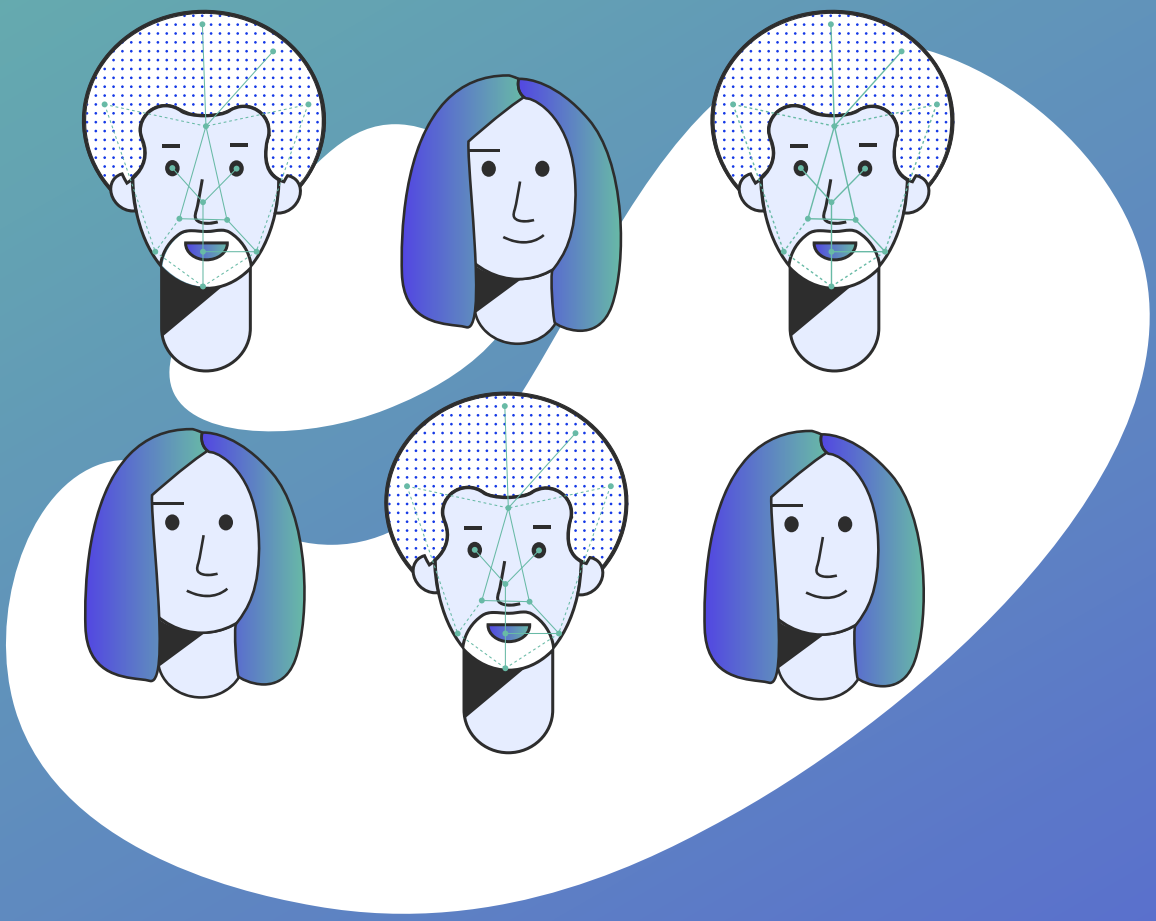
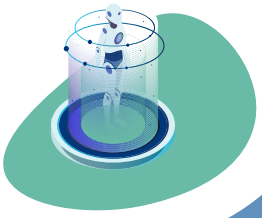
From Zero to Hero #6



Détectez des visages dans une vidéo grâce au deep metric learning

cellenza

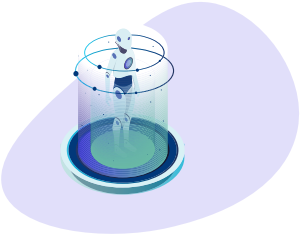
SOMMAIRE



PARTIE 1

UN PEU DE THÉORIE : QUELQUES RAPPELS SUR **LA COMPUTER VISION**

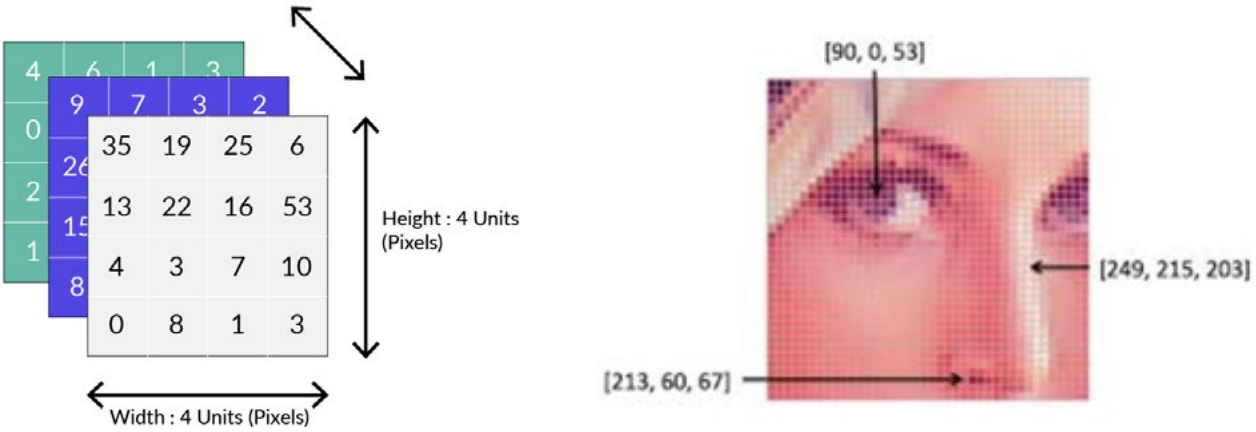
Pour comprendre le lab qui va suivre, nous avons besoin de nous remémorer quelques bases. Tout d'abord, comprendre … de quoi se compose une image (1.) et comment sont utilisés les réseaux de neurones dans le traitement d'images (2.). Nous verrons qu'il existe aujourd'hui des réseaux pré-formés qui facilitent grandement les projets de computer vision (3.), puis nous finirons en abordant une méthode que nous utiliserons dans ce lab : le deep metric learning (4.).



Mais au fait, qu'est-ce qu'une image ?

Avant de rentrer dans le vif du sujet, rappelons de quoi se compose le sujet principal de notre travail : l'image.

Vous pouvez voir l'image comme une grande matrice multidimensionnelle, avec une largeur (nombre de colonnes), une hauteur (nombre de lignes), et une profondeur (nombre de canaux dans l'image). Une image standard RGB a une profondeur de 3 (Red, Green, Blue).



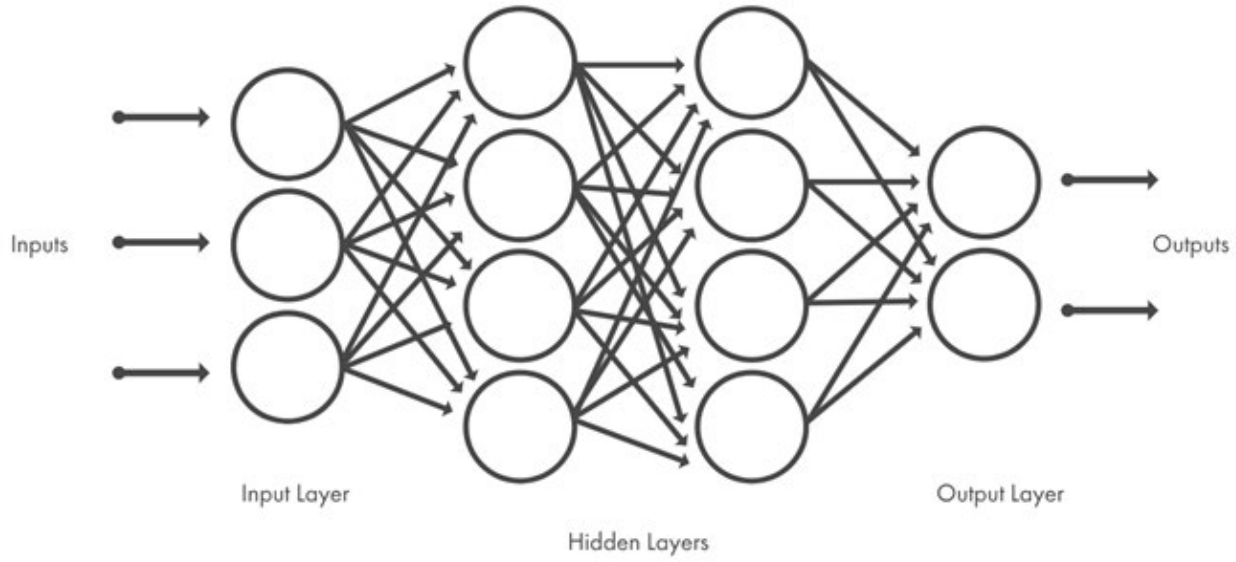
Source : deeplearningessentials

Comment sont utilisés les réseaux neuronaux dans la vision par ordinateur ?

Vous avez sans doute tous utilisé des applications ou logiciels faisant appel à de la vision par ordinateur (computer vision) et si vous travaillez dans ce domaine, vous allez forcément entendre parler des réseaux de neurones à convolution (CNN ou ConvNet), qui est l'une des méthodes les plus répandues dans le traitement d'image.

Au premier abord, le terme « convolution » peut apparaître complexe, mais en réalité il est fortement possible que vous l'ayez déjà utilisé sans le savoir : si vous avez utilisé Photoshop pour rendre une image plus nette par exemple.

Le CNN, comme les autres réseaux de neurones, comprend une couche d'entrée, une couche de sortie et plusieurs couches cachées entre les deux :



Source : mathworks

Nous pouvons trouver dans un **CNN** :

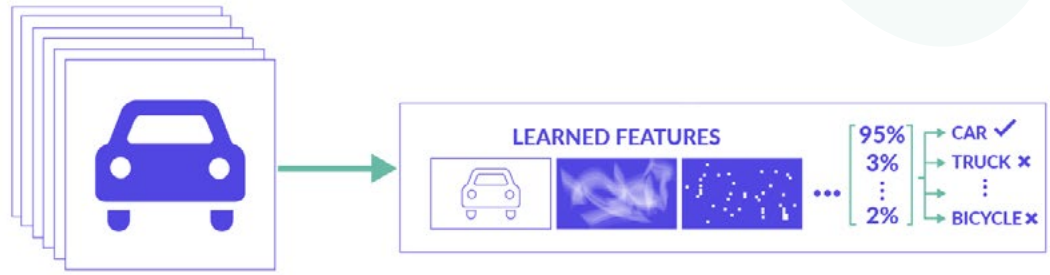
Une couche de convolution : pour faire simple, une convolution est le produit de deux matrices (élément par élément, pas un produit scalaire) suivie d'une somme, comme ceci.
Nous vous laissons vous amuser avec cet excellent outil de visualisation de kernel Setosa.io et si vous voulez connaître en détail comment cela fonctionne : Mandar Deshpande Convolutional Neural Network, 2018.

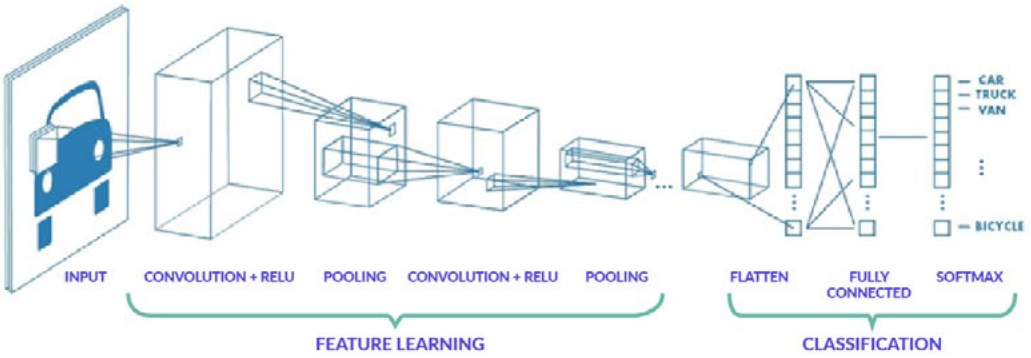
Une couche ReLU (*Rectified linear unit* ou couche d'activation) : qui permet un entraînement plus rapide en mappant les valeurs négatives à zéro et en maintenant des valeurs positives. Seules les caractéristiques (*features*) « activées » sont reportées dans la couche de neurones suivante ;

Une couche Pooling : qui va simplifier la sortie en effectuant un échantillonnage, réduisant le nombre de paramètres que le réseau a besoin d'apprendre.

Une dernière couche, SoftMax : qui va nous permettre de classifier (étiqueter) l'image.

Les opérations sont répétées sur plusieurs couches cachées, parfois plusieurs centaines, chaque couche apprenant à identifier des caractéristiques dans l'image.





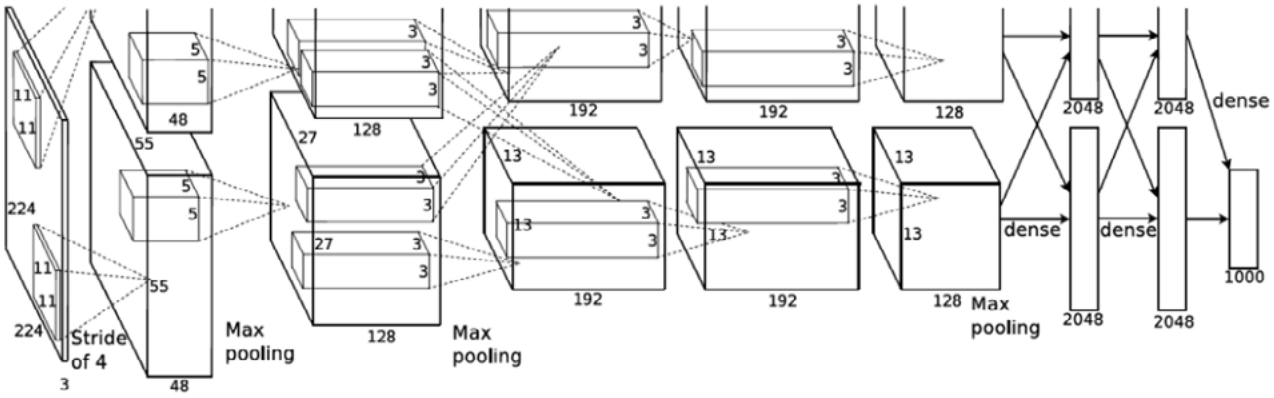
Source : mathworks

Cependant, cette phase peut prendre un temps considérable. Pour résoudre ce problème, nous ferons appel au *transfer learning*.

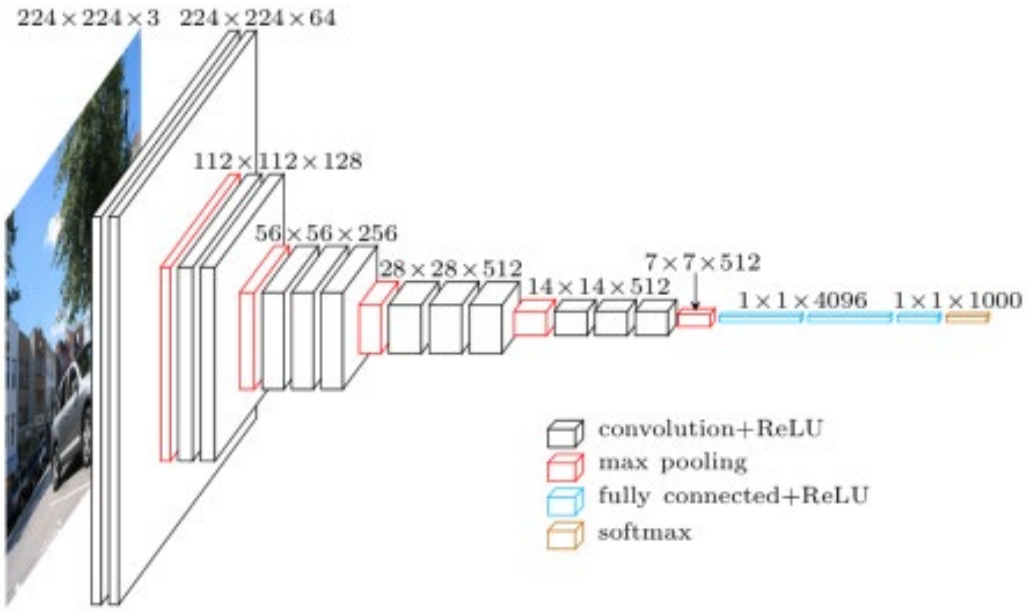
Pour aller plus loin : Dhanoop Karunakaran, Simple Image classification using deep learning, 2018.

Un gain de temps avec l'utilisation de réseaux de neurones pré-entrainés

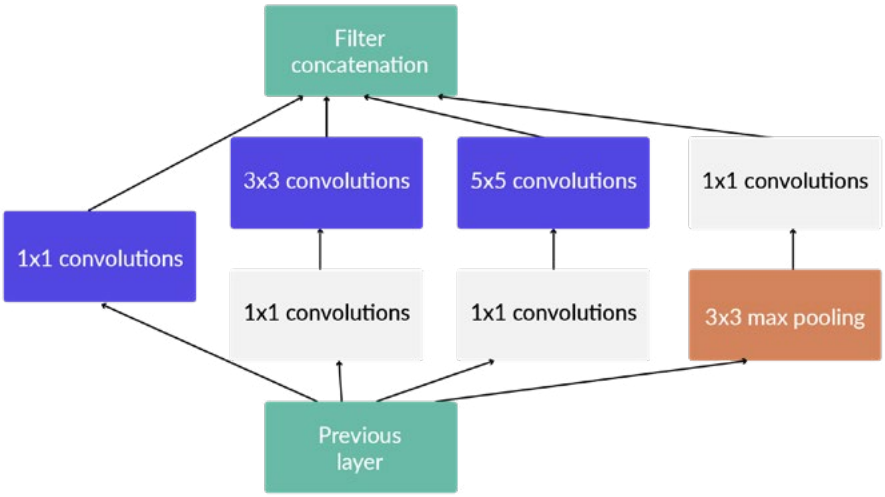
L'entraînement d'un réseau de neurones peut prendre énormément de temps et nécessiter énormément d'images. Ce problème peut être réglé en faisant appel à ce que l'on appelle le transfer learning ou réseau pré-entraîné. Il existe aujourd'hui plusieurs modèles pré-entrainés efficaces pour la computer vision :



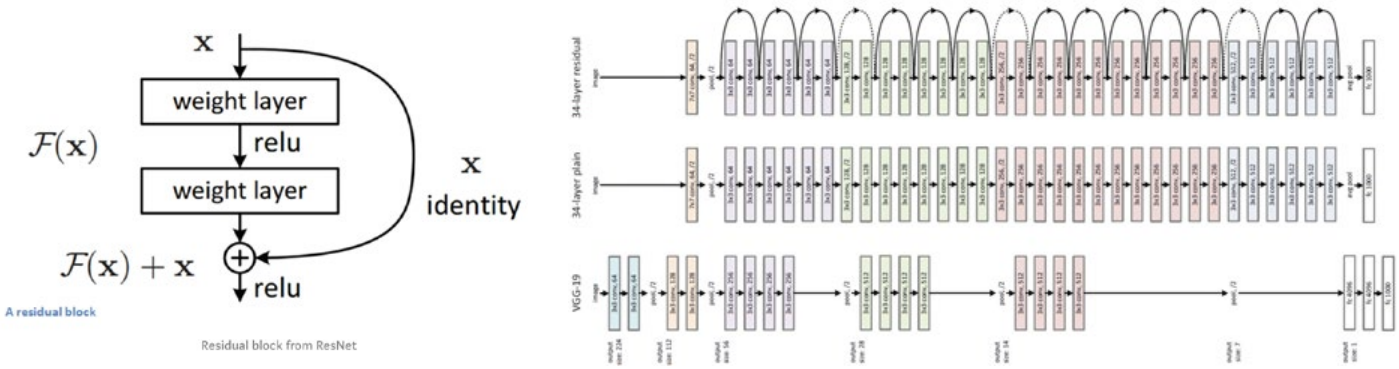
AlexNet, créé en 2012, par Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, de l'Université de Toronto



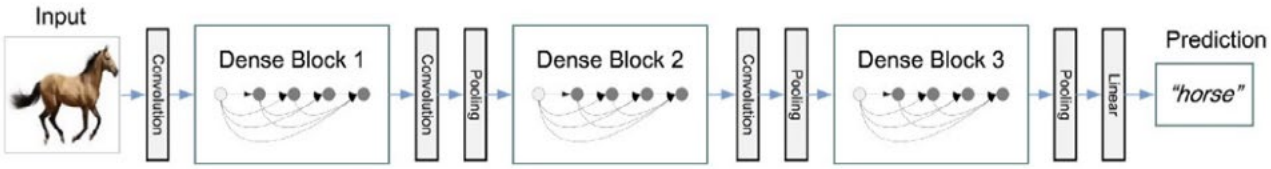
VGGNet, 2014, par Karen Simonyan & Andrew Zisserman, de l'Université d'Oxford



GoogLENet, 2015



ResNet, Microsoft, 2015

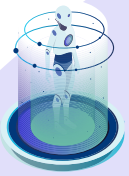


DenseNet visualization

DenseNet, 2018

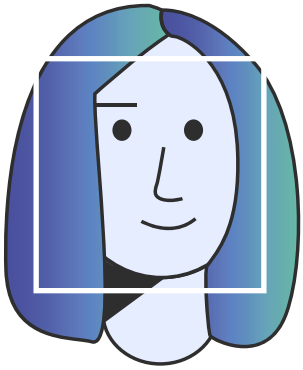
Pour aller plus loin : George Seif, Deep Learning for Image Recognition : why it's challenging, where we've been, and what'snext, 2018.

Nous avons encore une dernière « brique » à vous expliquer avant de pouvoir commencer notre lab : Le *deep metric learning*.



Une reconnaissance faciale de haute précision grâce au deep metric learning

Nous venons de voir que dans le deep learning en custom vision, nous formons généralement un réseau pour :
1 : lui donner une image en entrée,
2 : sortir une classification ou une étiquette pour cette image.



Cependant, il s'avère que des mesures qui nous semblent évidentes pour nous les humains (comme la couleur des cheveux, des yeux, etc) n'ont pas vraiment de sens pour un ordinateur qui analyse les pixels individuellement dans une image. Des chercheurs ont donc choisi de laisser l'ordinateur comprendre lui-même les mesures essentielles (déterminer les parties d'un visage les plus importantes) pour avoir les résultats les plus précis.

C'est ce qu'on appelle le *deep metric learning.*

Cette méthode fait appel à un CNN mais au lieu de former le réseau à reconnaitre des images des visages, nous allons l'entraîner pour générer un vecteur de mesures (caractéristiques) pour chaque visage.

Pour le réseau de face recognition *dlib*, le vecteur de caractéristiques en sortie est une liste de 128 nombres réels qui est utilisé pour quantifier le visage. Pour reconnaître un visage, le réseau est entrainé avec 3 images de visage :

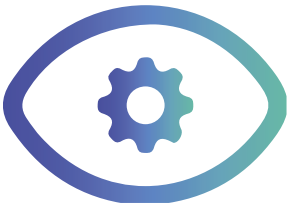
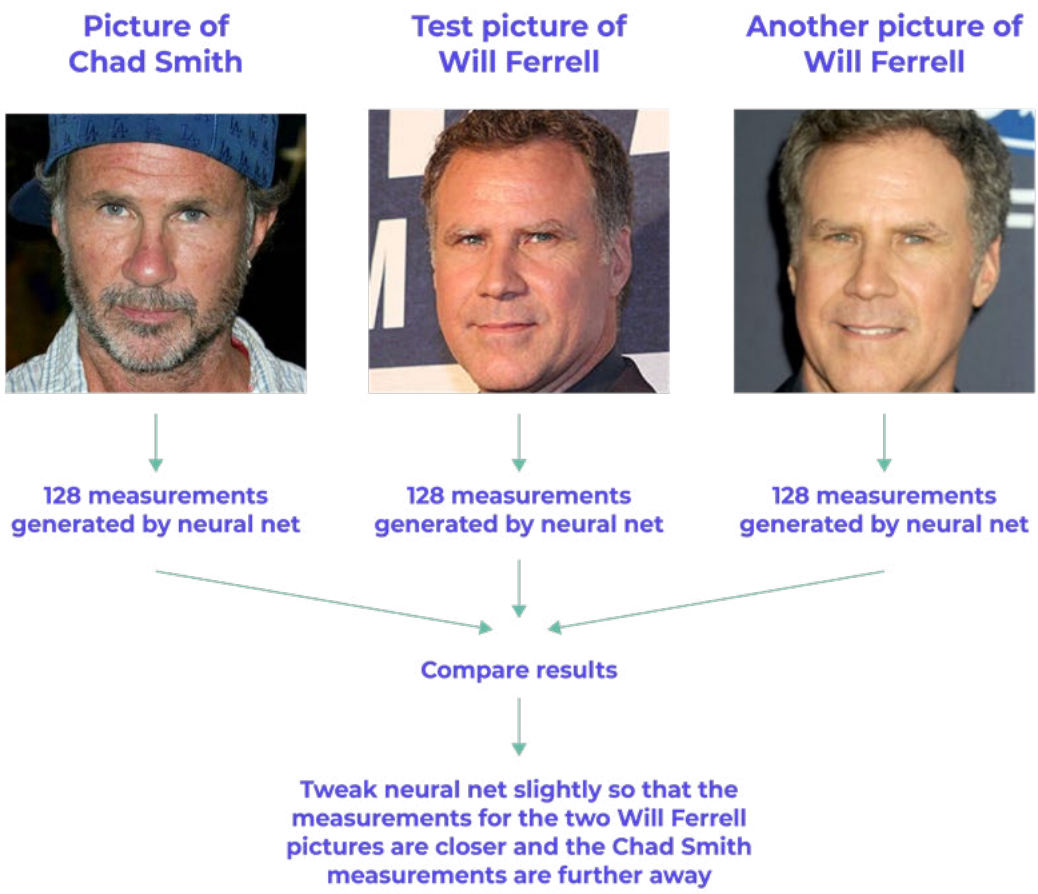


Image N°1 - 1 image de visage d'entraînement d'une personne
Image N°2 - 1 autre image de la même personne
Image N°3 - 1 dernière image d'une personne totalement différente

Ensuite l'algorithme examinera les mesures que le réseau de neurones génère pour chacune des 3 images et peaufine légèrement le réseau en modifiant les poids des neurones pour s'assurer que les images N°1 et N°2 soient légèrement plus proches et que N°2 et N°3 soient légèrement plus éloignées.



Source : https://medium.com/

Le réseau répète cette étape des millions de fois et pour des millions d'images de milliers de personnes différentes. Au lieu d'avoir une image avec des données brutes compliquées, nous aurons une version réduite de ces données dans une liste de valeurs générées par l'ordinateur.

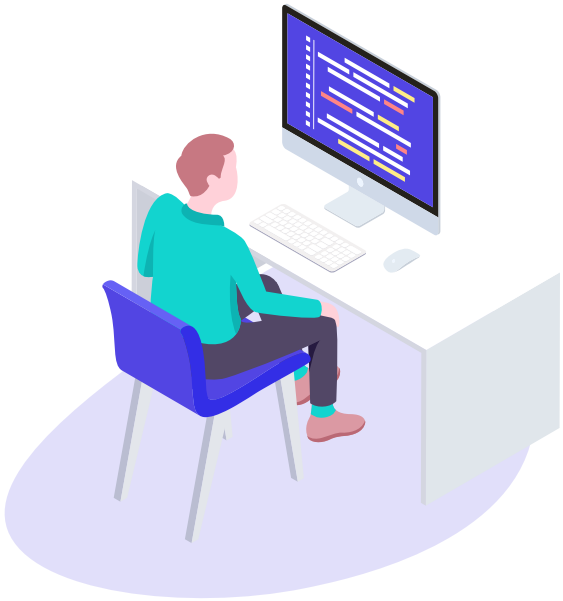
Pour aller plus loin :

Florian Schroff, Dmitry Kalenichenko, James Philbin, FaceNet:
A Unified Embedding for Face Recognition and Clustering, 2015
Davis King, High Quality Face Recognition with Deep Metric Learning, 2017
Adam Geitgey (auteur du module visage\recognition que nous utiliserons dans le lab), Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning, 2016

Nous utiliserons toutes ces notions dans notre lab. A vos machines !

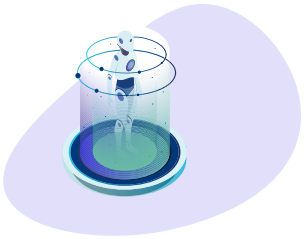
PARTIE 2

LE LAB :
LA RECONNAISSANCE DE VISAGES DANS UNE VIDÉO



A la fin de ce lab, vous serez en mesure de prendre n'importe quelle vidéo et de reconnaître automatiquement les visages sur celle-ci.

Commençons par créer notre environnement de développement !



Création d'un environnement virtuel dédié

Création d'un environnement virtuel Python

Nous vous recommandons vivement de créer un environnement virtuel pour isoler vos projets.

Note : avant de commencer ce lab, assurez-vous d'avoir :
• Python 3.6 disponible sur votre machine.
• Cmake : Disponible ici

Ouvrez votre terminal et créez votre environnement virtuel avec les bibliothèques nécessaires contenues dans le fichier *requirements.txt* :

Pour MacOS :

```
$ pip install virtualenv
$ cd user/projet/face-recognition-opencv
$ python3.6 -m venv fzth-opencv
$ source fzth-opencv/bin/activate
$ pip3.6 install -r requirements.txt
```

Pour Windows :

```
> pip install virtualenv
> python -m venv fzth-opencv
> fzth-opencv/Scripts/activate
> pip3.6 install -r requirements.txt
```

Installez la bibliothèque OpenCV :

```
$ pip install opencv-contrib-python
```

Pour faire de la reconnaissance faciale, nous devons installer trois bibliothèques spécifiques :

1 - dlib, maintenue par Davis King et qui contient notre implémentation de deep metric learning :

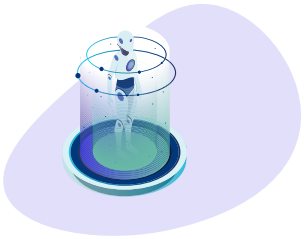
```
$ pip install dlib
```

2 - face_recognition, créée par Adam Geitgey, qui est une couche supplémentaire ajoutée à dlib.

```
$ pip install face_recognition
```

3 - imutils, créée par Adrian Rosebrock, qui contient un ensemble de fonctions utiles pour notre lab.

```
$ pip install imutils
```



La structure du projet

Notre projet a 4 niveaux principaux :

- **data/ :** contient les images des 3 personnes à reconnaître dans la vidéo, chaque personne ayant un dossier dédié.

- **fzth-opencv/ :** est l'environnement virtuel que nous venons de créer à l'aide du fichier *requirements.txt* avec toutes les bibliothèques dont nous aurons besoin pour notre lab.

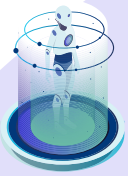
- **input/ :** contient les vidéos que nous allons utiliser pour notre reconnaissance faciale. Vous pouvez déjà y voir l'extrait d'un atelier de la Build Microsoft 2019 dirigé par Scott Hanselman (Partner Program Manager - Microsoft), et partagé avec Eduardo Laureano (Principal PM Manager – Microsoft) et Jeff Hollan (Product Manager – Microsoft).

- **output/ :** contiendra les vidéos de reconnaissance faciale traitées. Vous y voyez déjà l'extrait de la Build Microsoft après traitement.

Nous avons également 3 fichiers importants :

encodingFaces.py : génère les vecteurs – à 128 valeurs - à partir de notre jeu de données (dossier data/).
encodings.pickle : contient le résultat du script *encodingFaces.py*.
recognizingFaces.py : reconnaît les visages dans un fichier vidéo contenu dans le dossier input/

Note : vous pouvez remplacer la vidéo du dossier input par la vidéo de votre choix. Il vous faudra vous constituer un jeu de données avec des images des personnes présentes dans votre vidéo, comme cela est actuellement le cas dans le dossier data (1 personne = 1 dossier).



Première partie : encodage des visages avec Opencv

Comme vous l'avez compris, la première étape consiste à encoder les photos de visages en vecteurs de 128 valeurs.

Pour rappel, le réseau que nous utilisons dans ce lab pour créer ces encodages à 128 valeurs (embeddings) est un réseau pré-entrainé par David King sur un jeu de données d'environ 3 millions d'images. Autrement, il vous faudrait recueillir beaucoup plus d'images pour ce lab et plusieurs jours, voire semaines (... mois ?) pour arriver à de bons résultats en affinant les poids des neurones.

Pour cette première étape, détaillons le fichier **encodingFaces.py :**

Étape 1 - Importation des bibliothèques

```
1  # Importation des bibliothèques
2  import cv2
3  from imutils import paths
4  import face_recognition
5  import argparse
6  import pickle
7  import csv
8  import os
```

Étape 2 - Définissons nos arguments de ligne de commande

La bibliothèque argparse est utilisée pour analyser les arguments de la ligne de commande et fournir des informations complémentaires au script Python que vous exécutez sans quitter votre terminal.

```
9   # Construction des arguments
10  ap = argparse.ArgumentParser()
11  ap.add_argument(«-i», «--dataset», required=True,
12          help=»chemin vers le dossier contenant les images»)
13  ap.add_argument(«-e», «--encodings», required=True,
14          help=»chemin pour sauvegarder les visages encodés»)
15  ap.add_argument(«-d», «--detection-method», type=str, default=»cnn»,
16          help=»modèle de détection de visage à utiliser: `hog` ou `cnn`»)
17  args = vars(ap.parse_args())
```

dataset : est le chemin d'accès vers notre dataset.
encodings : est le chemin vers le fichier où seront sauvegardés nos encodages de visages.
detection-method : permet de choisir la méthode de détection du visage (« cnn » (plus rapide mais moins précis) ou « hog » (moins rapide mais plus précis)).

Étape 3 - Le coeur du script : l'encodage des images

Puis, décrivons les chemins d'accès aux fichiers de notre jeu de données et créons une liste de tous les chemins d'images qui y sont contenus :

```
19  # Saisir les chemins des images de notre dataset
20  print(«Calcul du nombre de visages...»)
21  img_paths = list(paths.list_images(args[«dataset»]))
```

Ensuite, nous allons initialiser deux listes avant notre boucle if : **known_encodings** (contenant les encodages de visage) et **known_names** (contenant les noms correspondants pour chaque personne dans le dataset) (**lignes 23 à 25**).

La boucle effectuera un cycle pour chaque chemin d'image de notre jeu de données - donc 15 dans notre cas (**ligne 27**). Puis nous extrayons le nom de la personne pour chaque *image_path*. Faites attention à ce que vos sous-répertoires soient nommés comme il convient : avec le nom de la bonne personne (**ligne 30**).

Nous allons ensuite charger l'image «img» en passant par le chemin *image_path* avec cv2.imread (**ligne 33**). Nous devons également faire attention à l'ordre des canaux de couleurs des images et effectuer un pré-traitement : OpenCV utilise des images en BGR et dlib attend du RGB (**ligne 34**). Enfin, localisons les visages (**ligne 37 et 38**) et calculons les encodages pour chaque image (**ligne 40 à 45**).

```
23  # Initialisation de la liste des encodages et des noms connus
24  known_encodings = []
25  known_names = []
26  # Boucler sur le chemin des images
27  for (i, img_path) in enumerate(img_paths):
28          # Extraire le nom des personnes
29          print(«Traitement de l'image {}/{}».format(i + 1, len(img_paths)))
30          name = img_path.split(os.path.sep)[-2]
31          # Charger l'image en entrée puis la convertir
32          # de RGB d'OpenCV en RGB dlib
33          img = cv2.imread(img_path)
34          rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
35          # Détecter les coordonnées (x, y) des boxes
36          # des visages sur l'image d'entrée
37          boxes = face_recognition.face_locations(rgb,
38                  model=args[«detection_method»])
39          # Calculer l'encodage du visage
40          encodings = face_recognition.face_encodings(rgb, boxes)
41          # Boucler sur les encodages ; Ajout de chaque encodage + nom
42          # à notre liste de nom et encodage connus
43          for encoding in encodings:
44                  known_encodings.append(encoding)
45                  known_names.append(name)
```

Étape 4 : La sauvegarde des visages encodés

Évidemment, nous allons devoir sauvegarder les résultats de ce script (encodages d'images) pour l'utiliser dans notre deuxième script qui va nous permettre la reconnaissance de visage :

```
47  # Sauvegarde des visages encodés et de leur nom au format pickle
48  print(«Sauvegarde des visages encodés...»)
49  data = {«encodings»: known_encodings, «names»: known_names}
50  # au format pickle:
51  file = open(args[«encodings»], «wb»)
52  file.write(pickle.dumps(data))
53  file.close()
```

Vous pouvez également visualiser les vecteurs générés :

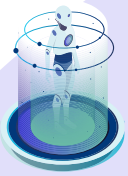
```
58  # au format txt:
59  data_txt = {«names»: known_names, «encodings»: known_encodings}
60
61  with open('encodings.txt','w') as f:
62          writer=csv.writer(f)
63          for k,v in data_txt.items():
64                  writer.writerow([k,v])
65
66  print(«Fichier 'encodings.txt' créé !»)
```

```
1   names,»['eduardo_laureano', 'jeff_hollan', 'scott_hanselman']»
2
3   encodings,»[array([-1.33473024e-01,  6.57220185e-02,  4.64127548e-02,  4.99860533e-02,
4         -1.01431236e-01, -4.13460471e-02, -1.24022020e-02, -4.00019512e-02,
5          2.24883765e-01,  3.72619517e-02,  1.87017843e-01,  6.05197772e-02,
6         -2.04267740e-01, -1.42571867e-01, -1.04539432e-01,  6.32182062e-02,
7         -9.36007500e-02, -1.41603604e-01, -3.05291060e-02, -7.41134062e-02,
8          8.64236429e-02,  4.18087393e-02,  5.87723739e-02,  6.42339885e-02,
9         -1.72579318e-01, -4.18645084e-01, -8.23170245e-02, -1.62345842e-01,
```

Voilà ! Maintenant vous pouvez exécuter le script avec la commande suivante :

```
$ python encodingFaces.py.py --dataset data --encodings encodings.pickle
```

Et passer à la dernière partie pour la reconnaissance de visages ...



Seconde partie : la reconnaissance de visages dans une vidéo

Grâce à nos encodages de visages, nous allons pouvoir reconnaître les visages dans une vidéo.

Étape 1 : Importation des bibliothèques

Dans un fichier *recognizingFaces.py.py*, nous allons commencer comme d'habitude par l'importation de bibliothèques que nous connaissons déjà.

```
# Importation des bibliothèques nécessaires
import argparse
import face_recognition
import imutils
import pickle
import time
import cv2
```

Étape 2 : Définition des arguments

Comme dans la première partie, nous définissons nos arguments de ligne de commande.

Étape 3 - Le coeur du script : la reconnaissance des visages

Nous allons charger les encodages de visages que nous avons créé tout à l'heure et indiquer le chemin de la vidéo à traiter :

```
1   # Charger les noms et encodages connus
2   print(«Chargement des encodages de visages...»)
3   data = pickle.loads(open(args[«encodings»], «rb»).read())
4
5   # Initialiser le pointeur sur le fichier vidéo
6   print(«Pré-traitement de la vidéo...»)
7   stream = cv2.VideoCapture(args[«input»])
8   writer = None
```

Maintenant, nous allons pouvoir commencer à traiter notre vidéo en bouclant sur les frames de celle-ci.
Comme pour le script précédant, nous allons convertir l'image et détecter les visages (**lignes 39 à 50**).

Puis (**lignes 51 à 74**), nous allons parcourir chacun des encodages et nous essayons de trouver des correspondances avec les visages de la vidéo.

```
32  # Boucler sur les images du flux vidéo
33  while True:
34          # Saisine de la prochaine image vidéo
35          (grabbed, frame) = stream.read()
36          # Arrêt lorsque nous avons atteint la fin de la vidéo
37          if not grabbed:
38                  break
39          # Conversion de l'image d'entrée de BGR en RGB
40          # et redimensionnement à 300px
41          rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
42          rgb = imutils.resize(frame, width=400)
43          r = frame.shape[1] / float(rgb.shape[1])
44          # Détection des coordonnées (x, y) des boxes
45          # de chaque visage en entrée,
46          # Et calcul de l'encodage de chaque visage
47          boxes = face_recognition.face_locations(rgb,
48                  model=args[«detection_method»])
49          encodings = face_recognition.face_encodings(rgb, boxes)
50          names = []
```

```
# Boucler sur chaque visage de la vidéo
# avec les visages de notre dataset (visages connus)
for encoding in encodings:
        # essayer de faire correspondre chaque visage détecté
        # dans l'image d'entrée à nos encodages connus
                matches = face_recognition.compare_faces(data[«encodings»],
                        encoding)
                name = «Unknown»
                # vérifier si nous avons trouvé une correspondance
                if True in matches:
                        # trouver les index de tous les visages correspondants
                        # puis initialiser un dictionnaire pour compter le nombre
                        # total de correspondances de chaque visage
                        matchedIdxs = [i for (i, b) in enumerate(matches) if b]
                        counts = {}
                        # boucler sur les index paires
                        # et compter chaque visage reconnu
                        for i in matchedIdxs:
                                name = data[«names»][i]
                                counts[name] = counts.get(name, 0) + 1
                        # déterminer le visage reconnu avec le plus grand nombre de votes
                        name = max(counts, key=counts.get)
                # mise à jour de la liste des noms
                names.append(name)
```

Ensuite, nous passons en boucle sur les visages reconnus et nous allons tracer un cadre autour du visage et afficher le nom de la personne :

```
# boucler sur les visages reconnus
        for ((top, right, bottom, left), name) in zip(boxes, names):
                # redimensionner les coordonnées du visage
                top = int(top * r)
                right = int(right * r)
                bottom = int(bottom * r)
                left = int(left * r)
                # Ecrire le nom du visage prédit sur l'image
                cv2.rectangle(frame, (left, top), (right, bottom),
                        (0, 255, 0), 2)
                y = top - 15 if top - 15 > 15 else top + 15
                cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
                        0.75, (0, 255, 0), 2)
```

Nous allons ensuite nous occuper de l'enregistrement de la vidéo traitée :

```
90  # Initialisons le graveur avec le chemin de fichier de sortie
91  # fourni dans les arguments de la ligne de commande
92          if writer is None and args[«output»] is not None:
93                  fourcc = cv2.VideoWriter_fourcc(*»MJPG»)
94                  writer = cv2.VideoWriter(args[«output»], fourcc, 24,
95                          (frame.shape[1], frame.shape[0]), True)
96          # si l'enregistreur est None, écrire la frame avec le visage reconnu
97          if writer is not None:
98                  writer.write(frame)
```

```
107 stream.release()
108
109 if writer is not None:
110         writer.release()
```

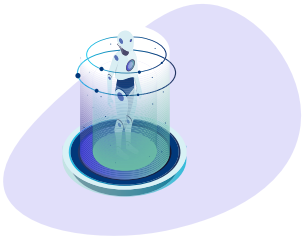
Nous initialisons VideoWriter_fourcc, FourCC étant un code de 4 caractères : dans notre cas, nous utiliserons le code « MJPG ». Nous passons ensuite cet objet fourcc dans VideoWriter avec le chemin du fichier de sortie, le nombre de frames par seconde ciblées, et la dimension des frames (**lignes 90 à 95**). Si le writer existe, nous pouvons enregistrer la frame (**lignes 96 à 98**).

Nous sommes prêts ! Exécutons notre script avec la commande :
$ python recognizingFaces.py --encodings encodings.pickle --input input/build_microsoft.mp4

Félicitations, vous avez construit votre premier projet de reconnaissance de visages !

Dans ce lab, vous avez réussi à construire un script de reconnaissance de visages avec OpenCV, Python et du deep learning. Nous avons également utilisé la bibliothèque dlib de David King, le module face_recognition d'Adam Geitgey et la bibliothèque imutils de Adrian Rosebrock qui facilitent la reconnaissance faciale.

N'hésitez pas à vous amuser avec ce script en prenant vos propres vidéos !



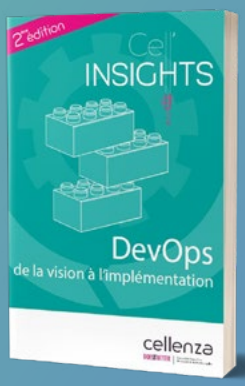
L’auteure

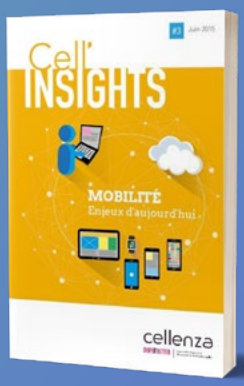
Nathalie Fouet
Consultante Junior IA

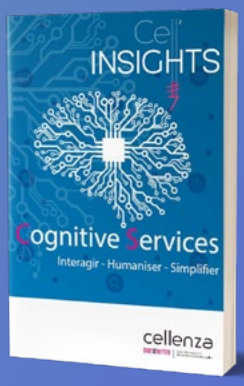
J’ai la particularité d’avoir un parcours de juriste et de développeuse en Data / IA. Je souhaite apporter des éléments de réponses à tous ceux qui s’intéressent aux technologies qui nous entourent et qui s’installent dans nos sociétés jusqu’à parfois s’immiscer dans notre vie privée.

Développeu.r.se.s ou esprits curieux, j’espère pouvoir vous éclairer sur ces sujets tant sous l’angle technique que juridique.

NOS PUBLICATIONS







Retrouvez aussi l'ensemble des articles techniques rédigés par nos Cellenzans :

blog.cellenza.com

cellenza

156 Boulevard Haussmann - 75008 PARIS
Tél : +33 (0)1 45 63 14 29 - info@cellenza.com

Rejoignez-nous sur