Cas pratique

# Introduction

L’objectif est d’améliorer une application d’authentification qui détecte les employés d’une entreprise à partir d’un scan d’un de leurs yeux. Il faut à la fois améliorer les modèles de détection, et l'application qui les utilise.

# Table des matières

[Introduction 1](#_Toc158208714)

[Table des matières 1](#_Toc158208715)

[1. Partie modèle 2](#_Toc158208716)

[1.1 Données 2](#_Toc158208717)

[1.2 Gestion des données 2](#_Toc158208718)

[1.3 Architecture du notebook d’entraînement 3](#_Toc158208719)

[1.4 Modèle initial 4](#_Toc158208720)

[1.5 Test de modèles 4](#_Toc158208721)

[1.6 Modèle final et résultats 7](#_Toc158208722)

[2. Partie application 8](#_Toc158208723)

[2.1 Application initiale 8](#_Toc158208724)

[2.2 Nouvelle application 9](#_Toc158208725)

# 1. Partie modèle

## Données

Nous disposons de 5 photos de chaque œil des 45 employés de l’entreprise.

Pour les employés 8, 13 et 43, les dossiers des yeux gauches et droits étaient inversés.

## Gestion des données

Pour gérer les données à l'entraînement, j’ai recréé un dossier data avec, à la place des photos d’yeux, des dossiers train / val / test contenant les photos d’yeux (3, 1 et 1 car 5 photos par yeux par employés).

Cela permet d'importer toujours les mêmes images à l'entraînement.

Type de jeu de données

Côté œil

Images œil

Numéro employé

1

left

2

Nouveau dossier data

train

3

val

right

Image 1

...

test

Image 2

Figure 1 : Architecture des données

J’ai ensuite une fonction qui génère directement mes jeux d'entraînement, de validation et de test, en fonction du modèle que l’on souhaite entrainer : œil gauche, œil droit, différence œil gauche-droit.

## Architecture du notebook d’entraînement

Voici les différentes parties qui constituent mon notebook d’entraînement, elles sont écrites en "markdown" pour pouvoir replier-déplier les cellules de code :

**Initialisation** : Import des librairies

**Préparation des jeux de données** : - Recréation du dossier data

- Import des jeux d'entrainement, validation et test

- Vérification de la correspondance photo / label

**Data augmentation :** - Définition du type de data augmentation + fonction de visualisation du rendu

**Prétraitement des données** : - Application du "LabelEncoder" sur les labels

- Application du "OneHotEncoder" sur les labels pour le modèle tensorflow

- Deuxième vérification de la correspondance X / y

**Modèles** : - Couche d'entrée

**-** Couches de data augmentation

**-** Couches de prétraitement (redimensionnement et standardisation des valeurs)

- Modèle

- Couche de sortie

**Entraînement**

**Prédiction :** - Inversion du label encoder sur les prédictions pour avoir les vrais résultats à la matrice de confusion

**Métriques, matrice de confusion**

**Sauvegarde du modèle, du label encoder et des métriques obtenues**

## 1.4 Modèle initial

Le modèle utilisé par l’ancien développeur pour reconnaitre les yeux est VGG16.

Il s’agit d’un modèle CNN (Convolutionnal Neural Network), pré-entrainé sur une large quantité d’images.

Une couche « flatten » (aplatissement) et une couche « dense » correspondant à la sortie ont été ajoutées.

Les données sont redimensionnées et standardisées lors de leur import dans l’IDE (Integrated Development Environment).

## 1.5 Test de modèles

Les tests sont effectués pour le modèle qui détecte l’employé avec l'œil gauche.

J’ai implémenté le redimensionnement et la normalisation dans le modèle, pour qu’il soit plus robuste à l’export.

Voici les résultats obtenus en jouant sur le "batch size" et les "epochs" avec le modèle initial :

Tableau 1 : Résultats

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Batch\_size = 10  Epochs = 5 | Batch\_size = 30  Epochs = 5 | Batch\_size = 60  Epochs = 5 | Batch\_size = 10  Epochs = 50 | Batch\_size = 30  Epochs = 50 | Batch\_size = 60  Epochs = 50 |
| Loss | 7.5\*10^-4 | 0.04 | 0.06 | 5.1\*10^-4 | 4.4\*10^-5 | 9.3\*10^-4 |
| Accuracy | 1 | 0.99 | 0.98 | 1 | 1 | 1 |
| Val\_loss | 0.83 | 0.86 | 0.46 | 1.09 | 0.29 | 0.6 |
| Val\_accuracy | 0.76 | 0.71 | 0.84 | 0.8 | 0.93 | 0.84 |

La note la plus parlante est la précision sur le jeu de validation (val\_accuracy), elle représente le taux d'images bien prédites.

On voit que les résultats sont meilleurs lorsque l'’on augmente la durée d’entraînement (epochs) et que le batch\_size est aux alentours de 30, jusqu’à 0.93 d’accuracy.

Notons que 0.93 n'apparaît pas tout le temps, les notes peuvent aller d'environ 0.8 à 0.93. Le facteur aléatoire est énorme donc l'entraînement est mauvais.

Pour les entrainements rapides (5 epochs), on a une meilleure val\_accuracy avec un batch\_size de 10. Lorsque l'on augmente la durée de l'entraînement, on a une meilleur accuracy avec un batch\_size de 30. Cela démontre d'autant plus que l'initialisation aléatoire des poids dans le modèle joue un rôle fort, probablement car les données ne sont pas assez variées pour apporter un effet de généralisation.

J’ai donc implémenté de la data augmentation dans le modèle, afin d’augmenter la capacité du modèle à généraliser sur les données. Il n’y avait que 3 images par label dans le jeu d’entraînement.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Figure 2: Paramètres utilisés pour la data augmentation

Les valeurs sont multipliées par un facteur 2Pi, donc cela donne environ 0.1 de rotation et de zoom. J’ai minimisé l’augmentation pour ne pas trop dénaturer les images.

J'ai créé une fonction pour visualiser les images issues de la data augmentation.

Voici les résultats en ajustant d’autres paramètres, combinés à la data augmentation.

Tableau 2 : Résultats avec la data augmentation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Batch\_size = 30  Epochs = 50 | Batch\_size = 45  Epochs = 50 | Defriezing des 2 dernières couches de VGG16 | Ajout d’une couche Dense 300 neurones après vgg16 (activation = relu) |
| Loss | 9.10^-4 | 7.1.10^-4 | 0.015 | 0.07 |
| Accuracy | 1 | 1 | 0.99 | 1 |
| Val\_loss | 0.25 | 0.22 | 0.67 | 0.31 |
| Val\_accuracy | 0.93 | 0.93 | 0.8 | 0.96 |

Et les résultats avec une autre data augmentation et un autre modèle (non pré-entrainé).

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 3 : Data augmentation 2

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 4 : Autre modèle testé

Tableau 3 : Résultats 3

|  |  |  |
| --- | --- | --- |
|  | Data augmentation 2 sur le modèle avec la couche de 300 neurones | Modèle non pré-entrainé + data augmentation 1 |
| Val\_accuracy | 0.93 | 0.4 |

Avec la data augmentation, les résultats des modèles sont plus stables (plusieurs lancements) et meilleurs. On obtient jusqu'à 0.96 de val\_accuracy avec l'ajout d'une couche de neurones intermédiaire.

Une image contenant texte, motif, point, tissu

Description générée automatiquementJ’ai réalisé une matrice de confusion pour analyser où le modèle faisait ses erreurs :

Figure 5 : Matrice de confusion

Le test 2 a été confondu avec le sujet 12, les deux yeux se ressemblent.

Le sujet 43 a été confondu avec le sujet 25.

Cela m'a permis de remarquer que j’avais mal replacé les dossiers d'yeux gauches et droits pour le sujet 43.

Après réentrainement, le modèle tient plus souvent les 0.96 de val\_accuracy.

## 1.6 Modèle final et résultats

J’ai gardé le modèle avec le meilleur score : la première data augmentation et la couche dense intermédiaire.

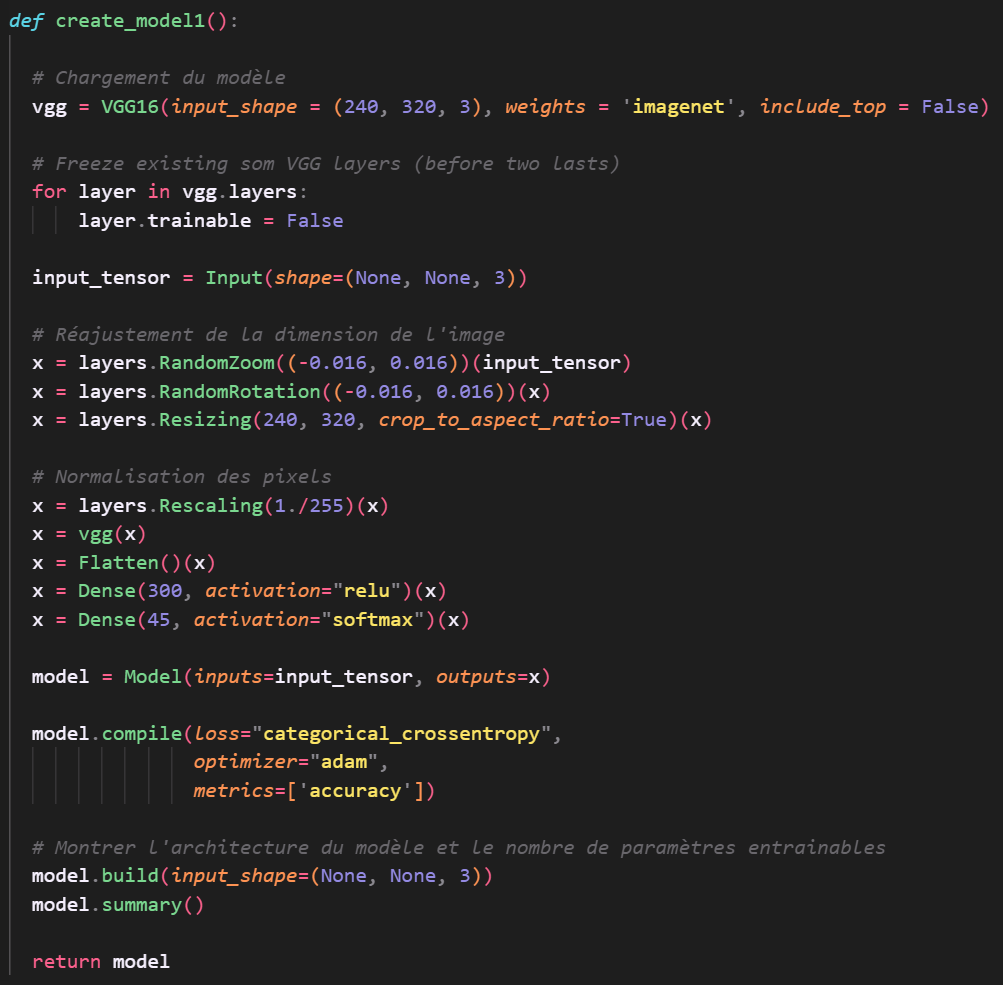


Figure 6 : Modèle final

Voici les résultats d'accuracy sur le jeu de validation, et de f1 score sur le jeu de test pour les trois modèles.

Je n'ai relevé que la val accuracy et le f1 score du test, car j'ai relevé les métriques à la main sur un fichier texte lors de la sauvegarde des modèles, et c'était selon moi les métriques les plus parlantes.

Tableau 4 Performances des modèles

|  |  |  |  |
| --- | --- | --- | --- |
|  | Oeil gauche | Oeil droit | Différence gauche-droit |
| Val accuracy | 0.96 | 0.82 | 1 |
| F1 score jeu de test | 0.97 | 0.94 | 1 |

Le modèle semble moins bien fonctionner sur l'oeil droit, ça vient probablement du manque de diversité dans les images d'entraînement.

On a donc réussi à augmenter les performances du modèle initial (qui étaient de 0.71 voir 0.93 avec plus d'epochs et sur coup de chance).

# 2. Partie application

Cette partie correspond à la compétence C14 :

Améliorer l’application d’intelligence artificielle en développant une évolution fonctionnelle pour répondre à un besoin exprimé par un client ou un utilisateur

## 2.1 Application initiale

On dispose d’un fichier app.py qui forme une application "tkinter", ce fichier est composé d’une classe qui contient l’application.

Cette classe contient un bouton pour charger une image et un bouton pour prédire l'image. De plus, elle contient une fonction chargement modèle, chargement image, prédiction image.

L’application ne se lance pas car le modèle à charger n’est pas reconnu.

Quand on force le lancement :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 7 : Design initial de l'application

Améliorations à envisager :

- Charger modèles, label encoders, informations employés

- Bouton de choix d'une image, associée à une fonction de prédiction

- La fonction de prédiction comprend : - le preprocessing de l'image

- la prédiction du modèle de différenciation d'oeil

- la prédiction d'un des modèles d'identification d'employé selon le côté de l'oeil

- transformation du résultat avec le label encoder pour avoir le résultat réel

- Placement de l'image, du chemin de l'image et des identifiants de l'employé prédit

## 2.2 Nouvelle application

L'import de deux de mes trois modèles a échoué. Le problème venait du fait que tous les éléments n'étaient pas définis dans la fonction du modèle (data augmentation, nombre de sorties, taille de redimensionnement image). Par ailleurs, le fait de définir le modèle dans une fonction a permis une augmentation considérable de la rapidité d'entraînement.

Une image contenant texte, capture d’écran

Description générée automatiquementVoici l'application finale :

Figure 8 : Application au lancement

Une image contenant texte, yeux, capture d’écran, organe

Description générée automatiquement

Figure 9 : Application après le choix d'une image