entrée dans le but d'induire en erreur le système de reconnaissance faciale.

L'analyse porte sur les objectifs de l'attaque, les paramètres utilisés (par exemple, le taux d'apprentissage, la régularisation L2), ainsi que son impact sur la précision du système de reconnaissance. L'attaque cherche à produire des images visuellement similaires à l'original mais dont les embeddings faciaux sont suffisamment modifiés pour provoquer une mauvaise identification.

4.1 Objectif d'Optimisation pour les Attaques Adversariales

L'objectif principal de l'attaque ReFace est de générer des perturbations adversariales qui maximisent la distance cosinus entre les plongements des images originale et adversariales, tout en minimisant la perceptibilité de la perturbation. La formulation mathématique est la suivante :

$$\forall x \in X \quad \max \left[d(F(x_{\text{adv}}), F(x)) - \lambda L_{\text{pips}}(x_{\text{adv}}, x) \right] \tag{1}$$

où

$$x_{\text{adv}} = \text{clip}_{[0,1]} \left(x + g_\theta(x) \right) \tag{2}$$

sous la contrainte

$$\|g_\theta(x)\|_\infty < \epsilon \tag{3}$$

Explication :

- **Équation 1** : L'objectif maximise la distance cosinus $d(F(x_{\text{adv}}), F(x))$ entre les plongements de l'image adversariales $F(x_{\text{adv}})$ et de l'image originale $F(x)$, où F est le modèle de reconnaissance faciale. Le terme $-\lambda L_{\text{pips}}(x_{\text{adv}}, x)$ utilise la perte *Learned Perceptual Image Patch Similarity* (LPIPS) pour minimiser la perceptibilité visuelle, avec λ comme coefficient de pondération.
- **Équation 2** : L'image adversariales x_{adv} est créée en ajoutant la perturbation $g_\theta(x)$, générée par le réseau de transformation adversarial (ATN) paramétré par θ, à l'image originale x. La fonction $\text{clip}_{[0,1]}$ garantit que les valeurs des pixels restent dans l'intervalle $[0, 1]$.
- **Équation 3** : La contrainte sur la norme L_∞ limite la perturbation $g_\theta(x)$ à une distorsion maximale ϵ, assurant une quasi-imperceptibilité.

Cette formulation cible l'espace des plongements pour perturber les tâches de vérification et d'identification faciale en éloignant les plongements adversariales de leurs homologues originaux sur la sphère unitaire.

4.2 Perturbation du Réseau de Transformation Adversarial (ATN)

L'ATN génère des perturbations à l'aide d'une architecture encodeur-décodeur neuronal. La perturbation est définie comme suit :

$$g_\theta(x) = \epsilon \cdot \tanh(N_\theta(x)) \tag{4}$$

Explication :

- La perturbation $g_\theta(x)$ est calculée en passant l'image d'entrée x à travers le réseau neuronal N_θ, suivi d'une fonction tangente hyperbolique ($\tanh$) mise à l'échelle par ϵ. La fonction $\tanh$ borne la sortie entre $[-1, 1]$, et la mise à l'échelle par ϵ garantit que la perturbation satisfait $\|g_\theta(x)\|_\infty < \epsilon$.
- Cela permet à l'ATN de produire des perturbations spécifiques à l'entrée en un seul passage avant, ce qui est beaucoup plus efficace que les méthodes itératives basées sur le gradient, comme PGD.

4.3 Procédure d'Entraînement de l'Attaque par Ensemble

L'ATN est entraîné pour cibler un ensemble de modèles de reconnaissance faciale à l'aide de la procédure suivante (Algorithme 1 dans l'article) :

Explication :

- L'algorithme entraîne l'ATN N_θ pour générer des perturbations adversariales pour un ensemble de modèles de reconnaissance faciale victimes F.
- Les images adversariales sont générées à l'aide de l'Équation 4 pour chaque lot.

Algorithm 1 Procédure d'Entraînement de l'Attaque par Ensemble

Entrées : Modèles victimes $F = \{F_1, \ldots, F_n\}$, jeu de données d'images X
Sortie : Paramètres θ du générateur de perturbations g_θ
Hyperparamètres : Taux d'apprentissage α, borne L_∞ ϵ, coefficient de perte LPIPS λ
Initialiser l'ATN : N_θ
Extraire un lot d'images d'entraînement : $X_{\text{batch}} \leftarrow \text{Batch}(X)$
for époque de 0 à N_{epoch} **do**
    for x dans X_{batch} **do**
        $x_{\text{adv}} \leftarrow \text{clip}_{[0,1]}\left(x + \epsilon \cdot \tanh(N_\theta(x))\right)$
        perte $\leftarrow 0$
        for F_i dans F **do**
            perte $\leftarrow$ perte $+ \left(-d(F_i(x), F_i(x_{\text{adv}}))\right)$
        end for
        perte $\leftarrow$ perte/len(F)
        perte $\leftarrow$ perte $+ \lambda L_{\text{pips}}(x_{\text{adv}}, x)$
        $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta(\text{perte})$
    end for
end for
Retourner θ

- La perte est la moyenne négative des distances cosinus sur tous les modèles victimes, encourageant les plongements adversariales à s'éloigner des originaux. La perte LPIPS minimise les différences perceptives.
- Les paramètres θ sont mis à jour via une descente de gradient avec un taux d'apprentissage α.
- L'entraînement par ensemble améliore la transférabilité des perturbations à des modèles non vus, comme démontré dans les expériences de l'article.

5 Explication de code

5.1 Attaque Reface (`reface_attack.py`)

La fonctionnalité principale de `reface_attack.py` est d'exécuter une attaque adversariale afin de manipuler l'encodage d'une image de visage pour qu'il ressemble à un encodage cible. L'extrait clé est la boucle d'optimisation qui perturbe l'image d'entrée.

```
for step in range(num_steps):
    optimizer.zero_grad()
    emb = model(perturbed)  # Encodage de  l image  perturb e
    cos_sim = F.cosine_similarity(emb, target_emb)
    loss = -cos_sim.mean()
    perturb = perturbed - source_tensor
    l2_loss = torch.norm(perturb.reshape(perturb.size(0), -1), p=2)
    loss += l2_weight * l2_loss
```

```
    loss.backward()
    optimizer.step()
    with torch.no_grad():
        perturbed.clamp_(-1, 1)  # Limiter les valeurs des pixels
```

Cette boucle ajuste de manière itérative l'image d'entrée (`perturbed`) à l'aide de l'optimiseur Adam pour maximiser la similarité cosinus (`cos_sim`) avec l'encodage cible (`target_emb`). Un terme de régularisation (`l2_loss`) limite l'amplitude de la perturbation, et les valeurs de pixels sont contraintes pour rester dans des plages valides.

5.2 Génération des Encodages (`generate_embeddings.py`)

Le fichier `generate_embeddings.py` génère les encodages faciaux pour des individus connus en utilisant un modèle pré-entraîné InceptionResnetV1. L'extrait principal concerne la génération et la sauvegarde des encodages.

```
for name in names:
    img = Image.open(f"data/{name}.jpg")
    tensor = transform(img).unsqueeze(0)
    with torch.no_grad():
        embedding = model(tensor)
    torch.save(embedding, f"embeddings/{name}.pt")
```

Ce code charge une image pour chaque personne, applique des transformations (redimensionnement et normalisation), la passe dans le modèle InceptionResnetV1 pour générer un encodage, puis sauvegarde cet encodage pour une utilisation ultérieure en reconnaissance faciale.

5.3 Reconnaissance Faciale (`face_recognition.py`)

Le fichier `face_recognition.py` implémente une reconnaissance faciale en temps réel à l'aide d'une webcam, avec une option pour exécuter une attaque Reface. L'extrait clé concerne la reconnaissance faciale et le déclenchement de l'attaque dans la boucle principale.

```
face_tensor = preprocess_face(face)
with torch.no_grad():
    emb = model(face_tensor)
name, score = recognize(emb, known_embeddings)
if key == ord('c'):
    print("Lancement attaque Reface...")
    target_emb = known_embeddings['Hisham']  # cible de l'attaque
    perturbed = reface_attack(model, face_tensor, target_emb,
       num_steps=100, lr=0.05)
    with torch.no_grad():
        emb_adv = model(perturbed)
    name_adv, score_adv = recognize(emb_adv, known_embeddings)
```

Cet extrait traite un visage détecté, génère son encodage et identifie la personne en le comparant aux encodages connus. Si l'utilisateur appuie sur la touche

'c', cela déclenche l'attaque Reface pour manipuler le visage afin qu'il corresponde à une identité cible (Hisham), puis réévalue l'identité de l'image perturbée.

5.4 Détecteur d'Attaque (`adversarial_detector.py`)

Le fichier `adversarial_detector.py` détecte les perturbations adversariales en comparant les images originales et perturbées. La fonction entière est concise et essentielle.

```
def detect_adversarial(original_tensor, perturbed_tensor, threshold
   =0.1):
    perturb = perturbed_tensor - original_tensor
    l2_norm = torch.norm(perturb.reshape(perturb.size(0), -1), p=2).
       item()
    if l2_norm > threshold:
        return True
    else:
        return False
```

Cette fonction calcule la norme L2 de la perturbation entre les images originale et perturbée. Si cette norme dépasse un seuil (0.1), elle signale l'image comme étant potentiellement issue d'une attaque adversariale.

6 Conclusion

Ce projet a permis de concevoir un système de reconnaissance faciale robuste, capable non seulement d'identifier des visages en temps réel grâce à l'apprentissage profond, mais aussi de simuler et de détecter des attaques adversariales sophistiquées à l'aide de la méthode Reface. L'intégration d'éléments tels que MTCNN, InceptionResnetV1 et des réseaux de transformation adversariale a permis de démontrer la faisabilité de perturbations visuelles imperceptibles mais efficaces, remettant en question la fiabilité des systèmes de reconnaissance faciale.

L'utilisation d'un entraînement par ensemble pour l'attaque adversariale a également mis en évidence la transférabilité des perturbations à différents modèles, soulignant ainsi la nécessité d'une meilleure défense. Le détecteur basé sur la norme L2 constitue une première barrière contre de telles attaques, mais il ouvre également la voie à des méthodes plus avancées de détection.

7 References

- Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and Harnessing Adversarial Examples.
- Sandler, M., et al. (2018). FaceNet: A Unified Embedding for Face Recognition and Clustering.
- PyTorch and FaceNet-PyTorch documentation.