DAKAR INSTITUTE OF TECHNOLOGY

Rapport de projet de fin d'année

Master 1 Intelligence Artificielle

Thème :

Mise en place d'un système de reconnaissance d'images pour l'accès dans une entreprise

Présenté par :

Mor Codou SECK

Lauriane MBAGDJE DORENAN

Date de soutenance : 15 Décembre 2023

Année académique: 2022-2023

Table des matières

Table des illustrations

INTRODUCTION

À l'ère où la sécurité et la surveillance sont devenues des impératifs, la nécessité de déterminer l'identité d'un individu de manière efficace et précise représente un défi majeur. Au quotidien, notre identité est soumise à une vérification, que ce soit lors d'un accès à notre lieu de travail, d'une utilisation de nos cartes bancaires, d'une connexion à des réseaux informatiques, ou dans d'autres situations aussi courantes.

Traditionnellement, deux méthodes principales sont utilisées pour l'identification individuelle. La première repose sur une clé connue uniquement par l'utilisateur, comme un mot de passe ou un code d'activation. La seconde est basée sur la possession d'un objet, tel qu'une pièce d'identité, une clé ou un badge. Cependant, ces approches présentent des inconvénients, allant de la gestion complexe de multiples mots de passe au risque de perte ou de vol d'objets d'identification physique.

Face à ces défis, l'exploitation des caractéristiques biométriques émerge comme une solution prometteuse. En effet, chaque individu possède des traits uniques tels que la voix, les empreintes digitales, les traits du visage, la forme de la main, la signature, voire l'ADN. Ces données biométriques offrent une voie vers une identification plus sûre et pratique, ouvrant ainsi la porte à des solutions novatrices.

C'est dans ce contexte que s'inscrit notre projet : la mise en place d'un système de reconnaissance d'images pour l'accès dans une entreprise. En mettant particulièrement l'accent sur la reconnaissance faciale, notre objectif est de mettre en place un générateur capable de créer dynamiquement des modèles de reconnaissance d'images utilisables par n'importe quelle entreprise. Ce générateur constituera une solution flexible, adaptative et évolutive, permettant une implémentation personnalisée dans divers environnements professionnels. Au fil des sections suivantes, nous plongerons dans les détails des différentes étapes clés de ce projet, mettant en lumière les modèles et les bibliothèques utilisés.

CHAPITRE I : Phase de création d'un modèle

I. Collecte de données (data collection)

Considérée comme le nouveau pétrole du XXIe siècle , la donnée est devenue l'un des actifs les plus précieux de notre époque. Elle est utilisée par les entreprises, les gouvernements et les organisations de tous types pour prendre des décisions, améliorer les produits et services, et comprendre le monde qui nous entoure. La collecte de données quant à elle est une étape cruciale pour tout projet d'intelligence artificielle (IA).

Dans notre contexte, la collecte de données (images), s'est révélée être un défi significatif en raison de la complexité liée à leur obtention. Ainsi, dans l'exemple que nous allons voir par la suite, nous avons opté pour une collection d'images disponible sur la plateforme Kaggle (lien à préciser). Ces images, représentant les visages des héros de l'univers Marvel, seront utilisées pour entraîner un modèle de reconnaissance d'images. Le choix des Avengers comme ensemble de données vise à exploiter la diversité des visages et des caractéristiques uniques de chaque personnage, renforçant ainsi la capacité du modèle à généraliser et à identifier efficacement une variété de visages.

De ce qui précède, la qualité et la représentativité des données sont des éléments cruciaux pour garantir la performance et la robustesse du modèle. De ce fait, un traitement minutieux des données collectées sera effectué, afin d' éliminer tout biais potentiel et d' assurer une expérience d'apprentissage optimale pour le système.

II. Prétraitement des données(data preprocessing)

La préparation des données, également appelée prétraitement ou preprocessing, est une étape importante dans le processus de développement d'un modèle, surtout lorsqu'il s'agit de données brutes telles que des images. Cette phase vise à nettoyer et à organiser les données en vue du traitement ultérieur. Pendant cette étape, une attention particulière est portée à la vérification minutieuse de ces données pour détecter d'éventuelles anomalies telles que les données de qualité médiocre, redondantes, incomplètes ou incorrectes afin de créer des données de haute qualité qui garantiront la performance du modèle.

Dans le contexte spécifique des images, le prétraitement consiste à apporter divers ajustements à ces images tels que les modifications de la taille, de l'orientation et de la couleur. L'objectif ultime du prétraitement est d'améliorer la qualité de l'image, permettant une analyse plus efficace. Il vise à éliminer les distorsions indésirables tout en améliorant des caractéristiques spécifiques essentielles à l'application en cours.

En ce qui concerne notre projet, différentes techniques ont été utilisés :

- Rescale (Normalisation) : cette technique normalise les valeurs des pixels des images en les divisant par 255. Cela ramène les valeurs des pixels à une plage entre 0 et 1, facilitant ainsi l'entraînement du modèle.
- Shear Range (Data augmentation) : C'est une transformation géométrique qui déplace les pixels d'une ligne par rapport à une autre. En définissant shear_range, on applique

un léger cisaillement aux images, introduisant de la variabilité dans les données d'entraînement.

- Zoom Range (Data augmentation) : cette technique permet d'appliquer un zoom aléatoire aux images. Elle augmente la variabilité des données en modifiant l'échelle des pixels de manière aléatoire.

- Horizontal Flip (Data augmentation) : Il effectue une symétrie par rapport à l'axe vertical. En activant cette option, certaines images peuvent être retournées horizontalement, ajoutant une autre forme de variabilité aux données.

Ces techniques sont mises en œuvre à l'aide de la classe ImageDataGenerator de Keras, qui permet de générer des lots d'images avec des transformations appliquées en temps réel. Ces transformations normalisent et augmentent la taille effective du jeu de données d'entraînement, améliorant ainsi la capacité du modèle à généraliser et à traiter des variations dans les données réelles.

Ci-dessous l'application de la préparation de données.

```python
def _prepare_data(self, train_data_path, validation_data_path = None):
    # Récupération des chemins des données ...
    current_dir = os.getcwd()
    train_dir = os.path.join(current_dir, train_data_path)
    val_dir = os.path.join(current_dir, validation_data_path)
    self.class_names = os.listdir(train_dir)
    # Préparation des générateurs ...
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
    )
    val_datagen = ImageDataGenerator(rescale=1./255)
    # Récupération des données de train et test et val ...
    batch_size = 120
    target_size = (220, 220)

    self.train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=target_size,
        batch_size=batch_size,
    )
    self.val_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=target_size,
        batch_size=batch_size,
    )
    print("data preparation complete ....")
```

Figure 1:Preparation des données

III. Création d'un modèle

Dans le cadre de notre projet, nous avons opté pour l'utilisation d'un modèle Réseaux de Neurones Convolutionnels (CNN), spécialement conçu pour le traitement des données en images. Les CNN se révèlent particulièrement efficaces dans la reconnaissance d'images et autres tâches de vision par ordinateur grâce à leur architecture exploitant des couches de convolution. Ces couches permettent au réseau d'apprendre des caractéristiques locales de l'image, telles que des bords, des textures et des motifs de manière hiérarchique.

Il existe ce pendant plusieurs modèles pré-entraînés pour la reconnaissance d'images à savoir : **ResNet, Inception, MobileNet, EfficientNet, ImageNet** etc.

Cependant, nous avons fait le choix de construire un modèle en suivant les étapes ci-après :

1. **Premières couches (Convolution et MaxPooling):**

- Couche convolutionnelle : **Conv2D** avec 16 filtres et un masque de taille (3, 3) pour extraire des caractéristiques de l'image.
- La couche **MaxPooling2D** réduit la dimension de l'image tout en préservant les caractéristiques importantes.

2. **Couches suivantes (Convolution et MaxPooling):** Deux autres blocs de convolution et de max pooling sont ajoutés pour augmenter progressivement la complexité et la richesse des caractéristiques apprises.
3. **Couche d'aplatissement (Flatten) :** Convertit la sortie des couches convolutionnelles en un vecteur plat, préparant ainsi les données pour la couche entièrement connectée.
4. **Couche entièrement connectée (Dense) :** Une couche dense avec 100 neurones .
5. **Couche de sortie :**
   - La couche de sortie a un nombre d'unités égal au nombre de classes .
   - Une fonction d'activation **softmax** est utilisée pour convertir les scores en probabilités.

En somme, notre modèle présente une architecture simple, potentiellement efficace pour des tâches de classification d'images moins complexes. Cependant, pour des tâches plus avancées, l'utilisation de modèles pré-entraînés comme cités précédemment pourrait être nécessaire.

Ci-après la création d'un modèle

```
def _create_model(self):
    output = len(self.class_names)
    model = Sequential()
    model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(220, 220, 3)))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(output, activation='softmax'))

    self.model = model

    print("model creation complete ....")
```

Figure 2: Création du modèle

IV. Entraînement et évaluation d'un modèle

1. Entraînement

Lors de l'entraînement d'un modèle, les choix des paramètres sont cruciaux pour garantir un apprentissage efficace. Dans notre cas, nous avons opté pour les paramètres suivants:

1. **Optimiseur (Adam):** L'optimiseur est responsable de la mise à jour des poids du réseau afin de minimiser la fonction de perte. Adam (Adaptive Moment Estimation) est un optimiseur populaire qui ajuste les taux d'apprentissage de chaque paramètre individuellement. Il combine les avantages de l'optimisation par descente de gradient stochastique (SGD) avec ceux de l'optimisation adaptative.

2. **Fonction de perte (Categorical Crossentropy) :** La perte (loss) est une mesure de l'écart entre les prédictions du modèle et les véritables étiquettes. Dans le cas d'une classification multi-classe (plus de deux classes), la perte catégorielle cross-entropy est couramment utilisée. Elle mesure la divergence entre les distributions de probabilité prédites par le modèle et les distributions réelles des classes.

3. **Métrique (Accuracy) :** La métrique accuracy est couramment utilisée pour évaluer les performances d'un modèle de classification. Elle mesure le pourcentage de prédictions correctes parmi toutes les prédictions. C'est un indicateur intuitif de la performance globale du modèle.

   Ci-dessous la partie entraînement

```python
def _train_model(self):
    # Compilation du modèle ...
    self.model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    # Entrainement du modèle ...
    epochs = 20
    history = self.model.fit(
        self.train_generator,
        epochs=epochs,
        validation_data=self.val_generator
    )

    # Visualisons les loss d'entrainement et de validation ...
    plt.figure(figsize=(14, 6))
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

    print("\nmodel training complete ....\n")
```

Figure 3: Entraînement du modèle

2. Evaluation

L'évaluation d'un modèle revêt une importance pour plusieurs raisons. Tout d'abord, elle fournit une mesure concrète de la performance réelle du modèle sur des données qu'il n'a pas rencontrées pendant l'entraînement, permettant ainsi de déterminer sa capacité à généraliser. De plus, l'évaluation permet de détecter des problèmes potentiels tels que le surapprentissage (overfitting) ou le sous-apprentissage (underfitting), offrant ainsi des indications précieuses pour améliorer la robustesse du modèle. Elle est également essentielle pour sélectionner le meilleur modèle parmi plusieurs candidats et optimiser les paramètres du modèle. L'évaluation joue un rôle clé dans la validation des objectifs spécifiques de la tâche, que ce soit la classification d'images, la détection d'objets ou d'autres tâches complexes.
L'évaluation d'un modèle est un élément fondamental pour assurer son utilité, sa fiabilité et sa pertinence dans des situations réelles.
Dans notre contexte, nous évaluons le modèle en nous basant sur la perte(loss) et la métrique (accuracy).

Ci-dessous la partie évaluation

```python
def evaluate(self, test_data_path):
    batch_size = 120
    target_size = (220, 220)

    # Récupération du chemin des données de test ...
    current_dir = os.getcwd()
    test_dir = os.path.join(current_dir, test_data_path)

    # Préparation du générateur ...
    test_datagen = ImageDataGenerator(rescale=1./255)

    # Récupération des données de test ...
    test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=target_size,
        batch_size=batch_size,
    )

    # Evaluation du modèle ...
    score = self.model.evaluate(test_generator, verbose=0)
    print(f'Test loss     : {score[0]:4.4f}')
    print(f'Test accuracy : {score[1]:4.4f}')

    print("\model evaluation complete ....")
```

Figure 4: Evaluation du modèle

3. Sauvegarde d'un modèle

La sauvegarde d'un modèle est nécessaire pour préserver les poids, les architectures, et les configurations apprises pendant l'entraînement. Cela permet de réutiliser le modèle ultérieurement sans avoir à le réentraîner, ce qui peut être coûteux en temps et en ressources.
Dans notre cas , la sauvegarde s'est faite de la manière suivante :

- Le chemin complet vers le modèle est décomposé pour extraire le répertoire et le nom du modèle.
- Les noms de classe utilisés pour l'entraînement sont sauvegardés dans un fichier texte. Chaque ligne du fichier contient l'indice de la classe et son nom.
- Le modèle lui-même est sauvegardé dans le chemin spécifié (**model_path**) en utilisant la méthode **save** de l'objet modèle.

La sauvegarde des noms de classe est utile lors de la récupération du modèle pour permettre une correspondance entre les indices prédits et les étiquettes de classe réelles lors des prédictions ultérieures.

Ci-après la partie sauvegarde

```python
def save(self, model_path):
    path = model_path.split('/')[0]
    model_name = model_path.split('/')[-1].split('.')[0]

    with open(f'{path}/labels_{model_name}.txt', 'w') as f:
        for index, label in enumerate(self.class_names):
            f.write(f'{index} {label}\n')
        f.close()

    self.model.save(model_path)
```

Figure 5: Sauvegarde du modèle

Chapitre II : Phase d’utilisation

I. Lecture ou chargement d'un modèle

La lecture ou chargement d'un modèle fait référence au processus de récupération des informations associées à un modèle sauvegardé, telles que l'architecture, les poids, et toute configuration. Dans notre cas, la lecture inclut la récupération des noms de classe à partir d'un fichier texte en plus du modèle.

Ci-dessous la partie chargement.

```python
def read(self, model_path):
    path = model_path.split('/')[0]
    model_name = model_path.split('/')[-1].split('.')[0]

    # load class names
    with open(f'{path}/labels_{model_name}.txt', 'r') as f:
        self.class_names = [a[:-1].split(' ')[1] for a in f.readlines()]
        f.close()

    # load specifical model
    self.model = load_model(model_path, compile=False)
    print("done!!!")
```

Figure 6: Chargement du modèle

II. Prédiction et utilisation

Tout modèle développé en IA à pour vocation d'être utilisé dans le monde réel ou, dans ce cas précis, faire des prédictions.

Pour faciliter l'exploitation du modèle développé, nous avons mis en place deux méthodes d'utilisations :

- A partir de la console : On fait appel à une fonction (open_on_console) à qui on fournit le modèle chargé et l'image qu'on veut prédire.
  Ainsi, elle affiche sur la console la prédiction (la personne prédite).

- A partir du navigateur : On fait appel à une fonction (open_on_web) à qui on fournit le modèle chargé ainsi qu'une image de fond. On lance avec streamlit.
  L'application sur le navigateur en affichant une interface permettant à l'utilisateur de charger l'image qu'il veut prédire.

  Ci-dessous la partie utilisation et prédiction.

```python
def open_on_console(model, image_path):
    class_name, conf_score = model.predict(image_path)
    print("## {}".format(class_name))
    print("### score: {}%".format(int(conf_score * 1000) / 10))


def open_on_web(model, bg_image_path):
    with open(bg_image_path, "rb") as f:
        img_data = f.read()

    b64_encoded = base64.b64encode(img_data).decode()
    style = f"""
        <style>
        .stApp {{
            background-image: url({b64_encoded});
            background-size: cover;
        }}
        </style>
    """
    st.markdown(style, unsafe_allow_html=True)

    # set title
    st.title('Face Recognition App')

    # set header
    st.header('Please upload an avenger face')

    # upload file
    file = st.file_uploader('', type=['jpeg', 'jpg', 'png'])
```

```python
    # upload file
    file = st.file_uploader('', type=['jpeg', 'jpg', 'png'])

    # display image
    if file is not None:
        image = Image.open(file).convert('RGB')
        st.image(image, use_column_width=True)

        # classify image
        class_name, conf_score = model.predict(file)

        # write classification
        st.write("## {}".format(class_name))
        st.write("### score: {}%".format(int(conf_score * 1000) / 10))
```

Figure 7:Prédiction

Ci-dessous l'interface de l'application avec streamlit

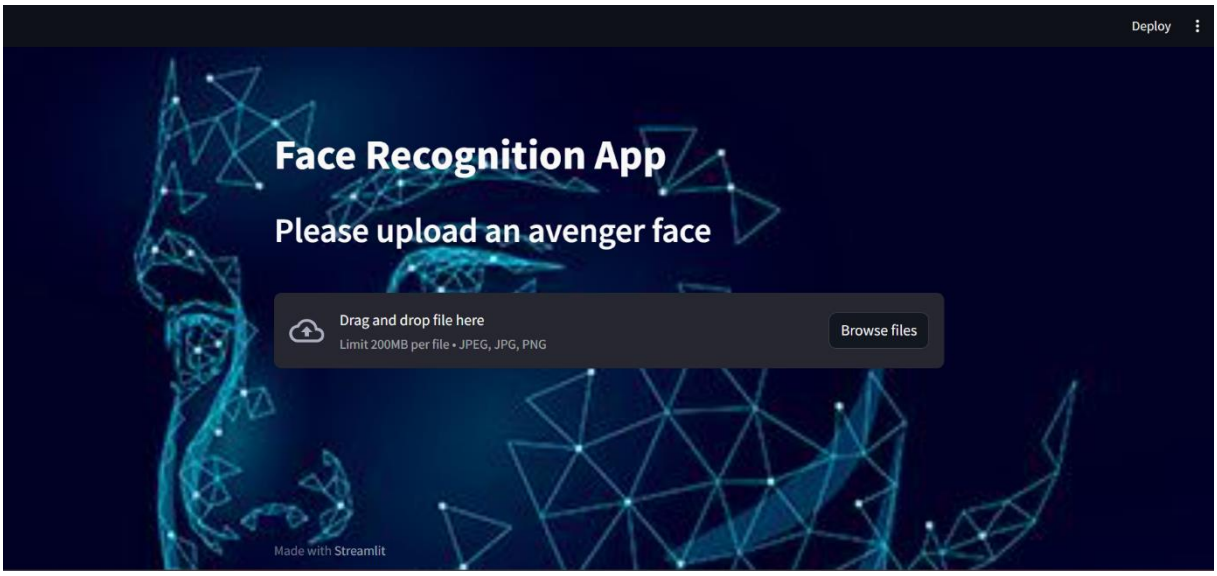

Figure 8: Interface streamlit

III. Perspectives

Nous avons développé un modèle qui fonctionne et donne une prédiction satisfaisante. Pour optimiser davantage ses performances, nous envisageons les axes d'amélioration suivants :

1. **Utilisation de Modèles Pré-Entraînés :** Nous utiliserons des modèles de réseaux neuronaux pré-entraînés, tels que ceux disponibles dans Keras Applications (VGG16, ResNet, etc.). Cela peut améliorer les performances de notre modèle en tirant parti des connaissances acquises par ces modèles sur de grandes bases de données.
2. **Keras Tuner :** Nous utiliserons Keras Tuner pour rechercher automatiquement les meilleures combinaisons d'hyperparamètres. Cela peut aider à optimiser la structure du réseau et les paramètres d'entraînement.

Conclusion

En définitive, notre projet de mise en place d'un système de reconnaissance d'images pour l'accès dans une entreprise a été une expérience enrichissante. Il nous a permis de consolider nos connaissances acquises en classe et de les appliquer de manière concrète. Malgré le défi majeur lié à l'accessibilité aux données, nous avons élaboré avec succès un générateur de modèles dynamiques. Cette solution flexible peut être adaptée à divers types de données liées à la reconnaissance d'images, offrant ainsi une solution polyvalente pour toute entité intéressée par ce domaine. Nous sommes confiants dans le potentiel d'expansion et d'application de notre approche dans d'autres contextes similaires.