Rapport de soutenance 2

QUBE

Clément Castiglione

Alexis Busson

Jacques Dai

Mathys Abonnel

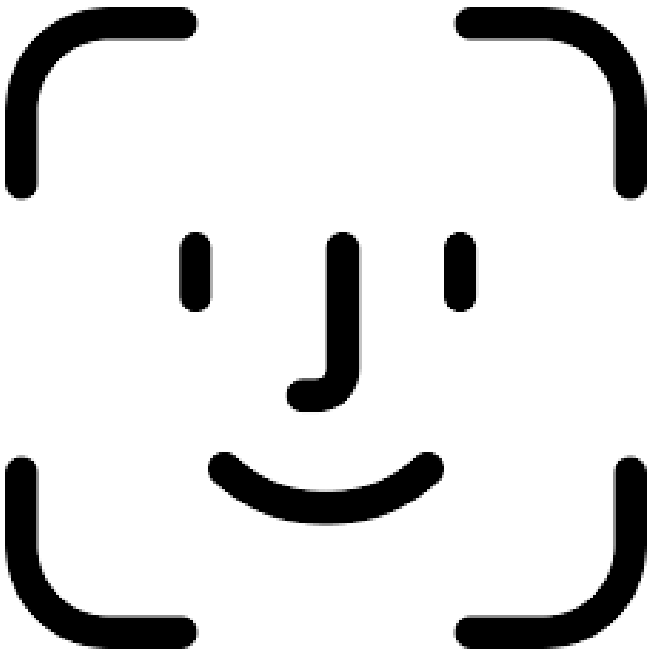


Table des matières

1 Introduction

Notre projet vise à créer une solution de reconnaissance faciale rapide et robuste dont des démonstrations pourront être effectuées à l'aide d'une application graphique. Les grandes étapes de ce projet : 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 Tâches à réaliser

3.1 Détection des visages

3.1.1 Introduction

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 avons 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 fonctionne-

ment.

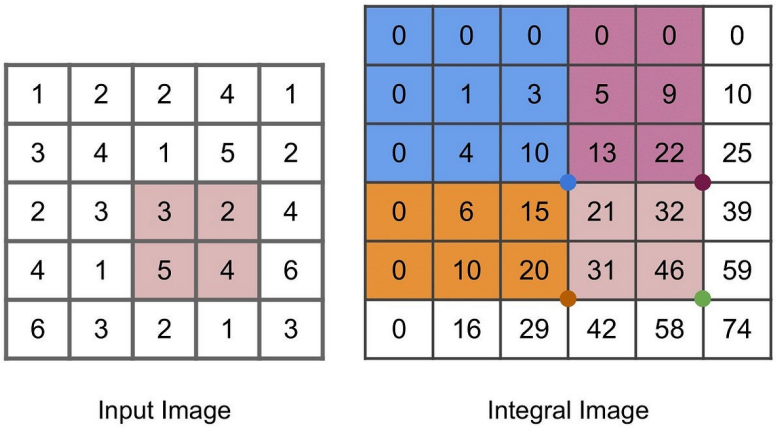
Le choix de la bibliothèque image de Rust a rendu la détection impossible et nous a fait perdre énormément de temps. En revanche, sa réalisation avec OpenCV nous fera gagner beaucoup de temps. De plus, un avantage de ce choix réside dans l'apprentissage de l'utilisation de bibliothèques externes qui 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

Actuellement, 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 :

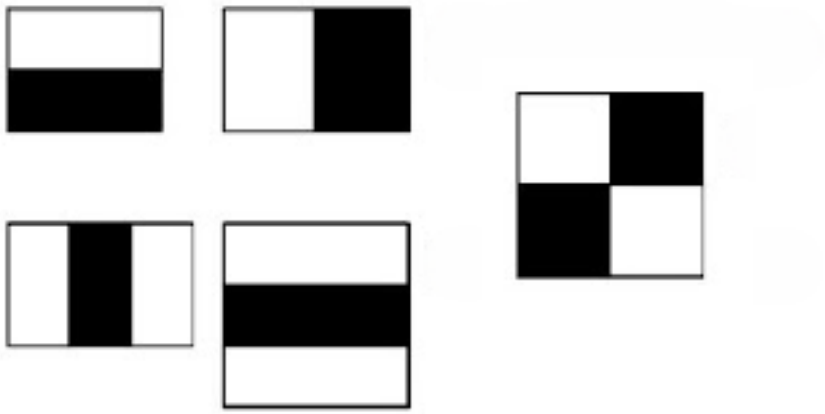


Pour atteindre cet objectif, nous comptons notamment utiliser la méthode de détection d'objets de Viola-Jones. Cette approche classique de détection de visages dans les images s'appuie sur la détection rapide de caractéristiques simples, telles que les bords, les coins et les zones de contraste, par 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 prévoyons également d'explorer des techniques de prétraitement d'images pour optimiser les performances de la détection de visages. Cette approche pourrait permettre une détection efficace des visages en minimisant le temps de traitement et en améliorant les performances globales de l'algorithme.

Pour l'instant notre programme se concentre sur le traitement d'image, ce qui est essentiel pour la détection ultérieure des visages. Rien ne nous empeche de l'adapter à un flux vidéo. Qui est un enchainement d'image.

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 : :getpixel()" 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, nous devrons continuer d'essayer de transférer leurs données dans nos patterns. Voici une image d'exemple de Ratio :



3.1.3 Objectif

Pour la dernière soutenance, notre objectif principal sera la précision, visant à réduire, voire éliminer, le taux de faux visages détectés. De plus, il serait intéressant de donner des instructions à l'utilisateur, par exemple "approchez votre visage" ou "tournez légèrement la tête sur votre gauche".

Nous envisageons également de permettre à l'utilisateur de choisir le nombre de visages à détecter (maximum 2), ce qui offrira une flexibilité supplémentaire dans l'utilisation de notre programme. En cas de détection de plusieurs visages, nous pourrions envisager une comparaison des visages présents sur l'image. Par exemple, nous pourrions comparer les deux visages présents sur cette image.

3.1.4 Bilan de ce qui a été réalisé

Lors de la premiere soutenance nous avons pu mettre au point des outils nous permettant de détecter un visage. Cependant nous n'avions pas l'élément principal qui sont les patterns afin de reconnaitre un visage, ce qui était prévu. Nous avions atteint nos objectifs.

3.1.5 Avancement actuel

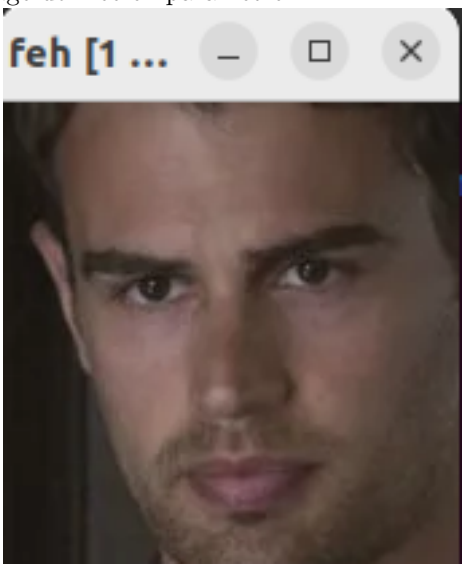
Comme mentionné précédemment, nous avons constaté un manque de patterns pour la détection de visages sur une image. Initialement, nous avions prévu de les créer nous-mêmes, mais nous avons réalisé que cette approche ne permettait pas une détection précise des visages, voire même qu'elle pouvait détecter des éléments qui ne sont pas des visages en raison du manque de précision. La création de milliers de patterns serait nécessaire pour obtenir une détection de visage adéquate, ce qui nous demanderait un temps colossal sans garantie de résultats probants. Nous avons donc choisi d'utiliser la bibliothèque OpenCV et de récupérer les patterns à partir du fichier XML officiel sur GitHub. Cela devrait nous offrir une précision accrue dans la détection de visages..

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 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. Nous sommes donc dans les temps pour cette deuxième soutenance. Voici le résultat produit par notre programme sur une image donnée en paramètre.



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

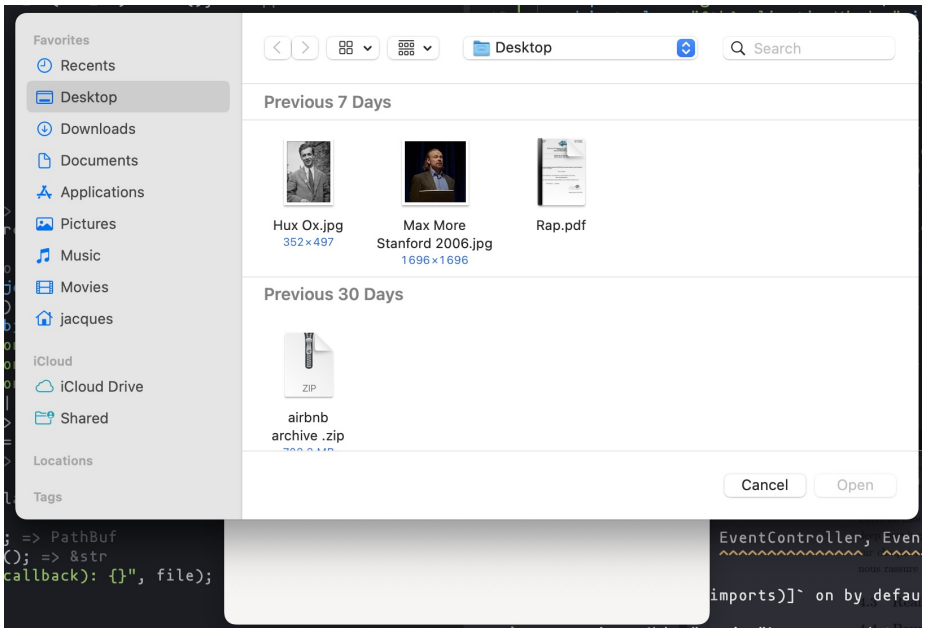
La réalisation de l'interface graphique a été une phase cruciale et enrichissante de ce projet, dans laquelle nous avons utilisé la bibliothèque gtk4-rs. Cette bibliothèque, bien qu'elle soit construite sur la base de GTK 3 en langage C, présente des spécificités qui ont nécessité une compréhension approfondie. 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.


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.

////////////////////////////////////

En effet, en utilisant une simple distance euclidienne sur cette "soupe" 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 existe, 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 soit, 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 nous sommes familiarisés avec tch-rs et avons créé un réseau de neurone convolutif (CNN) entrainé pour reconnaitre les dix catégories que contient CIFAR-10. Ce dataset de 60K images est composé de dix catégories (chien, avion, camion, etc.) utilisé pour benchmarker une architecture ou justement dans notre cas préentrainer un modèle. Le modèle, dont l'architecture suit, a été entrainé avec une logloss en fonction coût et Adam, optimizer réputé pour ses bons résultats avec une convergence rapide et des batchs de 32 images.

Ce modèle atteint une précision de 95% sur le jeu de test après 50 époques, ce qui prend une quinzaine de minutes. Nous sommes satisfaits des résultats et sommes confiants pour faire évoluer l'architecture du réseau en ajoutant de nouveaux blocs de convolution afin de le préentrainer sur CIFAR-10, voire

Type	Output Shape
Input	$[3, 32, 32]$
Conv2D $(3, 32, 3 \times 3)$	$[32, 32, 32]$
MaxPool2d	$[32, 16, 16]$
ReLU	$[32, 16, 16]$
Conv2D $(32, 64, 3 \times 3)$	$[64, 16, 16]$
MaxPool2d	$[64, 8, 8]$
ReLU	$[64, 8, 8]$
Conv2D $(64, 128, 3 \times 3)$	$[128, 4, 4]$
MaxPool2d	$[128, 2, 2]$
ReLU	$[128, 2, 2]$
Flatten	$[512]$
Linear $(512, 1024)$	$[1024]$
ReLU	$[1024]$
Linear $(512, 1024)$	$[10]$
Softmax	$[10]$

TABLE 1 – Architecture du CNN

CIFAR-100 puis transférer l'apprentissage sur CelebA afin de décortiquer les caractéristiques d'un visage.

4 Tableau de répartitions des tâches

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

5 Planning de réalisation

En avance sur la réalisation du projet, nous maintenons notre planning.

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