Table des matières

[I. Présentation Face Key 2](#_Toc508324230)

[II. Gestion de projet 2](#_Toc508324231)

[III. Base de données 2](#_Toc508324232)

[IV. Réseau 2](#_Toc508324233)

[V. Application mobile 2](#_Toc508324234)

[VI. IHM 2](#_Toc508324235)

[VII. Reconnaissance faciale 2](#_Toc508324236)

[a. Contexte 2](#_Toc508324237)

[b. Intégration de la reconnaissance faciale dans le projet 2](#_Toc508324238)

[c. Construction de la base de données d’images 4](#_Toc508324239)

[d. Apprentissage/Reconnaissance des visages 7](#_Toc508324240)

[Découpe de la base de données 7](#_Toc508324241)

[Méthode par régression logistique 7](#_Toc508324242)

[Méthode par perceptron multicouches 9](#_Toc508324243)

[Méthode par réseau de neurones convolutif 9](#_Toc508324244)

[e. Implémentation dans Face Key 9](#_Toc508324245)

[f. Conclusion : Avancement & améliorations possibles 9](#_Toc508324246)

[VIII. Sécurité 9](#_Toc508324247)

[IX. Conclusion 9](#_Toc508324248)

# Présentation Face Key

# Gestion de projet

# Base de données

# Réseau

# Application mobile

# IHM

# Reconnaissance faciale

## Contexte

Depuis quelques mois, avec la sortie de l’IphoneX et sa technologique Face ID pour le déverrouillage du téléphone, la reconnaissance faciale est sur le devant de la scène des technologies en vogue. En réalité ce problème de reconnaissance de visages est un problème qui remonte aux années 60 avec les travaux de Bledsoe, Helen Chan et Charles Bisson. Ce n’est que récemment avec le développement du deeplearning que le niveau de reconnaissance devient suffisant pour être utilisé de manière fiable dans des applications. 2014 DeepFace 97,35% de reconnaissance et 2015 FaceNet 99,63% de reconnaissance.

Nous avons donc décidé d’intégrer de la reconnaissance faciale à notre gestionnaire de mots de passes afin de rendre l’expérience utilisateur la plus simple possible. Le but du gestionnaire de mots de passes est de réduire le nombre de mot de passe que nous utilisons quotidiennement à un seul mot de passe sécurisé. Avec la reconnaissance faciale nous pouvons réduire à zéro le nombre de mot de passe et rendre la connexion à un site rapide, facile et fiable. Contrairement à Face ID de Apple qui nécessite un capteur infrarouge spécial, nous voulions que le système puisse tourner sur n’importe quelle machine, ordinateur voir même mobile, et donc n’utilisant qu’un capteur optique. Nous avons donc décidé d’expérimenter différentes techniques de machine learning pour reconnaitre nos utilisateurs.

Le machine learning est un type d’algorithmes nécessitant un apprentissage, supervisé ou non, permettant d’effectuer des tâches de classification d’images par exemple. Cela nous permettrai donc de classifier nos utilisateurs de manière que si on lui présente une photo d’une personne, il puisse nous dire de quel utilisateur il s’agit. Pour nous, le principe est d’entrainer de manière supervisé un modèle avec une base de données de visage de nos utilisateurs afin qui puisse apprendre les reconnaitre et que si on lui présente une nouvelle image d’un utilisateur qu’il a appris il puisse dire de quelle personne il s’agit.

## Intégration de la reconnaissance faciale dans le projet

Le problème de reconnaissance est découpé en 3 parties : la localisation du visage dans l’image de la webcam de l’utilisateur, la reconnaissance de ce visage qui implique l’apprentissage du visage.

Le problème de certaines techniques de machine learning, en particulier le deeplearning, c’est qu’il demande beaucoup de ressources au moment de l’apprentissage (temps et puissance de calcul). Cette étape ne peut donc pas être réalisée sur la machine de l’utilisateur. Nous allons donc devoir la déporter sur le serveur qui devra avoir une puissance de calcul suffisante.

Pour réaliser ce genre de calcul qui deviennent de plus en plus courant, le département informatique de L’Université en partenariat avec Orange à investit dans un ordinateur pour réaliser des calculs sur carte graphique. En effet les cartes graphiques permettent de paralléliser un maximum de calcul, ce qui est particulièrement efficace pour les calculs de type vectoriel et matriciel utilisé en machine learning. Nous avons donc eu l’honneur et la responsabilité de monter, installer, configurer pour le calcul sur carte graphique et inaugurer cette machine dans le cadre de notre projet. C’est donc cette machine qui nous servira de serveur.

Si la partie d’apprentissage du modèle demande beaucoup de ressources, l’utilisation du modèle entrainé pour reconnaitre les visages peut se faire avec très peu de ressources et peut donc être réalisé sur la machine de l’utilisateur.

De plus la détection du visage n’est également pas très couteuse et peut être faite sur la machine de l’utilisateur.

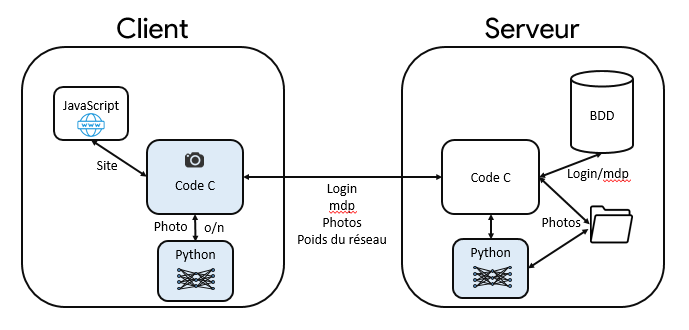
Nous avons donc sur notre client le code principal en C qui possède une fonction qui prend une photo, détecte le visage de l’utilisateur et envoi la partie de l’image correspondant au visage au code de reconnaissance. Ce dernier, en python, possède une version entrainée du modèle de reconnaissance et va prédire la classe de l’image qu’il a reçu et la renvoyer au code principal. Le code principal vérifie qu’il s’agit bien de la bonne classes en fonction du numéro de l’utilisateur et si c’est bien le cas il fait la demande de login/mot au serveur. En réalité, pour augmenter la fiabilité, le système va prendre plusieurs images et regarder la moyenne des identifications. Nous avons fixé le nombre d’images prisent à 5 avec un très léger temps entre les images pour minimiser la possibilité d’erreur. Si la personne est bien identifié les images du visage sont envoyaient au serveur pour l’apprentissage continue.

Figure Localisation des blocs de reconnaissance dans l'architecture de Face Key, en bleu.

Sur le serveur se déroule l’apprentissage du modèle de reconnaissance, il est réalisé une fois par jour. Il prend en compte les nouvelles classes créées (nouveaux utilisateurs) et les images utilisateurs courant qu’il a reçus à chaque connexion à un site. Ce système permet un apprentissage continu des classes. Si un utilisateur se laisse pousser la barbe tout en continuant à utiliser notre application alors le modèle apprendra à le reconnaitre avec sa nouvelle barbe. Une fois le nouveau modèle appris, il le transmet à tous les utilisateurs pour que les prochaines reconnaissances sur les machines des utilisateurs prennent bien en compte les nouvelles images apprissent.

## Construction de la base de données d’images

La partie commune des différentes méthodes de machine learning que nous allons utiliser est qu’elles nécessitent une base de données pour entrainer le modèle de reconnaissance. Cette base de données doit être labélisé car nous utilisons des méthodes d’apprentissage dites supervisées, c’est-à-dire que lors de l’entrainement, on présente l’image à apprendre au modèle et on lui dit à quelle classe appartient cette image. Ensuite, selon la technique d’apprentissage, la modèle va se modifier pour faire la corrélation entre l’image et la classe lui ont été donnée. La construction de la base de données est donc une étape primordiale car si les images et les labels qui lui sont présentés ne sont pas cohérant alors le modèle apprendra de mauvaises choses.

Pour notre application il faut donc construire une base d’images des visages de nos utilisateurs, labélisé avec leur nom. Pour des raisons de relatif anonymat nous n’utilisons pas le nom des utilisateurs comme label mais leur ID client que nous sommes les à connaitre.

Le problème est de réunir assez de photos du visage d’un utilisateur sans que ce processus soit trop lourd pour l’utilisateur.

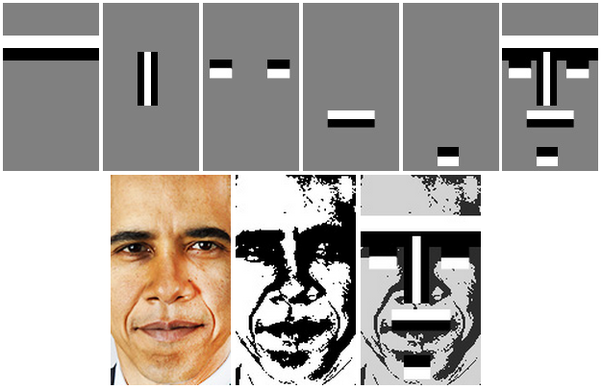
#### Comment l’utilisateur entre-t-il dans la base de données Face Key ?

Pour cela, à la création du compte Face Key, nous demandons à l’utilisateur s’il veut que le programme apprenne à reconnaitre son visage. S’il refuse, il ne pourras pas utiliser la reconnaissance faciale mais pourra utiliser Face Key comme un simple gestionnaire de mot de passe. Si il accepte, la fonction d’enregistrement d’images se lance, sa webcam s’allume, on détecte le visage de l’utilisateur dans chaque image du flux vidéo et on les envois au serveur pour entrainement. Le flux vidéo est pris pendant 1 minutes, pendant ce temps nous demandons à l’utilisateur de tourner sa tête sous différents angles et de réaliser différentes expression faciale pour que le modèle apprenne à reconnaitre l’utilisateur dans différents contextes. Les images transmissent au serveur sont des images couleur de 100px/100px.



#### Comment détecte-on le visage de l’utilisateur dans une image ?

Pour ce faire nous utilisons un algorithme appelé Haar Cascade qui utilise le principe de détection de features. Un visage est décomposable selon certain nombre de features simple.

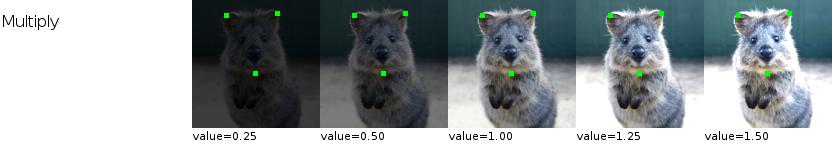
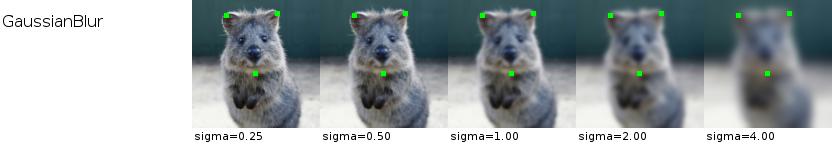
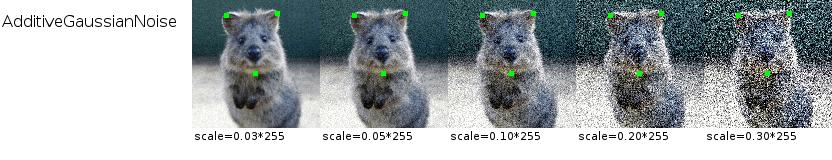


Ensuite on utilise nos features simple comme des convolutions sur notre image et on regarde a quel endroit de la nouvelle image obtenu se trouve le pattern ressemblant le plus à notre modèle de visage simplifié.

Le projet portant plus sur la reconnaissance de visage que la localisation de visage, nous avons fait le choix de ne pas implémenter l’algorithme même si initialement nous voulions implémenter une technique appelé HOG[[1]](#footnote-1) (Histogram Oriented Gradient) permettant également de détecter des visages mais en vue de la charge de travail et du peu de temps impartit nous avons décidé de ne pas prioriser cette partie. Nous avons donc utilisé la fonction HaarCascadeDetection d’OpenCV.

#### Et une fois les photos collectées sur le serveur ?

Une fois les photos des utilisateurs enregistrées sur le serveur, elles sont passées dans un augmenteur de données. En effet certaines technique de machine learning, principalement le deeplearning, nécessite un grand nombre d’exemple d’entrainement pour être performant. Le but de l’augmenteur de données va être d’à partir d’une image de la base, créer de nouvelles images en ajoutant du bruit, floutant l’image, baisser et augmenter la luminosité …



Ce système va permettre de créer artificiellement de nouveaux contextes et donc d’aider notre modèle à bien généralisé les classes qu’il apprend. Pour cela nous avons utilisé la librairie imaug[[2]](#footnote-2) dédié au machine learning et nous avons appliqué 13 transformations différentes sur nos images.

#### Comment avez-vous concrètement construit votre base ?

Pour construire la base de données il nous fallait des utilisateurs à pouvoir identifier. Nous avons donc fait un appel au prêt des étudiants du département informatique et des étudiants de CMI. 13 de ces derniers ainsi que 2 professeurs ont répondu à notre appel et nous avons pu construire une base de données de 17 personnes (en ajoutant nos 3 classes et en retirant la classe de Pierre F. qui a était victime délétion involontaire). Nous les avons fait suivre le protocole d’une création de compte Face Key comme décrit plus haut. Cette collecte nous a permis d’enregistrer 19462 images répartit relativement uniformément sur 17 classes et de construire une base de données de 225306 images après augmentation (soit 1.3Go d’images). Ces chiffres sont relevés après le tri de la base. En effet Haar Cascade (ou du moins son implémentation dans OpenCV) est relativement sensible à l’environnement. Des conditions de luminosité changeante pouvaient générer entre 2% et 15% d’images indésirable, des images où Haar Cascade voyait un visage alors qu’il n’y en avait pas. N’ayant pas trouvé de moyen simple pour automatiser la délétion des images indésirables, nous avons dû nettoyer la base de données à la main.

#### Remarques

La construction de la base de données est une étape extrêmement importante avant d’entrainer un modèle de machine learning. Nous avons passé beaucoup de temps à construire la basse de données cependant, avec le temps, nous remarquons que cette étape à était mal faite. En effet nous nous sommes laissé embarquer dans l’objectif d’avoir une base de données très large car d’après les dires de tout le monde, le deeplearning nécessite un très grand nombre de données. Dans notre cas la base est beaucoup trop redondante. Nous tirons les images d’un flux vidéo donc de l’une à l’autre les images se ressemble beaucoup. De nous avons mal fait l’augmentation de données car trop hésitant à faire de grosse transformation. On se retrouve donc avec des images qui se ressemblent beaucoup les unes aux autres et on les duplique 13 fois sans faire de transformations suffisamment importantes. La base est donc constituée d’énormément de redondance, ce qui n’aidera pas lors de la phase d’apprentissage. Il aurait mieux valu une base beaucoup plus petite mais beaucoup plus diversifiée.

## Apprentissage/Reconnaissance des visages

Maintenant que la base de données des visages des utilisateurs est constituée et le système de captation des visages mit en place, il nous faut entrainer notre modèle pour qu’il soit capable par la suite d’identifier les utilisateurs.

Nos travaux de références sont ceux de DeepFace et FaceNet qui obtiennent respectivement 97.35% et 99.63% de reconnaissance, c’est-à-dire que dans plus de 97% des cas ils arrivent à bien classifier le visage qui leur est présenté. Notre objectif est de se rapprocher le plus possible de ces scores.

Nous allons donc essayer différentes méthodes d’apprentissage.

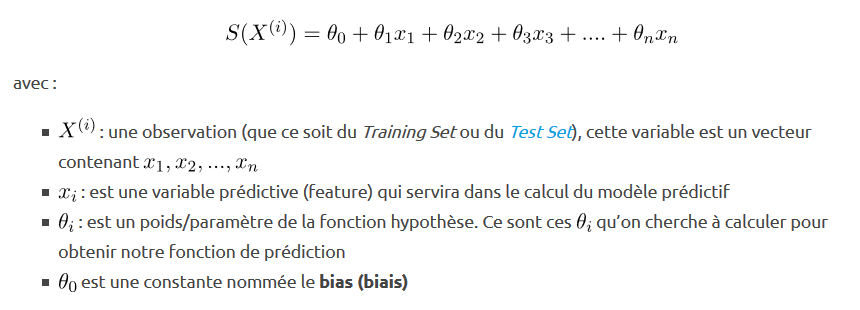
### Découpe de la base de données

A FAIRE

### Méthode par régression logistique

#### Théorie

La régression logistique est un modèle de classification linéaire probabiliste. C’est un outil simple de classification en machine learning, que l’on utilise souvent en premier pour tester son la complexité de son problème. Il est initialement fait pour faire de la classification binaire, classification en 2 classes.

Il est composé d’une fonction de coût S :

Cette fonction est ensuite passée dans une fonction sigmoid qui donnera un résultat y entre 0 et 1. Ce résultat peut être interprété comme la probabilité que le vecteur d’entré X soit de la classe 1. On a donc :

Sigmoid(S(X)) = P(y>=0,5 | X ; θ) et donc P(y<0,5 | X ; θ) = 1 - P(y>=0,5 | X ; θ)

Pour faire de la classification multi-classes on peut utiliser la technique du One-vs-Rest, c’est-à-dire que de la même manière, on calcul la probabilité 1 sorte par rapport au reste des classes. On obtient donc un ensemble de poids θ pour chaque classe.

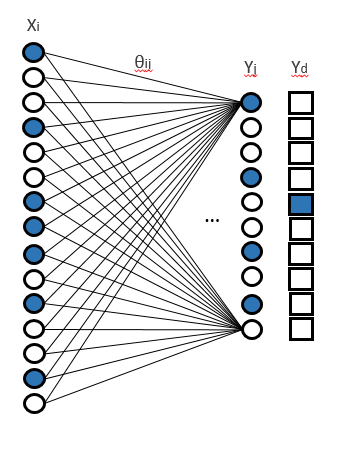
Pour qu’il y ait adaptation il faut que les poids θi soient appris. On va donc les mettre à jour en fonction de l’erreur faite par le modèle.

Tout d’abord on calcul l’erreur faites par la fonction Sigmoid par rapport au résultat attendu yd : err = yd – y

Ensuite on calcul de combien on va mettre à jour le poids θi : ∆θi = λ . err . xi avec λ un coefficient d’apprentissage.

Enfin on met à jour le poids : θi(t+1) = θi(t) + ∆θi

On met ainsi à jour tous les poids du modèle de façon à ce qu’ils s’adaptent en fonction de l’erreur qu’ils ont provoqués.

La régression logistique peut aussi se représenter sous forme de réseau de neurones où les neurones prennent un vecteur d’entré qu’il passe dans une fonction sigmoid et renvoi le résultat en sortie. Le réseau sera donc défini comme suit :

#### Implémentation

On utilise donc les images de visages de nos utilisateurs (en échelle de gris) sous forme de vecteur, c’est-à-dire avec les colonnes mise les unes à la suite des autres, pour alimenter ce modèle. L’implémentation de l’algorithme est faite « from scratch » en C.

#### Résultats

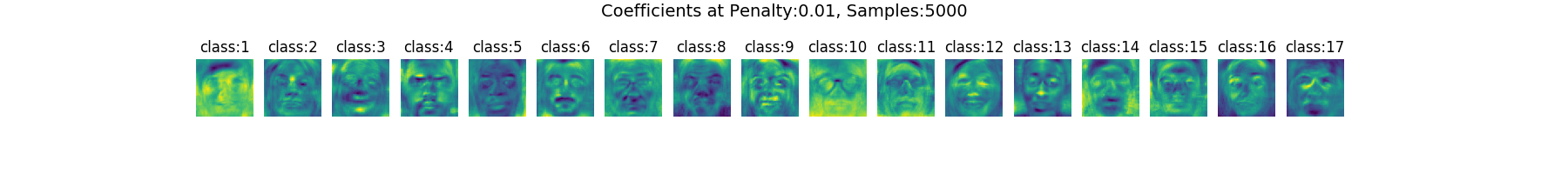
Après avoir passé l’entièreté du training set on obtient 33,2% de reconnaissance sur le test set.

Ce résultat très faible, mais au-dessus de l’aléatoire qui est à 6%, n’est pas étonnant car la régression logistique est une méthode de classification linéaire et notre problème semble être non linéaire.

Nous ne retiendrons pas cette méthode pour le système d’identification de Face Key.

Nous avons également essayé avec la bibliothèque sklearn qui utilise une version amélioré avec une fonction softmax à la place de la sigmoide et offre un affichage facile des résultats. Surprenament nous obtenons un score beaucoup plus élevé de 62.3% de reconnaissance sur 5000 exemple d’apprentissage.

Si l’on s’amuse à afficher les ensembles θ de chaque classe on peut voir quels pixels ont de l’importance dans la reconnaissance de chaque classe et on voit se dessiner des ébauches des visages.



### Méthode par perceptron multicouches

#### Théorie

#### Implémentation

#### Résultats

### Méthode par réseau de neurones convolutif

#### Théorie

#### Implémentation

#### Résultats

## Implémentation dans Face Key

## Conclusion : Avancement & améliorations possibles

# Sécurité

# Conclusion

1. medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78 [↑](#footnote-ref-1)
2. github.com/aleju/imgaug [↑](#footnote-ref-2)