

PROJET FIN D'ETUDE
PHOTOPHORE

**Adrien BAUDE
Raphael PRESBERG**

**Master 2
Systèmes d'Informations & Data**

Encadrant : M. Cédric Coussinet

Table des matières

Remerciements	3
Introduction	4
I. Spécifications Techniques	5
1. Cahier des charges	5
2. Spécifications techniques	5
II. Première phase du projet	7
1. Le logiciel	7
a) L'interface graphique	7
b) L'étiquetage manuel des photos.....	9
2. Open CV.....	10
a) Les Descripteurs.....	10
b) Matching entre descripteurs.....	13
c) Pourcentage de ressemblance	15
d) Les limites d'Open CV	17
III. Deuxième phase du projet	20
1. Machine learning	20
a) Définition & Principes de base	21
b) Principes du Deep Learning	23
c) Le Deep Learning, techniquement parlant	24
d) Introduction Keras	28
2. Etiquetage automatique des images	29
a) Un modèle à entraîner	29
b) Un modèle pré-entraîné	31
IV. Analyse	34
1. Cahier des charges	34
2. Comparaison des performances	36
3. Difficultés & Enseignements.....	37
Conclusion.....	38
Bibliographie	39

Remerciements

Nous aimerais tout d'abord remercier notre encadrant de projet, Monsieur Cédric Coussinet, qui nous a proposé son projet, fait confiance pour sa réalisation, et assisté durant toute la période de recherche et de réalisation.

Nous souhaiterions de plus remercier Monsieur Abdulhaim Dandoush, directeur du laboratoire Systèmes d'Information & Data pour son envie et sa générosité dans sa nouvelle fonction.

Nous ajoutons une petite pensée pour Monsieur Maxime Robin, professeur de Client/Serveur - Microservice - Data Visualisation, pour ses conseils précieux sur les différentes technologies de machine learning.

Enfin, nous souhaitons remercier l'institution ESME Sudria, ainsi que tous les professeurs nous ayant suivi durant notre cursus, pour la formation d'ingénieur fournie. Nous espérons être messagers de cette école de l'autre côté de l'Atlantique.

Introduction

Après un mois de vacances avec son groupe de 12 amis, un utilisateur souhaite créer un album photos sur son ordinateur ; pour cela, il demande à tous ses amis de lui envoyer toutes les photos prises pendant les vacances sur la conversation Whatsapp.

Il se retrouve ainsi avec des centaines de photos mal prises, identiques, et non triées.

Il recherche donc une solution fournissant une aide au tri de sa bibliothèque d'images.

Un logiciel permettant à l'utilisateur de trier les photos en un seul clic, suivant leur ressemblance ou les préférences personnelles de l'utilisateur, de supprimer les images non désirées ou mal prises, et enfin, de facilement étiqueter les photos.

Photophore est une bibliothèque d'images répondant parfaitement à ce besoin. Effectivement, grâce à Photophore, un utilisateur pourra trier ses photos comme il le souhaite, soit en fonction de sa préférence, soit en fonction de la ressemblance entre les photos. De plus, le logiciel permet d'étiqueter automatiquement et manuellement les photos.

En quelques clics, on peut donc simplement et rapidement organiser une bibliothèque d'images afin de la partager avec ses amis.

Nous avons choisi ce projet en tant que projet de fin d'étude à l'ESME Sudria, et travaillé sur le sujet pendant trois mois.

Depuis le début, nous travaillons ensemble avec Monsieur Coussinet, notre encadrant, en utilisant la méthode de gestion de projet Agile: *"Une méthode agile est une approche itérative et incrémentale, qui est menée dans un esprit collaboratif avec juste ce qu'il faut de formalisme. Elle génère un produit de qualité tout en prenant en compte l'évolution des besoins des clients et de l'évolution de la mise en place de la solution "*

De même, pour répondre de la meilleure des façons aux exigences de l'équipe encadrante qui nous a fait confiance sur ce projet, nous avons suivi le process suivant :

- Elaboration des besoins utilisateurs
- Recherche de solutions
- Phase de développement
- Intégration de la solution sur l'IHM

I. Spécifications Techniques

Avant de commencer tout projet, il est nécessaire de rédiger une ligne directrice, des consignes à respecter fournies par l'encadrant, un cahier des charges en somme. A cela nous ajoutons un panel de règles techniques et de technologies à utiliser pour le bon déroulement du projet.

1. Cahier des charges

Après avoir choisi ce projet et en avoir discuté plusieurs avec Monsieur Coussinet, encadrant du projet, nous avons rédiger ce cahier des charges.

Celui-ci est amovible et modifiable en raison de notre départ de l'école en janvier.

Effectivement, trois mois est une courte durée pour un projet de fin d'étude, il était donc impératif de pouvoir modifier le projet au cours du semestre en fonction de notre avancé.

Ainsi nous avons, avec confirmation de notre encadrant, rédigé le cahier des charges suivant :

- Développer une interface graphique
- Télécharger des photos sur la bibliothèque d'images
- Supprimer des photos
- Déplacer les photos suivant les préférences utilisateur
- Trier les photos en fonction leur ressemblance
- Etiqueter manuellement les images
- Etiqueter automatiquement les images

Au début du projet, nous espérions répondre à chacune de ces problématiques. De ce fait, ce cahier des charges a été estimé réalisable sur trois mois.

2. Spécifications techniques

La première exigence technique à laquelle nous avons répondu concerne la séparation du projet en deux phases : la première phase qui porte sur l'IHM et la reconnaissance entre les images, la deuxième phase se concentre sur l'étiquetage des images.

Sur un projet total de trois mois, nous avons séparé arbitrairement chacune des phases à un mois et demi.

Concernant les spécifications techniques, il nous a été spécifiquement demandé d'utiliser certaines technologies bien précises, sur certains domaines ; pour le reste, nous devions choisir nos outils. Nous avons pris soin d'expliquer un maximum nos choix dans ce rapport.

PROJET PHOTOPHORE

Tout d'abord, pour l'interface graphique, il nous a été demandé d'utiliser les librairies de qT. Nous avons nous-même choisi d'utiliser Python par souci de confort et de cohérence avec les algorithmes de machine learning que nous souhaitions implanter. Dans le cadre de la ressemblance entre les images, il a été précisé que nous devions utiliser OpenCV et notamment les descripteurs SURF.

Enfin concernant le système d'étiquetage des images, cette technologie nous a été laissée au choix ; il a été question au début de sac de mots mais notre volonté commune de découvrir la technologie à la mode nous a conduit vers le machine et le deep learning.

Pour travailler à deux sur le code et les bibliothèques d'images, il a été fortement conseillé d'utiliser un gestionnaire de version. Nous avons donc utilisé Github, gestionnaire de version en cloud qui plus est. Suite à quelques problèmes techniques, nous avons dû créer deux repository différents.

Ci-dessous, la liste des commit sur notre repository git entre le 27 octobre et le 10 novembre.

```

commit 5a0a500af557a7200c606a7c4d2c6b44d2b1c2c7 (HEAD)
Author: Adrien BAUDE <Vendicare@localhost.localdomain>
Date:   Fri Nov 10 10:40:53 2017 +0100

    commit

commit 934514bc9040cca3abd01fd5949b0817a5568eb0
Author: rapha94 <raphael.presberg@gmail.com>
Date:   Thu Nov 9 13:25:01 2017 +0100

    images + codes ac images; next step : integrer le code au logiciel

commit d36dd8a75e13d72cd58ca8331753c7d75e40a6ce
Author: rapha94 <raphael.presberg@gmail.com>
Date:   Thu Nov 9 11:44:22 2017 +0100

    avec fichier matching images

commit 7eae59d4dc063a451ad246a778cd0783f6edcb0e
Merge: 00ec5fb f259743
Author: Adrien BAUDE <Vendicare@localhost.localdomain>
Date:   Fri Oct 27 10:33:40 2017 +0200

    commit

```

Figure 1: liste des commit sur le repository git

Lien du repository git : https://github.com/rapha94/Photophore_Final.git

II. Première phase du projet

Voici donc la première phase du projet pour laquelle on a passé environ la moitié du temps. Durant ce premier mois et demi nous avons notamment construit une IHM stable et solide, et terminé la partie concernant la ressemblance entre les images.

1. Le logiciel

a) L'interface graphique

Nous avons réalisé l'interface graphique en Python3 et en utilisant les librairies de qT associées à Python (pyQt)

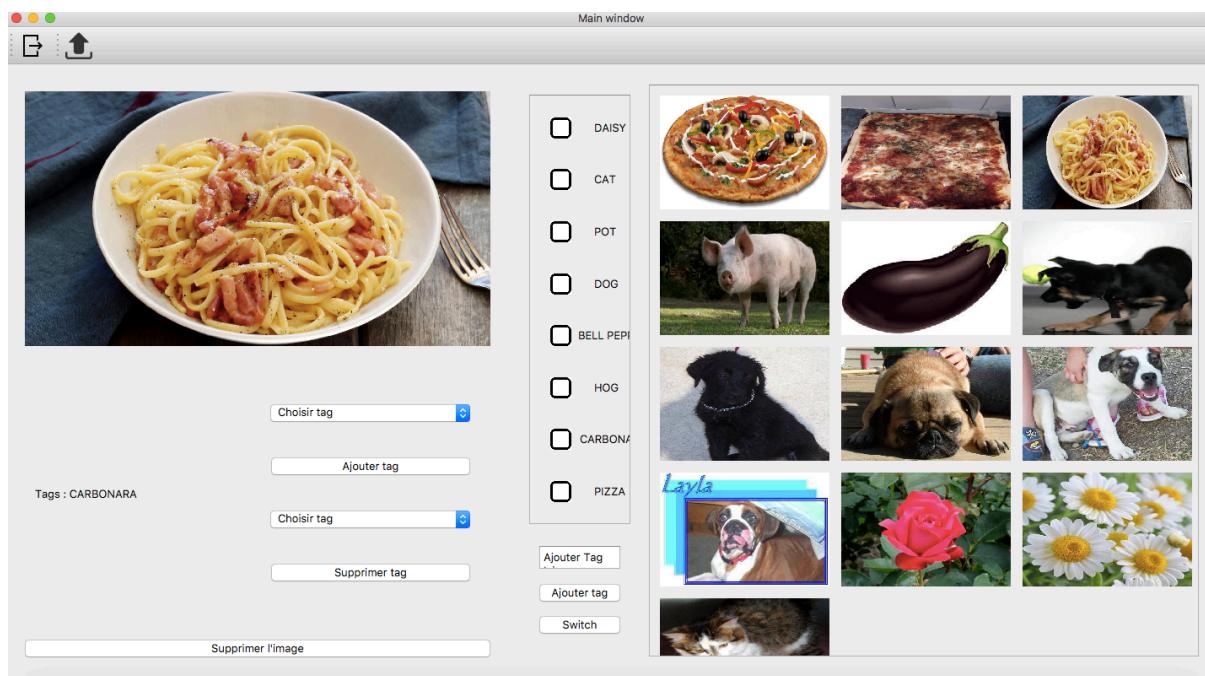


Figure 2: architecture de l'interface graphique

L'interface est séparée en trois parties : la photo focus à gauche, la liste des étiquettes au centre, et la liste des images téléchargées dans le logiciel à droite.

On ajoute aussi les boutons permettant de supprimer et ajouter les photos, ajouter et supprimer les tags, et associer ou séparer un tag avec une photo.

L'architecture du code est bien évidemment en MVC (Model-View-Controller).

Le model contient les données, soit la liste des images, la liste des tags et l'id de l'image focus. Il gère aussi l'accès à la base de données sqlite qui contient toutes les données du logiciel, les images, les tags, les distances entre les photos (pour la ressemblance entre les images), ...etc.

La View contient les éléments graphiques : les labels contenant les images et les tags. Le contrôleur récupère toutes les actions utilisateurs et chapote la view et le model dans leur bon fonctionnement.

Concernant le code, nous avons lancé un Doxygen. La documentation générée permet d'avoir une vision assez globale de l'architecture du code. Tous les commentaires dans le code sont accessibles directement dans le code.

Pour obtenir la documentation de Photophore, exécutez le fichier index.html dans le dossier html du projet.

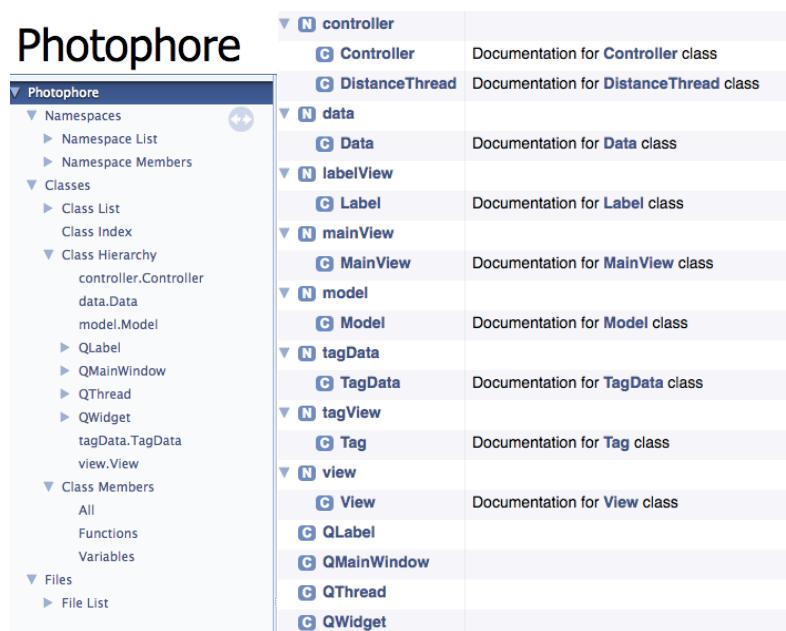
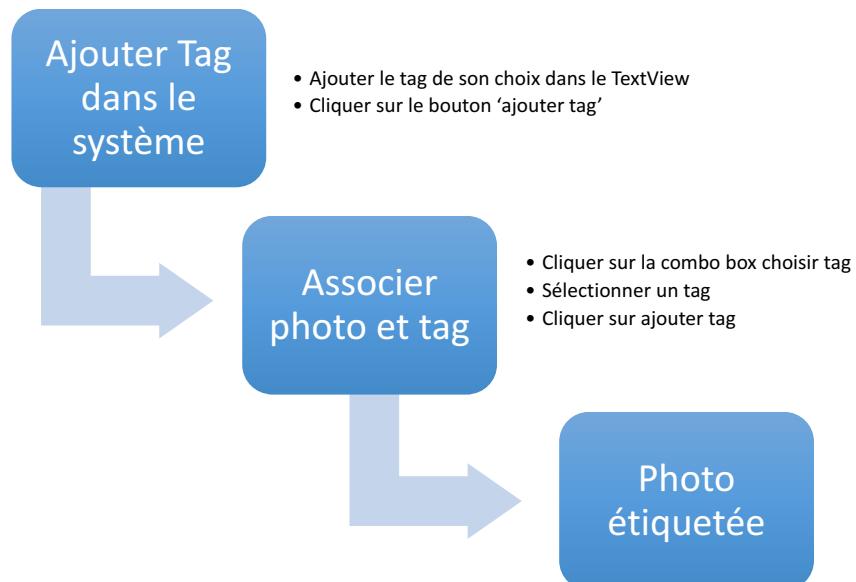


Figure 3: Documentation Doxygèn

b) L'étiquetage manuel des photos

Les images sur le logiciel sont des objetsPixmap. Afin d'avoir une interface assez responsive, les dimensions sont en pourcentage d'écran ; sur la partie de droite, avec toutes les photos, celles-ci sont téléchargées avec des dimensions prédéfinies, soit 15% de l'écran en hauteur, et un maximum de trois photos en longueur sur le scroll area. La taille de l'image en focus est de 40% en hauteur, et la longueur s'adapte à la photo avec une limite de 40% de l'écran.

Process d'étiquetage des images :



2. Open CV

Afin de répondre aux besoins utilisateur concernant le tri des images par leur ressemblance, nous avons réalisé un algorithme calculant une distance mathématique entre deux images qui nous fournit ainsi un pourcentage de ressemblance. Ces calculs de distance sont d'ailleurs réalisés par des threads en arrière-plan pour ne pas bloquer l'IHM.

Pour cela, nous avons utilisé les bibliothèques open source d'Open CV, spécialisées dans le traitement d'images.

Les différentes étapes dans la réalisation de cet algorithme sont les suivantes :

- Définir et afficher les descripteurs
- Matcher les descripteurs de deux images
- Déduire un pourcentage de ressemblance
- Intégrer le programme dans le logiciel

a) Les Descripteurs

Sémantiquement parlant, un descripteur est un mot clé qui définit le contenu d'un document et qui permet de le retrouver dans un fichier.

Dans le domaine du traitement d'images, les descripteurs sont les points clés ou points d'intérêts d'une image. Généralement, ce sont les coins et autres côtés d'objets suite à la superposition des plans par exemple, ou encore les différences d'intensité et de couleur sur l'image.

Il existe plusieurs bibliothèques dans OpenCV permettant d'obtenir les descripteurs d'une image : SIFT, SURF, ORB, ... Ces bibliothèques appartiennent à la bibliothèque Feature2D, qui appartient elle-même à la bibliothèque FeatureDetector.

Un petit aperçu de la position de la bibliothèque SURF dans OpenCV.

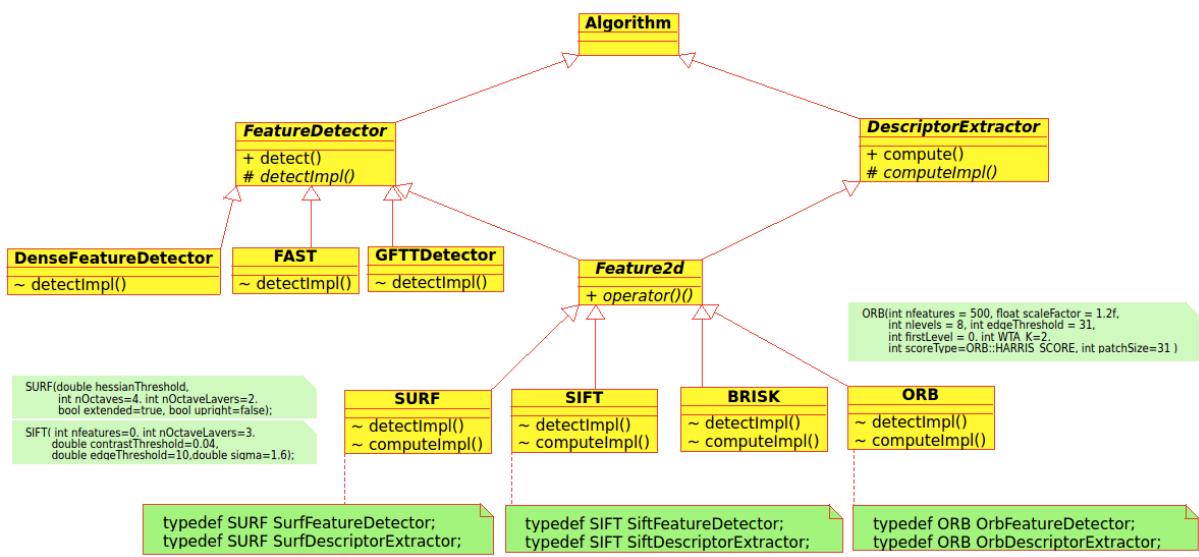


Figure 4: architecture feature2D d'OpenCV

On voit bien que SURF est au même titre que SIFT ou BRISK une bibliothèque de Feature2d.

Nous dissocions ainsi feature point et descripteur. Le feature point est un point potentiellement descripteur qui est généré par une fonction 'DETECT', alors que le descripteur est un feature point assez discriminant généré par la fonction 'COMPUTE'

Dans l'ensemble, l'utilisation de SURF, SIFT ou autre, fournit un résultat très similaire mis à part quelques légers détails. Afin de répondre au cahier des charge, nous avons utilisé SURF.

Afin d'avoir un point contenant assez d'informations pour le décrire descripteur de l'image, SURF utilise aussi les informations autour de ce point. Une fois que le point est assez discriminant, il est défini descripteur de l'image.

Autre paramètre à prendre en compte, l'orientation des informations autour du point est un des plus sensibles facteurs dans le traitement d'image. SIFT et SURF par exemple utilisent l'orientation normalisée pour détecter un feature point et le compute en descripteur ou non.

Ci-dessous, une modélisation de quatre feature points d'une image, qui sont transformés en descripteur avec justement des orientations différentes.

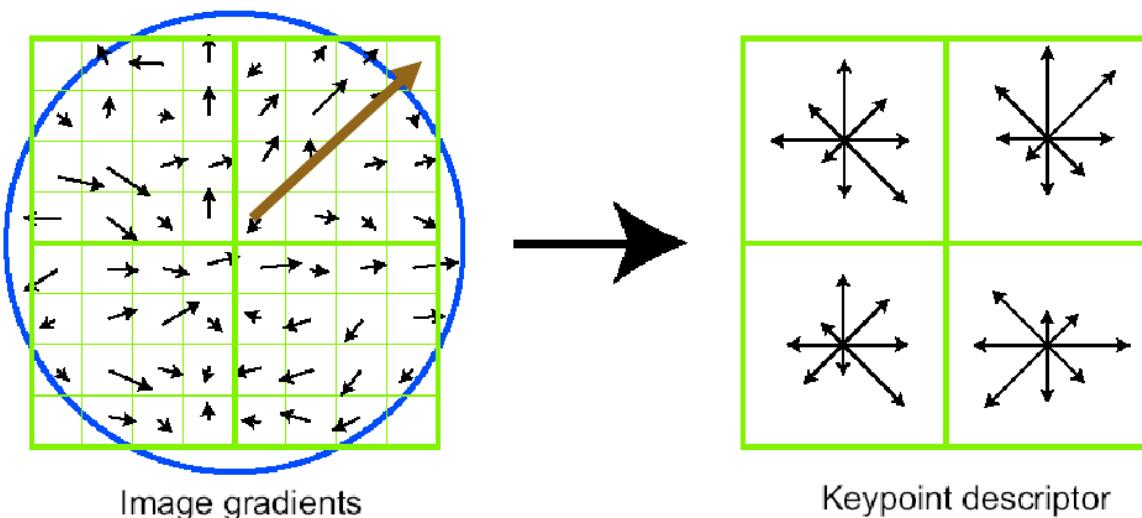


Figure 5: feature points

De plus, concentrons-nous sur le code qui affiche les descripteurs sur une image

```
import cv2
import matplotlib.pyplot as plt

import numpy as np

img1 = cv2.imread("eiffel1.jpg", 0)
#img2 = cv2.imread("image2.jpeg", 0)

surf = cv2.xfeatures2d.SURF_create(400)

kp1, des1 = surf.detectAndCompute(img1,None)

img2 = cv2.drawKeypoints(img1,kp1,None,(255,0,0),4)

plt.imshow(img2)
plt.show()
```

Figure 6: code génération des descripteurs

La fonction `imread` d'OpenCV nous permet de lire l'image et de l'associer à une variable `img1`. Nous associons de plus la variable `surf` à la bibliothèque SURF que l'on appelle en passant par la librairie `feature2d`.

Nous utilisons ensuite la fonction detectAndCompute de SURF pour détecter les feature point et les compute en descripteurs.

Nous dessinons ensuite les points et les affichons en utilisant matplotlib.

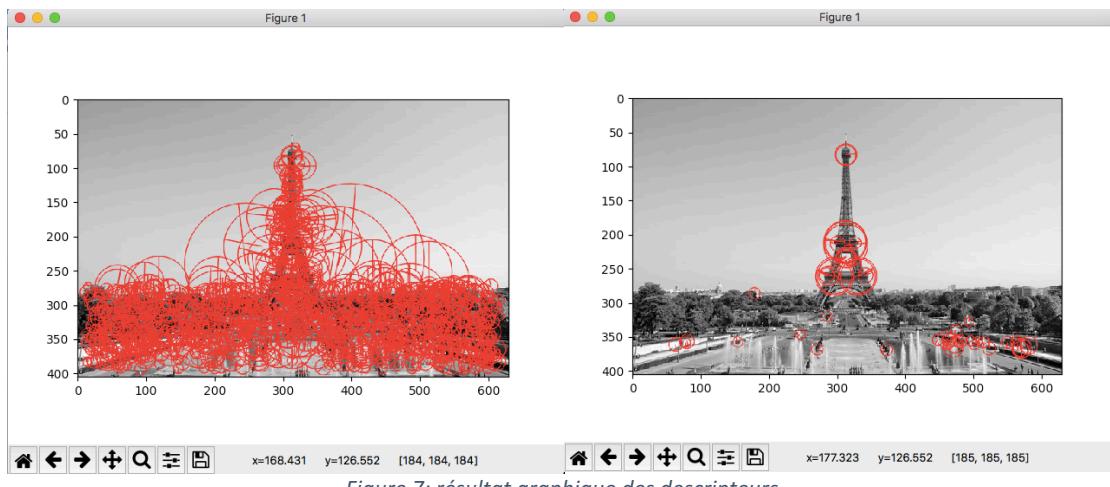


Figure 7: résultat graphique des descripteurs

L'image de gauche est la configuration utilisée dans notre programme. Afin d'observer les différentes possibilités de SURF, nous avons augmentés la précision des descripteur en utilisant la fonction SURF_create(2000). L'image de droite est le résultat du code renvoyant les descripteurs les plus discriminants de l'image. Cette configuration est néanmoins inutilisable pour comparer les images entre elles, car trop précise.

b) Matching entre descripteurs

Dans le process, il faut ensuite faire matcher les descripteurs SURF de deux images entre eux.

```
#mes deux images
img1 = cv2.imread("eiffel1-1.jpg", 0)
img2 = cv2.imread("eiffel1.jpg", 0)

# Initiate SURF detector
surf = cv2.xfeatures2d.SURF_create(400)

# find the keypoints and descriptors with SURF de chacune des images
kp1, des1 = surf.detectAndCompute(img1,None)
kp2, des2 = surf.detectAndCompute(img2,None)

#initialise flann index
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)

#unir flannBaseMatcher avec method knnMatch
flann = cv2.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)

# stocker les good matches
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
```

Figure 8: code génération des match entre descripteurs

Pour obtenir un résultat graphique, nous récupérons tous les bons matchs entre descripteur en utilisant les fonctions FlannBasedMatcher et KnnMatches qui permet de calculer une distance entre deux descripteurs.

La liste des bons matchs sont ensuite stockés dans un tableau que l'on appelle good. On récupère uniquement les matchs entre descripteurs de plus de 70%.

Le résultat graphique est assez cohérent à première vue. Effectivement, nous avons fait quelques tests en affichant uniquement les matchs entre descripteurs.

Voyez ci-dessous les résultats les plus représentatifs : le match parfait, le match à 50%, et le non match.

- Match parfait

Ici les deux images sont exactement les mêmes.

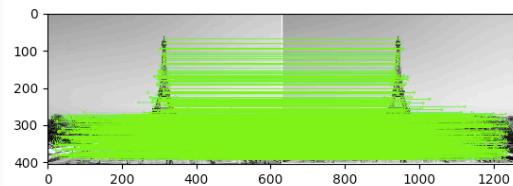


Figure 9: résultat graphique match parfait

On voit bien que tous les descripteurs matchent d'une photo à l'autre

- Match environ 50 %

Nous prenons une image et nous l'a rognons afin qu'elle représente la moitié de l'image initiale. Autrement dit, le pattern d'image est identique.

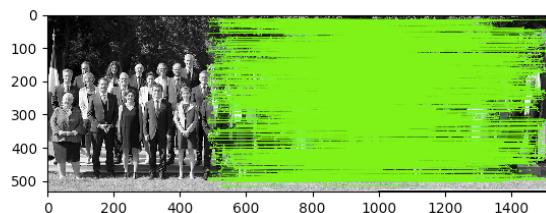


Figure 10: résultat graphique match 50%

On note que tous les descripteurs de la photo rognée matchent avec la moitié des descripteurs de l'image initiale.

- Match nul

Lorsque nous sélectionnons deux photos totalement différentes, nous observons qu'effectivement, mis à part quelques descripteurs, il n'y a aucun match.

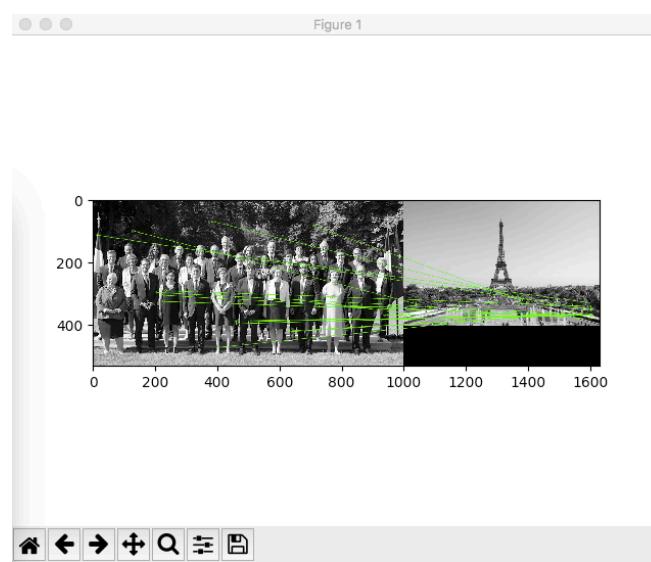


Figure 11: résultat graphique match NULL

c) Pourcentage de ressemblance

Dans le logiciel photophore, nous ne verrons pas la ressemblance entre les images graphiquement. Néanmoins, pour obtenir le tri en fonction de la ressemblance, nous avons besoin de cette technologie.

Nous avons donc personnalisé notre programme de matching d'image en récupérant un pourcentage de match entre deux images. C'est donc cette valeur qui sera utilisée pour Photophore.

Dans le programme ci-dessous, nous affichons tout d'abord les potentiels matchs entre descripteurs sur une première image, puis les vrais matchs en fonction de la deuxième photo (contenu dans le tableau good).

A partir de ces deux valeurs, nous pouvons donc calculer un pourcentage de ressemblance entre deux images.

Pour la démonstration graphique de ce programme, nous affichons les matchs et la décision finale.

```
#a titre indicatif: les potentiels matchs, les bon matchs stockés dans good et le
all_matches = len(matches)
print ("nb de matchs: ", all_matches)

good_matches = len(good)
print ("nb de bons matchs: ", good_matches)

pourcentage_matching = good_matches*100/all_matches
print ("resultat de matching: ", pourcentage_matching, "%")

# image + final decision(the image won't be used in the software rather ...
# ... than the final decision)

# choisir le calibrage de ressemblance
if pourcentage_matching > 25:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)

    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

    h,w = img1.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
    dst = cv2.perspectiveTransform(pts,M)

    img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)

    print ("Final decision ==> Matching")

else:
    print ("Final decision ==> No matching")
matchesMask = None
```

Figure 12: code décision sur terminal

Reprendons l'exemple avec les mêmes photos que précédent (voir 2. Matching entre descripteurs)

Dans un premier temps, lorsque les images sont identiques, nous obtenons bien évidemment le même nombre de matchs potentiels et de bons matchs car les deux images génèrent exactement les mêmes descripteurs.

```
→ Photophore git:(master) ✘ python3
nb de matchs: 1064
nb de bons matchs: 1064
resultat de matching: 100.0 %
Final decision ==> Matching
```

Figure 13: détails sur terminal match 100%

La valeur du pourcentage est ainsi à 100, et la décision Matching.

Lorsque l'on coupe la photo à la moitié (le rognage étant manuel, nous n'avons pas coupé à 50% de la photo exactement), nous obtenons environ 50% de matching.

```
+ Photophore git:(master) ✘ python3
nb de matchs: 4705
nb de bons matchs: 2221
resultat de matching: 47.20510095642
Final decision ==> Matching
```

Figure 14: détails sur terminal match 50%

Enfin, dans le dernier cas pour lequel les photos sont parfaitement différentes, quelques descripteurs matchent mais il s'agit du plus grand hasard. Le pourcentage n'est que de 0,4 ; valeur assez dérisoire. Nous affirmons qu'il ne s'agit pas de la même photo.

```
+ Photophore git:(master) ✘ python3
nb de matchs: 4705
nb de bons matchs: 19
resultat de matching: 0.403825717321
Final decision ==> No matching
```

Figure 15: détails sur terminal match NULL

Les résultats de cette valeur appelée pourcentage étant cohérent avec les essais graphiques, nous l'intègrerons donc dans Photophore. Cela nous permettra d'avoir la fonctionnalité de de tri par ressemblance.

d) Les limites d'Open CV

OpenCV est absolument essentiel dans l'écosystème du traitement d'image, néanmoins on note quelques limites à l'utilisation des bibliothèques.

Effectivement, la fonction permettant le matching d'images en comparant les descripteurs initiés par SURF fonctionne parfaitement avec des images ayant un même pattern. Néanmoins, deux images avec un même objet pris en photo sous plusieurs angles différents ou avec des expositions à la lumière différentes ne seront pas considérées comme ressemblantes.

A contrario, deux images n'ayant aucun rapport, mais possédant des descripteurs assez similaires et susceptibles de matcher entre eux, pourraient être considérés comme ressemblants.

Ainsi si situe la limite de cette technologie qui ne prend pas en compte tous les paramètres nécessaires afin d'avoir un résultat optimal.

Prenons par exemple la tour Eiffel :

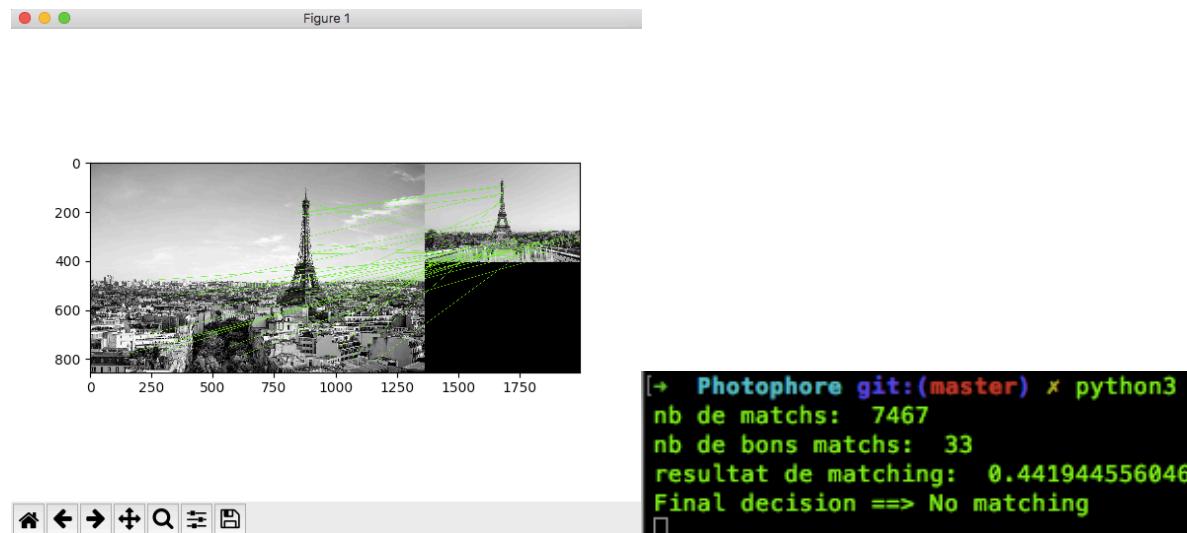


Figure 16: limites d'OpenCV

La tour Eiffel a été photographié sous deux angles différents, et avec deux luminosités un peu différentes, on aurait souhaité obtenir un pourcentage de ressemblance conséquent. Nous n'obtenons malheureusement que 0,44 %...

Nous obtenons un faible pourcentage pour la ressemblance entre deux laptops pourtant identiques.

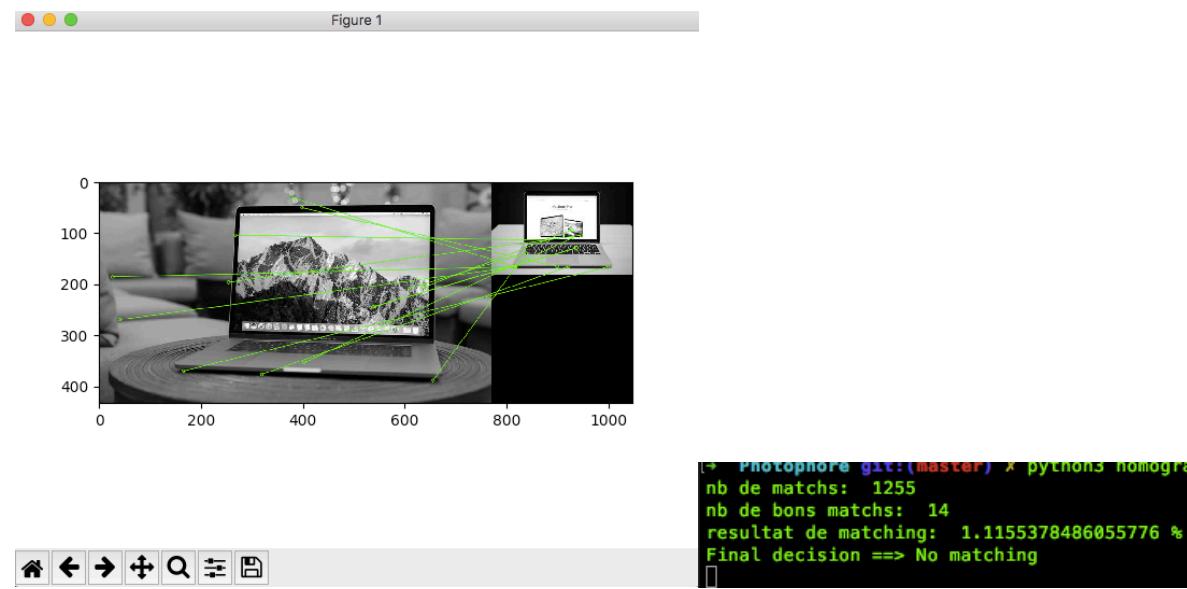


Figure 17: limites d'OpenCV (2)

Effectivement, nous n'obtenons que 1,1% de ressemblance.

Dans notre cas, les limites d'openCV sur la ressemblance entre les images ne nous affectent pas.

Dans le logiciel nous ne prenons que la valeur du pourcentage de ressemblance entre deux images et nous stockons cette valeur. Si l'utilisateur désire choisir le mode de visualisation des images par ressemblance, seules les 10 images les plus ressemblantes apparaîtront.

III. Deuxième phase du projet

La deuxième phase se concentre sur l'étiquetage automatique des photos.

Pour commencer, il était nécessaire d'avoir une interface graphique fonctionnelle sur laquelle il était possible de télécharger des photos, de les supprimer, et de les trier que cela soit par préférence ou par ressemblance.

Une fois cette interface graphique stable et solide, nous avons pu développer une solution pour l'étiquetage automatique des images.

1. Machine learning

Dans le but de rendre le logiciel intelligent, nous avons souhaité intégrer un algorithme de Machine Learning afin d'automatiser le système d'étiquetage des photos.

Ainsi, dès lors que l'on télécharge une photo dans le software, celui-ci va associer le tag le plus évident à la photo.

En image, prenons par exemple l'étiquette « human face »

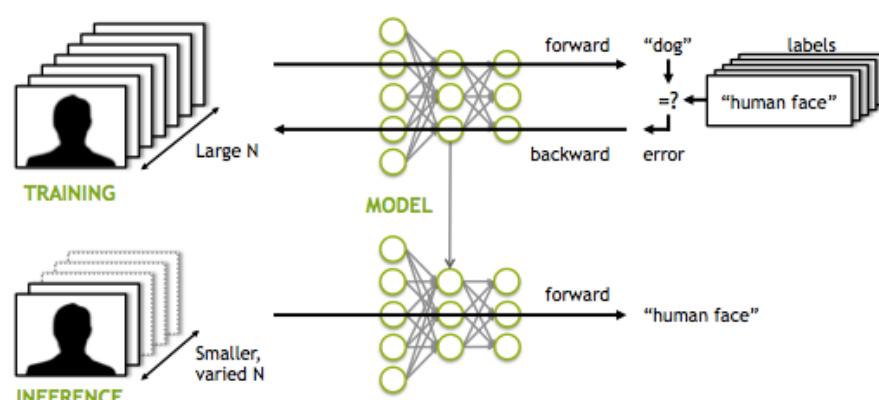


Figure 18: introduction machine learning

On entraîne un modèle sur un tag en particulier, que l'on connaît. Une image de visage humain est ensuite envoyé au logiciel et le modèle doit comprendre qu'il s'agit du tag « human face ».

Pour faire cela avec Photophore, nous avons dû comprendre, apprendre et appliquer différentes technologies et différents modèles, que cela soit de la simple régression linéaire aux technologies de Deep Learning, en passant par les différentes formes de classification.

a) Définition & Principes de base

Pour lebigdata.fr :

« Le Machine Learning est une technologie d'intelligence artificielle permettant aux ordinateurs d'apprendre sans avoir été programmés explicitement à cet effet. Pour apprendre et se développer, les ordinateurs ont toutefois besoin de données à analyser et sur lesquelles s'entraîner. De fait, le Big Data est l'essence du Machine Learning, et le Machine Learning est la technologie qui permet d'exploiter pleinement le potentiel du Big Data. »

Aujourd'hui on utilise principalement deux méthodes basiques de machine learning : la régression et la classification.

Un algorithme de régression permet de trouver un modèle (une fonction mathématique) en fonction des données d'entraînement. Le modèle calculé permettra de donner une estimation sur une nouvelle donnée non encore vue par l'algorithme (qui ne faisait pas partie des données d'entraînement).

Les algorithmes de régression peuvent prendre plusieurs formes en fonction du modèle qu'on souhaite construire. La régression linéaire est le modèle le plus simple : Il consiste à trouver la meilleure droite qui s'approche le plus des données d'apprentissage. La fonction de prédiction sera donc une droite.

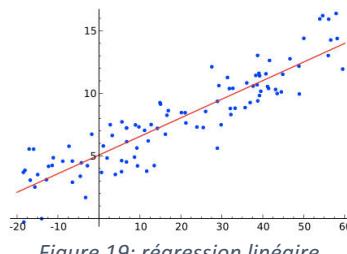


Figure 19: régression linéaire

Parmi les algorithmes de classification, on retrouve : Support Vector Machine (SVM), Réseaux de neurones, Naïve Bayes, Logistic Regression...

Chacun de ses algorithmes a ses propres propriétés mathématiques et statistiques. En fonction des données d'entraînement et nos features, on optera pour l'un ou l'autre de ces algorithmes. Toutefois, la finalité est la même : pouvoir prédire à quelle classe appartient une donnée.

Quand l'ensemble des valeurs possibles d'une classification dépasse deux éléments, on parle de classification multi-classes. L'image suivante illustre les deux types de classifications.

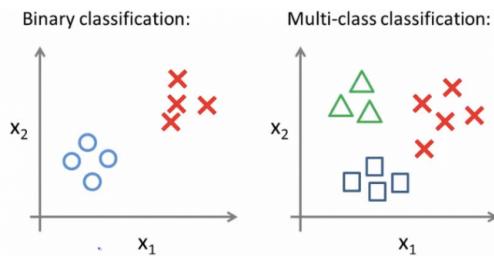


Figure 20: classification

Dans le projet photophore, il paraît évident et logique que nous utilisons une classification multi-classes.

Ensuite, pour choisir quel algorithme utiliser dans un problème, il faut se demander quelle sortie nous recherchons. Si nous souhaitons une valeur continue (un nombre) alors nous allons choisir une régression. Par contre, pour une valeur discrète (une catégorie) nous choisirons une classification.

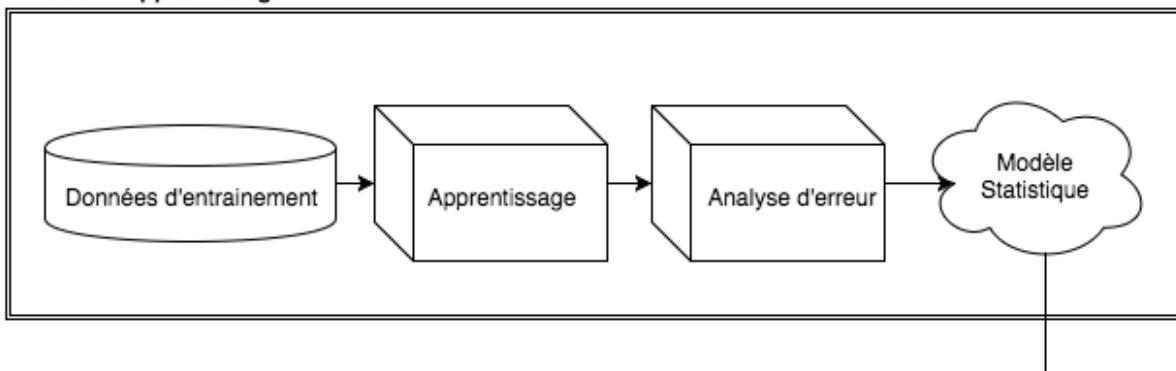
Le modèle est testé avec un dataset dont on connaît d'ores et déjà le résultat. Nous pouvons ainsi comparer le résultat obtenu et le résultat théorique afin d'obtenir un taux d'accuracy qui permettra de valider ou non le modèle.

Concernant le taux d'accuracy, il n'y a pas de règle prédefinie, généralement, cela dépend du besoin, du niveau d'exigence, et de la qualité des données et des inputs.

Un problème de machine learning peut être séparé en deux phases.

Il y a une première phase qui est l'apprentissage. Cela consiste à entraîner un modèle sur un set de données. S'en suit une deuxième phase qui est la réalisation.

Phase 1: Apprentissage



Phase 2: Réalisation de la tâche

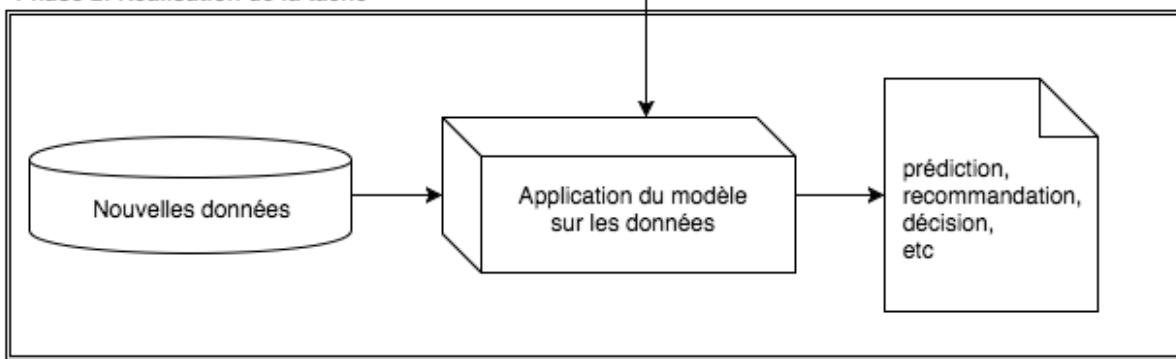


Figure 21: 2 phases

On remarque bien la distinction entre les deux phases. L'idéal est d'avoir un data set que l'on sépare en deux (1/3-2/3 par exemple), le plus gros échantillon servira de données d'entraînement, le deuxième échantillon sera le data set pour analyser l'erreur (c'est-à-dire pour trouver l'accuracy et éventuellement modifier le modèle jusqu'à obtenir des performances satisfaisantes).

Une fois le modèle statistique, régression ou classification, bien entraîné, il peut être appliqué sur un nouveau dataset afin de réaliser une prédition, une recommandation, ou tout autre besoin fonctionnel ou business.

b) Principes du Deep Learning

Afin de mieux comprendre le fonctionnement des technologies Deep Learning, prenons par exemple un utilisateur voulant identifier des photos qui comportent un chat. Pour cela, l'algorithme doit être en mesure de distinguer les différents types de chats, et de reconnaître un chat de manière précise quel que soit l'angle sous lequel il est photographié.

Afin d'y parvenir, le réseau de neurones doit être entraîné ; il est donc nécessaire de choisir un échantillon d'images qui servira de dataset d'entraînement. Cet ensemble va regrouper des milliers de photos de chats différents, images qui seront ensuite converties en matrices 2D, puis 1D, et transférées sur le réseau.

Les neurones artificiels assignent ensuite un poids aux différents éléments du modèle initié. La couche finale de neurones va alors rassembler les différentes features pour déduire s'il s'agit ou non d'un chat.

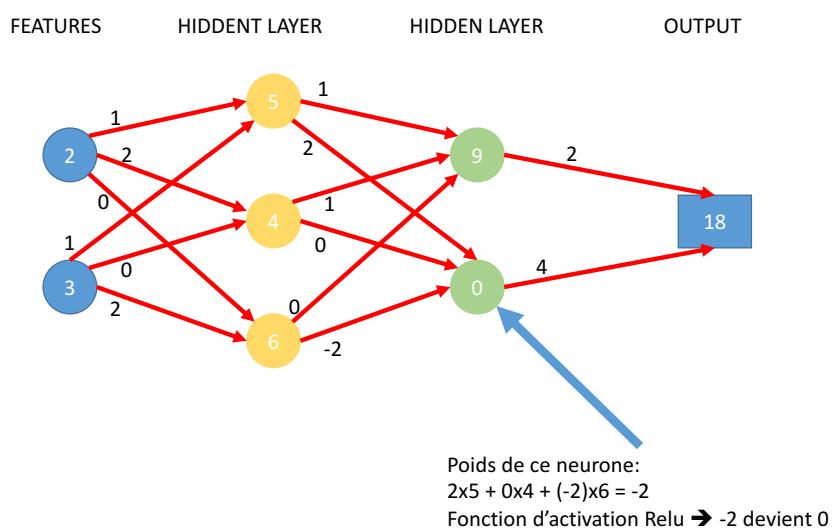
Le réseau de neurones va ensuite comparer ces réponses théoriques aux bonnes réponses indiquées manuellement par les humains. Si les réponses correspondent, le réseau garde cette réussite en mémoire et s'en servira plus tard pour reconnaître les chats. Dans le cas contraire, le réseau prend note de son erreur et ajuste le poids placé sur les différents neurones pour corriger son erreur. Le processus est répété des milliers de fois jusqu'à ce que le réseau soit capable de reconnaître un chat sur une photo dans toutes les circonstances.

Il existe aujourd'hui plusieurs framework de deep learning sur le marché. Le plus connu est développé par Google, il s'agit de TensorFlow, mais on peut trouver d'autres framework tout aussi fonctionnel comme CNTK, Theano, ou MXNet.

c) Le Deep Learning, techniquement parlant

Schématisons un simple réseau de neurones comportant :

- 2 features
- 1 Hidden Layer de 3 neurones
- 1 Hidden Layer de 2 neurones
- 1 output
- Fonction d'activation Relu
- Batch Size de 5



Les features correspondent aux entrées. Il s'agit de propriétés propres à chaque problème de machine learning que l'on utilise pour différencier nos classes.

Reprenons l'exemple du chat par rapport aux autres animaux, les features pourraient être la taille des yeux, la forme du museau, les moustaches, les griffes, etc...

L'output est la sortie. Comme expliqué précédemment, cela peut être une régression ou une classification.

Ce que l'on appelle Hiden Layer sont les couches de neurones. Chaque couche possède un ou plusieurs neurones en fonction du modèle et chaque neurone dans une couche est associé à un poids.

Le Batch Size est un paramètre concernant la descente du gradient. La valeur de 5 a été choisie arbitrairement.

PROJET PHOTOPHORE

La fonction d'activation est appliquée à un signal en sortie. Dans le cas ci-dessus, on l'applique à la sortie de chaque neurone. La fonction Relu remet à 0 un potentiel signal négatif ; si le signal est positif, la fonction retourne la valeur du signal.

Modélisation de la fonction d'activation :

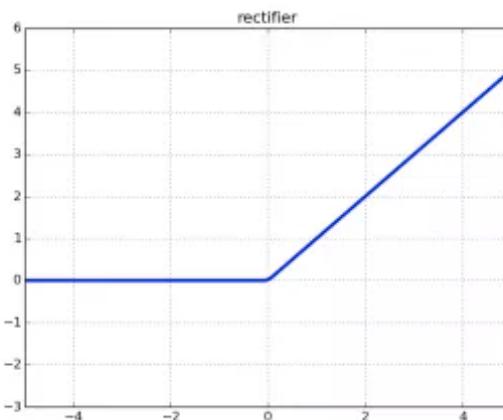


Figure 22: fonction d'activation RELU

Soit x la valeur du signal d'entrée ; on s'aperçoit que si $x < 0$ alors la fonction retourne 0, si $x \geq 0$, la fonction retourne x .

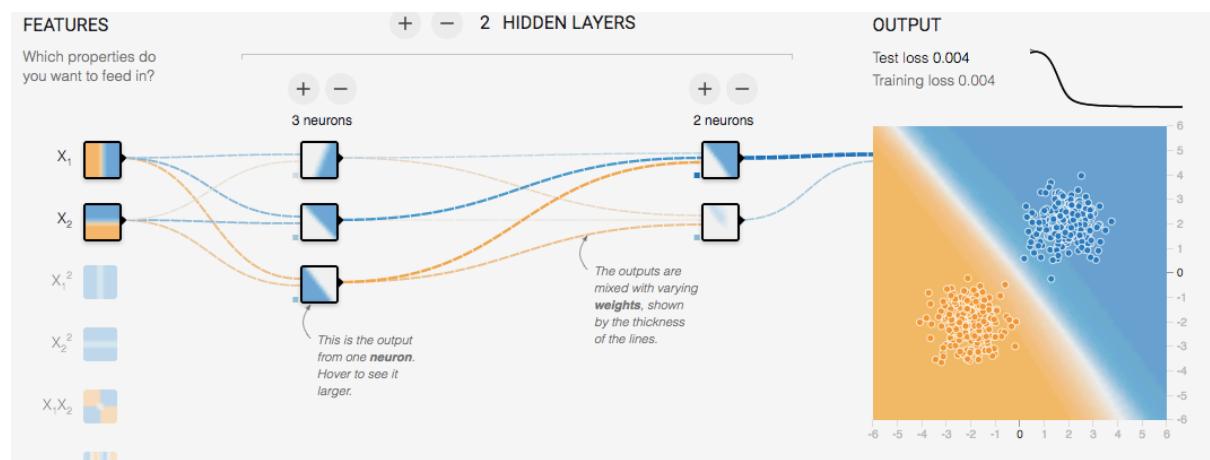
Pour entraîner un réseau de neurone, on envoie un dataset des milliers de fois sur notre modèle.

Un Epoch est lorsque le dataset est passé une fois dans le modèle. Ainsi, pour obtenir le modèle le plus performant possible, il faut compter plusieurs epoch.

Modélisation du réseau de neurones avec un dataset simple :

Nous avons choisi un dataset simple avec des points bleus en haut à droite et des points orange en bas à gauche.

Le réseau de neurones a ensuite été modélisé sur le playground TensorFlow en ligne et nous obtenons le résultat ci-dessous :

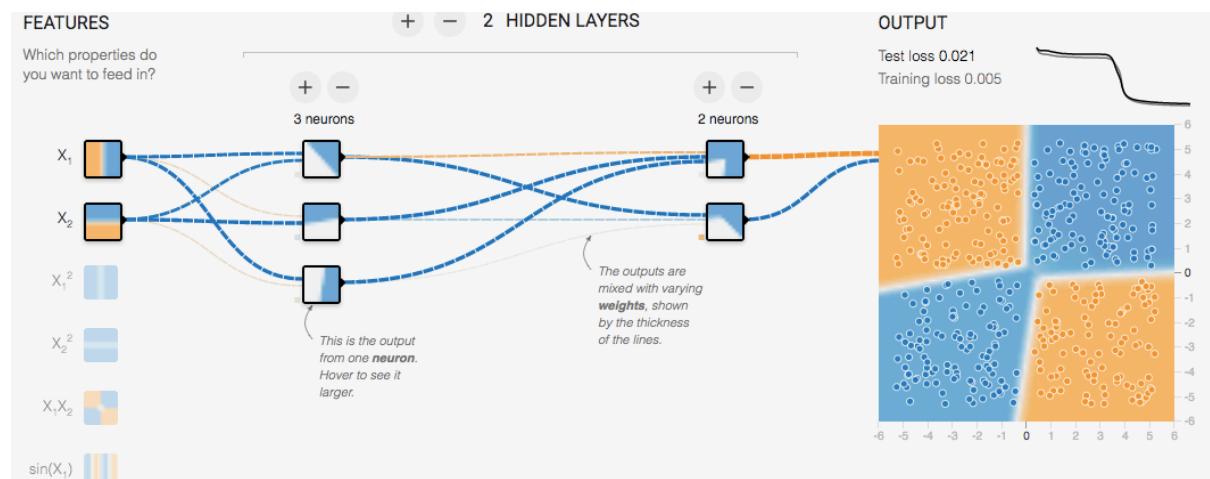


On note que le modèle, bien que très simple, renvoie un résultat parfait après environ 350 epoch.

Modélisation du réseau de neurones avec un dataset plus complexe :

Ici, nous choisissons un dataset plus complexe, effectivement nous avons des points orange en haut à gauche de l'image et en bas à droite, ainsi que des points bleus dans les deux autres coins.

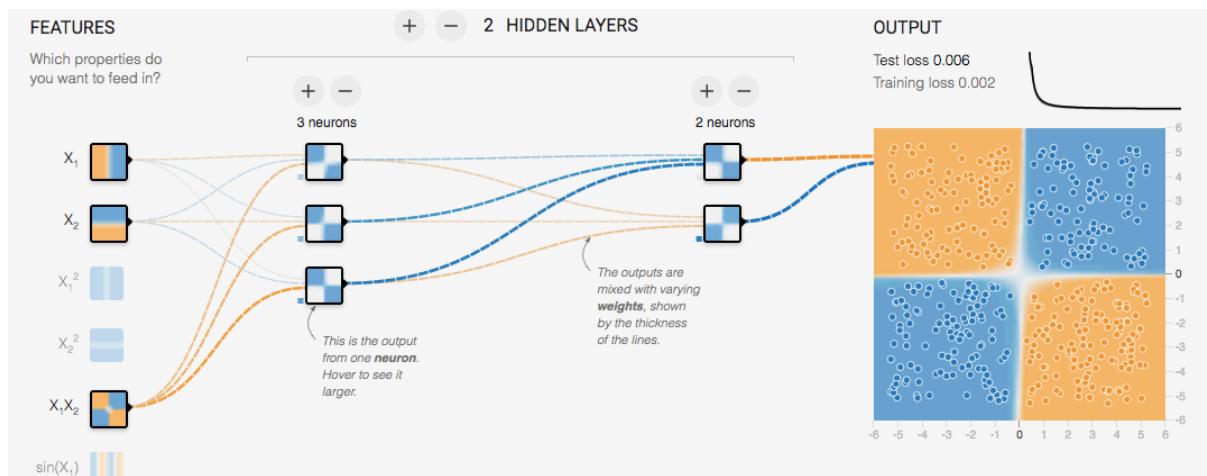
En réalisant le modèle ci-dessus, nous obtenons un résultat peu satisfaisant.



Après 1500 epoch, nous obtenons le résultat ci-dessus qui est convenable, mais pas parfait. On analyse que le rapport qualité du résultat/temps (ou nombre d'epoch) est finalement mauvais.

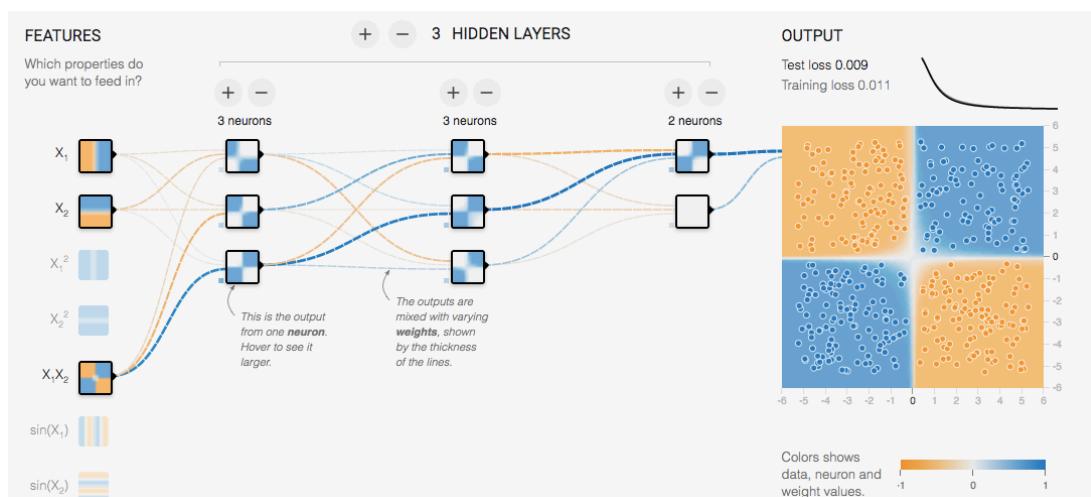
Suite à cela, nous procérons à plus de tests. Par exemple nous rajoutons des neurones à une couche, ou bien nous rajoutons une nouvelle couche d'abord avec 2 neurones, puis 4..ect.

Après un certain temps de tâtonnement, nous obtenons enfin un résultat très positif :



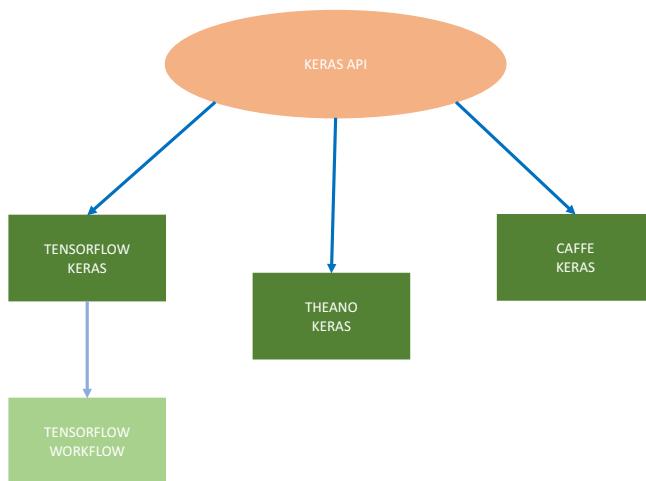
Effectivement, en rajoutant une propriété (ou feature), la multiplication des deux inputs initiaux en l'occurrence, le résultat est parfait après seulement 500 epoch.
On peut donc affirmer que le ratio qualité du résultat/temps nous convient pour répondre à ce problème.

Cependant, en effectuant par intérêt quelques tests en plus, nous trouvons un nouveau modèle répondant à ce dataset après seulement 175 epoch



d) Introduction Keras

Pour réaliser cet algorithme de machine learning, nous avons utilisé Keras qui est une librairie pour une implémentation plus haut niveau des réseaux de neurones, avec le choix d'un moteur d'exécution en TensorFlow, Theano, ou autre framework de Deep Learning.



Cela permet notamment d'écrire les programmes en moins de lignes de code, et répond à une logique de développement modulaire et de simplification de la gestion des dépendances et des performances.

De plus, Keras permet de simplifier la gestion du GPU pour le calcul, mais également de développer des modèles facilement compatibles avec Tensorflow.

2. Etiquetage automatique des images

A la base solide de la phase 1, nous avons amélioré notre système d'étiquetage en automatisant la tâche.

Il a ainsi été ajouté un algorithme de machine learning qui permet l'affichage d'un tag lorsque l'on télécharge une image sur Photophore.

Pour ce faire nous avons utilisé un modèle pré-entraîné plutôt qu'un modèle à entraîner nous même bien que nous ayons essayé les deux méthodes.

a) Un modèle à entraîner

Afin d'automatiser le système d'étiquetage, nous avons développé un modèle de machine learning qui avait une accuracy assez performante.

Cinq bibliothèques d'images (chats, chiens, et différents types de fleurs) ont été choisie arbitrairement afin d'en faire les tags par défaut du logiciel. Nous souhaitions entraîner le logiciel Photophore à reconnaître ces images avec le programme suivant :

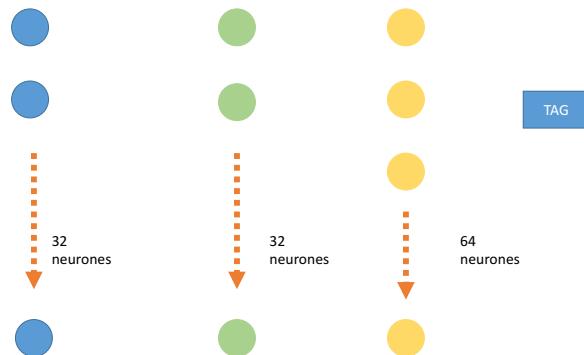
```

32
33
34 model = Sequential()
35 model.add(Conv2D(32, (3, 3), input_shape=(3, 150, 150)))
36 model.add(Activation('relu'))
37 model.add(MaxPooling2D(pool_size=(2, 2)))
38
39 model.add(Conv2D(32, (3, 3)))
40 model.add(Activation('relu'))
41 model.add(MaxPooling2D(pool_size=(2, 2)))
42
43 model.add(Conv2D(64, (3, 3)))
44 model.add(Activation('relu'))
45 model.add(MaxPooling2D(pool_size=(2, 2)))
46
47
48
49 model.add(Flatten()) # this converts our 3D feature maps to 1D feature vectors
50 model.add(Dense(64))
51 model.add(Activation('relu'))
52 model.add(Dropout(0.5))
53 model.add(Dense(1))
54 model.add(Activation('sigmoid'))
55
56 model.compile(loss='binary_crossentropy',
57                 optimizer='rmsprop',
58                 metrics=['accuracy'])
59
60 batch_size = 16
61
62 # this is the augmentation configuration we will use for training
63 train_datagen = ImageDataGenerator(
64     rescale=1./255,
65     shear_range=0.2,
66     zoom_range=0.2,
67     horizontal_flip=True)
68

```

Figure 23: code modèle à entraîner

Plus graphiquement :



Ce modèle développé avec Keras nous fournissait une accuracy d'environ 80% sur les fleurs, résultat qui était assez satisfaisant.

Néanmoins, cette solution a très vite rencontré deux problèmes principaux :

- Le temps et la puissance nécessaire à l'entraînement du modèle

Effectivement, pour entraîner un échantillon de 10 000 photos (2000 par classe, 5 classes différentes), il fallait compter environ 3 heures. Faire cela après chaque modification du modèle pour tenter d'améliorer l'accuracy au maximum nous a fait perdre beaucoup de temps.

De plus, entraîner un modèle avec des images en local exige un ordinateur très performant avec un bon processeur et un CPU ou une carte graphique puissant.

L'autre solution aurait été d'entraîner le modèle sur un serveur mais malheureusement nous n'avons pas eu accès aux serveurs de Sudria.

- L'exclusivité trop restrictive des cinq tags

De fait, n'entraîner le logiciel que sur 5 classes d'images (ce qui était notre volonté dans un premier temps) ne nous permettait pas de mettre d'autres images que ces 5 classes.

Par exemple, si un utilisateur souhaite télécharger une photo de pizza, un tag s'affichant automatiquement, le système aurait sélectionné un tag parmi chien, chat, daisy ..etc ; tag qui n'aurait donc eu aucun rapport avec la photo.

b) Un modèle pré-entraîné

En pratique, nous aurions préféré entraîner nous même un modèle sur nos propres classes, par manque de temps et surtout de puissance de nos ordinateurs, nous avons ainsi choisi une solution alternative.

En effet, nous avons utilisé un modèle de machine learning pré-entraîné et nous l'avons intégré à notre code.

Nous chargeons ainsi le modèle pré-entraîné de Keras, auquel nous envoyons le chemin vers la photo. Le code nous renvoie ensuite un vecteur contenant tous les tags identifiés par le VGG16.

Nous filtrons ensuite les pourcentages de ressemblance ; nous n'affichons que les tags avec plus de 40% de précision. D'après nos tests, cette valeur est suffisante pour obtenir des tags correspondants aux images sans que cela soit trop exigeant.

Aussi, l'inconvénient principal du modèle pré-entraîné est la dé personnalisation du système d'étiquetage. Les tags sont imposés et parfois trop précis pour ce que l'on souhaite.

Prenons par exemple les chats. Le VGG16 reconnaît 6 races de chats.

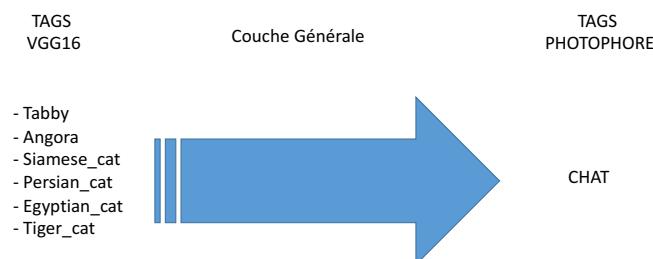
	A
1	tabby
2	Egyptian_cat
3	tiger_cat
4	Persian_cat
5	Angora
6	Siamese_cat
7	

Figure 24: couche générale (.csv)

Afin d'afficher le tag 'chat' sur Photophore, et donc d'être plus général, nous avons construit une couche plus générale.

Dans un fichier csv, nous avons récupérer les races de chats. Si un chat est associé par exemple au tag 'Angora', il sera affiché sur l'IHM 'chat'.

Modélisation d'une couche générale :



Pour personnaliser un maximum les tags, nous devrions faire cela pour chacune des classes.
Pour le moment, nous avons réalisé cette opération uniquement pour les chat et les chiens.

Nous aurions pu le faire pour les fleurs mais nous avons préféré laisser les détails du VGG16 pour l'exemple.

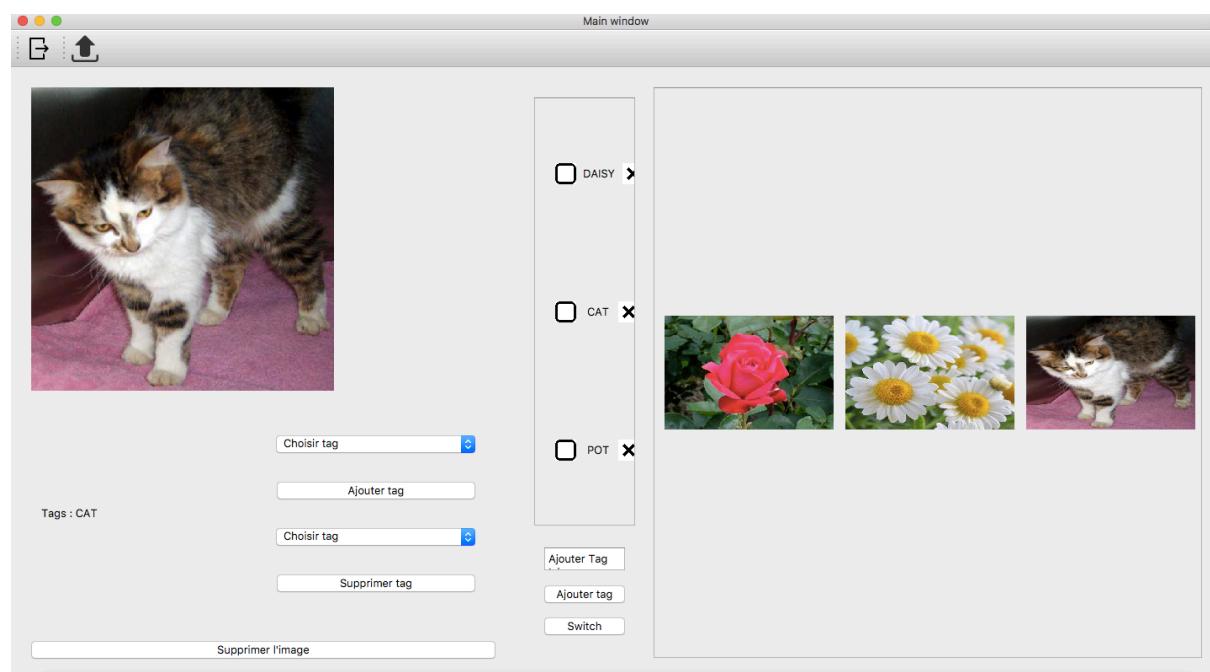


Figure 25: aperçu étiquettes 'chat'

Le chat, bien que cela soit une race très particulière, est ainsi étiqueté 'chat'.

PROJET PHOTOPHORE

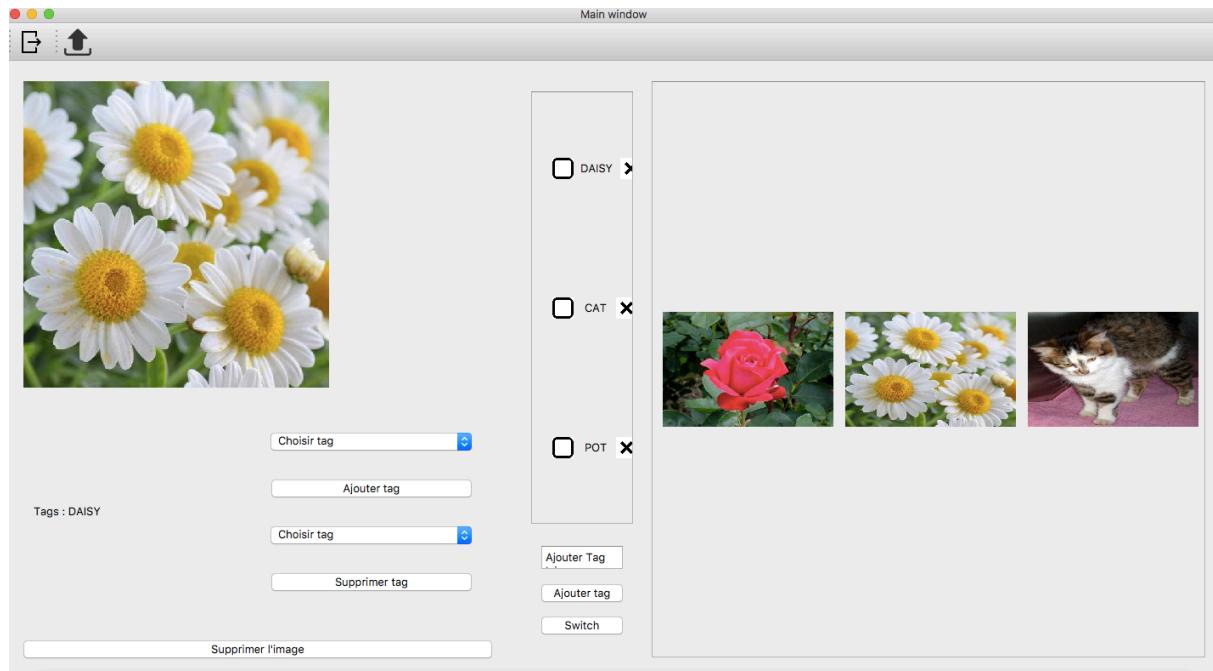


Figure 26: aperçu étiquette 'daisy'

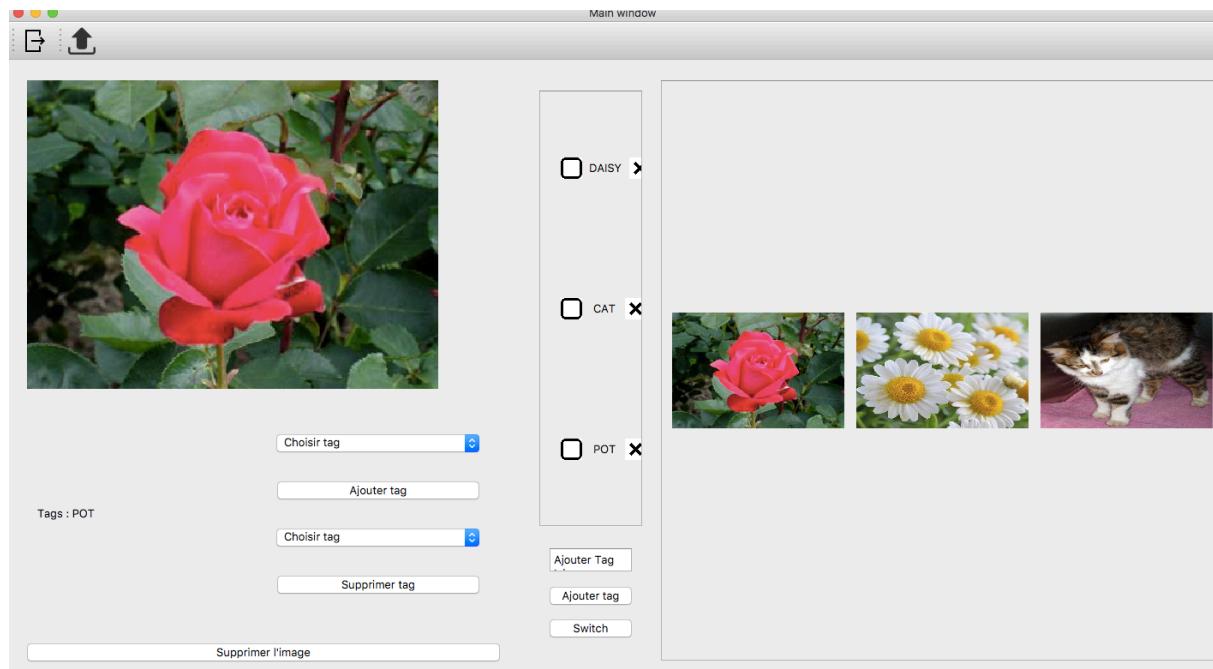


Figure 27: aperçu étiquette 'pot'

En revanche, concernant les fleurs, on remarque que les étiquettes correspondent au type de fleur.

IV. Analyse

Nous avons travaillé environ trois mois sur ce projet, et maintenant que le projet touche à sa fin, il est important de prendre du recul et de ne pas hésiter à s'autocritiquer afin de continuellement apprendre de nos erreurs.

1. Cahier des charges

Reprendons le cahier des charges rédigé en amont du projet :

- Développer une interface graphique
- Télécharger des photos sur la bibliothèque d'images
- Supprimer des photos
- Déplacer les photos suivant les préférences utilisateur
- Trier les photos en fonction leur ressemblance
- Etiqueter manuellement les images
- Etiqueter automatiquement les images

Nous pouvons affirmer que le cahier des charges a été rempli.

Cependant, pour un futur plus ou moins proche, il est possible d'apporter quelques modifications à Photophore :

- Entrainer manuellement le logiciel

Si une connexion au serveur de Sudria est possible, ou si vous disposez d'un ordinateur assez puissant, il serait intéressant de ne plus utiliser le VGG16 et d'entrainer vous-même les classes.

Cette technique offre une vraie personnalisation du logiciel afin qu'un maximum d'utilisateurs utilisent cette interface pour ordonner leur photos après un beau voyage.

- Modèle hybride VGG16 – entrainement manuel

Il aurait été également possible de se baser sur le modèle pré-entraîné, puis de passer le vecteur de sorti dans un autre classifieur afin d'entrainer sur nos propres classes.

Pour cela, il aurait fallu un modèle multi-classe qui n'est pas « tout-ou-rien », c'est à dire que le modèle aurait du pouvoir sortir un tag « autre », on aurait donc pris par défaut les tags du modèle pré-entraîné. Nous n'avons pas retenu cette méthode car trop complexe à développer en si peu de temps, mais c'est bien de savoir que cela est possible.

- Améliorer le design de l'interface graphique

Cela n'était pas le but de notre projet donc il est vrai que nous l'avons un peu, voir totalement délaissé. L'interface graphique a néanmoins été conçue dans un temps record et elle est parfaitement fonctionnelle et responsive.

Il reste cependant du design à ajouter sur cette interface, afin de rendre Photophore plus ergonomique et plus simple d'utilisation.

- Connecter la plateforme sur internet

Il serait important aussi, afin de rendre Photophore plus accessible, de mettre la plateforme sur un Cloud. Oublions qT et l'interface graphique un instant, et implantons le logiciel directement sur le Cloud.

Les codes de ressemblance d'image ou encore de machine learning étant fonctionnel, il ne suffirait qu'à les réimplanter sur une interface web.

De plus, avec cette interface web, il aurait été possible de continuer l'apprentissage du modèle en continu avec les images ajoutées et manuellement étiquetées par l'utilisateur.

2. Comparaison des performances

Ayant rencontré plusieurs technologies différentes durant le projet, nous pouvons analyser leur performance sur un exemple bien concret.

L'exemple utilisé est bien évidemment l'étiquetage des images. De plus, il se peut que nous manquions d'objectivité et que nos affirmations, basées sur les tests effectués pendant le projet, ne soient pas complètes. Effectivement, il existe une multitude de technologies différentes, d'assemblage de technologies, ou de modèles possibles.

Cela dit, nos tests semblent cohérents et notre étude de la performance aussi :

- Machine Learning/OpenCV (descripteurs SURF)

Sur la ressemblance entre les images, on a parlé plus haut des limites de OpenCV.

Les algorithmes de machine learning ont des performances limitées certes, mais les résultats obtenus avec une classification Scikit Learn basique sont bien meilleurs que ce que l'on peut obtenir avec les descripteurs SURF.

Ainsi, un modèle de machine learning sera bien plus performant et identifiera mieux les objets que SURF, mais celui-ci sera sûrement plus compliqué à développer qu'un code utilisant la bibliothèque feature2D d'OpenCV.

- Machine Learning /Deep Learning

Un modèle de machine learning basique est toujours plus simple à mettre en place qu'un modèle de deep learning, plus complexe, et plus long à implémenter et tester.

L'analyse concernant le deep learning est assez simple. Il s'agit d'une nouvelle technologie exceptionnelle en qualité et en puissance la réalisation de ce projet permet de se rendre compte à quel point cette nouvelle technologie n'est pas encore maîtrisé aujourd'hui. On ne connaît ni les limites, ni les performances maximales, ni les modèles à utiliser à l'avance.

Pour un simple modèle de quelques points séparés dans un carré, nous avons passé 1 mois à faire des tests avec différents modèles, différents inputs, différentes fonctions d'activation, etc..

Imaginez les possibilités de modèles pour un problème beaucoup plus complexe, très fonctionnel, et avec plusieurs dizaines d'inputs.

Généralement, pour un problème simple, il est préférable d'utiliser un algorithme simple de machine learning, dès lors que le problème se complexifie, les performances seront bien meilleures avec un réseau de neurones.

3. Difficultés & Enseignements

Si nous devions résumer ces trois mois de projet, nous le ferions en un mot : découverte. En effet, le fil conducteur sur cette période a été la découverte de nombreuses technologies qui, assemblées ensemble, fournissent un résultat d'autant plus performant.

En plus d'approfondir nos connaissances en qT, qui a été vu en Master 1 à Sudria, nous avons découvert OpenCV, la bibliothèque la plus performante pour les thématiques de traitement d'image, le machine learning, avec notamment la régression et la classification, et le deep learning avec ses différents framework,

Quoi de mieux que d'avoir réalisé le projet Photophore pour débuter sa carrière professionnelle dans un vaste domaine comme le machine learning.

Nous avons néanmoins rencontré quelques difficultés, et c'est sur ces points bien précis que j'aimerai insister car on apprend beaucoup sur les nouvelles technologies en lisant les documentations, mais on apprend encore plus de ses erreurs.

Tout d'abord, la durée du projet a été très courte. Donc bien sûr, cela est dû à notre départ aux Etats Unis donc il n'y avait pas le choix, cependant, quel dommage de ne pas avoir pu développer cette plateforme en l'a connectant sur un serveur.

Ensuite, la complémentarité du binôme a été un réel atout dans le sens où les tâches étaient bien réparties durant la première phase et que nous avons travaillé ensemble dans la recherche du meilleur modèle de machine learning. Même si nous avions des connaissances techniques et fonctionnelles différentes et bien que nous travaillions avec des rythmes différents, nous avons réussi à travailler ensemble.

Cela dit, la multitude d'informations n'a pas été intégré à la même vitesse. Nous avons donc dû nous adapter l'un à l'autre, exercice qui s'est révélé très formateur dans le sens où ce cas de figure risque d'arriver aux Etats-Unis ; au moins, nous serons préparés.

Conclusion

Notre enseignement supérieur touche à sa fin avec ce projet ; 5 ans après notre arrivée à l'ESME Sudria, nous sommes heureux d'avoir parcouru ce chemin. Lorsque l'on regarde le projet de fin d'étude que l'on rend, une légère sensation nous émeut et on se rappelle forcément de ce premier jour d'école. Nous étions jeunes, nous ne connaissions rien au monde de l'entreprise et nous étions à des années lumières de devenir ingénieur. Nous voici à rendre notre dernier projet en tant qu'étudiant à Sudria, avant de partir pour de nouvelles aventures outre-Atlantique.

Nous sommes fiers d'avoir réalisé ce projet, qui nous a servi d'initiation (et d'approfondissement) sur différentes technologies.

En plus de revoir les interfaces graphiques qT et leur architecture en MVC, nous avons découvert OpenCV, référence dans l'écosystème du traitement d'images, ainsi que la nouvelle technologie à la mode et en corrélation avec le Big Data : le machine learning.

Nous souhaitons bonne chance aux élèves qui potentiellement continueront le projet. Aussi, nous avons proposé certains axes d'amélioration ou tout simplement de continuation de Photophore pour que le travail commencé ce semestre perdure et soit utilisé par le plus grand nombre d'utilisateurs.

Plus personnellement, en définitive, Photophore aura été un projet formateur techniquement et fonctionnellement, formateur sur le plan émotionnel aussi avec cette courte durée de projet, et fort en attraction dans l'optique de rechercher un stage/premier emploi dans le domaine de la Data Science, de l'Intelligence Artificiel, ou du Machine Learning.

Bibliographie

<https://blog.octo.com/classification-dimages-les-reseaux-de-neurones-convolutifs-en-toute-simplicite/>
<https://opencv.org/>
<https://www.qt.io/>
<https://openclassrooms.com/courses/initiez-vous-au-machine-learning>
<https://www.datacamp.com/home>
<https://www.kaggle.com/>
<https://www.tensorflow.org/>
<http://playground.tensorflow.org/>
<https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>
<https://keras.io/>
<http://caffe.berkeleyvision.org/>
<http://deeplearning.net/software/theano/>
<https://github.com/>
<https://stackoverflow.com/>
<https://www.tutorialspoint.com/>
<https://www.youtube.com/>
<https://www.python.org/>
<http://scikit-learn.org/stable/>
<https://www.lebigdata.fr/>
<https://www.linkedin.com/>
<https://www-01.ibm.com/>
<http://www.futura-sciences.com/>
<https://fr.mathworks.com/>
<https://www.sas.com/>
<http://www.journaldunet.com/>
<https://www.coursera.org/learn/machine-learning>
<https://www.lesechos.fr/>
<https://azure.microsoft.com/fr-fr/services/machine-learning-studio/>
<https://www.pluralsight.com/courses/understanding-machine-learning>
<https://www.codingame.com/>
<https://www.elastic.co/products/x-pack/machine-learning>
<https://console.bluemix.net/catalog/services/machine-learning>
https://docs.opencv.org/3.2.0/d9/df8/tutorial_root.html
<https://pub.phyks.me/sdz/sdz/introduction-a-la-vision-par-ordinateur.html>
<http://www.robindavid.fr/opencv-tutorial/>
<https://www.pyimagesearch.com/2016/12/19/install-opencv-3-on-macos-with-homebrew-the-easy-way/>
<https://www.learnopencv.com/install-opencv3-on-macos/>
<http://www.mobileway.net/2016/12/15/install-opencv-for-python-on-mac-os-x-or-linux/>