
Projet de Fin d'Etudes : ExaFace

Licence Sciences et Techniques Génie logiciel

Sujet : Application Web pour la Vérification d'Identité par Reconnaissance Faciale dans le Contexte des Examens

Réalisé par :

- ZEHNOUNE RABII
- ZENNOU CHARAF EDDINE

Soutenu devant le Jury :

Aaaaa

Bbbbb

ccccc

DEDICACE



Une pensée forte,

À nos familles et à nos parents, qui ont été les piliers de notre parcours et nous ont soutenus inconditionnellement. Leur croyance en nous et leur encouragement constant nous ont permis de surmonter les obstacles et de devenir les meilleures versions de nous-mêmes.

À tous ceux qui ont contribué, de près ou de loin, à notre succès. Que ce soit par des conseils précieux, des encouragements chaleureux ou des opportunités offertes, vous avez joué un rôle essentiel dans notre développement personnel et professionnel.

À nos professeurs, nos amis et toutes les personnes qui croisent notre chemin au quotidien, vous nous enseignez la valeur de la persévérance et de l'effort constant. Votre présence nous inspire et nous motive à atteindre de nouveaux sommets.

C'est avec une profonde gratitude, un amour sincère et une admiration sans limites que nous vous dédions ces mots. Vous êtes les témoins de notre succès, et nous vous exprimons notre plus profonde reconnaissance et notre plus haute estime !



REMERCIEMENTS

Tout d'abord, nous exprimons notre profonde gratitude à Allah, le Tout-Puissant, le Très Miséricordieux, qui nous a accordé une bonne santé et nous a permis de mener à bien cette tâche difficile dans les meilleures conditions.

Nous souhaitons remercier nos parents, nos frères et nos sœurs qui ont toujours été à nos côtés, nous soutenant dans nos choix et nous apportant leur soutien lors des moments les plus difficiles.

Nos sincères remerciements vont également à notre professeur encadrant, le **Pr Fouad Yaakoubi**, qui a consacré de nombreux efforts pour nous accompagner tout au long de la réalisation de notre projet. Ses précieux conseils, et son soutien infaillible nous ont permis de surmonter les obstacles et de progresser.

Nous tenons à exprimer notre reconnaissance envers tout le personnel administratif et le corps enseignant de l'Université Moulay Ismaël, en particulier de la Faculté des Sciences et Techniques d'Errachidia, qui se battent chaque jour pour transmettre leur savoir à de jeunes étudiants, indépendamment de leurs origines ou de leurs antécédents.

Nous n'oublions pas de remercier les éminents membres du jury : **AAAAAA** et **BBBBBB** qui ont accepté de consacrer leur temps et leurs compétences pour évaluer notre travail.

Enfin, nous adressons nos remerciements les plus chaleureux à tous ceux qui ont honoré notre projet de leur présence. Votre soutien et votre présence nous touchent profondément et nous inspirent à continuer de nous dépasser.

Table des matières

DEDICACE	2
REMERCIEMENTS	3
Liste des figures	5
Chapitre 1 : Présentation du Projet	7
1. Introduction	7
1.1. Description du projet	7
1.2. Problématique et Solution	7
Chapitre 2 : Théorie du Deep Learning et Fondamentaux de Django	9
2.1. Théorie du Deep Learning	9
2.1.1. Introduction aux CNN (Réseaux de Neurones Convolutifs)	10
2.1.2. Introduction aux Réseaux Siamois	14
2.2. Fondamentaux de Django	25
2.2.1. Introduction à Django	25
2.2.2. Avantages de Django	25
2.2.3. Composants Principaux de Django	25
Chapitre 3 : Analyse et conception	27
1. Introduction	Error! Bookmark not defined.
2. Rôle des Diagrammes dans l'Analyse et la Conception :	27
2.1. Diagramme de cas d'utilisation :	27
2.2. Diagramme de Séquence :	29
2.3. Diagramme de Classe :	34
3. Choix de Méthodologie :	36
4. Spécification fonctionnelle :	37
5. Conception du code :	37
Chapitre 4 : Réalisation du projet	39
1. Introduction	39
2. Langages et Outils de Développement	39
1.1. Langages et Scripts :	39
1.2. Frameworks et Bibliothèques	40
1.3. Outils de développement :	41
3. CODE ET DÉVELOPPEMENT	41
4. Interfaces Utilisateurs	56
Bibliographie	60

Liste des figures

Figure 1 : Architecture de CNN pour la Reconnaissance d'Images.....	12
Figure 2 : Réseau Siamois à 2 Couches Cachées pour Classification Binaire	15
Figure 3 : embedding structure	15
Figure 4: Architecture Convolutionnelle pour la Vérification	16
Figure 5 : Diagramme de cas d'utilisation.....	27
Figure 6 : Diagramme de Séquence- Authentification	29
Figure 7 : Diagramme de Séquence- Vérification de l'Identité de l'Étudiant.....	30
Figure 8 : Diagramme de Sequence-le model de verification	31
Figure 9 : Diagramme de Séquence- Importer un Répertoire d'Étudiants	33
Figure 10 : Diagramme de classe	35
Figure 11 : interface pour la page de login	56
Figure 12 : interface pour la sélection de la classe et du semestre.....	57
Figure 13 : Interface pour la Reconnaissance Faciale	57
Figure 14 : Interface pour la Page de Résultat.....	58
Figure 15 : interface pour Importer un Répertoire d'Images	58

Introduction Générale

La technologie de la reconnaissance faciale a connu une expansion significative ces dernières années, offrant des solutions innovantes dans divers domaines, notamment la sécurité, la surveillance et l'identification. L'une des applications les plus prometteuses de cette technologie est son utilisation dans le domaine de l'éducation pour vérifier l'identité des étudiants pendant les examens. Cette application offre une solution efficace pour prévenir la fraude académique, tout en assurant l'intégrité des examens.

Le présent projet vise à concevoir et développer une plateforme web innovante appelée "ExaFace" qui intègre la technologie de reconnaissance faciale dans le processus d'administration des examens. Cette plateforme permettra de vérifier de manière fiable l'identité des étudiants en temps réel en utilisant des algorithmes de reconnaissance faciale avancés. L'objectif principal est de garantir que les étudiants passent leurs examens de manière équitable et sécurisée, tout en fournissant aux établissements éducatifs un outil précieux pour renforcer la confiance dans l'intégrité de leurs processus d'examen.

Ce rapport détaille les différentes étapes du développement de l'application "ExaFace", en commençant par une présentation de l'idée du projet, suivie d'une analyse approfondie des problématiques rencontrées et des solutions envisagées. Le cahier des charges de l'application est également défini pour guider le processus de développement. Enfin, une explication détaillée de l'architecture et des fonctionnalités de l'application est fournie, mettant en évidence son potentiel et ses avantages dans le domaine de l'éducation.

Chapitre 1 : Présentation du Projet

1. Introduction

1.1. Description du projet

Ce projet se situe dans le cadre de l'éducation et de l'évaluation des étudiants. L'identification précise des individus y joue un rôle crucial pour garantir l'équité et l'intégrité des examens.

L'objectif est de mettre en place un système utilisant des techniques de reconnaissance faciale. Ce système doit automatiser le processus d'identification des étudiants.

Les visages des étudiants seront vérifiés par rapport à une base de données préalablement enregistrée. Ainsi, nous pourrions assurer une identification rapide et fiable, minimisant les risques de fraude et améliorant l'efficacité des procédures d'examen.

1.2. Problématique et Solution

La principale problématique abordée par ce projet est la nécessité de garantir l'identité des étudiants lors des examens, en particulier dans les environnements où la fraude et la tricherie sont préoccupantes. La solution proposée consiste à utiliser des algorithmes de reconnaissance faciale pour authentifier les étudiants en temps réel, en comparant leurs visages capturés lors de l'examen avec des visages enregistrés dans une base de données

• CAHIER DES CHARGES

- ✓ **Introduction :** Le présent cahier des charges détaille les spécifications et les exigences fonctionnelles de l'application "ExaFace", une plateforme web de reconnaissance faciale conçue pour l'administration sécurisée des examens dans les établissements éducatifs. L'objectif principal de cette application est de vérifier de manière fiable l'identité des étudiants pendant les examens, en
- ✓ permettant aux surveillants d'utiliser la caméra de leur téléphone pour capturer et vérifier les visages des étudiants.

Fonctionnalités Principales :L'application "ExaFace" devra inclure les fonctionnalités suivantes :

- ✓ Authentification des Surveillants : Les Surveillants auront des comptes sécurisés avec des identifiants uniques pour accéder à l'interface d'administration de l'application.
L'authentification se fera à l'aide de noms d'utilisateur et de mots de passe sécurisés.
- ✓ Reconnaissance Faciale Automatisée : Les images des visages des étudiants seront capturées à partir des caméras des téléphones des surveillants.
L'application utilisera des algorithmes avancés de reconnaissance faciale pour identifier les étudiants en comparant les visages capturés aux images enregistrées dans la base de données de l'application pour vérification.

- **EXIGENCES TECHNIQUES**

- ✓ Plateforme Web : L'application sera une plateforme web accessible via un navigateur internet standard.
Elle sera développée en utilisant le framework Django en Python pour le backend et HTML/CSS/JavaScript pour le frontend.
- ✓ Stockage des Données : Les données des utilisateurs, y compris les images faciales et les informations d'identification, seront stockées de manière sécurisée dans une base de données
- ✓ Bibliothèques de Deep Learning : Pour la partie d'apprentissage profond, nous utiliserons la bibliothèque TensorFlow en Python. TensorFlow est une plateforme open-source de bout en bout pour l'apprentissage automatique développée par Google. Elle offre des outils flexibles pour la création et l'entraînement de modèles d'apprentissage profond, y compris les réseaux de neurones convolutionnels (CNN) utilisés pour la reconnaissance faciale.
- ✓ Bibliothèques de Traitement d'Images : Nous utiliserons également des bibliothèques telles que OpenCV (Open Source Computer Vision Library) pour le traitement d'images. OpenCV est une bibliothèque open-source qui offre des outils et des algorithmes pour le traitement d'images en temps réel, y compris la détection de visage et la manipulation d'images.

Chapitre 2 : Théorie du Deep Learning et Fondamentaux de Django

2.1. Théorie du Deep Learning

Introduction au Deep Learning : Le Deep Learning est une sous-branche de l'intelligence artificielle (IA) qui s'inspire du fonctionnement du cerveau humain pour résoudre des problèmes complexes en utilisant des réseaux de neurones artificiels. Contrairement aux méthodes traditionnelles d'apprentissage automatique, le Deep Learning permet aux machines d'apprendre des représentations hiérarchiques de données en utilisant plusieurs couches de traitement pour capturer des motifs abstraits.[1]

Utilisation de réseaux de neurones artificiels : Les réseaux de neurones artificiels, qui sont la pierre angulaire du Deep Learning, sont des modèles informatiques composés de plusieurs couches de neurones interconnectés. Chaque neurone est une unité de calcul qui reçoit des entrées, applique une transformation mathématique et produit une sortie qui est transmise à la couche suivante. Les réseaux de neurones profonds, qui ont plusieurs couches cachées, sont capables d'apprendre des représentations de données complexes et non linéaires.

Révolution dans divers domaines : Le Deep Learning a révolutionné de nombreux domaines de l'informatique et de la science des données, notamment :

- **Vision par ordinateur :** En utilisant des réseaux de neurones convolutionnels (CNN), le Deep Learning a permis des avancées significatives dans la reconnaissance d'objets, la détection de visages, la segmentation d'images et d'autres tâches liées à la vision par ordinateur.
- **Reconnaissance vocale :** Les réseaux de neurones récurrents et les réseaux de neurones convolutifs sont utilisés pour convertir la parole humaine en texte écrit, permettant ainsi le développement de systèmes de reconnaissance vocale précis.

- **Biologie et médecine : Le Deep Learning est utilisé pour l'analyse d'imagerie** médicale, la découverte de médicaments, la prédiction du risque de maladies et d'autres applications liées à la santé.

2.1.1. Introduction aux CNN (Réseaux de Neurones Convolutifs)

Les réseaux de neurones convolutifs (Convolutional Neural Networks - CNN) représentent une classe spécifique de réseaux de neurones artificiels, particulièrement adaptée à l'analyse et au traitement d'images. Depuis leur introduction, les CNNs ont révolutionné le domaine de la vision par ordinateur, surpassant de manière significative les méthodes traditionnelles dans diverses tâches telles que la classification d'images, la détection d'objets, la segmentation d'images et la reconnaissance de visages. (1)

CONTEXTE HISTORIQUE : Les concepts fondamentaux des CNNs remontent aux années 1980 avec les travaux de Kunihiko Fukushima, qui a développé le "neocognitron". Cependant, c'est Yann LeCun et ses collègues qui ont véritablement popularisé les CNNs dans les années 1990 avec leur application réussie à la reconnaissance de caractères manuscrits, notamment pour le tri du courrier postal. Le modèle LeNet-5 de LeCun est devenu une référence dans le domaine et a jeté les bases des architectures modernes de CNN.

PROBLEMES ET APPLICATIONS

Les réseaux de neurones convolutifs (CNN) sont particulièrement efficaces pour les tâches de reconnaissance d'images et de classification grâce à leur capacité à capturer les caractéristiques spatiales et hiérarchiques des données d'image. Voici quelques exemples d'applications où les CNN se sont avérés extrêmement utiles :

- **Classification d'Images :** Cette tâche consiste à attribuer une étiquette à une image entière, représentant ainsi l'identification de l'objet principal dans l'image. Par exemple, un modèle CNN peut être entraîné pour reconnaître différentes races de chiens ou pour différencier entre divers types de fleurs.
- **Détection d'Objets :** Contrairement à la classification d'images, la détection d'objets vise à localiser plusieurs objets dans une seule image et à identifier leurs positions exactes. Les CNNs sont utilisés pour dessiner des cadres autour des objets d'intérêt et les classer simultanément. Par exemple, dans une image de rue, un modèle peut détecter et identifier des voitures, des piétons et des panneaux de signalisation.
- **Segmentation Sémantique :** Cette application consiste à attribuer une étiquette à chaque pixel d'une image, de manière à délimiter les différents objets ou régions de l'image. Cela permet une compréhension plus fine des éléments présents dans l'image. Par exemple, dans une image de paysage, chaque pixel peut être classé comme appartenant à une catégorie telle que ciel, arbre, route ou bâtiment.

- **Reconnaissance Faciale** : Les CNNs sont largement utilisés pour identifier ou vérifier l'identité d'une personne à partir de son visage. Ce processus inclut la détection des visages dans une image, l'extraction des caractéristiques uniques et la comparaison avec une base de données de visages connus. Les applications courantes incluent le déverrouillage des smartphones, la surveillance de sécurité et le marquage automatique des photos sur les réseaux sociaux.
- **Analyse Biomédicale** : Les CNNs jouent un rôle crucial dans l'interprétation des images médicales pour le diagnostic de maladies. Par exemple, ils peuvent être utilisés pour identifier les tumeurs dans les scans IRM, détecter les anomalies dans les radiographies pulmonaires ou segmenter les structures anatomiques dans les images échographiques. Ces modèles assistent les médecins en offrant des diagnostics plus rapides et parfois plus précis.

FONCTIONNEMENT DES CNNs : Les CNNs diffèrent des réseaux de neurones traditionnels principalement par leur architecture spécialisée, qui inclut des couches de convolution, de pooling et de normalisation, permettant une extraction efficace des caractéristiques à partir des données d'image.

- **Couche Convolutionnelle** : Elle applique des filtres de convolution pour extraire des caractéristiques locales de l'image, telles que les bords, les textures et les motifs. Chaque filtre "convolue" l'image d'entrée pour produire une carte de caractéristiques (feature map).
- **Couche de Pooling** : Elle réduit la dimensionnalité des cartes de caractéristiques en regroupant les valeurs, ce qui permet de diminuer la complexité computationnelle tout en conservant les informations essentielles. Le max pooling et l'average pooling sont les techniques les plus courantes.
- **Couche de Normalisation** : Elle stabilise et accélère l'apprentissage en normalisant les activations, souvent à l'aide de la batch normalization.
- **Couche Entièrement Connectée** : Elle est similaire à celle des réseaux de neurones traditionnels et sert à combiner les caractéristiques extraites pour produire la sortie finale du réseau, telle qu'une probabilité pour chaque classe dans une tâche de classification.

AVANTAGES DES CNNs : Les CNNs présentent plusieurs avantages par rapport aux méthodes traditionnelles et aux autres architectures de réseaux de neurones :

- **Invariance aux translations** : Grâce aux opérations de convolution et de pooling, les CNNs peuvent détecter les mêmes caractéristiques dans différentes parties de l'image, rendant le modèle plus robuste aux variations de position.
- **Réduction des paramètres** : En partageant les poids à travers les filtres, les CNNs nécessitent moins de paramètres, ce qui réduit le risque de surapprentissage et diminue les besoins en mémoire.
- **Extraction hiérarchique des caractéristiques** : Les couches successives des CNNs permettent de capturer des caractéristiques de plus en plus complexes, allant des bords et des textures aux objets entiers.

ARCHITECTURE D'UN CNN : Un CNN typique est composé de plusieurs couches, chacune ayant un rôle spécifique :

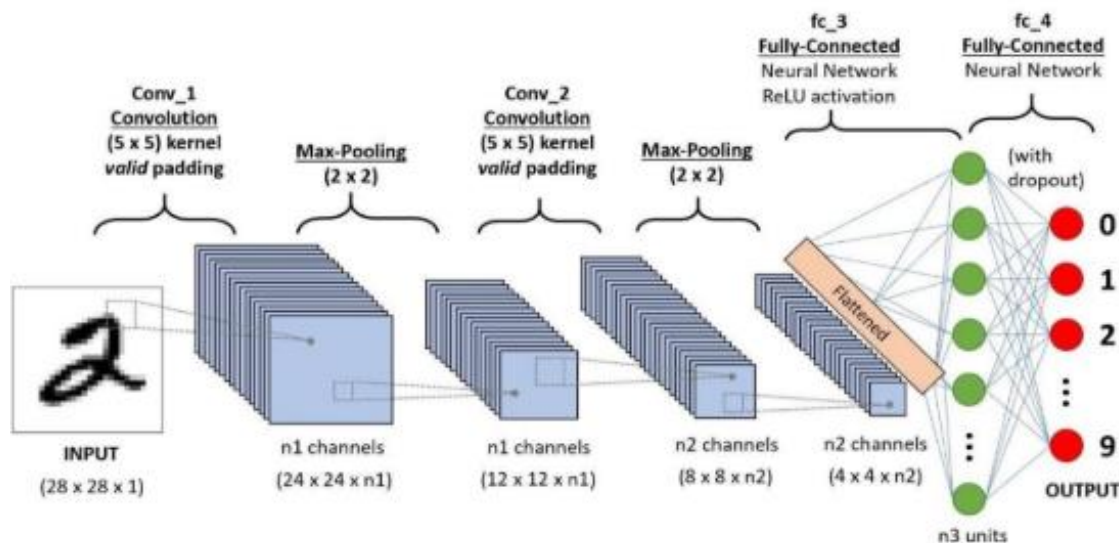


Figure 1 : Architecture de CNN pour la Reconnaissance d'Images

- **Couche Convolutionnelle** : Cette couche applique une opération de convolution à l'image d'entrée en utilisant plusieurs filtres (ou noyaux) pour extraire des caractéristiques locales. Mathématiquement, la convolution est définie comme suit :

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n)$$

- I est l'image d'entrée,
 - K est le filtre,
 - (i, j) sont les coordonnées du pixel dans l'image de sortie.
 - **Filtres (Kernels)** : Chaque filtre glisse sur l'image d'entrée et effectue un produit scalaire entre les éléments du filtre et les éléments de l'image. Ce processus est appelé convolution.
 - **Feature Maps** : Le résultat de chaque convolution est une carte de caractéristiques (feature map), qui détecte des caractéristiques spécifiques telles que les bords, les textures, et les motifs.
 - **Stride et Padding** : Le stride détermine le pas de la convolution, et le padding ajoute des bordures à l'image d'entrée pour contrôler la taille des feature maps.
- **Couche de Pooling** : La couche de pooling réduit la dimensionnalité de l'image tout en conservant les caractéristiques importantes. La méthode la plus courante est le max pooling. Mathématiquement, pour un pooling 2x2, cela se traduit par :

$$P(i, j) = \max\{I(2i, 2j), I(2i, 2j + 1), I(2i + 1, 2j), I(2i + 1, 2j + 1)\}$$

- où P est la sortie du pooling.
- **Max Pooling** : Sélectionne le maximum d'une région définie (par exemple, 2x2) de la feature map, réduisant ainsi la dimensionnalité tout en conservant les informations importantes.
- **Average Pooling** : Calcule la moyenne des valeurs dans une région définie de la feature map.

- **Couche de Normalisation** : La normalisation est utilisée pour stabiliser et accélérer l'apprentissage en ajustant et en mettant à l'échelle les activations. Une méthode courante est la Batch Normalization, qui normalise les activations sur un mini-batch :

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

où :

- $\hat{x}^{(k)}$ est l'activation de la k-ième couche,
- μ_B et σ_B sont la moyenne et l'écart-type calculés sur le mini-batch,
- ϵ est une petite constante pour éviter la division par zéro.

- **Couche Entièrement Connectée** : Les fonctions d'activation introduisent des non-linéarités dans le réseau, ce qui permet de modéliser des relations complexes. Les plus courantes sont :

- ReLU (Rectified Linear Unit) :

$$f(x) = \max(0, x)$$

- Sigmoid :

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Tanh :

$$f(x) = \tanh(x)$$

PROPAGATION ET APPRENTISSAGE :

- **Propagation Avant (Forward Propagation)** : Pendant la propagation avant, les données d'entrée passent à travers les couches du réseau, où chaque couche applique ses transformations pour produire une sortie. Pour une couche

convolutionnelle, la sortie est obtenue en appliquant la convolution suivie d'une fonction d'activation.

- **Fonction de Perte (Loss Function) :** La fonction de perte mesure l'erreur entre la sortie prédite par le réseau et la vérité terrain (ground truth). Une fonction de perte couramment utilisée pour la classification est l'entropie croisée (cross-entropy) :

$$L = - \sum_{i=1}^N y_i \log(\hat{y}_i)$$

où :

- y_i est la vraie étiquette,
 - \hat{y}_i est la probabilité prédite pour la classe i .
- **Rétropropagation (Backpropagation) :** La rétropropagation est utilisée pour mettre à jour les poids du réseau en fonction de l'erreur calculée par la fonction de perte. Les gradients de la perte par rapport aux poids sont calculés en utilisant la règle de la chaîne (chain rule). L'algorithme de descente de gradient (gradient descent) ou ses variantes (comme Adam) sont ensuite utilisés pour ajuster les poids.

Les CNNs sont puissants pour l'analyse d'images en raison de leur capacité à capturer des caractéristiques locales et à les combiner hiérarchiquement. La combinaison de couches convolutionnelles, de pooling, de normalisation et entièrement connectées, avec des fonctions d'activation non linéaires et des techniques d'apprentissage efficaces, permet aux CNNs de traiter et de comprendre des images de manière très efficace. [2]

2.1.2. Introduction aux Réseaux Siamois

Les réseaux siamois sont une architecture de réseaux de neurones particulièrement adaptée pour des tâches de comparaison et de vérification, telles que la reconnaissance faciale, la vérification de signature, et d'autres applications où la comparaison directe entre paires de données est cruciale. Ce type de réseau est spécialement conçu pour déterminer si deux entrées (par exemple, des images) sont similaires ou dissemblables. Les réseaux siamois convertissent les images en vecteurs de caractéristiques appelés embeddings. Ces embeddings capturent les caractéristiques essentielles des images et permettent une comparaison efficace des données en utilisant diverses métriques de distance.[6]

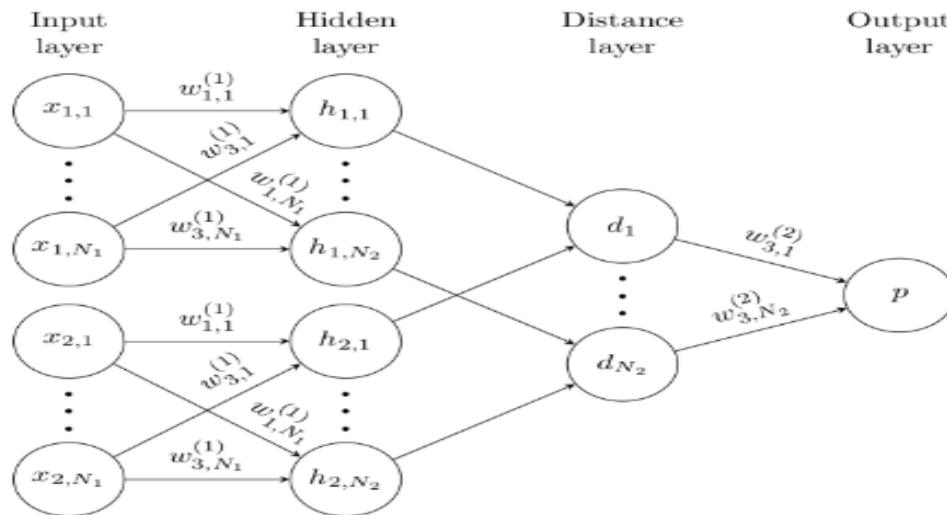


Figure 2 :Réseau Siamese à 2 Couches Cachées pour Classification Binaire

DEFINITION DE L'EMBEDDING :

Embedding est le processus de transformation d'objets discrets en vecteurs de nombres réels dans un espace vectoriel de dimension fixe. Ce processus permet de capturer les relations et les similarités entre ces objets dans un format mathématiquement manipulable par des algorithmes d'apprentissage automatique.



Figure 3 : embedding structure

ARCHITECTURE DU RESEAU SIAMOIS :

L'architecture des réseaux siamois se divise en deux parties principales :

- **La Partie d'Extraction des Embeddings** : Cette partie consiste en deux sous-réseaux identiques et parallèles, chacun prenant une entrée distincte et produisant un vecteur de caractéristiques (embedding) de dimension réduite.

Les sous-réseaux partagent les mêmes poids et les mêmes paramètres, garantissant que les deux entrées sont traitées de manière identique. Ce processus d'extraction permet de convertir chaque entrée en une représentation vectorielle riche qui capture les caractéristiques essentielles de l'image.

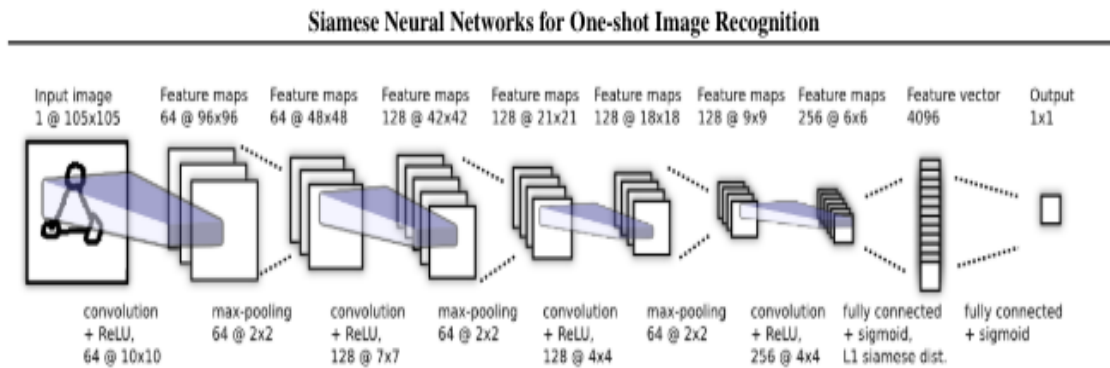


Figure 4: Architecture Convolutionnelle pour la Vérification

- Entrée (Image) :
 - Taille : 1×105×105
- Première Couche Convolutionnelle :
 - Convolution avec 64 filtres de taille 10×10
 - Fonction d'activation : ReLU
 - Sortie : 64 cartes de caractéristiques de taille 96×96
- Première Couche de Pooling (Max-Pooling :
 - Pooling de taille 2×2
 - Sortie : 64 cartes de caractéristiques de taille 48×48
- Deuxième Couche Convolutionnelle :
 - Convolution avec 128 filtres de taille 7×7
 - Fonction d'activation : ReLU
 - Sortie : 128 cartes de caractéristiques de taille 42×42
- Deuxième Couche de Pooling (Max-Pooling) :
 - Pooling de taille 2×2
 - Sortie : 128 cartes de caractéristiques de taille 21×21

- Troisième Couche Convolutionnelle :
 - Convolution avec 128 filtres de taille 4×4
 - Fonction d'activation : ReLU
 - Sortie : 128 cartes de caractéristiques de taille 18×18
- Troisième Couche de Pooling (Max-Pooling) :
 - Pooling de taille 2×2
 - Sortie : 128 cartes de caractéristiques de taille 9×9
- Quatrième Couche Convolutionnelle :
 - Convolution avec 256 filtres de taille 4×4
 - Fonction d'activation : ReLU
 - Sortie : 256 cartes de caractéristiques de taille 6×6
- Couche Entièrement Connectée :
 - Taille : 4096
 - Fonction d'activation : Sigmoid ou ReLU

Le résultat final de cette partie est un vecteur d'embedding de taille 4096 pour chaque entrée.

- **La Partie de Comparaison des Embeddings :** Une fois les embeddings générés pour les deux entrées, cette partie du réseau utilise une métrique de distance pour comparer les vecteurs d'embedding et déterminer leur similarité ou dissimilarité. Des distances telles que la distance euclidienne, la distance de Manhattan ou la similarité cosinus peuvent être utilisées. La comparaison est ensuite optimisée à l'aide d'une fonction de perte appropriée, comme la contrastive loss, pour apprendre à discriminer efficacement entre les paires similaires et dissemblables.

- Calcul de la Distance entre les Embeddings : Embeddings Produits :
 - Embedding de la première image : E_1^i
 - Embedding de la deuxième image : E_2^i
- Calcul de la Distance : Plusieurs métriques de distance peuvent être utilisées pour comparer les embeddings :
 - Distance Euclidienne (L2 Norm) :

$$D_{\text{euclidienne}} = \sqrt{\sum_{i=1}^n (E_1^i - E_2^i)^2}$$

- *Distance de Manhattan (L1 Norm) :*

$$D_{\text{manhattan}} = \sum_{i=1}^n |E_1^i - E_2^i|$$

- *Distance Cosine :*

$$D_{\text{cosine}} = 1 - \frac{E_1 \cdot E_2}{\|E_1\| \|E_2\|}$$

- **Fonction de Perte et Décision :** Pour entraîner le réseau siamois, une fonction de perte adaptée est utilisée pour optimiser les distances entre les embeddings :

- **Fonction de Perte(Contrastive Loss) :**

$$\mathcal{L}(x_1^{(i)}, x_2^{(i)}) = y(x_1^{(i)}, x_2^{(i)}) \log p(x_1^{(i)}, x_2^{(i)}) + (1 - y(x_1^{(i)}, x_2^{(i)})) \log(1 - p(x_1^{(i)}, x_2^{(i)})) + \lambda w^T$$

- $y(x_1^{(i)}, x_2^{(i)}) \log p(x_1^{(i)}, x_2^{(i)})$: Ce terme pénalise les cas où la paire appartient à la même classe mais la probabilité prédite est faible.
- $(1 - y(x_1^{(i)}, x_2^{(i)})) \log(1 - p(x_1^{(i)}, x_2^{(i)}))$: Ce terme pénalise les cas où la paire n'appartient pas à la même classe mais la probabilité prédite est élevée.
- λw^T : Ce terme de régularisation aide à prévenir le surapprentissage en pénalisant les poids de grande amplitude.

La contrastive loss est utilisée pour minimiser la distance entre les embeddings de paires similaires et maximiser la distance entre les embeddings de paires dissemblables

ENTRAINEMENT D'UN RESEAU SIAMOIS :

L'entraînement d'un réseau siamois implique plusieurs étapes clés, de la préparation des données à la mise à jour des poids du modèle. Voici un aperçu détaillé de chaque étape :

- **Preparation des donnees :** AVANT DE COMMENCER L'ENTRAINEMENT, IL EST CRUCIAL DE PREPARER LES DONNEES DE MANIERE APPROPRIEE.

a. Chargement et Prétraitement des Données :

- **Charger les images :** Charger les images à partir de la base de données.
- **Normalisation :** Normaliser les images pour que les pixels aient des valeurs comprises entre 0 et 1.
- **Augmentation des données :** Appliquer des transformations (rotation, mise à l'échelle, recadrage) pour augmenter la variété des données.

b. Création des Paires de Données :

- **Paires similaires :** Sélectionner des paires d'images appartenant à la même classe.
- **Paires dissemblables :** Sélectionner des paires d'images appartenant à des classes différentes.
- **Étiquetage :** Assigner une étiquette 111 aux paires similaires et 000 aux paires dissemblables.

- **Initialisation du Modèle :** Initialiser les paramètres du réseau siamois.

a. Initialisation des Poids

- **Poids des couches convolutionnelles :** Initialiser les poids des filtres de convolution (par exemple, avec une distribution gaussienne).
- **Poids des couches entièrement connectées :** Initialiser les poids des couches denses.
- **Forward Pass :** Pour chaque paire d'images dans le batch, effectuer une passe en avant à travers le réseau.

a. Calcul des Embeddings

- **Propagation avant :** Passer chaque image d'une paire à travers les sous-réseaux pour obtenir les embeddings.
- **Extraction des caractéristiques :** Obtenir les vecteurs d'embedding des deux images.

b. Calcul de la Distance

- **Distance entre embeddings :** Calculer la distance entre les deux embeddings (par exemple, distance euclidienne).
- **Calcul de la Perte :** Calculer la fonction de perte pour chaque paire dans le batch.

a. Perte de Cross-Entropy

- **Étiquettes :** Utiliser les étiquettes (0 ou 1) pour calculer la cross-entropy entre la prédiction et la vérité terrain.
- **Régularisation :** Ajouter le terme de régularisation pour éviter le surapprentissage.

- **Backward Pass :** Effectuer une passe arrière pour calculer les gradients.

a. Calcul des Gradients

- **Propagation arrière** : Calculer les gradients de la perte par rapport aux poids du modèle en utilisant la rétropropagation.
- **Gradients des couches convolutionnelles et denses** : Calculer les gradients pour chaque couche.
- **Mise à Jour des Poids** : Mettre à jour les poids du modèle en utilisant une méthode d'optimisation (par exemple, descente de gradient stochastique).

a. Optimisation

- **Taux d'apprentissage** : Multiplier les gradients par le taux d'apprentissage.
- **Mise à jour des poids** : Soustraire les gradients multipliés du taux d'apprentissage aux poids actuels.
- **Boucle d'Entraînement** : Répéter les étapes de forward pass, calcul de la perte, backward pass et mise à jour des poids pour plusieurs itérations.

a. Epochs et Minibatches

- **Minibatch** : Diviser les données d'entraînement en petits lots (minibatches).
- **Epochs** : Répéter l'entraînement sur l'ensemble des données pour un certain nombre d'epochs.
- **Validation et Ajustement** : Valider périodiquement le modèle sur un ensemble de données de validation pour éviter le surapprentissage et ajuster les hyperparamètres.

a. Validation Croisée

- **Évaluation périodique** : Évaluer la performance du modèle sur des données non vues pendant l'entraînement.
- **Ajustement des hyperparamètres** : Ajuster les paramètres tels que le taux d'apprentissage, la marge de la loss function, etc., en fonction des performances de validation.

L'entraînement d'un réseau siamois implique une série de passes avant et arrière à travers le modèle, la mise à jour des poids en fonction des gradients calculés, et une validation régulière pour assurer une bonne généralisation du modèle. En suivant ces étapes de manière itérative, le réseau siamois apprend à produire des embeddings qui capturent efficacement la similarité ou la dissimilarité entre les images.[6]

2.1.3. Algorithme de Réseau Siamois

- **Fonction Créer_Sous_Réseau**

plaintext

Copier le code

```
Fonction Créer_Sous_Réseau(input_shape)
  Entrée : input_shape - La forme des images en entrée (par exemple, (105, 105,
1) pour des images de taille 105x105 en niveaux de gris)
  Sortie : Un modèle de sous-réseau pour extraire les caractéristiques des images

  Début
    // Création d'un sous-réseau pour l'extraction de caractéristiques
    Initialiser input de taille input_shape
    Ajouter Convolution(64 filtres, taille 3x3, activation 'relu')
    Ajouter MaxPooling(taille 2x2)
    Ajouter Convolution(128 filtres, taille 3x3, activation 'relu')
    Ajouter MaxPooling(taille 2x2)
    Ajouter Flatten()
    Ajouter Dense(128 unités, activation 'relu')
    Retourner le modèle défini
  Fin
```

Explication : Cette fonction crée un modèle de sous-réseau convolutif pour extraire les caractéristiques des images en entrée. Elle utilise des couches de convolution pour détecter des motifs visuels et des couches de pooling pour réduire la dimensionnalité des caractéristiques extraites.

- **Fonction Distance_Euclidienne**

plaintext

Copier le code

```
Fonction Distance_Euclidienne(vecteurs x, y)
  Entrée : x, y - Les vecteurs de caractéristiques à comparer
  Sortie : La distance euclidienne entre les deux vecteurs

  Début
    // Calcul de la distance euclidienne entre deux vecteurs
    sum_square <- Somme(Carre(x - y))
    distance <- RacineCarrée(Max(sum_square, epsilon))
    Retourner distance
  Fin
```

Explication : Cette fonction calcule la distance euclidienne entre deux vecteurs de caractéristiques extraites des images. Elle est utilisée pour mesurer la similarité entre deux paires d'images basée sur les caractéristiques extraites par les sous-réseaux.

- **Fonction Perte_Contrastive**

plaintext

Copier le code

```
Fonction Perte_Contrastive(y_vrai, y_prédit)
  Entrée : y_vrai - Étiquette indiquant si les paires d'images sont similaires
  (1) ou non (0)
  y_prédit - La distance prédite entre les paires d'images par le modèle
  Sortie : La perte contrastive calculée pour entraîner le modèle

  Début
    // Définition de la fonction de perte contrastive pour l'entraînement du
réseau
    margin <- 1
    square_pred <- Carre(y_prédit)
```

```

margin_square <- Carre(Max(margin - y_prédit, 0))
perte <- Moyenne(y_vrai * square_pred + (1 - y_vrai) * margin_square)
Retourner perte
Fin

```

Explication : Cette fonction définit la fonction de perte contrastive, utilisée pendant l'entraînement du modèle siamois. Elle compare la distance prédite entre les paires d'images avec les étiquettes réelles (similaires ou non) pour ajuster les poids du modèle afin de minimiser cette perte.

- **Algorithme Réseau siamois**

```

plaintext
Copier le code
Algorithme Réseau_Siamois
  Entrée : X_train_a, X_train_b (paires d'images d'entraînement)
           Y_train (étiquettes des paires, 1 si similaire, 0 sinon)
  Sortie : Modèle entraîné

  Début
    // Initialisation des formes d'entrée et création des sous-réseaux
    Initialiser input_shape <- (105, 105, 1)
    Initialiser input_a <- Input(input_shape)
    Initialiser input_b <- Input(input_shape)

    base_network <- Créer_Sous_Réseau(input_shape)
    processed_a <- base_network(input_a)
    processed_b <- base_network(input_b)

    // Calcul de la distance entre les sorties des sous-réseaux
    distance <- Distance_Euclidienne(processed_a, processed_b)

    // Définition du modèle final et compilation
    modèle <- DéfinirModèle([input_a, input_b], distance)
    modèle <- Compiler(modèle, Perte_Contrastive, Optimiseur 'adam')

    // Entraînement du modèle avec les données d'entraînement
    modèle <- Entraîner(modèle, [X_train_a, X_train_b], Y_train, batch_size=32,
epochs=10)

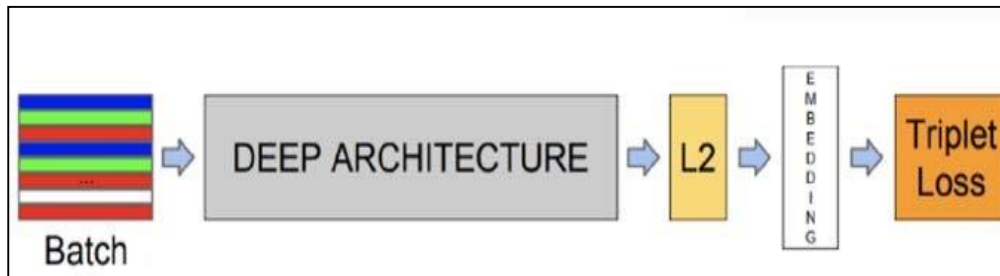
    Retourner modèle
  Fin

```

Explication : Cet algorithme global combine les fonctions précédentes pour créer, entraîner et utiliser un réseau siamois. Il utilise des paires d'images d'entraînement avec des étiquettes pour ajuster les poids du modèle de manière à minimiser la perte contrastive, facilitant ainsi la comparaison de similitudes entre paires d'images.

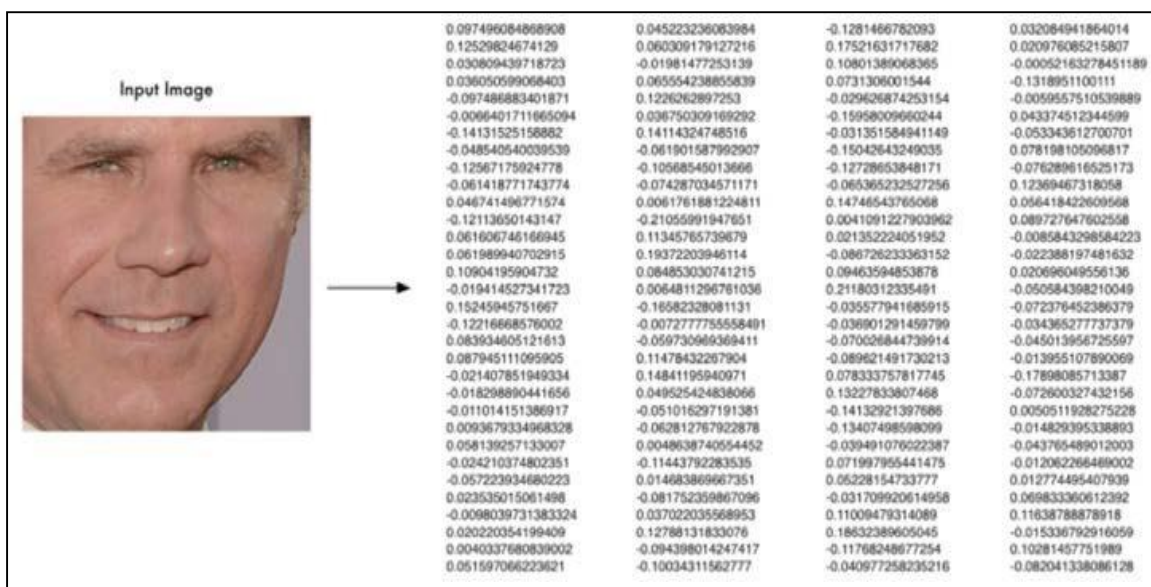
2.1.4. FaceNet :

faceNet est un réseau neuronal de pointe pour la reconnaissance, la vérification et le regroupement des visages. Il s'agit d'un réseau neuronal profond à 22 couches, publié en 2015 par les chercheurs de Google, Schroff et al



Le réseau profond illustré à la figure est issu de l'architecture GoogleNet (avec de nombreuses révisions). L'article de FaceNet ne traite pas beaucoup du fonctionnement interne de l'architecture GoogleNet, et considère le réseau neuronal profond comme une boîte noire.

FaceNet prend une image d'un visage en entrée et sort un vecteur de 128 nombres qui représentent les caractéristiques les plus importantes d'un visage. En apprentissage automatique, ce vecteur est appelé embeddings parce que toutes les informations importantes d'une image sont intégrées dans ce vecteur. Fondamentalement, FaceNet prend le visage d'une personne et le compresse dans un vecteur de 128 chiffres. Idéalement, les incorporations de visages similaires sont également similaires.

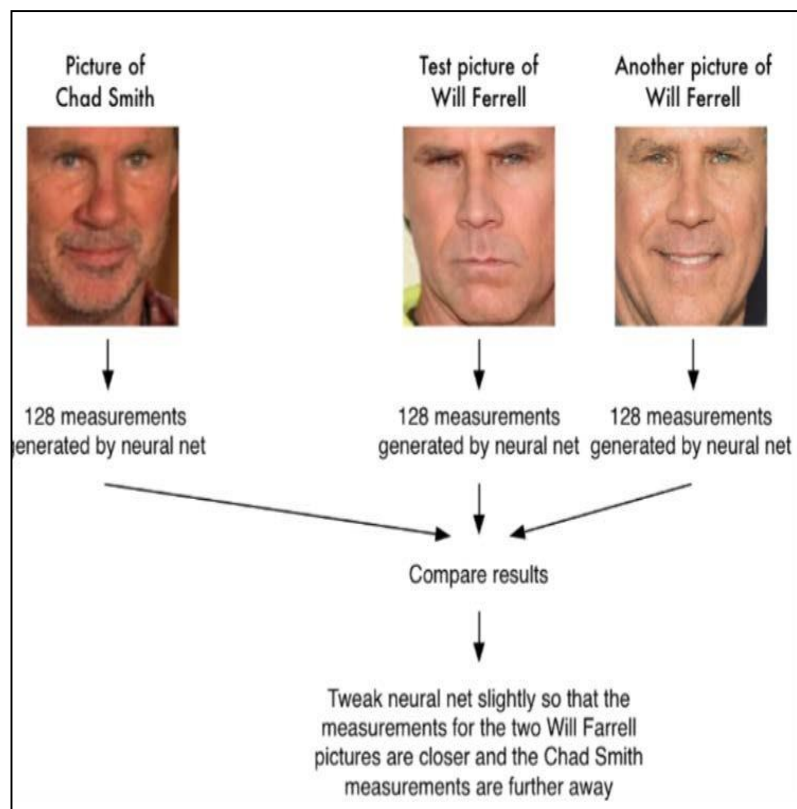


Les chercheurs ont découvert que l'approche la plus précise consiste à laisser l'ordinateur déterminer lui-même les mesures à collecter. L'apprentissage profond parvient mieux que les humains à déterminer les parties du visage qu'il est important de mesurer.

L'idée de FaceNet est donc former un réseau neuronal convolutif profond et l'entraîner à générer 128 mesures pour chaque visage. Le processus d'entraînement fonctionne en examinant 3 images de visage à la fois :

- Charger une image de visage d'une personne connue
- Chargez une autre image de la même personne connue
- Charger une image d'une personne totalement différente

L'algorithme examine ensuite les mesures qu'il génère pour chacune de ces trois images. Il modifie ensuite légèrement le réseau neuronal afin de s'assurer que les mesures qu'il génère pour les images 1 et 2 sont légèrement plus proches et que les mesures pour les images 2 et 3 sont légèrement plus éloignées : Cette méthode d'apprentissage est appelée perte triplet (triplet loss).



Après avoir répété cette étape des millions de fois pour des millions d'images de milliers de personnes différentes, le réseau neuronal apprend à générer de manière fiable 128 mesures pour chaque personne. Dix photos différentes d'une même personne devraient donner à peu près les mêmes mesures.

2.2. Fondamentaux de Django

2.2.1. Introduction à Django

Django est un framework de développement Web open-source, écrit en Python, qui encourage le développement rapide et propre d'applications Web complexes. Créé par des développeurs expérimentés chez Lawrence Journal-World en 2003, Django est conçu pour simplifier le processus de développement en fournissant une structure solide et des fonctionnalités prêtes à l'emploi. [5]

2.2.2. Avantages de Django

- ✓ Productivité accrue : Django propose une architecture MVC (Modèle-Vue-Contrôleur) qui facilite la séparation des préoccupations et la réutilisation du code. Avec ses nombreux composants intégrés et ses bibliothèques tierces, Django permet aux développeurs de se concentrer sur la logique métier plutôt que sur des tâches de routine.
- ✓ Sécurité : Django est livré avec des fonctionnalités de sécurité intégrées telles que la protection contre les attaques par injection SQL, la protection contre les attaques CSRF (Cross-Site Request Forgery) et l'authentification utilisateur robuste. De plus, Django encourage les meilleures pratiques en matière de sécurité, ce qui contribue à réduire les risques liés à la sécurité.
- ✓ Scalabilité : Django est conçu pour être évolutif, ce qui signifie qu'il peut facilement s'adapter à des charges de travail croissantes. Grâce à son architecture modulaire et à son support pour les bases de données relationnelles et non relationnelles, Django peut répondre aux besoins des applications de toutes tailles.
- ✓ Documentation exhaustive : Django offre une documentation complète et bien organisée, qui couvre tous les aspects du développement Web avec le framework. Cette documentation détaillée facilite l'apprentissage pour les nouveaux développeurs et constitue une référence précieuse pour les développeurs expérimentés. [5]

2.2.3. Composants Principaux de Django

- ✓ Modèle de Données : En Django, les modèles de données sont définis à l'aide de classes Python. Un modèle représente une table dans la base de données, et chaque attribut de la classe représente une colonne de cette table. Par exemple, si nous avons un modèle pour les étudiants, il pourrait ressembler à ceci :

```
class Student(models.Model):
```

```
name = models.CharField(max_length=100)
age = models.IntegerField()
grade = models.CharField(max_length=2)
```

#Dans cette classe, name, age, et grade sont les attributs du modèle Student#

- ✓ Vue :Les vues dans Django sont des fonctions Python qui prennent une requête HTTP en entrée et renvoient une réponse HTTP. Les vues peuvent être utilisées pour traiter les données d'entrée, interagir avec les modèles, et rendre des templates HTML. Par exemple :

```
from django.shortcuts import render
```

```
def index(request):
    return render(request, 'index.html')
```

#Cette vue index renvoie le template index.html en réponse à la requête HTTP.#

- ✓ Template :Les templates dans Django sont des fichiers HTML qui peuvent contenir des balises et des instructions spéciales pour l'insertion de données dynamiques. Les templates peuvent être rendus à l'aide de vues pour générer des pages Web dynamiques. Par exemple :

```
<!DOCTYPE html>
<html>
<head>
    <title>Mon Site Web</title>
</head>
<body>
    <h1>Bienvenue sur mon site !</h1>
    <p>{{ message }}</p>
</body></html>
```

#Dans ce template, `{{ message }}` est une balise qui sera remplacée par une valeur dynamique lorsque le template sera rendu.#

- ✓ URLs :Les URLs dans Django sont définies dans le fichier urls.py de chaque application. Chaque URL est associée à une vue spécifique. Par exemple :

```
from django.urls import path
from . import views
```

```
urlpatterns = [
    path('', views.index, name='index'),]
```

#Cette URL vide est associée à la vue `index` et sera utilisée pour accéder à la page d'accueil de l'application.#

Chapitre 3 : Analyse et conception

Dans ce chapitre, nous examinerons les aspects clés de l'analyse et de la conception de notre système de vérification d'identité. En utilisant divers types de diagrammes, nous fournirons une représentation visuelle claire des composants du système et de leurs interactions. Ces outils sont essentiels pour modéliser, documenter et comprendre l'architecture et le fonctionnement de notre projet, assurant ainsi une conception structurée et efficace.

1. Rôle des Diagrammes dans l'Analyse et la Conception :

La conception d'un système logiciel complexe nécessite une représentation visuelle de ses composants, de leurs interactions et des processus qu'ils exécutent.

Les diagrammes jouent un rôle essentiel dans ce processus en fournissant une méthode structurée pour modéliser et documenter les différents aspects du système.

Dans cette section, nous utilisons plusieurs types de diagrammes pour explorer en détail l'architecture et le fonctionnement du système de vérification d'admission des étudiants lors des examens.(2)

2.1. Diagramme de cas d'utilisation :

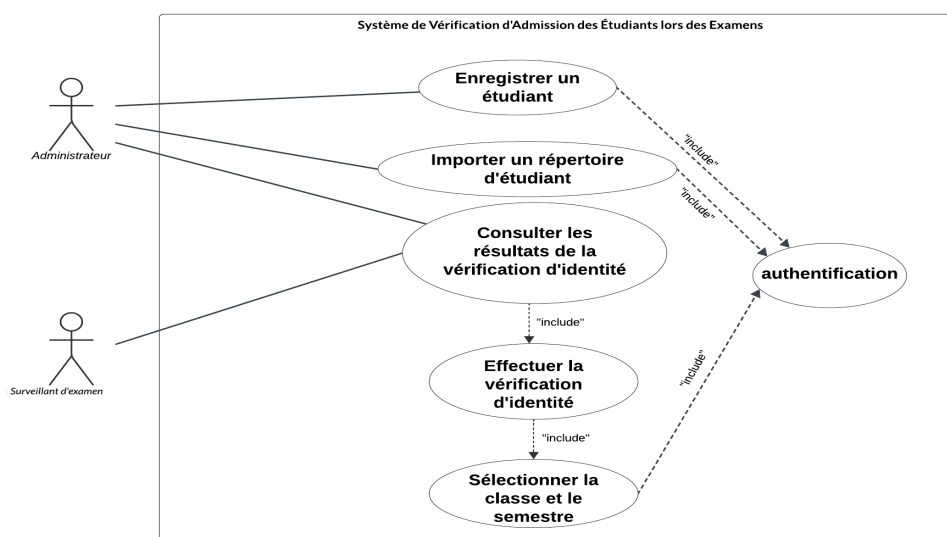


Figure 5 : Diagramme de cas d'utilisation

Ce diagramme de cas d'utilisation représente les différentes interactions entre les acteurs du système et les fonctionnalités principales de notre système de vérification d'admission des étudiants lors des examens.

Acteurs

- **Administrateur** : Responsable de la gestion des étudiants et des données associées.
- **Surveillant d'examen** : Responsable de la vérification de l'identité des étudiants pendant les examens.

Cas d'Utilisation

1. **Enregistrer un étudiant**
 - **Acteur** : Administrateur
 - **Description** : L'administrateur peut ajouter de nouveaux étudiants au système en enregistrant leurs informations personnelles et leurs données faciales.
2. **Importer un répertoire d'étudiants**
 - **Acteur** : Administrateur
 - **Description** : L'administrateur peut importer un répertoire contenant les données faciales de plusieurs étudiants pour les enregistrer en bloc dans le système.
3. **Consulter les résultats de la vérification d'identité**
 - **Acteur** : Administrateur
 - **Description** : L'administrateur peut consulter les résultats des vérifications d'identité effectuées, pour s'assurer que tous les étudiants ont été correctement vérifiés.
4. **Effectuer la vérification d'identité**
 - **Acteur** : Surveillant d'examen
 - **Description** : Le surveillant d'examen peut vérifier l'identité des étudiants en comparant leur visage à leurs données enregistrées dans le système.
5. **Sélectionner la classe et le semestre**
 - **Acteur** : Surveillant d'examen
 - **Description** : Le surveillant d'examen peut sélectionner la classe et le semestre des étudiants avant de procéder à la vérification de leur identité.

Relation d'Inclus (<<include>>)

- **Authentification**
 - **Acteurs** : Administrateur et Surveillant d'examen
 - **Description** : L'authentification est une action commune à plusieurs cas d'utilisation. Les utilisateurs doivent être authentifiés pour accéder aux fonctionnalités du système.

Chaque interaction de l'utilisateur avec le système passe par une étape d'authentification pour garantir la sécurité et l'intégrité des données.

2.2. Diagramme de Séquence :

i. Diagramme de Séquence- Authentification :

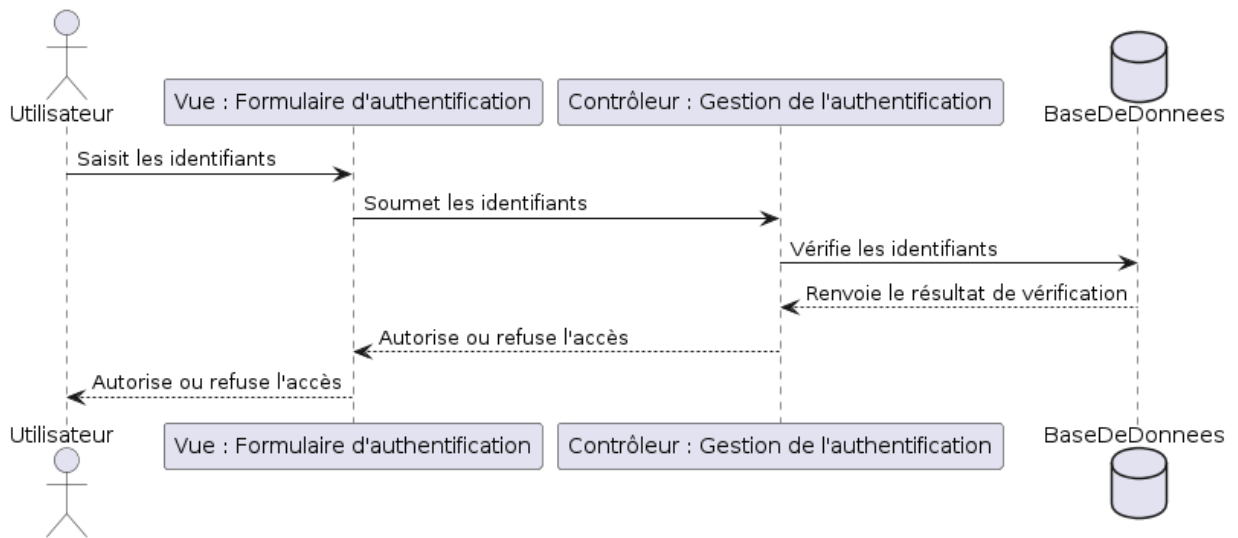


Figure 6 : Diagramme de Séquence- Authentification

Description du Diagramme

1. **Utilisateur :**
 - L'utilisateur saisit ses identifiants (nom d'utilisateur et mot de passe) via l'interface de formulaire d'authentification.
2. **Vue : Formulaire d'authentification :**
 - L'utilisateur soumet ses identifiants pour vérification.
3. **Contrôleur : Gestion de l'authentification :**
 - Le contrôleur reçoit les identifiants soumis et envoie une requête à la base de données pour vérifier les informations.
4. **Base de Données :**
 - La base de données vérifie les identifiants reçus du contrôleur.
 - La base de données renvoie le résultat de la vérification (succès ou échec) au contrôleur.
5. **Contrôleur :**
 - Le contrôleur reçoit le résultat de la base de données et décide d'autoriser ou de refuser l'accès à l'utilisateur.
 - Le contrôleur envoie la décision (autorisation ou refus) à la vue.
6. **Vue : Formulaire d'authentification :**
 - La vue affiche le résultat de l'authentification à l'utilisateur, l'autorisant ou lui refusant l'accès.

Explication des Composants et des Interactions

- **Vue : Formulaire d'authentification :** Interface où l'utilisateur entre ses identifiants.
- **Contrôleur : Gestion de l'authentification :** Composant qui gère la logique de vérification des identifiants et de décision d'accès.
- **Base de Données :** Stocke les identifiants des utilisateurs et vérifie les informations soumises par le contrôleur.

Scénarios Alternatifs

- **Échec de l'authentification** : Si les identifiants sont incorrects, le contrôleur renvoie un message d'erreur à la vue, qui informe l'utilisateur de l'échec de l'authentification.
- **Tentatives de connexion multiples** : En cas de tentatives de connexion multiples échouées, le système peut verrouiller le compte de l'utilisateur temporairement pour des raisons de sécurité.

ii. *Diagramme de Séquence- Vérification de l'Identité de l'Étudiant :*

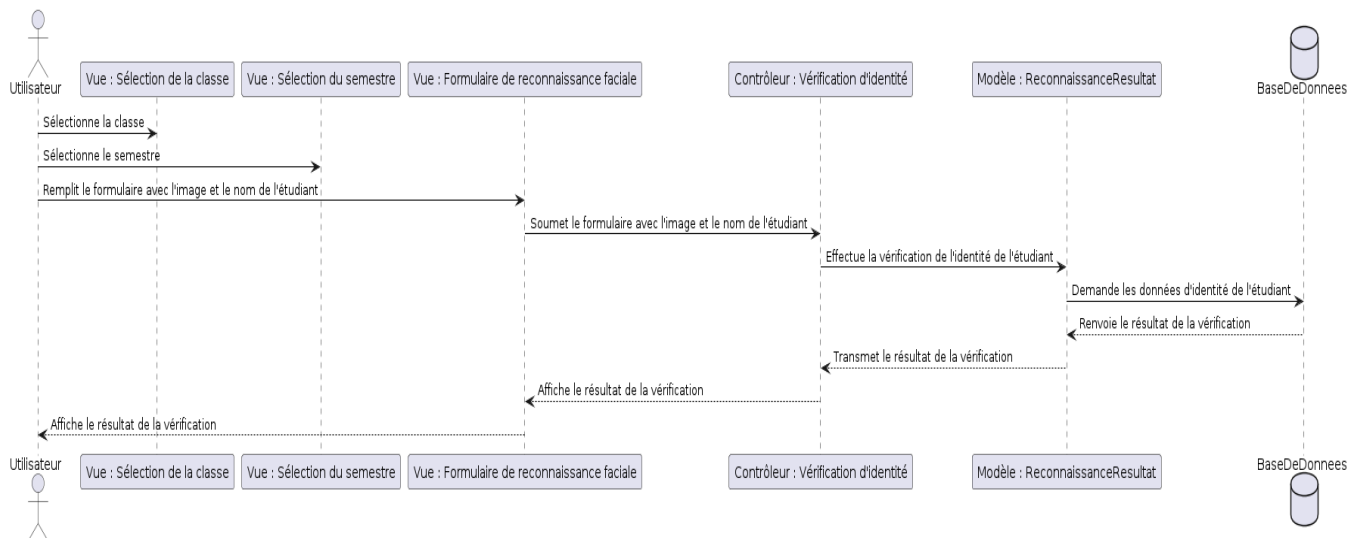


Figure 7 : Diagramme de Séquence- Vérification de l'Identité de l'Étudiant

Description du Diagramme

- Utilisateur :**
 - L'utilisateur (étudiant) commence par sélectionner une classe via l'interface utilisateur.
- Vue : Sélection de la classe :**
 - L'utilisateur sélectionne la classe, ce qui déclenche la vue de sélection du semestre.
- Vue : Sélection du semestre :**
 - L'utilisateur sélectionne le semestre, ce qui mène à la vue du formulaire de reconnaissance faciale.
- Vue : Formulaire de reconnaissance faciale :**
 - L'utilisateur remplit le formulaire avec son image et son nom.
 - L'utilisateur soumet le formulaire avec ces informations.
- Contrôleur : Vérification d'identité :**
 - Le contrôleur reçoit le formulaire soumis et effectue la vérification de l'identité de l'étudiant.
 - Le contrôleur transmet les informations nécessaires au modèle pour la reconnaissance faciale.
- Modèle : ReconnaissanceResultat :**
 - Le modèle effectue la vérification de l'identité en comparant l'image soumise avec les données stockées dans la base de données.
 - Le modèle renvoie le résultat de la vérification au contrôleur.
- Contrôleur :**
 - Le contrôleur affiche le résultat de la vérification via l'interface utilisateur.

Explication des Composants et des Interactions

- **Vue : Sélection de la classe** : Interface permettant à l'utilisateur de choisir la classe.
- **Vue : Sélection du semestre** : Interface permettant à l'utilisateur de choisir le semestre après avoir sélectionné la classe.
- **Vue : Formulaire de reconnaissance faciale** : Interface où l'utilisateur soumet son image et son nom pour vérification.
- **Contrôleur : Vérification d'identité** : Composant qui gère la logique de vérification d'identité en coordonnant avec le modèle et la base de données.
- **Modèle : ReconnaissanceRésultat** : Composant responsable de la comparaison des images et de la détermination du résultat de la vérification.
- **Base de Données** : Stocke les données nécessaires pour la reconnaissance faciale, comme les images référentielles des étudiants.

Scénarios Alternatifs

- **Échec de la vérification** : Si l'identité de l'étudiant ne peut pas être vérifiée, le système peut renvoyer un message d'erreur ou une notification d'échec à l'utilisateur.
- **Sélection incorrecte** : Si l'utilisateur sélectionne une classe ou un semestre incorrect, il peut être redirigé pour refaire la sélection.

iii. *Diagramme de Sequence-le model de verification :*

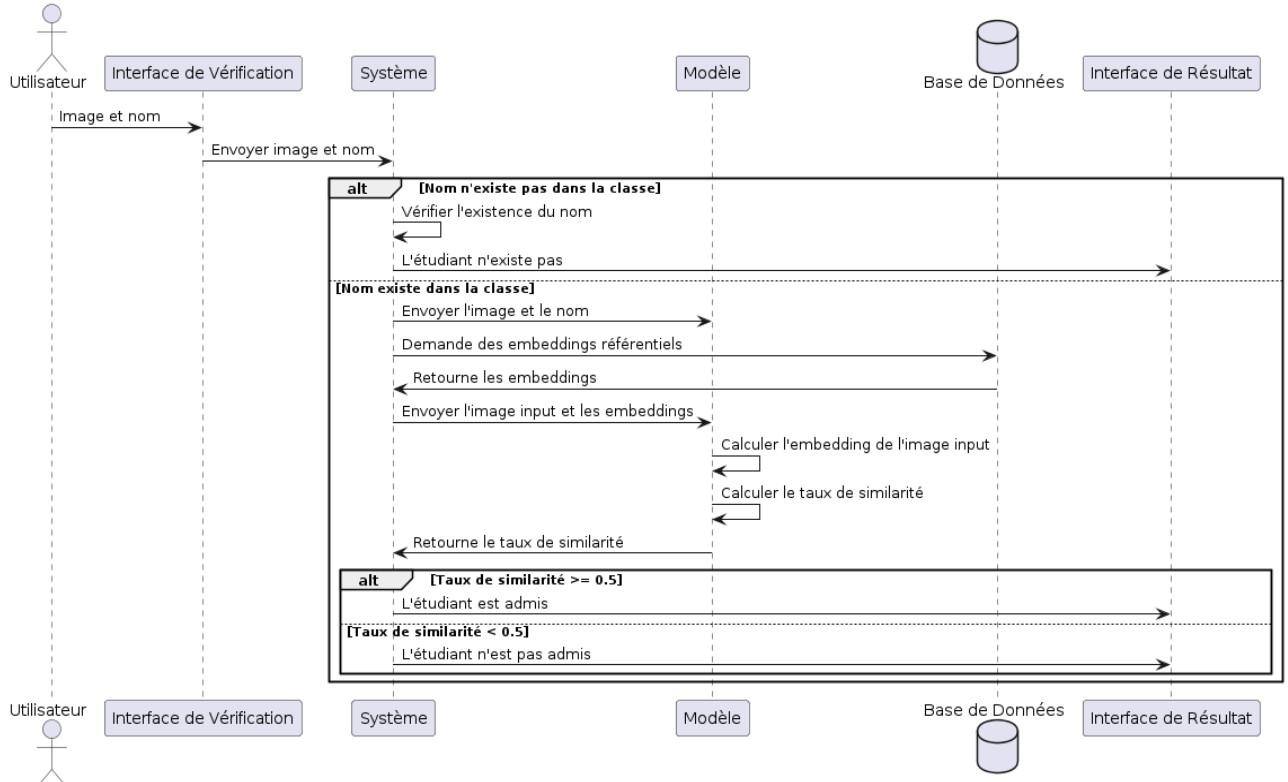


Figure 8 :Diagramme de Sequence-le model de verification

Description du Diagramme

1. **Utilisateur :**
 - L'utilisateur (étudiant) envoie une image et un nom via l'interface de vérification.
2. **Interface de Vérification :**
 - L'interface de vérification transmet l'image et le nom au système.
3. **Système :**
 - Le système vérifie l'existence du nom dans la classe.
 - **Si le nom n'existe pas dans la classe :**
 - Le système informe l'utilisateur que l'étudiant n'existe pas.
 - **Si le nom existe dans la classe :**
 - Le système envoie l'image et le nom au modèle.
4. **Modèle :**
 - Le modèle demande les embeddings référentiels à la base de données.
 - La base de données retourne les embeddings au modèle.
 - Le modèle envoie l'image d'entrée et les embeddings reçus pour calculer l'embedding de l'image d'entrée.
 - Le modèle calcule le taux de similarité entre l'embedding de l'image d'entrée et les embeddings référentiels.
 - Le modèle retourne le taux de similarité au système.
5. **Système :**
 - **Si le taux de similarité est supérieur ou égal à 0.5 :**
 - Le système informe l'utilisateur que l'étudiant est admis.
 - **Si le taux de similarité est inférieur à 0.5 :**
 - Le système informe l'utilisateur que l'étudiant n'est pas admis.

Explication des Composants et des Interactions

- **Interface de Vérification :** Il s'agit de l'interface utilisateur où l'étudiant soumet son nom et son image pour vérification.
- **Système :** Le composant central qui gère le flux de données entre l'interface de vérification, le modèle et la base de données.
- **Modèle :** La partie du système qui contient la logique de traitement de l'image et la comparaison des embeddings.
- **Base de Données :** Stocke les embeddings référentiels des étudiants enregistrés.
- **Interface de Résultat :** Affiche le résultat final de la vérification (admis ou non admis) à l'utilisateur.

Scénarios Alternatifs

- **Nom n'existe pas dans la classe :** Si le nom soumis par l'utilisateur n'est pas trouvé dans la classe, le système informe l'utilisateur que l'étudiant n'existe pas.
- **Taux de similarité :** Le taux de similarité est utilisé pour déterminer si l'étudiant est admis ou non. Un seuil de 0.5 est utilisé dans cet exemple.

iv. Diagramme de Séquence- Importer un Répertoire d'Étudiants :

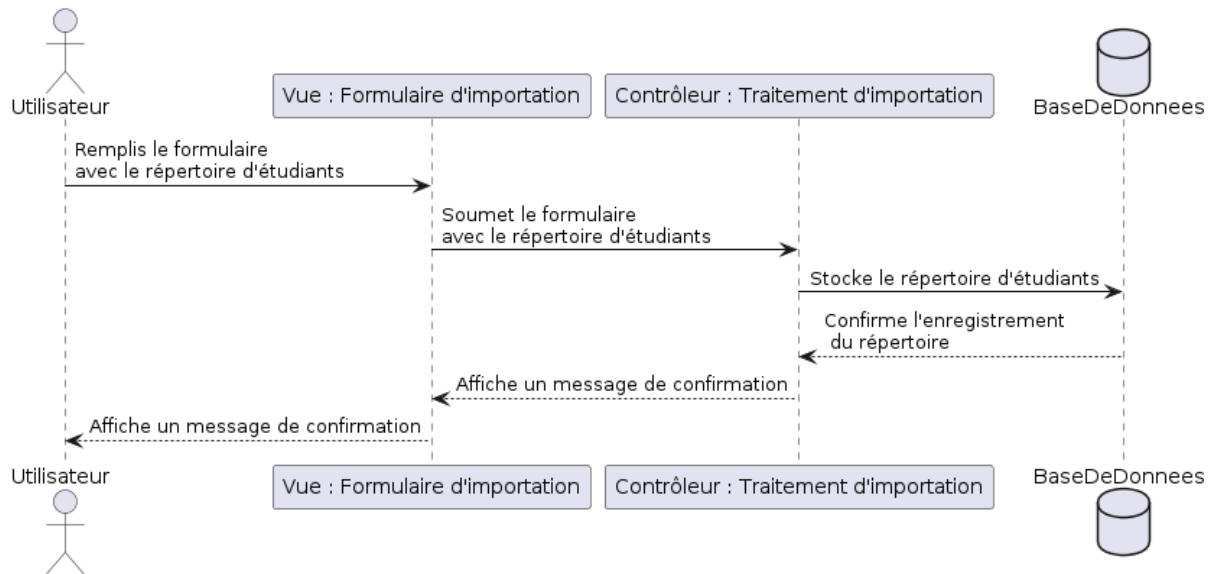


Figure 9 : Diagramme de Séquence- Importer un Répertoire d'Étudiants

Description du Diagramme

1. **Utilisateur :**
 - L'utilisateur souhaite importer un répertoire contenant les données faciales des étudiants.
2. **Vue : Formulaire d'importation :**
 - L'utilisateur accède à la vue d'importation et téléverse un fichier ZIP contenant les images des étudiants.
3. **Contrôleur : Gestion de l'importation :**
 - Le contrôleur reçoit le fichier téléversé et commence le processus d'importation.
4. **Base de Données :**
 - Le contrôleur enregistre les informations des étudiants dans la base de données après avoir extrait et analysé les images du fichier ZIP.
5. **Modèle : Gestion des données d'étudiants :**
 - Le modèle est responsable de créer les enregistrements des étudiants et de stocker les embeddings faciaux pour la reconnaissance future.
6. **Vue : Confirmation de l'importation :**
 - Une fois l'importation réussie, la vue affiche un message de confirmation à l'utilisateur.

Explication des Composants et des Interactions

- **Vue : Formulaire d'importation :** Interface où l'utilisateur peut téléverser un fichier ZIP contenant les images des étudiants.
- **Contrôleur : Gestion de l'importation :** Composant qui gère la logique d'extraction des images, de création des embeddings et d'enregistrement des informations dans la base de données.
- **Base de Données :** Stocke les informations des étudiants et les embeddings faciaux.
- **Modèle : Gestion des données d'étudiants :** Structure les données des étudiants et génère les embeddings faciaux pour la reconnaissance.

Étapes du Processus

1. **Téléversement du Fichier :**
 - L'utilisateur sélectionne un fichier ZIP et le téléverse via le formulaire d'importation.
2. **Traitement du Fichier :**
 - Le contrôleur extrait les images du fichier ZIP.
 - Pour chaque image, le contrôleur génère un embedding facial.
 - Le contrôleur enregistre les informations des étudiants et les embeddings faciaux dans la base de données.
3. **Confirmation :**
 - Une fois l'importation terminée, le système affiche une confirmation de succès à l'utilisateur.

Scénarios Alternatifs

- **Échec de l'importation :** Si le fichier téléversé est corrompu ou ne contient pas les images nécessaires, le contrôleur envoie un message d'erreur à la vue, qui informe l'utilisateur de l'échec de l'importation.
- **Fichier non valide :** Si l'utilisateur tente de téléverser un fichier qui n'est pas au format ZIP, le système rejette le fichier et affiche un message d'erreur.

2.3. Diagramme de Classe :

Le diagramme de classes est un élément fondamental de la modélisation orientée objet qui permet de représenter la structure statique du système. Il décrit les différentes classes qui composent le système, ainsi que leurs attributs, méthodes et les relations entre elles. Ce diagramme est essentiel pour comprendre comment les composants du système interagissent et comment ils sont organisés.(2)

Dans le cadre de notre projet de vérification d'identité des étudiants lors des examens, le diagramme de classes illustre les principales entités du système ainsi que leurs interactions. Ce diagramme nous aide à visualiser la structure du système de manière claire et précise, facilitant ainsi le développement et la maintenance du code.

Les principales classes identifiées dans notre système sont les suivantes :

- **Utilisateur** : Classe parent représentant un utilisateur générique du système.
- **Étudiant** : Sous-classe d'Utilisateur, représentant les étudiants qui doivent être vérifiés.
- **Surveillant** : Sous-classe d'Utilisateur, représentant les surveillants qui capturent les images des étudiants.
- **Admin** : Sous-classe d'Utilisateur, représentant les administrateurs qui gèrent les enregistrements des étudiants.
- **SystèmeDeVerification** : Classe principale orchestrant le processus de vérification d'identité.
- **BaseDeDonnées** : Classe responsable de la gestion des données des utilisateurs et des étudiants.
- **ReconnaissanceFaciale** : Classe implémentant les algorithmes de reconnaissance faciale pour comparer les images.

Chaque classe possède des attributs et des méthodes spécifiques, et les relations entre les classes, ainsi que les cardinalités, sont définies pour représenter les interactions entre les différentes entités. Ce diagramme de classes sert de plan détaillé pour le développement de notre système, assurant une architecture bien structurée et cohérente.

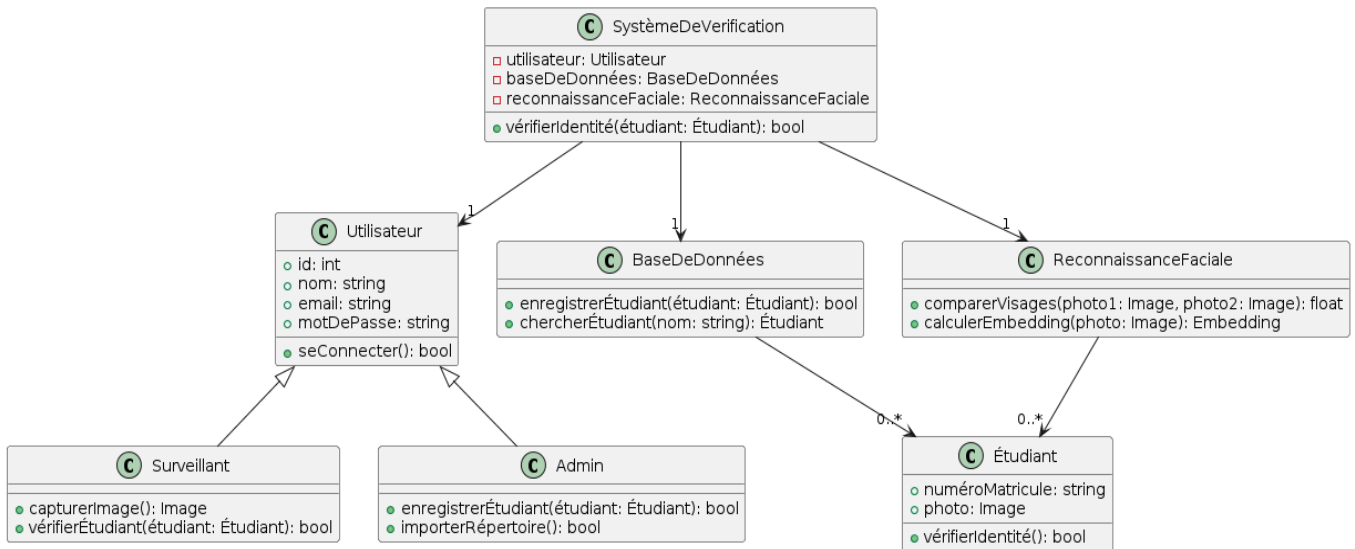


Figure 10 : Diagramme de classe

Description des classes :

- **Utilisateur** : La classe de base pour les utilisateurs du système. Inclut des attributs pour l'identifiant, le nom, l'email et le mot de passe, ainsi qu'une méthode pour se connecter.
- **Étudiant** : Hérite de `Utilisateur`. Ajoute des attributs spécifiques à un étudiant comme le numéro de matricule et la photo. Inclut une méthode pour vérifier l'identité.
- **Surveillant** : Hérite de `Utilisateur`. Inclut des méthodes pour capturer une image et vérifier l'identité d'un étudiant.
- **Admin** : Hérite de `Utilisateur`. Inclut des méthodes pour enregistrer un étudiant et importer un répertoire d'étudiants.
- **SystèmeDeVerification** : Représente le système de vérification d'identité. Contient des références à `Utilisateur`, `BaseDeDonnées`, et `ReconnaissanceFaciale`, et inclut une méthode pour vérifier l'identité d'un étudiant.
- **BaseDeDonnées** : Représente la base de données du système. Inclut des méthodes pour enregistrer et chercher des étudiants.
- **ReconnaissanceFaciale** : Inclut des méthodes pour comparer les visages et calculer des embeddings à partir de photos. (2)

2. Choix de Méthodologie :

Le choix de la méthodologie pour le développement de notre système de vérification d'admission des étudiants lors des examens est une étape cruciale qui influence la planification, l'exécution et la qualité du projet. Après une analyse approfondie des besoins et des contraintes du projet, nous avons opté pour la méthodologie Agile en raison de ses avantages et de sa pertinence pour notre contexte.(3)

✓ Justification du Choix :

- Flexibilité et Adaptabilité : La méthodologie Agile est connue pour sa capacité à s'adapter aux changements et aux évolutions fréquentes des exigences du projet. Dans notre cas, où les besoins des utilisateurs peuvent évoluer au fil du temps ou être clarifiés progressivement, cette flexibilité est essentielle pour assurer la satisfaction du client et la réussite du projet.(3)
- Livraisons Itératives : L'approche Agile privilégie les cycles de développement courts et itératifs, permettant ainsi des livraisons continues de fonctionnalités utilisables. Cette approche incrémentale nous permet de recueillir rapidement les retours des utilisateurs finaux, d'identifier les problèmes et d'apporter des ajustements en conséquence, assurant ainsi la pertinence et la qualité des livrables. (3)
- Collaboration Client : La méthodologie Agile encourage une collaboration étroite entre l'équipe de développement et le client tout au long du projet. Cette collaboration continue favorise une meilleure compréhension des besoins du client, une communication transparente et une prise de décision plus rapide, ce qui conduit à des solutions plus adaptées et satisfaisantes pour toutes les parties prenantes. (3)
- Gestion du Risque : En adoptant une approche incrémentale et en mettant l'accent sur la livraison rapide de petites fonctionnalités, la méthodologie Agile permet de mieux gérer les risques associés au projet. Les problèmes sont identifiés plus tôt dans le processus, ce qui réduit les impacts potentiels sur le projet et facilite leur résolution rapide. (3)
- Implémentation de l'Agile : Pour mettre en œuvre la méthodologie Agile dans notre projet, nous avons choisi d'utiliser le Framework Scrum. Scrum fournit une structure claire avec des rôles définis, des artefacts et des événements spécifiques, ce qui facilite la gestion et la coordination des activités de développement. Nous avons également adapté certaines pratiques Agile, telles que les réunions quotidiennes de stand-up, les revues de sprint et les rétrospectives, pour répondre aux besoins et à la dynamique de notre équipe de développement. (3)

3. Spécification fonctionnelle :

Le système utilise des formulaires Django pour interagir avec l'utilisateur. Il y a des vues dédiées pour l'authentification des utilisateurs, la sélection de la classe et du semestre, ainsi que la vérification de l'identité des étudiants.

- *Reconnaissance faciale :* Le cœur du système réside dans la vue `facial_recognition`. Cette vue utilise des algorithmes de reconnaissance faciale pour comparer les visages en temps réel lors de la vérification d'identité des étudiants.
- *Chargement et stockage des données :* Le code charge les embeddings et les labels à partir de fichiers numpy, qui contiennent les données faciales des étudiants enregistrés pour chaque classe et semestre. Les résultats de la vérification d'identité sont enregistrés dans la base de données Django.
- *Importation de répertoire :* Il existe une fonctionnalité pour importer un répertoire contenant les données faciales des étudiants pour une classe spécifique. Cette fonctionnalité utilise des formulaires pour téléverser et extraire les fichiers ZIP contenant les images des étudiants.

4. Conception du code :

- *Modularité :* Le code est bien structuré en utilisant des vues Django distinctes pour chaque fonctionnalité. Les fonctions sont découpées en modules logiques, ce qui facilite la maintenance et l'extension du code.
- *Utilisation de bibliothèques :* Le code utilise des bibliothèques Python populaires telles que OpenCV, dlib et numpy pour la reconnaissance faciale et le traitement d'image. Cela permet une implémentation efficace des fonctionnalités sans avoir à réinventer la roue.
- *Gestion des erreurs :* Le code gère les erreurs de manière adéquate en affichant des messages d'erreur conviviaux pour les utilisateurs lorsque des problèmes surviennent, par exemple, en cas d'échec de la vérification d'identité ou d'erreur dans le téléchargement du répertoire.

- Sécurité : Bien que le code n'aborde pas directement les aspects de sécurité tels que le chiffrement des données ou la gestion des autorisations d'accès, il utilise les fonctionnalités intégrées de Django telles que l'authentification des utilisateurs pour assurer un niveau de sécurité de base.
- Extensibilité : Le code est conçu de manière à pouvoir être étendu facilement pour ajouter de nouvelles fonctionnalités ou améliorer les fonctionnalités existantes. Par exemple, il serait relativement simple d'ajouter une fonctionnalité de suivi des activités suspectes ou de génération de rapports sur les résultats de la vérification d'identité.

Chapitre 4 : Réalisation du projet

1. Introduction

Dans cette partie, nous allons détailler les outils et environnements nécessaires à la réalisation de notre projet de vérification d'admission des étudiants par reconnaissance faciale et deep learning. Nous aborderons divers éléments tels que les langages de programmation, les frameworks, les bibliothèques, ainsi que les logiciels indispensables comme les éditeurs de code, les plateformes de gestion de projets et les outils de visualisation et de présentation.

L'objectif est de mettre en lumière les ressources utilisées, qui sont majoritairement libres de droits, disponibles en version d'essai, ou proposant des alternatives open source. Cette approche garantit non seulement la conformité aux réglementations en vigueur, mais aussi l'accessibilité et la durabilité des outils nécessaires à la réalisation de notre projet.

2. Langages et Outils de Développement

1.1. Langages et Scripts :

- **HTML5 :**



Le terme HTML est l'acronyme de "HyperText Markup Language", qui se traduit par "langage de balisage pour l'hypertexte". Il est utilisé pour la création et la représentation du contenu ainsi que de la structure d'une page web. La dernière révision majeure de HTML est le HTML5, qui a été finalisé le 28 octobre 2014. Cette version apporte de nombreuses nouveautés telles que les formats de date et d'email, ainsi que l'introduction des balises footer, header, section, etc. Cette évolution du HTML offre des fonctionnalités étendues pour la structuration et la présentation du contenu web.(1)

Dans notre projet, HTML5 a été utilisé pour développer les templates de l'interface utilisateur dans le cadre du framework Django. Ces templates permettent de créer des pages web dynamiques et interactives, facilitant ainsi l'interaction entre les surveillants d'examens et le système de vérification faciale.



- **CSS :**

CSS, ou "Cascading Style Sheets", est un langage de feuille de style utilisé pour décrire la présentation d'un document écrit en HTML ou XML. CSS permet de séparer le contenu de la présentation, offrant ainsi une flexibilité accrue dans la conception de l'apparence des pages web. Il permet de contrôler la mise en page, les

couleurs, les polices, et de créer des designs réactifs qui s'adaptent aux différentes tailles d'écran.

Dans notre projet, CSS a été utilisé pour styliser les templates HTML5 et améliorer l'expérience utilisateur en créant une interface utilisateur esthétique et conviviale. Grâce à CSS, nous avons pu assurer une présentation cohérente et professionnelle de notre application web.(1)

- **Python :**



Python est un langage de programmation de haut niveau, connu pour sa simplicité et sa lisibilité. Il est largement utilisé dans le développement de projets d'intelligence artificielle et de deep learning grâce à ses nombreuses bibliothèques spécialisées, comme TensorFlow et Keras. Dans notre projet, Python a été utilisé pour développer les modules de reconnaissance faciale et les algorithmes de deep learning nécessaires à l'identification des étudiants.

- **JavaScript :**



JavaScript est un langage de programmation essentiel pour le développement web, permettant de rendre les pages web interactives. Dans notre projet, JavaScript a été utilisé pour améliorer l'interactivité des pages web, notamment pour les fonctionnalités de capture d'images en temps réel et d'affichage dynamique des résultats de vérification.(1)

1.2. Frameworks et Bibliothèques

- **Django :**

Django est un framework web Python de haut niveau qui encourage le développement rapide et le design propre et pragmatique. Dans notre projet, Django a été utilisé pour structurer l'application web, gérer les bases de données et intégrer les templates HTML5 pour créer une interface utilisateur intuitive.

- **OpenCV et dlib**

OpenCV (Open Source Computer Vision Library) et dlib sont des bibliothèques utilisées pour le traitement d'images et la reconnaissance faciale. OpenCV offre une grande variété de fonctions pour la manipulation d'images, tandis que dlib est particulièrement utile pour la détection et l'alignement des visages. Ces bibliothèques ont été intégrées dans notre projet pour capturer, traiter et analyser les images faciales des étudiants.

- TensorFlow et Keras

TensorFlow est une bibliothèque open-source de Google utilisée pour le calcul numérique et le machine learning. Keras est une API de réseau de neurones de haut niveau, écrit en Python et capable de s'exécuter au-dessus de TensorFlow. Dans notre projet, ces outils ont été utilisés pour développer et entraîner les modèles de deep learning nécessaires à la reconnaissance faciale.

1.3. Outils de développement :

- Visual Studio Code



Visual Studio Code (VS Code) est un éditeur de code source développé par Microsoft, apprécié pour sa flexibilité et ses nombreuses extensions. Nous avons utilisé VS Code pour écrire et déboguer le code de notre projet.

- PyCharm

PyCharm est un environnement de développement intégré (IDE) pour Python, développé par JetBrains. Il offre des outils puissants pour le développement Python et a été utilisé pour la partie back-end de notre projet.

- Git et GitHub

Git est un système de contrôle de version décentralisé, et GitHub est une plateforme de collaboration pour le développement de logiciels. Nous avons utilisé Git pour le versioning de notre code et GitHub pour la gestion des projets et la collaboration en équipe.

3. CODE ET DÉVELOPPEMENT

✓ Implémentation d'un Réseau siamois en Python pour la Comparaison d'Images

un Réseau Siamois : Un réseau siamois se compose de deux sous-réseaux identiques (partageant les mêmes poids et architecture) qui prennent deux entrées distinctes et produisent deux vecteurs d'embedding. Une couche de distance (comme L1 ou L2) est ensuite appliquée pour mesurer la similitude entre ces deux vecteurs. Le réseau est formé pour minimiser la distance entre les embeddings de paires similaires et maximiser la distance entre les embeddings de paires différentes.

○ Construire le Sous-Réseau (Architecture Partagée)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Lambda
from tensorflow.keras import backend as K

def make_embedding():
    inp = Input(shape=(100, 100, 3), name='input_image')

    # Premier bloc
    c1 = Conv2D(64, (10, 10), activation='relu')(inp)
    m1 = MaxPooling2D(64, (2, 2), padding='same')(c1)

    # Deuxième bloc
    c2 = Conv2D(128, (7, 7), activation='relu')(m1)
    m2 = MaxPooling2D(64, (2, 2), padding='same')(c2)

    # Troisième bloc
    c3 = Conv2D(128, (4, 4), activation='relu')(m2)
    m3 = MaxPooling2D(64, (2, 2), padding='same')(c3)

    # Bloc final d'embedding
    c4 = Conv2D(256, (4, 4), activation='relu')(m3)
    f1 = Flatten()(c4)
    d1 = Dense(4096, activation='sigmoid')(f1)

    return Model(inputs=inp, outputs=d1, name='embedding')
```

Explication de code

Importation des Bibliothèques

numpy est importé pour manipuler les données sous forme de tableaux.

tensorflow et tensorflow.keras sont importés pour créer et entraîner le modèle de réseau neuronal

Définition de la Fonction make_embedding

La fonction commence par définir l'entrée du modèle avec une forme de (100,100,3) ce qui signifie des images de 100x100 pixels avec 3 canaux de couleur (RGB).

Premier Bloc Convolutif

Un calque convolutif avec 64 filtres de taille 10x10 et une fonction d'activation relu est appliqué à l'entrée.

Un calque de max-pooling est ensuite appliqué avec une taille de fenêtre de 2x2, ce qui réduit les dimensions de moitié, tout en conservant le même remplissage pour garder la même taille de sortie.

Deuxième Bloc Convolutif

Un deuxième calque convolutif avec 128 filtres de taille 7x7 et une fonction d'activation relu est appliqué à la sortie du premier bloc.

Un calque de max-pooling avec les mêmes paramètres que précédemment est appliqué à la sortie de cette couche.

Troisième Bloc Convolutif

Un troisième calque convolutif avec 128 filtres de taille 4x4 et une fonction d'activation relu est appliqué à la sortie du deuxième bloc.

Un calque de max-pooling avec les mêmes paramètres que précédemment est appliqué à la sortie de cette couche.

Bloc Final d'Embedding

Un quatrième calque convolutif avec 256 filtres de taille 4x4 et une fonction d'activation relu est appliqué à la sortie du troisième bloc.

La sortie de cette couche est aplatie (convertie en un vecteur 1D).

Un calque dense avec 4096 unités et une fonction d'activation sigmoid est appliqué pour obtenir l'embedding final.

Création du Modèle

Le modèle est créé en utilisant l'entrée définie initialement et la sortie du dernier calque dense.

Le modèle est nommé 'embedding'.

- **Définir la Fonction de Distance L1**

Ce code définit une classe personnalisée L1Dist pour calculer la distance L1 entre deux embeddings dans un réseau siamois en utilisant TensorFlow. Cette distance est également connue sous le nom de distance de Manhattan ou somme des valeurs absolues des différences des coordonnées correspondantes.

```
# Siamese L1 Distance class
class L1Dist(Layer):

    # Init method - inheritance
    def __init__(self, **kwargs):
        super().__init__()

    # Magic happens here - similarity calculation
    def call(self, input_embedding, validation_embedding):
        return tf.math.abs(input_embedding - validation_embedding)
```

Explication de code :

Déclaration de la Classe

La classe L1Dist hérite de Layer, une classe de base fournie par TensorFlow pour créer des couches personnalisées.

Méthode `__init__` :

Le constructeur `__init__` initialise la classe.

`super().__init__()` appelle le constructeur de la classe parente (Layer) pour s'assurer que la classe de base est correctement initialisée.

Méthode `call`

La méthode `call` est la méthode principale où se produit le calcul de la distance L1.

Elle prend deux arguments : `input_embedding` et `validation_embedding`, qui sont les embeddings des images d'entrée et de validation respectivement.

`tf.math.abs(input_embedding - validation_embedding)` calcule la distance L1 en prenant la valeur absolue de la différence entre les embeddings correspondants.

Utilisation :

Dans un réseau siamois, cette classe peut être utilisée pour mesurer la similitude entre deux embeddings en calculant leur distance L1. Voici comment cette classe peut être intégrée dans un modèle siamois :

1. **Définition du modèle** : Inclure cette couche après avoir obtenu les embeddings des deux branches du réseau siamois.
2. **Calcul de la perte** : Utiliser cette distance pour calculer la perte pendant l'entraînement, par exemple en utilisant une fonction de perte comme la perte contrastive ou la perte de triplet.

- Construire le Modèle Siamois

Nous allons utiliser la fonction `make_embedding` pour créer deux instances de l'embedding, partager leurs poids et ensuite calculer la distance L1 entre les deux embeddings.

```
def build_siamese_network(input_shape):  
    # Créer Les deux branches du réseau siamois  
    embedding_network = make_embedding()  
  
    input_a = Input(shape=input_shape, name='input_a')  
    input_b = Input(shape=input_shape, name='input_b')  
  
    # Générer Les embeddings pour Les deux entrées  
    embedding_a = embedding_network(input_a)  
    embedding_b = embedding_network(input_b)  
  
    # Calculer la distance L1 entre Les deux embeddings  
    distance = Lambda(l1_distance, name='l1_distance')([embedding_a, embedding_b])  
  
    # Ajouter une couche Dense pour obtenir La prédiction finale  
    output = Dense(1, activation='sigmoid', name='output')(distance)  
  
    # Construire Le modèle siamois  
    siamese_model = Model(inputs=[input_a, input_b], outputs=output, name='siamese_network')  
  
    return siamese_model
```

Explication de code :

Création du Réseau d'Embedding :

`make_embedding()` : Cette fonction crée un sous-réseau convolutif utilisé pour générer des embeddings à partir des images d'entrée.

`embedding_network` : Ce réseau sera utilisé pour transformer les images d'entrée en vecteurs d'embedding.

Définition des Entrées :

`Input(shape=input_shape)` : Crée deux entrées pour le modèle siamois, chacune ayant la forme spécifiée par `input_shape`.

`name='input_a'` et `name='input_b'` : Noms des entrées pour les différencier.

Génération des Embeddings :

`embedding_network(input_a)` et `embedding_network(input_b)` : Applique le réseau d'embedding aux deux images d'entrée pour obtenir leurs embeddings respectifs

Calcul de la Distance L1 :

Lambda(l1_distance) : Utilise une couche Lambda pour appliquer la fonction l1_distance aux embeddings.

name='l1_distance' : Nom de la couche Lambda.

[embedding_a, embedding_b] : Liste des deux embeddings en entrée de la couche Lambda.

Ajout d'une Couche Dense pour la Prédiction Finale

Dense(1, activation='sigmoid') : Ajoute une couche dense avec une seule unité et une activation sigmoïde.

name='output' : Nom de cette couche de sortie.

distance : La distance L1 entre les deux embeddings, entrée de la couche Dense

Construction du Modèle Siamois :

Model(inputs=[input_a, input_b], outputs=output) : Crée le modèle siamois en spécifiant les entrées et la sortie.

name='siamese_network' : Nom du modèle

Retourner le Modèle :

Retourne l'instance du modèle siamois créé.

- la Fonction train_step

```
@tf.function
def train_step(batch):

    # Record all of our operations
    with tf.GradientTape() as tape:
        # Get anchor and positive/negative image
        X = batch[:2]
        # Get label
        y = batch[2]

        # Forward pass
        yhat = siamese_network(X, training=True)
        # Calculate loss
        loss = binary_cross_loss(y, yhat)
    print(loss)

    # Calculate gradients
    grad = tape.gradient(loss, siamese_network.trainable_variables)

    # Calculate updated weights and apply to siamese model
    opt.apply_gradients(zip(grad, siamese_network.trainable_variables))

    # Return loss
    return loss
```

La fonction `train_step` est une partie essentielle du processus d'entraînement d'un modèle d'apprentissage profond. Dans le contexte d'un modèle siamois, elle est responsable de calculer la perte (ou `loss`), de calculer les gradients par rapport aux poids du modèle, et de mettre à jour ces poids en fonction des gradients calculés à chaque itération de l'entraînement

Explication de code :

La fonction `train_step` est décorée avec `@tf.function`, ce qui indique à TensorFlow de compiler cette fonction en un graphe TensorFlow, ce qui améliore généralement les performances d'exécution en utilisant des opérations optimisées.

Paramètres de la Fonction :

`batch` : Ce paramètre représente un lot de données d'entraînement, généralement sous la forme d'un tuple ou d'une liste contenant les entrées et les étiquettes associées.

Utilisation de GradientTape :

`tf.GradientTape()` : TensorFlow utilise `GradientTape` pour enregistrer les opérations effectuées à l'intérieur de ce bloc. Cela permet de calculer les gradients de la perte par rapport aux variables du modèle.

Extraction des Entrées et des Étiquettes

`X` : Représente les données d'entrée du modèle siamois, composées de deux images (généralement l'ancrage et une image positive ou négative).

`y` : Les étiquettes associées à ce lot, indiquant généralement la similarité entre les paires d'images.

Passage en Avant (Forward Pass)

`siamese_network(X, training=True)` : Effectue une passe en avant à travers le modèle siamois avec les données d'entrée `X`. L'argument `training=True` est important pour activer le comportement d'entraînement du modèle, ce qui peut influencer le comportement de certaines couches, comme l'activation de couches de dropout ou de batch normalization.

Calcul de la Perte

`binary_cross_loss` : Cette fonction (non définie ici) calcule la perte entre les prédictions `yhat` et les étiquettes réelles `y`. Elle est souvent utilisée pour des tâches de classification binaire.

Calcul des Gradients :

tape.gradient(loss, siamese_network.trainable_variables) : Calcule les gradients de la perte par rapport aux variables entraînables (trainable_variables) du modèle siamois. Ces gradients sont nécessaires pour mettre à jour les poids du modèle.

Mise à Jour des Poids du Modèle

opt : Représente un optimiseur TensorFlow déjà initialisé, tel que tf.keras.optimizers.Adam ou tf.keras.optimizers.SGD.

apply_gradients : Cette méthode applique les gradients calculés aux variables du modèle, ce qui met à jour les poids en fonction des gradients et du taux d'apprentissage spécifié par l'optimiseur.

Retour de la Perte :

return loss : La fonction train_step retourne la perte calculée pour ce lot d'entraînement. Cette valeur peut être utilisée pour suivre la performance du modèle au fil du temps ou pour l'afficher pendant l'entraînement.

○ la Fonction train :

La fonction train est responsable de l'entraînement itératif d'un modèle siamois sur plusieurs epochs à partir d'un ensemble de données (data). Elle utilise des métriques intégrées telles que la précision (Precision) et le rappel (Recall) pour évaluer la performance du modèle pendant l'entraînement.

```
# Import metric calculations
from tensorflow.keras.metrics import Precision, Recall
def train(data, EPOCHS):
    # Loop through epochs
    for epoch in range(1, EPOCHS+1):
        print('\n Epoch {}/{}'.format(epoch, EPOCHS))
        progbar = tf.keras.utils.Progbar(len(data))

        # Creating a metric object
        r = Recall()
        p = Precision()

        # Loop through each batch
        for idx, batch in enumerate(data):
            # Run train step here
            loss = train_step(batch)
            yhat = siamese_network.predict(batch[:2])
            r.update_state(batch[2], yhat)
            p.update_state(batch[2], yhat)
            progbar.update(idx+1)
        print(loss.numpy(), r.result().numpy(), p.result().numpy())

    # Save checkpoints
    if epoch % 10 == 0:
        checkpoint.save(file_prefix=checkpoint_prefix)
```


Explication de code

Paramètres de la Fonction :

data : Représente l'ensemble de données utilisé pour l'entraînement du modèle siamois.

EPOCHS : Nombre d'epochs pour lesquels le modèle sera entraîné.

Boucle des Epochs :

La boucle for itère sur chaque epoch de 1 à EPOCHS. Progbar est utilisé pour afficher une barre de progression pendant l'entraînement.

Initialisation des Métriques :

Recall() et Precision() : Ce sont des métriques de TensorFlow Keras pour calculer respectivement le rappel et la précision. Elles sont initialisées à chaque epoch pour calculer les valeurs moyennes sur tout l'ensemble de données après chaque mise à jour de modèle.

Boucle à travers les Lots de Données :

enumerate(data) : Itère sur chaque lot de données dans data.

train_step(batch) : Appelle la fonction train_step pour effectuer une étape d'entraînement sur le lot batch.

siamese_network.predict(batch[:2]) : Prédit les sorties du modèle siamois pour les entrées batch[:2].

r.update_state(batch[2], yhat) et p.update_state(batch[2], yhat) : Met à jour les métriques de rappel et de précision avec les étiquettes réelles batch[2] et les prédictions yhat.

progbar.update(idx+1) : Met à jour la barre de progression avec l'indice actuel du lot.

Affichage des Métriques et de la Perte :

Affiche la perte actuelle, le rappel moyen et la précision moyenne à la fin de chaque epoch.

Sauvegarde des Points de Contrôle :

Sauvegarde les poids du modèle à intervalles réguliers (tous les 10 epochs dans cet exemple) pour permettre une reprise de l'entraînement à partir d'un certain point si nécessaire.

✓ **Intégration d'un Réseau siamois dans une Application Django**

Nous avons implémenté l'architecture des réseaux Siamese dans une application Django appelée "ExaFace" pour la vérification des étudiants pendant les examens. L'application est destinée aux surveillants d'examen, leur permettant d'authentifier les étudiants et de sélectionner la classe qu'ils surveillent. Une fois connectés, les surveillants accèdent à une interface comprenant un

formulaire pour saisir le nom de l'étudiant et une vidéo en direct pour capturer l'image de ce dernier. Le serveur traite cette image en calculant son embedding à l'aide de notre modèle Siamese, puis le compare avec les embeddings référentiels enregistrés dans des fichiers NumPy sur le serveur. Le résultat de cette comparaison est affiché, indiquant si l'étudiant est admis ou non. Cette approche assure une vérification d'identité rapide et précise, garantissant l'intégrité des examens.

En plus de cela, nous avons ajouté une interface dédiée aux administrateurs de l'application. Cette interface permet de mettre à jour les fichiers NumPy avec de nouveaux embeddings pour les nouveaux étudiants. Lorsqu'un administrateur charge un répertoire contenant des images de nouveaux étudiants, le système extrait ces images, calcule leurs embeddings et les enregistre dans les fichiers appropriés en fonction de la classe et du semestre sélectionnés. Cette mise à jour automatisée assure que notre base de données d'embeddings reste actuelle et précise.

- [Authentification et Sélection de la Classe](#)

```
def user_login(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)

            return redirect('select_class')
        else:
            return render(request, 'login.html', {'error_message': 'Nom d'utilisateur ou mot de passe incorrect.'})
    else:
        return render(request, 'login.html')
```

[View pour Authentification](#)

```
from .forms import SelectClassForm
from django.urls import reverse

def select_class(request):
    if request.method == 'POST':
        form = SelectClassForm(request.POST)
        if form.is_valid():
            selected_class = form.cleaned_data['selected_class']
            selected_semester = form.cleaned_data['selected_semester']

            url = reverse('facial_recognition', kwargs={'selected_class': selected_class, 'selected_semester': selected_semester})

            return redirect(url)
        else:
            form = SelectClassForm()
            return render(request, 'select_class.html', {'form': form})
```

[View pour la selection de class](#)

Les surveillants peuvent se connecter à l'application pour authentifier leur identité.

Ils peuvent sélectionner la classe ou le groupe d'étudiants qu'ils surveillent.

- **View pour la verification facial**

```
def facial_recognition(request, selected_class, selected_semester):
    valid_classes = ['MIP', 'BCG']
    if selected_class not in valid_classes:
        messages.error(request, "Invalid class selection.")
        return redirect('select_class')

    if request.method == 'POST':
        form = FacialRecognitionForm(request.POST, request.FILES)
        if form.is_valid():

            if selected_class == "MIP":
                if selected_semester == "S1":
                    embeddings_data = np.load(r'reconnaissance\fichier de projet\mip\s1_mip.npy', allow_pickle=True)
                elif selected_semester == "S2":
                    embeddings_data = np.load(r'reconnaissance\fichier de projet\mip\s2_mip.npy", allow_pickle=True)
            elif selected_class == "BCG":
                if selected_semester == "S1":
                    embeddings_data = np.load(r'reconnaissance\fichier de projet\bcg\s1_bcg.npy', allow_pickle=True)
                elif selected_semester == "S2":
                    embeddings_data = np.load(r'reconnaissance\fichier de projet\bcg\s2_bcg.npy', allow_pickle=True)

            embeddings = embeddings_data.item().get('embeddings')
            labels = embeddings_data.item().get("labels")

            |
            label = form.cleaned_data['label']
```

```

captured_image_data = request.POST.get('captured_image')
if captured_image_data:
    format, imgstr = captured_image_data.split(';base64,')
    ext = format.split('/')[1]
    image = ContentFile(base64.b64decode(imgstr), name='captured_image.' + ext)
    print("Captured image processed")
else:
    image = form.cleaned_data['image']
    print("Uploaded image processed")

print(type(image))

if np.isin(label, labels):
    face = extract_face(image, MTCNN(), (160, 160))
    similarity_score = compare_images_with_label(embeddings, labels, face, label)

    recognition_result = RecognitionResult(label=label, image=image, similarity_score=similarity_score)
    recognition_result.save()

    return redirect('result_page')
else:
    messages.error(request, "Label not found.")
else:
    messages.error(request, "Invalid form data.")
else:
    form = FacialRecognitionForm()

return render(request, 'facial_recognition.html', {'form': form})

```

La vue `facial_recognition` gère la vérification faciale des étudiants dans une application Django. Elle commence par vérifier la validité de la classe sélectionnée (soit "MIP" ou "BCG"). Si la classe n'est pas valide, un message d'erreur s'affiche et l'utilisateur est redirigé vers la page de sélection de classe. Si une requête POST est reçue, la vue crée une instance du formulaire `FacialRecognitionForm` avec les données soumises. Si le formulaire est valide, les embeddings et les labels sont chargés à partir des fichiers NumPy correspondants selon la classe et le semestre sélectionnés. Ensuite, l'image capturée est extraite de la requête POST et décodée. Si le label fourni par l'utilisateur existe dans les labels chargés, la fonction procède à l'extraction du visage à partir de l'image en utilisant le modèle MTCNN, puis compare ce visage avec les embeddings référencés pour calculer un score de similarité. Le résultat de la reconnaissance est alors sauvegardé dans le modèle `RecognitionResult`, et l'utilisateur est redirigé vers la page des résultats. Si le label n'est pas trouvé, ou si les données du formulaire sont invalides, un message d'erreur approprié est affiché.

○ View result

```

def result_page(request):
    latest_result = RecognitionResult.objects.latest('id') #
    if latest_result.similarity_score >= 0.5:
        admission_status = "L'étudiant est admis."
    else:
        admission_status = "L'étudiant n'est pas admis."
    return render(request, 'result_page.html', {'latest_result': latest_result, 'admission_status': admission_status})

```

La vue `result_page` dans cette application Django est conçue pour afficher le résultat le plus récent de la reconnaissance faciale. Elle commence par récupérer l'enregistrement le plus récent dans le modèle

RecognitionResult, basé sur l'identifiant. Ensuite, la vue vérifie le score de similarité associé à cet enregistrement. Si ce score est supérieur ou égal à 0,5, cela signifie que l'étudiant est reconnu comme admis, et la variable admission_status est définie sur "L'étudiant est admis.". Sinon, admission_status est définie sur "L'étudiant n'est pas admis.". Enfin, la vue rend la page result_page.html, en passant le dernier résultat et le statut d'admission comme contexte au template.

- [View importer repertoire](#)

```
def importer_repertoire(request):
    if request.method == 'POST':
        form = ImporterRepertoireForm(request.POST, request.FILES)
        if form.is_valid():
            selected_class = form.cleaned_data['selected_class']
            selected_semester = form.cleaned_data['selected_semester']
            repertoire = request.FILES['repertoire']
            try:
                # Créer un répertoire temporaire pour extraire le contenu du fichier ZIP
                temp_dir = r'C:\hello\reconnaissance\tester'
                os.makedirs(temp_dir, exist_ok=True)
                print(temp_dir)
                # Enregistrer le fichier ZIP sur le système de fichiers
                repertoire_path = os.path.join(temp_dir, repertoire.name)
                with open(repertoire_path, 'wb+') as destination:
                    for chunk in repertoire.chunks():
                        destination.write(chunk)
                print(repertoire_path)

                with zipfile.ZipFile(repertoire_path, 'r') as zip_ref:
                    zip_ref.extractall(temp_dir)

                os.remove(repertoire_path)
                index_point = repertoire.name.rfind('.')
                if index_point == -1:
                    ok = repertoire.name
                else:
                    ok = repertoire.name[:index_point]

                print(ok)
                temp_dir = f'C:\\hello\\reconnaissance\\tester\\{ok}'

                X, Y = load_faces(temp_dir)
```

```

messages.success(request, "Répertoire importé avec succès.")

X_emb, label = calculate_embeddings_with_labels(X, Y)

chemin = os.path.join(settings.MEDIA_ROOT, 'reconnaissance', 'new_data', 'nouveau.npy')

# Sauvegarder les embeddings et les Labels
save_embeddings_and_labels(X_emb, label, chemin)

if selected_class=="MIP":
    if selected_semester == "S1":
        path1=r'reconnaissance\fichier de projet\mip\s1_mip.npy'
        os.remove(path1)
        chemin1 = os.path.join(settings.MEDIA_ROOT, 'reconnaissance', 'fichier de projet', 'MIP.npy')
        save_embeddings_and_labels(X_emb, label, chemin1)

    elif selected_semester == "S2":
        path1=r'reconnaissance\fichier de projet\mip\s2_mip.npy'
        os.remove(path1)
        chemin1 = os.path.join(settings.MEDIA_ROOT, 'reconnaissance', 'fichier de projet', 'MIP.npy')
        save_embeddings_and_labels(X_emb, label, chemin1)

elif selected_class=="BCG":
    if selected_semester=="S1":
        path1=r'reconnaissance\fichier de projet\bcg\s1_bcg.npy'
        os.remove(path1)
        chemin1 = os.path.join(settings.MEDIA_ROOT, 'reconnaissance', 'fichier de projet', 'MIP.npy')
        save_embeddings_and_labels(X_emb, label, chemin1)

```

```

    elif selected_semester == "S2":
        path1=r'reconnaissance\fichier de projet\bcg\s2_bcg.npy'
        os.remove(path1)
        chemin1 = os.path.join(settings.MEDIA_ROOT, 'reconnaissance', 'fichier de projet', 'MIP.npy')
        save_embeddings_and_labels(X_emb, label, chemin1)

    return JsonResponse({'progress': 100})

except Exception as e:
    return JsonResponse({'error': str(e)}, status=500)

else:
    form = ImporterRepertoireForm()

    return render(request, 'importer_repertoire.html', {'form': form})

```

Cette vue `importer_repertoire` permet aux administrateurs de télécharger un fichier ZIP contenant les images des nouveaux étudiants. Le fichier est extrait, et les images sont traitées pour calculer les embeddings. Les embeddings et les étiquettes sont ensuite sauvegardés dans des fichiers spécifiques selon la classe et le semestre sélectionnés, remplaçant les anciens fichiers pour assurer que les données restent à jour.

4. les résultats des modèles

1.4. Réseaux Siamois:

Nous avons entraîné ce modèle sur notre ensemble de données et nous avons obtenu comme résultats :

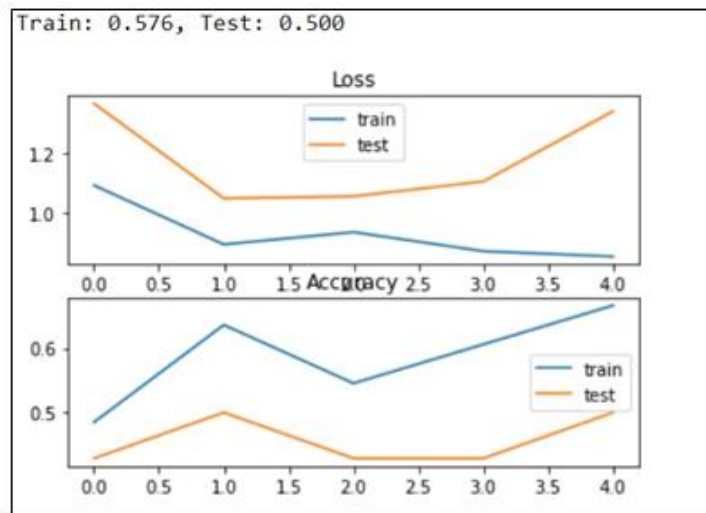


Figure 47 : Précision de CNN

La perte (loss) s'agit d'une somme des erreurs commises pour chaque exemple dans les ensembles d'apprentissage ou de validation donc pour SNN nous avons trouvé que pour l'apprentissage 57% de l'ensemble de données qui sont bien entraîné et pour le test ou bien la validation le modèle a pu savoir ou bien prédire juste 50% de l'ensemble de données

1.5. FaceNet :

Nous avons utilisé ce modèle pré-entraîné, et a eu les résultats suivants :

Modèle	Précision
FaceNet	98.96 %

1.6. Approche Proposée:

Nous nous sommes basées sur ce qui précède, et nous avons sélectionnés les méthodes avec les meilleures performances pour le système que nous proposons

- L'extraction des caractéristiques en utilisant le modèle de FACENET.
- Calcule de la distance entre les embeddings avec la distance euclidienne.

5. Interfaces Utilisateurs

Pour illustrer la mise en œuvre de notre projet, nous présentons ici les différentes interfaces utilisateurs ainsi que leur intégration dans notre application Django. Les interfaces sont définies à travers les vues, les modèles et les templates, permettant une interaction fluide et sécurisée avec les utilisateurs.

- **Interface pour la Page de Login**

Cette interface permet aux utilisateurs de se connecter à l'application en fournissant un nom d'utilisateur et un mot de passe.

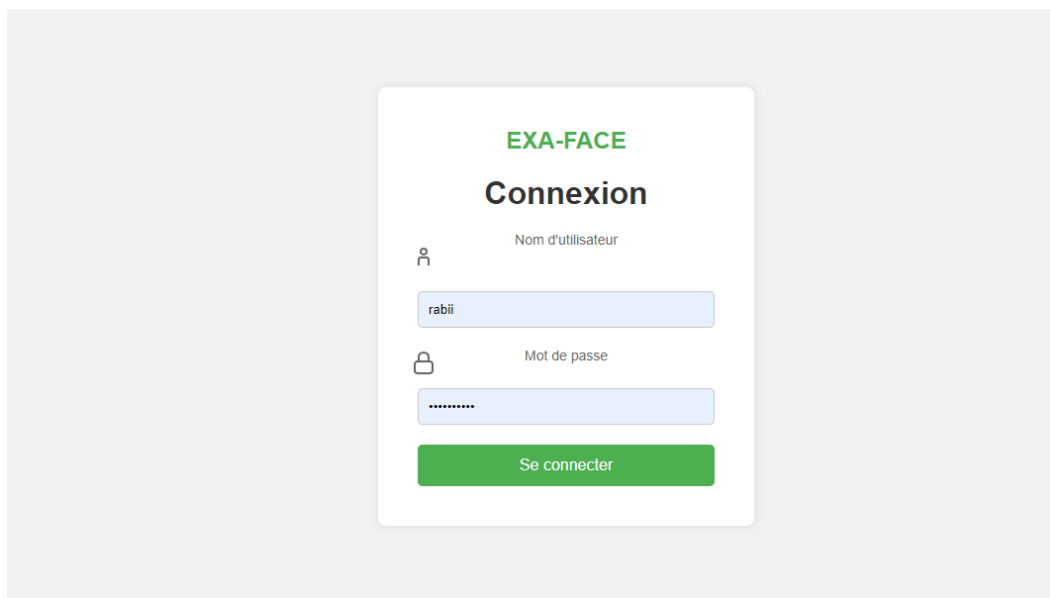
The image shows a login form for 'EXA-FACE'. The form is centered on a light gray background. It has a white background with rounded corners. At the top, it says 'EXA-FACE' in green and 'Connexion' in bold black. Below that, there are two input fields. The first is labeled 'Nom d'utilisateur' with a user icon and contains the text 'rabii'. The second is labeled 'Mot de passe' with a lock icon and contains masked characters '.....'. At the bottom of the form is a green button with the text 'Se connecter'.

Figure 11 : interface pour la page de login

- **Interface pour la Sélection de la Classe et du Semestre**

Cette interface permet aux utilisateurs de sélectionner la classe et le semestre pour lesquels ils souhaitent effectuer la reconnaissance faciale.

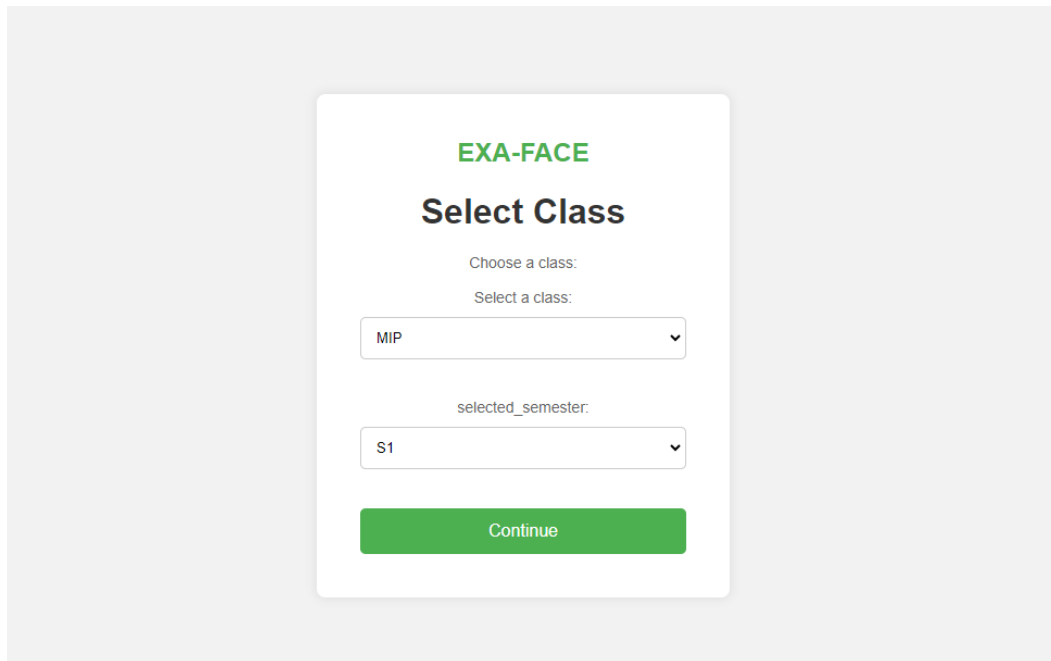


Figure 12 : interface pour la sélection de la classe et du semestre

- **Interface pour la Reconnaissance Faciale**

Cette interface permet aux utilisateurs de soumettre une image pour la reconnaissance faciale et vérifier leur identité.

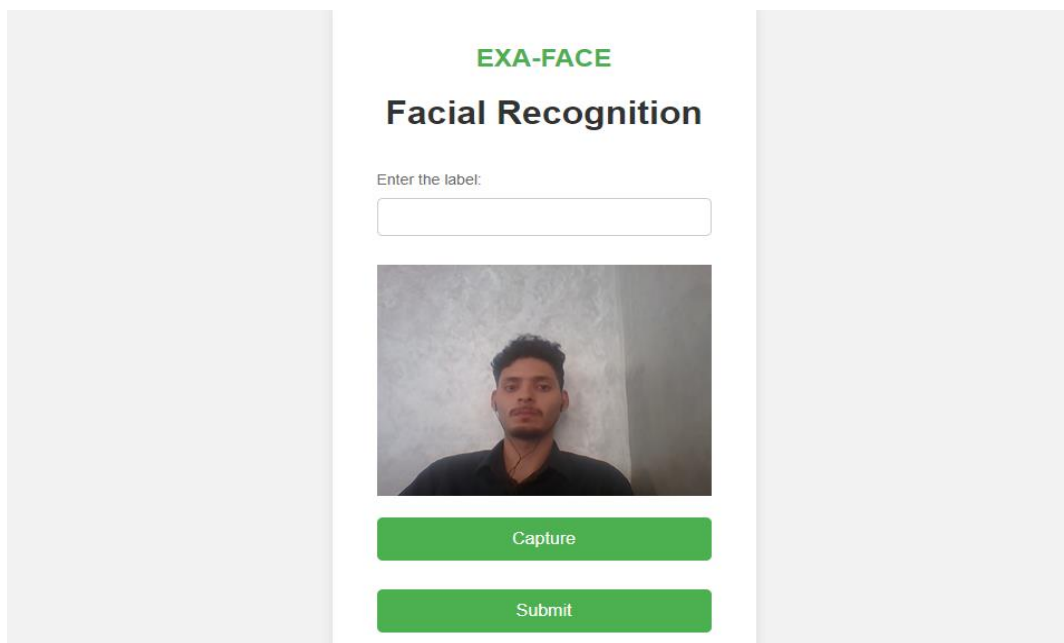


Figure 13 : Interface pour la Reconnaissance Faciale

- **Interface pour la Page de Résultat**

Cette interface affiche les résultats de la vérification d'identité, incluant le score de similarité et le statut d'admission de l'étudiant.



Figure 14 : Interface pour la Page de Résultat

- **Interface pour Importer un Répertoire d'Images**

Cette interface permet aux utilisateurs d'importer un répertoire d'images compressé (ZIP) pour entraîner le modèle de reconnaissance faciale avec de nouvelles données.

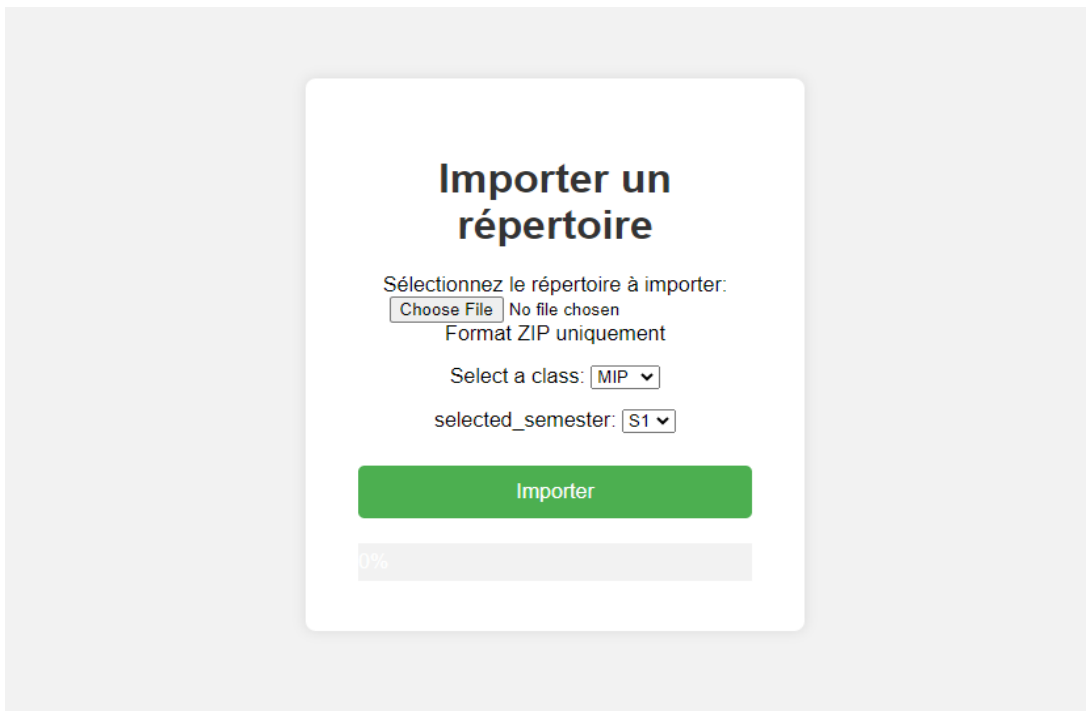


Figure 15 : interface pour Importer un Répertoire d'Images

Conclusion et Perspectives

Dans le cadre de notre projet de fin d'étude pour notre licence, nous avons développé un système de vérification d'identité des étudiants lors des examens en utilisant la reconnaissance faciale. Ce projet a été conçu pour répondre aux exigences définies dans le cahier des charges, nous permettant ainsi de découvrir de nouvelles architectures, d'implémenter des solutions innovantes et d'enrichir nos connaissances et notre expérience.

Ce projet s'inscrit dans le cadre d'une licence en informatique, option Génie Logiciel, au sein de l'UMI, Faculté des Sciences et Techniques d'Errachidia. Il nous a offert l'occasion d'approfondir les connaissances acquises durant notre formation. Nous avons implémenté les fonctionnalités essentielles permettant l'authentification des surveillants, la vérification faciale des étudiants, et la gestion des données faciales à travers une interface d'administration. Nous avons utilisé une architecture MVC en utilisant la programmation orientée objet avec Python, et nous avons fait en sorte que notre solution soit sécurisée et efficace.

Tout au long de ce travail, nous avons réalisé la création des différentes classes nécessaires pour gérer les utilisateurs, les images faciales et les vérifications. Nous avons intégré des bibliothèques comme TensorFlow et OpenCV pour assurer une reconnaissance faciale précise et rapide.

Cependant, notre projet, bien que complet, peut être amélioré. Compte tenu du temps et des ressources limitées dont nous disposons, nous nous sommes concentrés sur les fonctionnalités les plus importantes. Notre projet offre donc certaines perspectives d'amélioration et d'extension pour répondre aux besoins des utilisateurs de manière plus avancée.

Parmi les perspectives envisageables, nous pouvons mentionner :

- **Intégration de modules de machine learning** : L'utilisation de techniques de machine learning permettrait de proposer des fonctionnalités de vérification encore plus précises et d'identifier des comportements suspects en temps réel.
- **Utilisation de frameworks plus avancés** : Pour améliorer la qualité du code, la performance et la maintenabilité de l'application, il serait judicieux d'explorer des frameworks populaires tels que React pour le frontend et Django REST Framework pour l'API backend.
- **Sécurité renforcée** : Intégrer des mécanismes de chiffrement des données et de gestion des autorisations d'accès pour renforcer la sécurité de l'application.
- **Rapports et analyses** : Ajouter des fonctionnalités de génération de rapports et d'analyse des résultats de vérification pour offrir des insights utiles aux administrateurs.

En résumé, ce projet nous a permis de mettre en pratique nos compétences en programmation orientée objet et en développement web. Nous avons acquis une compréhension approfondie de l'architecture MVC et des fonctionnalités nécessaires pour un système de vérification d'identité. Cependant, nous sommes conscients des possibilités d'amélioration et des nouvelles technologies à explorer pour créer des solutions encore plus avancées et adaptées aux besoins du marché. Nous sommes enthousiastes à l'idée de continuer à apprendre et à développer de nouveaux projets en utilisant des technologies innovantes et des bonnes pratiques de développement.

Bibliographie

- (1): Pr. Fouad YAAKOUBI cours programmation web 2023/2024.
- (2) : Pr. Abdeslam JAKIMI cours UML 2023/2024.
- (3) : Pr. Jihane LAKHEOUIT cours Gestion de Projet 2023/2024.

Webographie

- [1] : **Recher, Arnaud Bodin et François.** *Deepmath : Mathématiques des réseaux de neurones.* 2020.
- [2] : **Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei.** *Mixed Pooling for Convolutional Neural Networks,*. 2016.
- [3] : **M. R. Gupta, N. P. Jacobson, E. K. Garcia,** OCR binarization and image pre-processing for searching historical documents, *Pattern Recognition*, 40, p. 389-397, 2007.
- [4] : **ICET2017, Antalya, Turkey 978-1-5386-1949-0/17/\$31.00 ©2017 IEEE,** Understanding of a Convolutional Neural Network
- [5] : **Brad Dayley,** [Sams Teach Yourself Django in 24 Hours Mar.](#)
- [6] : **Richard Zemel, Ruslan Salakhutdinov, Gregory Koch,** Department of Computer Science, University of Toronto. Toronto, Ontario, Canada.