Rapport de soutenance 3

QUBE

Clément Castiglione

Alexis Busson

Jacques Dai

Mathys Abonnel

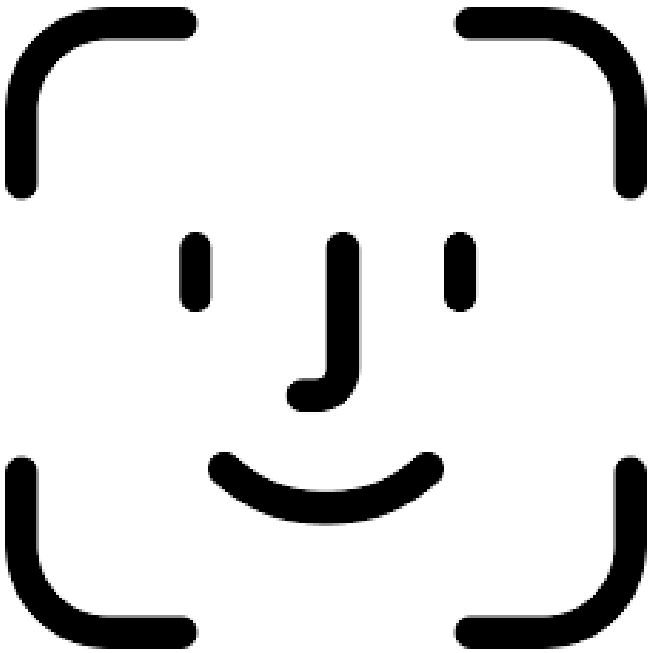


Table des matières

1 Introduction

Le projet QUBE, développé par une équipe d'étudiants d'Epita composée de Clément Castiglione, Alexis Busson, Jacques Dai, et Mathys Abonnel, représente un ambitieux défi technologique centré sur la reconnaissance faciale. Dès le début, notre objectif était de créer une application innovante qui pourrait identifier et analyser les visages avec une grande précision, tout en offrant une interface utilisateur intuitive et une expérience fluide. Ce projet a été motivé par notre passion pour l'intelligence artificielle et les technologies de pointe, et nous avons travaillé ensemble pour surmonter de nombreux défis techniques et de gestion. Les grandes étapes de ce projet sont : localiser un visage dans une image, reconnaitre un visage et la création de l'application.

2 Les technologies utilisées de nos jours

Les technologies utilisées de nos jours dans le domaine de la reconnaissance faciale ont considérablement évolué pour répondre aux demandes croissantes en matière de précision et de fiabilité. Les algorithmes d'apprentissage profond, tels que les réseaux de neurones convolutifs (CNN), sont devenus omniprésents pour la détection et la classification des visages. Ces réseaux sont capables d'apprendre des caractéristiques discriminantes à partir de grandes quantités de données d'entraînement, ce qui leur permet d'identifier efficacement des visages dans des conditions variables. De plus, l'utilisation de techniques de prétraitement d'images, telles que la normalisation et l'augmentation de données, contribue à améliorer la robustesse des modèles de reconnaissance faciale. Parallèlement, l'intégration de matériels spécialisés, comme les unités de traitement tensoriel (TPU) et les unités de traitement graphique (GPU), accélèrent les calculs nécessaires à l'inférence en temps réel. En combinant ces

avancées, les technologies actuelles offrent des performances remarquables en matière de reconnaissance faciale, ouvrant la voie à une gamme étendue d'applications allant de la sécurité biométrique à la personnalisation de l'expérience utilisateur.

3 Réalisation du projet

3.1 Détection des visages

3.1.1 Introduction

La détection de visages à été un processus enrichissant et moins complexe qu'il n'y parait.

La partie concernant la détection des visages sera illustrée à l'aide de l'image suivante :

Le choix de la bibliothèque a été un processus difficile pour nous. Bien que la bibliothèque OpenCV offre une large gamme de fonctionnalités, son utilisation ne semblait pas être la plus pertinente dans le cadre de notre projet scolaire à Epita. En tant qu'étudiants travaillant sur un projet de classe, nous avons estimé que l'achèvement de notre tâche en utilisant OpenCV ne serait pas très significatif. C'est pourquoi nous avions décidé d'opter pour une approche plus élémentaire en utilisant la bibliothèque "image" de Rust. Malheureusement, après avoir développé la grande majorité des fonctions, nous nous sommes retrouvés bloqués en raison de l'absence de Haar features disponibles. À cause

de cela, nous avons dû nous tourner vers la bibliothèque OpenCV. Grâce à nos recherches, nous avons désormais une bonne compréhension de son fonctionnement.

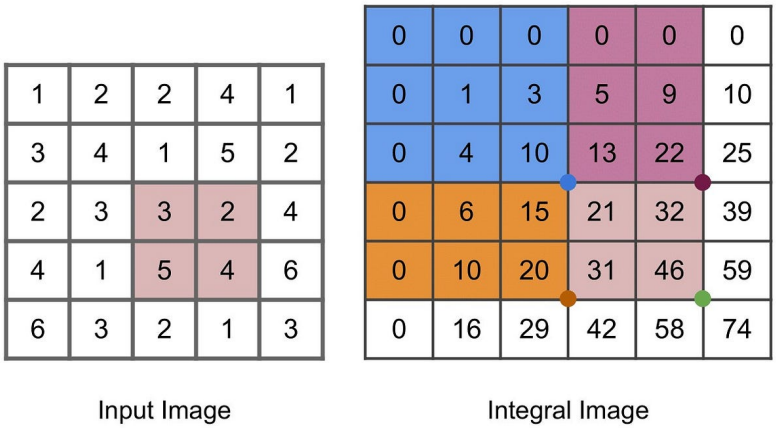
Le choix de la bibliothèque image de Rust a rendu la détection impossible et nous a fait perdre énormément de temps. En revanche, sa réalisation avec OpenCV nous a fait gagner beaucoup de temps. De plus, un avantage de ce choix réside dans l'apprentissage de l'utilisation de bibliothèques externes qui pourront nous être utiles dans nos projets futurs. En effet, la bibliothèque "OpenCV" nécessite le téléchargement de certains compilateurs externes, ce qui est particulièrement pratique pour obtenir un code efficace, peu importe le langage.

3.1.2 Réalisation première étape

Au début pour réaliser la détection de visages nous étions partis pour réaliser des fonctions ayant pour but de suivre l'algorithme de Viola-Jones et en appliquant des filtres similaire à ceux des haars cascades. Nous disposons d'une fonction qui transforme une image, représentée par un vecteur de pixels, en un vecteur d'entiers. Ces entiers sont compris entre 1 et 10 en fonction du niveau de luminosité des pixels correspondants. La création de ce tableau d'entiers a une complexité linéaire : pour chaque carré d'image, il suffit d'effectuer au maximum trois additions pour chaque pattern.

Le vecteur résultant représente l'intégrale de l'image, ce qui facilite la récupération des moyennes de pixels. Chaque zone de l'image peut être obtenue en effectuant une addition de trois composantes.

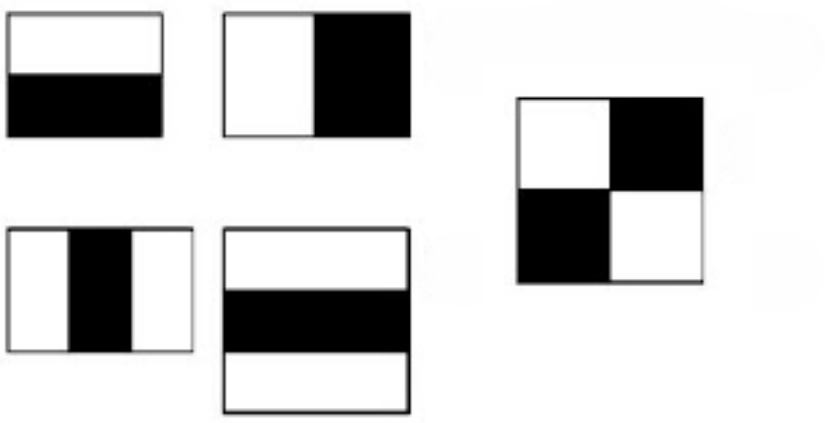
Voici un exemple de calcul de l'integrale de l'image :



Pour atteindre cet objectif,avons utilisé la méthode de détection d'objets de Viola-Jones. Cette approche classique de détection de visages dans les images s'appuie sur la détection rapide de caractéristiques simples, telles que les bords, les coins et les zones de contraste, par l'intermédiaire d'une représentation intégrale de l'image. En utilisant un classificateur en cascade d'Adaboost pour combiner et sélectionner les caractéristiques les plus pertinentes, la méthode de Viola-Jones peut identifier de manière efficace et rapide les visages dans des images, même dans des conditions de faible résolution ou d'éclairage variable. Nous avions également prévus d'explorer des techniques de prétraitement d'images pour optimiser les performances de la détection de visages. Cette approche aurait pu permettre une détection efficace des visages en minimisant le temps de traitement et en améliorant les performances globales de l'algorithme.

Notre implémentation, en dehors de celle réalisée avec OpenCV, est constituée de 6 structures qui permettent une compréhension rapide du code. Cela s'avère également plus pratique lors de la phase de développement. La structure principale est "IntegralImage", qui est caractérisée non seulement par ses paramètres mais aussi par ses implémentations. En effet, les fonctions "IntegralImage : :get-

pixel()" et "IntegralImage : :getsquare()" sont extrêmement pratiques. De plus, grâce à un travail conséquent pour gérer les erreurs, ces fonctions renvoient précisément le problème rencontré. Tout le code utilise des types Result¡¿, ce qui permet de gérer proprement les erreurs en Rust. La dernière structure créée est la structure "Pattern". Elle contient une structure "Square", représentant un rectangle dans le vecteur "integralimage", ainsi qu'une liste de ratios qui contient une liste de rectangles représentant les zones noires. Cela nous permet de valider ou non le filtre, en vérifiant si les zones noires du filtre correspondent à ce qui est attendu. Pour réussir à créer et utiliser les patterns OpenCV. Voici une image d'exemple de Ratio :



3.1.3 Réalisation deuxième étape

Nous avons rencontré un problème majeur. Malgrés le fait que nous avions toutes nos fonction pour suivre l'algorithme de Viola-Jones telles que le grayscale, les square qu'on découpe pour analyser l'image ainsi que les filtres et autres, il nous manquait le point principal : Les patterns. Pour pouvoir détecter un visage il faut appliquer une succession de pattern sur l'image transformée. Cependant cela demande des milliers de patterns à avoir. Il est évident que ce serait

une grosse perte de temps que de les refaire. Nous avons donc décider de prendre les patterns de haars cascade pour les appliquer sur nos image. Mais au final nous ne pouvions pas les utiliser, nous ne comprenions pas comment le fichier xmml fonctionnait pour pouvoir s'en servir correctement. Donc nous avons opté pour utiliser la fonction de détection de visage que propose OpenCV. C'était dur à encaisser car au final tout ce que nous avions fait n'a pas servi à faire avancer le projet. Mais ce n'est pas perdu, nous avons pu mieux comprendre son fonctionnement et n'avons pas non plus betement utilisé la bibliothèque.

Donc voila pour la deuxième étape du projet nous avons utilisé la fonction d'opencv detect-multi-scale() pour detecter un visage sur l'image donné en paramètre et transformé en noir et blanc au préalable. Nous extrayons simplement l'image et la transformons en png afin de pouvoir la visualiser. Cependant nous avons remarqué un manque de précision de l'algorithme, certaine fois il détectait des zones qui ne sont pas de visage, surement due a la luminosité. Et certaines fois il détectait pas les visages trop petit. Nous savons à quoi cela est due, lors de l'appel de la fonction on peut choisire la taille minimale et maximale des visages, il suffit de la modifier. Mais étant donné que ce n'est pas l'objectif du projet que de s'amuser à détecter des visages aussi petits qu'il soit, nous n'avons pas renforcé le parcours de l'image en appelant plusieurs fois la fonction avec différentes scales. L'utilisateur est sensé se présenter à une distance raisonnable de la caméra.

3.1.4 Réalisation dernière étape

Pour pouvoir poffiner le travail nous avons fait en sorte de pouvoir détecter plusieurs visages à la fois. Pour cela il a juste fallu exploité la sortie de la fonction de détection de visages pour enregistrer en png chaque image de la liste récupérée. Et pour éviter que l'image enregistrée n'écrase la précédente on

avait juste a modifier la chaine de charactères avec un simple compteur.

Voila nous sommes désormais capables d'extraire deux visages d'une image comme celle ci :

Nous pouvons simplement les envoyer à notre IA désormais.

3.1.5 Bilan de ce qui a été réalisé

Nous avons pu produire un algorithme simple et efficace pour détecter des visages sur une image. Nous pouvons les enregistrés en pdf pour les envoyer automatiquement dans un dossier afin que l'IA puisse les récupérer pour les traiter.

3.1.6 Opencv, ses joies et ses peines

OpenCV était la solution évidente pour améliorer la précision de notre détection de visages. Malheureusement, son utilisation nous a pris énormément de temps car ce n'est pas une bibliothèque native de Rust. Le projet ne compilait pas correctement et recommandait l'utilisation d'un compilateur C (en l'occurrence Clang), mais malgré cela, l'erreur persistait sans que nous en comprenions

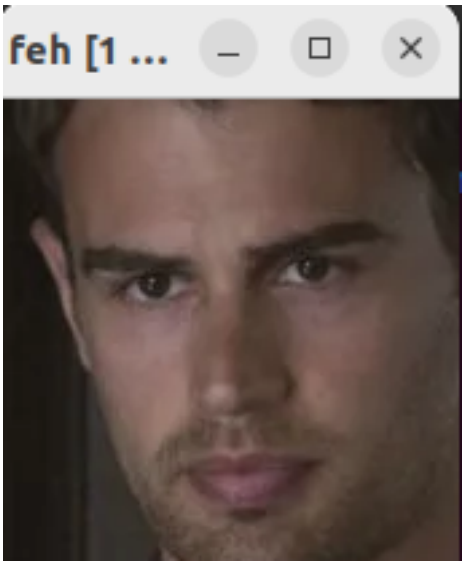
la raison. Après de longues recherches sur des forums, nous avons découvert qu'il fallait installer la bibliothèque clang-dev depuis le dépôt officiel et spécifier le chemin vers ce compilateur lors de la construction. Malgré cela, d'autres erreurs sont survenues, mais celles-ci étaient moins mystérieuses que la précédente. Après de multiples manipulations, la bibliothèque était enfin opérationnelle et utilisable sans problème.

3.1.7 Ressenti Alexis et Mathys

Pour ce projet nous avons pu apprendre d'avantage de chose sur la manipulation d'image. Le langage rust à beaucoup simplifier les choses, si on bloquait il nous diasait quelle bibliothèque utiliser et comment. Le début du projet à été le plus enrichissant car on à tout fait en partant de zéro. Ce qui nous à fait apprendre les fondement meme de la détection de visage. Cependant la bibliothèque OpenCV ainsi que celle d'image de rust on beaucoup trop simplifié les choses. Tout était déjà fait de manière soujacente. On avait juste à les utiliser sans se poser de questions. Si nous avions su nous nous serions fixés d'avantage de challenge pour ce projet en rajoutant des fonctionnalitées, afin de nous faire progresser même si nous avons déjà pas mal appris de nouvelles choses. Ce fut intéréssant et plutot amusant.

3.1.8 Conclusion

Grâce aux patterns d'OpenCV, nous sommes désormais capables de prendre une image en paramètre et de générer une nouvelle image ne comportant que la tête trouvée sur l'image initiale. Cette image pourra ensuite être transmise à notre IA pour traitement. Voici le résultat produit par notre programme sur une image donnée en paramètre. (C'est celle présenté en début de partie)


feh [1 ...

3.2 UI

3.2.1 Introduction

Pour concevoir l'interface graphique de notre application, nous avons opté pour la bibliothèque GTK-rs. Cette décision s'est appuyée sur notre expérience antérieure avec cette bibliothèque lors du projet de s3, en plus de sa maturité et de sa documentation approfondie, facteurs susceptibles de favoriser une rapide progression du développement de l'application.

Dans un premier temps, l'utilisateur pourra s'enregistrer soit en important des images de son visage, soit en utilisant la webcam de son ordinateur pour prendre des photos à l'intérieur même de l'application. Dans un second temps, afin de montrer les capacités de notre reconnaissance faciale, de nouvelles images pourront être prises ou importées, l'application donnera un retour en fonction de si oui ou non, il s'agit de la même personne enregistrée en premier lieu. Voici le site de la bibliothèque : `https://gtk-rs.org` Voici un site faisant part de l'état de l'art du gui en rust qui m'a aiguillé dans le choix de cette bibliothèque : `https://blog.logrocket.com/state-rust-gui-libraries`.

3.2.2 Réalisation

La réalisation de l'interface graphique a été une phase cruciale et enrichissante de ce projet, dans laquelle nous avons utilisé la bibliothèque gtk4-rs. Cette bibliothèque, bien qu'elle soit construite sur la base de GTK 3 en langage C, présente des spécificités qui ont nécessité une compréhension approfondie. L'une des différences notables réside dans les fonctions de rappel (callbacks) qui sont désignées autrement, avec la disparition ou l'obsolescence de certaines d'entre elles. Un défi majeur a été de constater l'absence de gpointer pour transmettre des variables à modifier dans les fonctions de rappel. Pour contourner cette limitation, nous avons adopté l'utilisation de Cell et RefCell, des mécanismes permettant de passer et de sauvegarder les variables lors des appels de fonctions de rappel.

Au fil du développement, il est devenu évident que cette bibliothèque est relativement nouvelle, ce qui se reflète dans la qualité de la documentation disponible. Souvent, les informations sont soit obsolètes, soit insuffisamment détaillées, avec de nombreuses fonctions manquant d'explications claires. Il est à noter également qu'il existe très peu de tutoriels sur l'utilisation de GTK4 avec Rust, bien que quelques ressources aient été trouvées, elles étaient souvent dépassées. Cependant, certaines de ces ressources se sont révélées précieuses pour comprendre les fonctionnalités non documentées.

Pour définir les éléments présents dans l'interface utilisateur (UI), j'ai utilisé la commande gtk4 tool simplify pour convertir le fichier .glade en fichier .ui, éliminant ainsi les balises XML incompatibles avec GTK4. Glade et les fichiers .ui sont tous deux écrits en XML et permettent de spécifier l'emplacement des éléments de l'interface utilisateur.

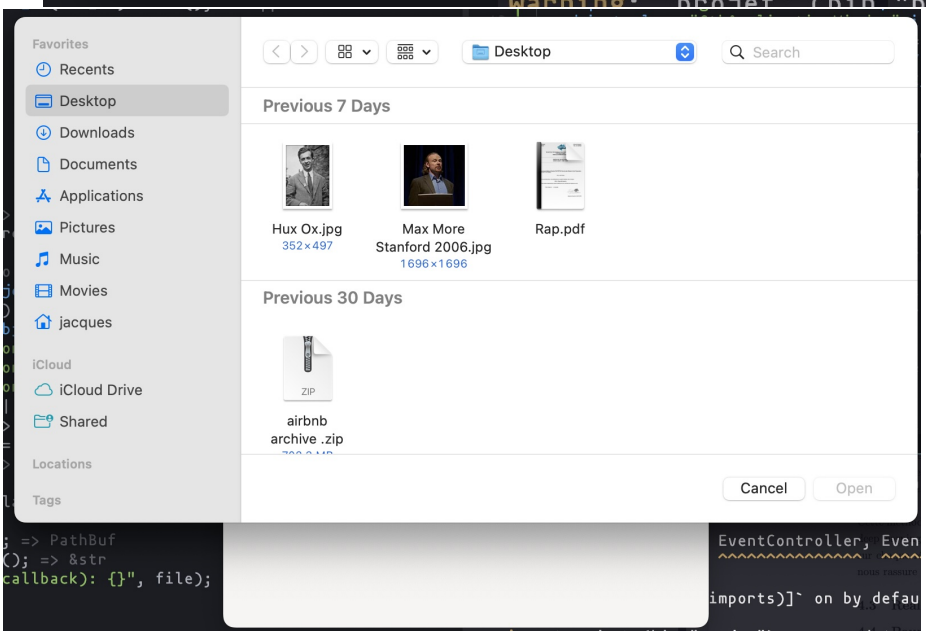
Un aspect particulièrement chronophage de ce projet a été de comprendre le fonctionnement de la bibliothèque GTK4 et de traduire les pratiques établies avec GTK3. Par exemple, dans GTK3 en langage C, l'élément d'interface utilisateur "filechooser" est devenu obsolète en GTK4, remplacé par "file dialog" avec une fonction "open" pour ouvrir la fenêtre de sélection de fichier. Cette migration a demandé du temps et de la patience pour s'adapter aux changements de fonctionnalités et de nomenclature, ainsi que pour maintenir la cohérence et la compatibilité avec les versions précédentes du logiciel.

Pour instant, ce qui à été fait sont , le ui avec 4 buttons dont un button qui permet de choisir l'image à afficher et la zone à afficher. Pour les prochaines soutenances, j'implémenterai les couleurs de l'interface et faire en sorte que la taille de l'image s'adapte dynamiquement à la taille de la fenêtre.

Pour affichage de la caméra j'ai pris beaucoups de temps pour trouver la librairie qui permettent d'intégrer les fonctions d'une camera dans gtk4, en essayant plusieurs librairies, tout ces librairies manquait soit des exemples de code obsolètes ou qui ne marchait pas tou court soit une documentation incomplète sur les fonction. Puis j'ai aussi essayé d'exécuter sur archlinux qui non plus de marchait pas . Le premier librairie que j'ai essayé n'avait pas d'intégration avec gtk4 nativement , il fallait spécifiquement un plugin appelé gst-plugin-gtk4 que je croyait n'existait pas sur cargo (faute d'indications sur comment l'utiliser) et donc qu'il fallait compiler manuellement avec cargoc-build et le compilateur clangd de c car tout la librairie est un binding de c donc qui appel des fonctions écrites en c et aussi installer les librairies en c sur brew ou sur un package manager .

Tout cela ne marchait pas sur Macbook pro m1 donc j'espère que tout cela marche sur archlinux en x86 . En plus j'ai essayé de compiler un projet qui essayait de faire la même chose que moi mais cela n'a pas compilé . Puis j'ai essayé de trouver une solution pour faire marcher le code en s'inspirant du code donné dans la repo de gst-plugin-gtk4 . Ce qui m'a bloqué le plus est le fichier cargo toml car pour chaque package il fallait mettre plus d'informations qu'au paravent la version de celui-ci suffisait. Cela a crée plus de confusion sur quelle version et pkg il fallait mettre pour que tout fonctionne. Grâce au lsp le language serveur protocole installé sur neovim qui corrige les erreurs de syntaxe et au documentations rudimentaires j'ai pu m'en sortir avec un interface graphique sur gtk4-rs qui intègre une webcam grâce à gstreamer-rs. Avec toute cette expérience de développement, je peux voir pourquoi la transition du c au rust prend autant de temps, les librairies sont souvent des bindings du c qui sont pas codé totalement en rust , et aussi les documentations sont souvent rudimentaires voire inexistante vu la jeunesse de celles ci. De plus, je constate un manque de tutoriel pour cette language et le peu de tutoriels ou d'exemples de code qui existent sont souvent obsolètes par exemple un code qui date de quelques mois avant ne ressemble pas du tout à celui d'aujourd'hui cela montre à quelle point ces librairies sont nouvelle et en constante évolution ce qui rend le développement difficile et plus lente car on doit souvent scruter le code source , deviner le fonctionnement du code , calquer sur du c , testé le code ...


GTK Window
Button 1
Button 2
Button 3
Button 4


Favorites
Recents
Desktop
Downloads
Documents
Applications
Pictures
Music
Movies
jacques
iCloud
iCloud Drive
Shared
Locations
Tags
Desktop
Search
Previous 7 Days
Hux Ox.jpg
352×497
Max More Stanford 2006.jpg
1696×1696
Rap.pdf
Previous 30 Days
ZIP
airbnb archive .zip
Cancel
Open

3.3 Reconnaissance faciale

3.3.1 Présentation

Cette étape a pour but de donner une distance entre deux visages qui pourra ensuite être utilisé avec un simple mécanisme de seuil pour considérer que oui ou non les deux visages présentés sont les mêmes. Le process intervient après la localisation du visage dans l'image et comparera l'image extraite du visage avec les différentes images de l'utilisateur préalablement enregistré.

Au vu de la difficulté de la tâche, uniquement des méthodes de deep learning sont envisageables pour extraire les caractéristiques d'un visage. Dans un premier temps, nous avons prototypé un réseau de neurone convolutif (CNN) de taille réduite afin de se familiariser avec la création de réseau de neurones en Rust. nous utilisons la bibliothèque tch-rs. Elle consiste en des liaisons Rust pour l'api C++ de PyTorch. Cela permet pour nous d'être en terrain connu et de s'assurer l'accès à une documentation riche. En effet, la seule alternative, le seul autre framework de Deep Learning en Rust, Burn, est encore à un stade primaire de développement. Nous pensons que ce n'est pas une bonne initiative que de baser notre projet sur des bibliothèques en "beta". L'idée avec ce CNN était de l'entrainer de manière traditionnelle pour reconnaitre les différentes caractéristiques d'un visage. Une fois privé de ses derniers niveaux totalement connectés, le réseau aurait servi à mapper des visages dans un espace à plusieurs dizaines de dimensions, un espace certes abstrait, mais dans lequel on aurait pu calculer des distances euclidiennes. Malheureusement, après de plus amples lectures, cette approche est possible mais demanderait beaucoup trop de postprocessing car toutes les dimensions de cet espace abstrait ne se vaudrait pas, l'espace ne serait pas orthonormé. Pour palier ce problème, nous nous sommes tournés vers des architectures de réseau de neurones beaucoup plus conséquentes, les auto-encodeurs variationnels, qui sont encore à l'état de l'art

dans certaines applications.

3.3.2 VAE

Dans un premier temps, pour comprendre ce qu'est un auto-encodeur variationnel, nous allons voir ce qu'est un auto-encodeur. Un auto-encodeur est une famille d'architecture de réseaux de neurones spécialement faite pour comprendre et résumé les caractéristiques de ses entrées (images, texte, scalaires, etc.). Cette architecture est composée de deux blocs, l'encodeur et le décodeur qui sont bien souvent le reflet de l'autre. L'encodeur réduit la dimensionnalité de l'information jusqu'à un goulot d'étranglement, puis le décodeur retrouve l'information initiale à partir des sorties de l'encodeur. On dit que le goulot d'étranglement projette l'information dans un espace latent, car toutes entrées similaires sont proches au sens de la distance dans cet espace. On peut imaginer utiliser ces modèles dans un scénario de compression-décompression de données.

Un VAE dérive d'un auto-encodeur par l'homogénéité qu'il garantît par ses projections dans l'espace latent. Similaire à un auto-encodeur par sa forme de sablier, il diffère en son cœur afin de garantir cette homogénéité. En effet, l'encodeur ne donne pas directement un point dans l'espace en sortie, mais une distribution normale dans cet espace, ceci à l'aide d'une moyenne et du logarithme de la variance (car plus simple à apprendre). Un point peut ensuite être tiré aléatoirement de cette distribution qui servira d'entrée au décodeur lors de l'entrainement. Un autre ajout permet les capacités d'un VAE, sa fonction coût. Celle-ci est composée de deux parties, la première, traditionnelle, compare l'entrée et la sortie du VAE et juge sa capacité à reconstruire l'information, la seconde, calcule la divergence de la distribution donnée par l'encodeur vis-à-vis de l'origine du repère de l'espace latent et juge à quelle point le modèle "étale" sa représentation, est utilisé pour cela la divergence de Kullback-Leibler. Ces deux

paramètres sont pondérés en fonction de l'importance portée à la capacité à reconstruire l'information où à l'homogénéité de l'espace latent. On peut imaginer comme cas où on porte plus d'attention à la reconstruction de l'augmentation de donnée dans le cadre de l'apprentissage d'un CNN sur un jeu de donnée simple comme MNIST. On pourrait explorer l'espace latent d'un VAE afin de générer de nouveaux chiffres. Dans le second cas où on porte plus d'importance à la position des données dans l'espace latent, on peut imaginer un cas où on cherche à calculer des distances dans cet espace latent, ce qui requiert une disposition homogène dans l'espace, par exemple dans le cas de la reconnaissance faciale !

3.3.3 Protocole

Entrainé un réseau de neurones directement pour de la reconnaissance faciale et inenvisageable car la tâche et ardu et qu'il n'existe pas de dataset contenant un nombre suffisant de personnes avec un nombre suffisant d'images de chacune de ces personnes dans des contextes suffisamment variés pour envisager entrainer directement un réseau de la sorte. L'idée est qu'au lieu d'espérer un réseau magique sortant une probabilité que deux visages soient similaires, on ait un réseau capable d'extraire les caractéristiques d'un visage qui pourront ensuite être comparées et traitées de manière séparée.

Notre jeu d'apprentissage, basé sur CelebA, est composé de plus de deux-cent-mille images de visages de plus de soixante-mille personnalités. À l'aide d'un script les images ont été recadrées en carré de 128x128 pixels centré sur le visage et zoomé afin que ceux-ci emplissent l'entièreté du cadre. La taille de 128x128 est un compromis entre la richesse de l'information et la vitesse d'apprentissage de notre modèle.

Notre réseau de neurones, notre VAE peut ensuite s'entrainer sur ce jeu de données de manière auto-supervisé, en effet l'apprentissage ne nécessite pas de labelles. L'encodeur prend une image en entrée et réduit sa dimensionnalité, le décodeur retrouve l'image initiale, les deux images en entrée et sortie sont comparées ce qui permet de calculer les dérivées en chaine de la backpropagation du décodeur jusqu'à l'encodeur.

une fois que nous avons un outil pour représenter chaque visage dans un espace le plus homogène possible, nous pouvons utiliser une simple distance euclidienne entre deux points de l'espace latent et en définissant de manière empirique un seuil, nous pouvons prédire si deux visages sont similaires. Si cette option n'est pas fructueuse, une alternative est possible, entrainer un petit réseau de type MLP afin de donner une mesure de distance, de ressemblance.

Cette méthode est similaire à ce qui se fait depuis des années dans le domaine du deep learning et ne présente pas de risque d'échec en soi, la difficulté se présente sur chaque réalisation individuelle plus que sur l'ensemble du protocole ce qui nous rassure afin de mener à bien ce projet.

3.3.4 Avancement

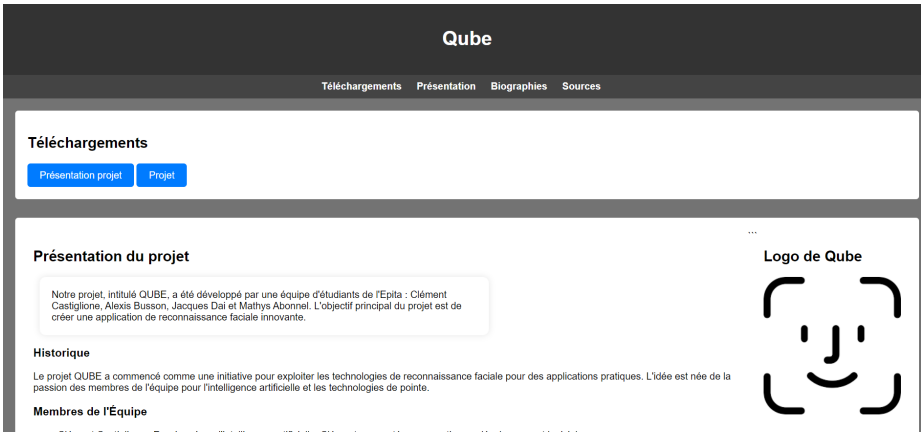
Nous avons jeté notre ancien simple classifier après avoir décidé de construire un VAE car il ne nous est plus utile. Fort de notre expérience avec tch-rs, nous avons pu créer un VAE de presque 19 millions de paramètres. Celui-ci est composé de couche de convolution puis de déconvolution en chaine. Pour ses performances reconnues, nous utilisons l'optimizer Adam. Son architecture est donnée en annexe.

Notre réseau atteint de bonnes performances après quatre heures d'entrainement en utilisant l'accélération gpu permise par CUDA sur une RTX 3080. Bien que les performances du modèle soient difficilement quantifiables car donner la valeur de la fonction coût n'a pas beaucoup de sens dans notre cas d'apprentissage auto-supervisé. Voici ci-dessus une image comparant différente reconstruction de visages, donnant un ordre d'idée de la performance du modèle et sa capacité à comprendre les composantes d'un visage.

À l'aide de cette représentation dans l'espace latent du modèle, nous pouvons calculer nos distances euclidiennes. Il nous reste plus qu'à définir un seuil expérimental pour trancher si deux visages en sont en fait qu'un. Nous comptons utiliser un simple algorithme de machine learning à cet effet. Ensuite, le réseau pourra être amélioré en retirant les visages qui ne sont pas face à la caméra

du jeu de donnée, en rajoutant des transformations sur le jeu de données et en augmentant la taille du réseau.

3.4 Site web



3.4.1 Introduction

Le projet QUBE, que nous avons développé, vise à créer une application de reconnaissance faciale innovante. Durant la conception et le développement du site web pour présenter ce projet, nous avons rencontré plusieurs défis techniques et de gestion. Ces défis ont non seulement testé notre expertise en développement web, mais nous ont également poussés à trouver des solutions efficaces pour garantir la performance, la sécurité.

3.4.2 Problèmes

Un défi majeur était la compatibilité mobile. Bien que notre site fonctionnait correctement sur les navigateurs de bureau, il posait des problèmes sur les smartphones et les tablettes. Le site n'est pas responsive, ce qui signifie que les utilisateurs doivent faire défiler horizontalement pour voir le contenu. Les éléments de l'interface utilisateur étaient souvent mal alignés ou trop petits pour être utilisés confortablement sur des écrans tactiles.

La collaboration au sein de l'équipe de développement a posé des défis. Travailler à distance et coordonner les efforts de plusieurs membres de l'équipe a nécessité l'utilisation d'outils de gestion de projet (GitHub).

L'hébergement du site sur GitHub a également présenté des défis uniques. Bien que GitHub Pages offre une solution gratuite et facile à utiliser pour héberger des sites statiques, il y a eu des problèmes avec les chemins relatifs et l'accès aux ressources, nécessitant des ajustements manuels pour garantir que tout fonctionnait correctement.

3.4.3 Solutions

Pour améliorer la collaboration au sein de l'équipe, nous avons mis en place des pratiques de développement collaboratif plus rigoureuses, telles que des revues de code régulières et des sprints de développement clairement définis. L'utilisation de Git et GitHub a également aidé à gérer les versions et à résoudre les conflits de code efficacement.

Enfin, pour surmonter les défis liés à l'hébergement sur GitHub, nous avons soigneusement configuré nos chemins relatifs et testé le site après chaque déploiement pour identifier et corriger rapidement les problèmes. Nous avons également tiré parti des fonctionnalités de GitHub Pages pour automatiser le déploiement, ce qui a facilité la gestion des mises à jour et des nouvelles fonctionnalités.

4 Conclusion

Notre projet de création d'une application de reconnaissance faciale a été une expérience extrêmement enrichissante et constructive. Malgré les défis rencontrés, nous avons pu surmonter les obstacles grâce à notre persévérance et notre esprit d'équipe.

Le projet a débuté avec une phase de recherche et de sélection des technologies, où nous avons opté pour l'utilisation de bibliothèques comme OpenCV et des langages de programmation tels que Rust. La détection des visages, bien que complexe, a été maîtrisée grâce à une compréhension approfondie des algorithmes de détection et de la manipulation des images.

Nous avons également dû faire face à des problèmes techniques, notamment l'intégration de bibliothèques externes et la gestion des erreurs de compilation. Cependant, ces défis nous ont permis d'acquérir des compétences précieuses en résolution de problèmes et en développement logiciel.

Enfin, le travail d'équipe a été un atout majeur tout au long du projet. Chacun a apporté ses compétences et son expertise, ce qui nous a permis de progresser efficacement et de réaliser un produit final fonctionnel et satisfaisant.

Ce projet a non seulement renforcé nos compétences techniques, mais a également prouvé que la collaboration et la détermination sont essentielles pour réussir dans le développement de projets complexes. Nous sommes fiers du travail accompli et convaincus que les connaissances et expériences acquises seront bénéfiques pour nos futurs projets.

5 Tableau de répartitions des tâches

Taches	Responsable
UI	Jacques
Détection visages	Mathys & Alexis
Reconnaissance faciale	Clément

6 Planning de réalisation

Planning prévu avant la premiere soutenance :

Taches	Soutenance 1	Soutenance 2	Soutenance 3
Détection des visages	20%	60%	100%
UI	20%	60%	100%
reconnaissance faciales	20%	60%	100%

Planning actuel :

Taches	Soutenance 1	Soutenance 2	Soutenance 3
Détection des visages	40%	90%	100%
UI	20%	60%	100%
reconnaissance faciales	20%	60%	100%

Layer (type)	**Output Shape**	**Param #**
Conv2d-1	[-1, 64, 64, 64]	1,792
BatchNorm2d-2	[-1, 64, 64, 64]	128
LeakyReLU-3	[-1, 64, 64, 64]	0
Conv2d-4	[-1, 128, 32, 32]	73,856
BatchNorm2d-5	[-1, 128, 32, 32]	256
LeakyReLU-6	[-1, 128, 32, 32]	0
Conv2d-7	[-1, 256, 16, 16]	295,168
BatchNorm2d-8	[-1, 256, 16, 16]	512
LeakyReLU-9	[-1, 256, 16, 16]	0
Conv2d-10	[-1, 512, 8, 8]	1,180,160
BatchNorm2d-11	[-1, 512, 8, 8]	1,024
LeakyReLU-12	[-1, 512, 8, 8]	0
Conv2d-13	[-1, 1024, 4, 4]	4,719,616
BatchNorm2d-14	[-1, 1024, 4, 4]	2,048
LeakyReLU-15	[-1, 1024, 4, 4]	0
Linear-16	[-1, 128]	2,097,280
Linear-17	[-1, 128]	2,097,280
Linear-18	[-1, 16384]	2,113,536
ConvTranspose2d-19	[-1, 512, 8, 8]	4,719,104
BatchNorm2d-20	[-1, 512, 8, 8]	1,024
LeakyReLU-21	[-1, 512, 8, 8]	0
ConvTranspose2d-22	[-1, 256, 16, 16]	1,179,904
BatchNorm2d-23	[-1, 256, 16, 16]	512
LeakyReLU-24	[-1, 256, 16, 16]	0
ConvTranspose2d-25	[-1, 128, 32, 32]	295,040
BatchNorm2d-26	[-1, 128, 32, 32]	256
LeakyReLU-27	[-1, 128, 32, 32]	0
ConvTranspose2d-28	[-1, 64, 64, 64]	73,792
BatchNorm2d-29	[-1, 64, 64, 64]	128
LeakyReLU-30	[-1, 64, 64, 64]	0
ConvTranspose2d-31	[-1, 64, 128, 128]	36,928
BatchNorm2d-32	[-1, 64, 128, 128]	128
LeakyReLU-33	[-1, 64, 128, 128]	0
Conv2d-34	[-1, 3, 128, 128]	1,731
Sigmoid-35	[-1, 3, 128, 128]	0

TABLE 1 – Model architecture details