Classification d'âge et de genre en utilisant des réseaux de neurones convolutionnels (CNN)

Membres du groupe :

CABO India

GIFFARD Axel

OUCHALLAL Samia

HAMSEK Fayçal

Institut Universitaire de Technologie de Villetaneuse

Mr FAYE & Mme AZZAG

2025-2026

Table des matières

Introduction

La reconnaissance faciale humaine permet d'identifier différentes caractéristiques d'un individu, telles que son genre, son âge, voire parfois son origine ethnique, ainsi que d'autres attributs.

Parmi l'ensemble des techniques existantes consacrées à la reconnaissance faciale, l'utilisation des réseaux de neurones convolutionnels (CNN) est particulièrement adapté pour résoudre ce type de problème. Ces réseaux de neurones convolutionnels vont permettre l'extraction des motifs complexes sur les visages humains, comme les traits de visage, les rides pour détecter l'âge de la personne, les imperfections sur la peau, la barbe pour qualifier un homme, … Ce sont des caractéristiques pertinentes et cruciales pour détecter un genre et un âge. Cependant, bien que la détection soit possible, elle n'en est pas moins certifiée véridique tout le temps. En effet, en tant qu'humain, nous arrivons à estimer le genre et l'âge d'une personne, mais on ne le sait pas à 100%. Idem, pour le modèle qui aura de grandes difficultés sur l'estimation de l'âge d'une personne. L'usage du Deep Learning dans ce contexte permet à un modèle de reconnaître l'âge et le genre d'une personne suivant des caractéristiques complexes. Les CNN utilisent des couches de convolution pour appliquer des flitres afin d'en extraire ces caractéristiques (bords, textures, couleurs, …). De plus, ils utilisent d'autres couches tout aussi importantes comme les couches d'activation, de pooling pour diminuer le nombre de paramètres et améliorer la robustesse, des couches « fully connected » pour la classification, et même des couches de normalisation.

L'objectif de ce projet va être de réaliser 4 modèles différents en utilisant le jeu de données UTKFace, à savoir :

- Modèle de Classification de Genre avec CNN
- Modèle de Classification d'âge avec une approche de régression
- Modèle de Classification simultanée de Genre et d'âge
- Modèle pré-entraîné avec l'utilisation du *transfer learning* et comparatifs avec d'autres modèles pré-entraînés

A partir de ces modèles-là, le but va être de les analyser afin de trouver les points forts et axes d'améliorations.

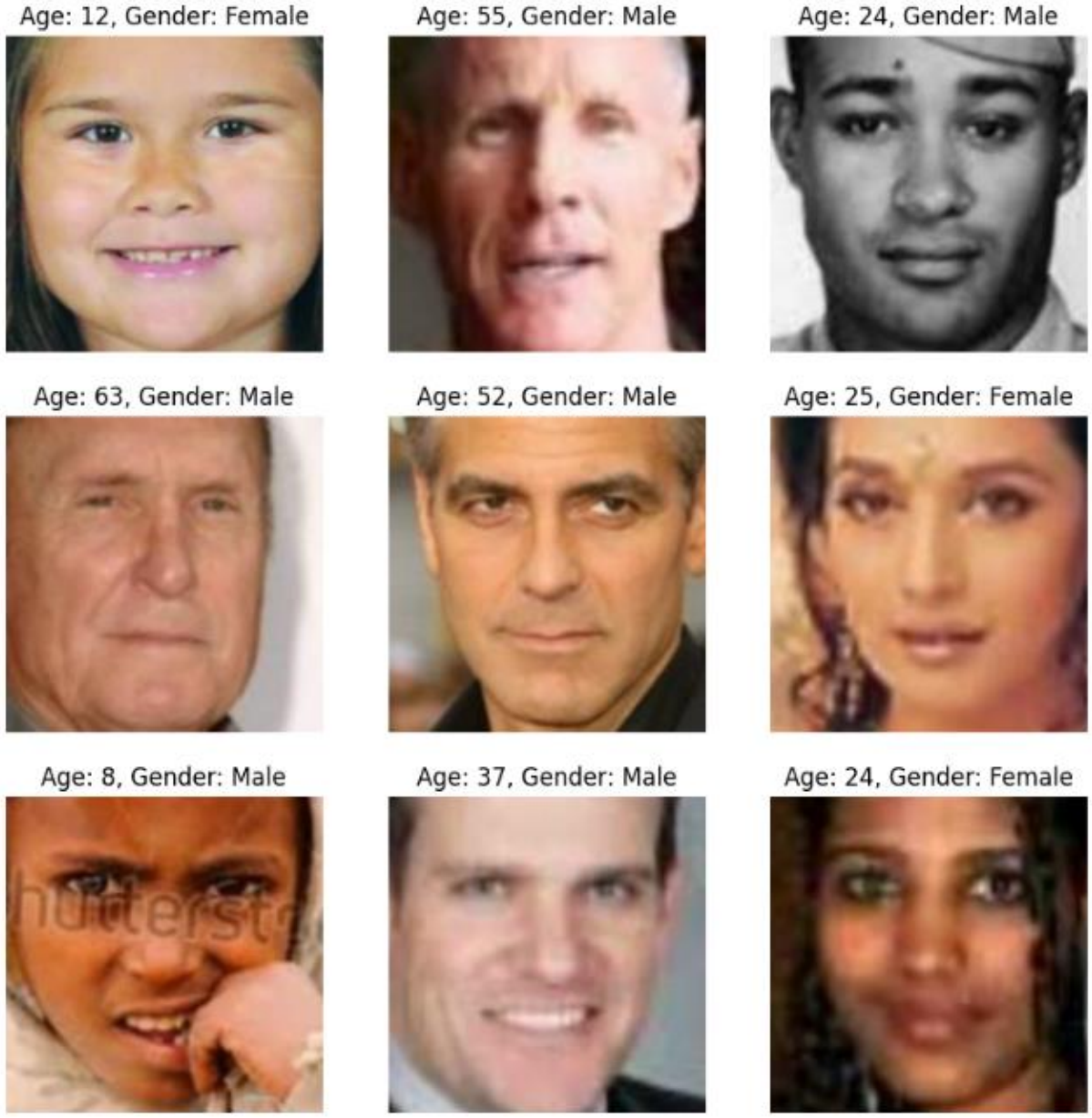
L'analyse des performances de chaque modèle sera cruciale. Nous utiliserons des métriques d'évaluation appropriées pour chaque tâche : l'accuracy et la matrice de confusion pour la classification de genre, et l'erreur absolue moyenne (MAE), l'erreur quadratique moyenne (MSE) et le RMSE (Root Mean Squared Error) pour la régression d'âge. Pour le modèle simultané, nous évaluerons les performances pour les deux tâches conjointement et séparément. Cette évaluation approfondie permettra de comparer les forces et faiblesses de chaque approche et de déterminer les axes d'amélioration potentiels, tels que l'ajustement des hyperparamètres, l'ajout de techniques de régularisation, ou l'exploration d'architectures plus complexes.

Par la suite, la création d'une interface utilisant la librairie Gradio va permettre de faciliter l'interaction Homme-Machine pour que les utilisateurs puissent tester les modèles.

Enfin, Ces modèles vont être hébergés sur HuggingFace pour les rendre accessibles en ligne.

UTKFace

Le dataset imposé à été UTKFace. C'est un dataset composé de plus de 20000 images (exactement 23708) allant de 0 à 116 ans. Ces images peuvent avoir des tons de couleurs différents, et des variations dans l'expression des visages.



Ces images sont en niveaux de gris ou en couleur RGB, avec des tailles variables, ce qui implique une étape de prétraitement des données. De plus, le dataset UTKFace présente une grande diversité en termes d'expressions faciales, et de conditions d'éclairage. Certaines images peuvent être légèrement floues ou occluses par des lunettes, des chapeaux, ou des cheveux, ajoutant un défi supplémentaire pour les modèles de reconnaissance faciale.

Concernant les annotations, chaque image est étiquetée avec l'âge et le genre de la personne représentée. L'âge est fourni en années, couvrant un large spectre, ce qui permet d'entraîner des modèles capables d'estimer l'âge sur une population variée. Le genre est étiqueté de manière binaire (0 pour homme, et 1 pour femme). Par exemple, le nom du fichier ci-dessous :

10_0_0_20170103200329407.jpg.chip.jpg

Indique :

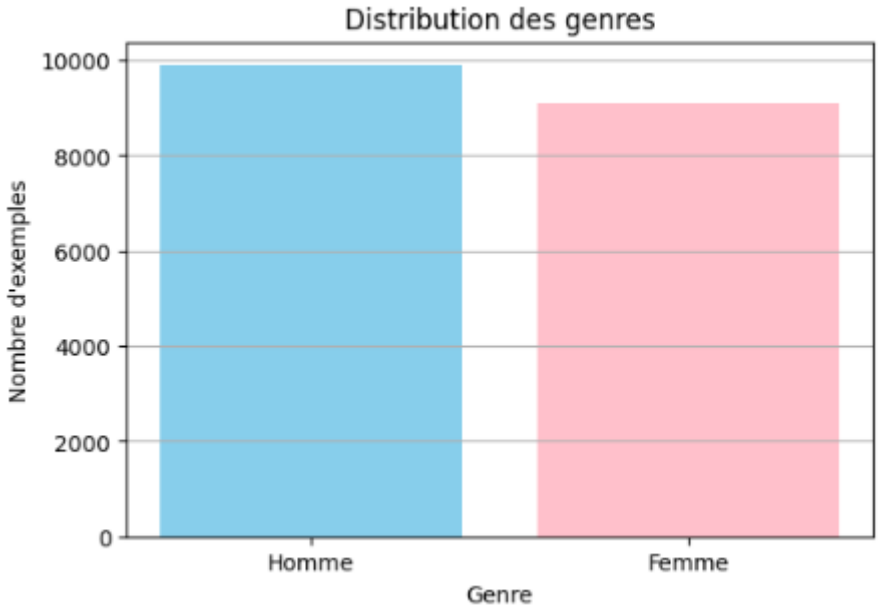
Age : 10

Genre : 0 -> Homme

Race : 0 (Non utilisé)

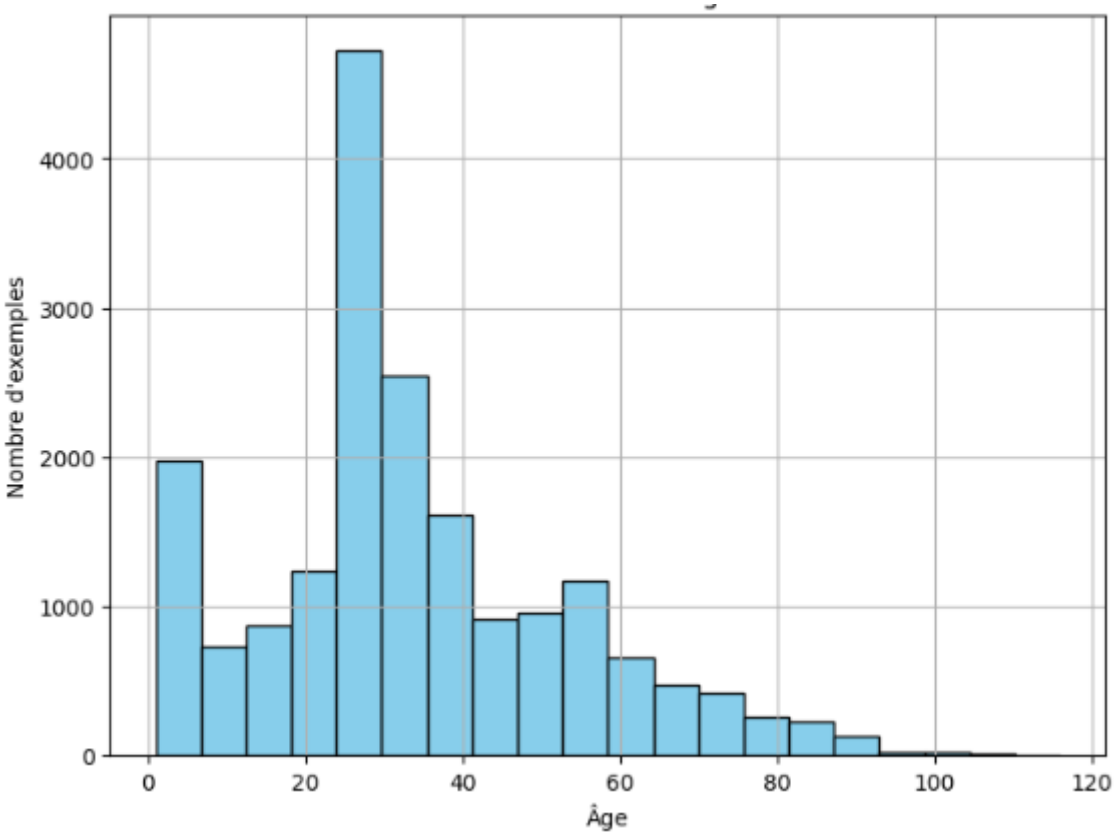
Proportions du dataset

Avant de nous orienter vers la définition des modèles, il est nécessaire de visualiser les distributions du Dataset par rapport aux différentes propriétés comme l'âge le plus fréquent, le nombre de femmes et d'hommes afin d'identifier d'éventuels déséquilibres entre les classes ou d'éventuels tendances dans les données. Cette analyse de données permettra par la suite d'optimiser les modèles en se basant sur les possibles biais et d'établir des stratégies d'équilibrage.



En se basant sur ce graphique montrant la distribution des genres entre hommes et femmes, on observe un nombre d'hommes plus élevées que le nombre de femmes dans le dataset (52.3% d'hommes et 47.7% de femmes). Cela peut conduire à un déséquilibre des classes, bien que cela reste minoritaire et bien échelonnée.

Cependant, la distribution des âges est une observation à prendre en compte, étant donné la diversité des images du dataset (0 à 116 ans).



Comme on peut l'apercevoir sur ce graphique montrant la distribution des âges, nous observons une surreprésentation de personnes ayant entre 20 et 40 ans. La moyenne à été fixé à 33 ans. De plus, on observe chez les personnes les plus âgés, une décroissance de la distribution de l'âge à partir de 60-70 ans. En conséquence, le modèle pourrait être plus performant pour estimer l'âge des personnes ayant entre 20 et 40 ans que pour estimer l'âge des personnes entre 60 et 116 ans.

En résumé, Il est important de tenir compte de cette distribution lors de l'entraînement et de l'évaluation des modèles, et d'être conscient du potentiel biais vers les jeunes adultes. Il sera susceptible d'évaluer les performances du modèle de l'âge par tranches d'âge pour ajuster au mieux le modèle. De plus, nous appliquerons de l'augmentation de données (Data Augmentation) pour améliorer la représentation des classes sous-représentés.

Méthodologie

Préparation des données

Nous préparons nos données en extrayant le genre et l'âge de chaque fichier et on charge les données pour les convertir en DataFrame avec la biblitohèque Pandas.

```python
def extract_age_gender(filename):
    """Extrait l'âge et le genre du nom de fichier UTKFace."""
    parts = filename.split("_")
    age = int(parts[0])
    gender = int(parts[1])
    return age, gender

def load_dataset(image_folder):
    """Charge les chemins d'image et labels depuis le dossier UTKFace."""
    image_files = [f for f in os.listdir(image_folder) if f.lower().endswith((".jpg", ".jpeg", ".png"))]
    image_paths = [os.path.join(image_folder, f) for f in image_files]
    ages, genders = zip(*[extract_age_gender(f) for f in image_files])
    return image_paths, list(ages), list(genders)

# Définir le chemin vers le dataset
DATA_DIR = "/kaggle/input/utkface-new/UTKFace"
image_paths, ages, genders = load_dataset(DATA_DIR)

# Convert to DataFrame for easier manipulation
df = pd.DataFrame({'image_path': image_paths, 'age': ages, 'gender': genders})

# Séparation initiale en ensembles train et validation (80% / 20%) avant le balancing
train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)

print("Taille initiale du dataset d'entraînement :", len(train_df))
print("Taille initiale du dataset de validation   :", len(val_df))
```

Les données sont ainsi divisées à l'aide de la méthode train_test_split. Nous avons choisi de diviser 80 % l'entraînement, et 20 % pour la validation pour avoir d'une part la même standardisation d'images, et d'autre part pour garantir un certain équilibre entre les classes.

Dataset	Nombre de données
Entraînement	18966
Validation	4742

Pré-traitement des données

L'objectif de cette partie, va être de rendre les images exploitables par les CNN. Les images sont redimensionnées à une taille uniforme : 64x64, 128x128 ou bien 224x224 pour le modèle pré-entraîné. Une fois leur conversion en tenseurs pour appliquer des algorithmes, une normalisation est nécessaire pour transformer les valeurs des pixels comprises entre 0 et 255 en valeurs beaucoup plus adaptés pour l'entraînement du modèle (0 à 1).

Pour les images, on va diviser par 255 les pixels :

```python
def preprocess_utkface(image, label, mode='train'):
    """Preprocessing and data augmentation function."""

    image = tf.image.convert_image_dtype(image, dtype=tf.float32)
    image = image / 255.0
```

Pour l'âge, on va diviser par l'âge maximale du dataset

```
normalized_age = tf.cast(age, tf.float32) / 116.0 # Normalization
return image, {'age_output': normalized_age, 'gender_output': tf.cast(gender, tf.float32)}
```

Conformément à nos recherches sur la taille de batch optimale, nous appliquons une taille de batch égale à 32 comme ce que conseille l'article suivant : wandb.ia de sorte à offrir un équilibre entre efficacité, stabilité de l'entraînement et contraintes matérielles, pour éviter d'épuiser la mémoire.

Une chose qu'on a pensé bien plus tard, a été d'augmenter les données. Au fur et à mesure des tests et des prédictions effectuées, nous avions remarqués que certains modèles commençaient à surapprendre (*overfitting*) et ne donnaient pas de résultats concrets. Une technique bien connue pour pallier ce problème là est la *Data Augmentation.* Son objectif est d'appliquer des transformations (Rotations, zoom, flip, etc…) pour rendre le modèle bien plus robuste.

La Data Augmentation étant propre à chacun des modèles, voici un exemple réalisé sur le modèle pré-entraîné d'EfficientNetB0 :

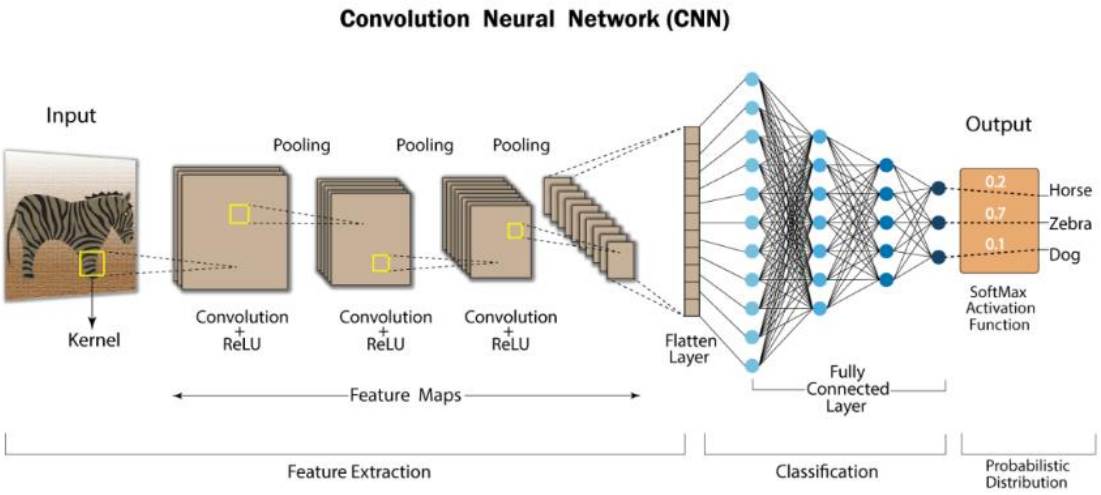
```
# Data Augmentation Layers (inchangées)
image = tf.image.random_flip_left_right(image)
image = tf.image.random_flip_up_down(image)
image = tf.image.random_brightness(image, max_delta=0.2)
image = tf.image.random_contrast(image, lower=0.7, upper=1.3)
image = tf.image.random_saturation(image, lower=0.8, upper=1.2)
image = tf.image.random_hue(image, max_delta=0.08)
image = tf.image.rot90(image, k=tf.random.uniform(shape=[], minval=0, maxval=4, dtype=tf.int32))

image = efficientnet_preprocess(image)
```

Les transformations appliquées incluent des flips, l'augmentation du contraste/luminosité/saturation/hue et également une rotation aléatoire sous des angles différents (0°, 90°, 180°, 270°). Enfin, l'image est également prétraitée avec la méthode efficientnet_preprocess pour adapter l'image aux exigences du modèle pré-entraîné. On observe le résultat sur le même set que la figure 1 :

Construction des modèles

Afin de respecter les besoins demandés, nous allons opérer sur des modèles basés sur l'architecture des CNN :



Les CNN appliquent 3 types d'opérations différentes :

- Convolution permettant d'extraire les motifs des images (Traits, couleurs, pilosité, …)
- Pooling permettant de réduire les dimensions tout en préservant l'information
- Fully Connected : Traditionnelles couches MLP qui effectuent la classification et/ou la régression suivie d'une fonction d'activation (généralement ReLu).

Un point crucial a été le choix des hyperparamètres. Ils sont « hyper » importants dans l'optimisation de chacun des modèles. Le *learning rate*, le *kernel*, le *stride*, le *padding*, et le *choix de l'optimiseur* sont à prendre en compte lors de l'élaboration de nos modèles. Il existe également des techniques comme l'ajout de droupout (pour éviter le surapprentissage en désactivant aléatoirement des neurones), ou de batchnormalization(pour stabiliser l'entraînement en normalisant les entrées de chaque couche) sont essentielles. Ils sont donc couplés avec l'optimiseur qu'on a choisi : Adam. Plus particulièrement, c'est le *learning rate* qui va faire nouer le lien avec l'optimiseur Adam. Enfin, chaque modèle a été entraîné entre 30 et 50 epochs.

Passons aux différents modèles avec chacun leurs spécificités, leurs architectures et leurs résultats.

Modèle 1. Genre :

Architecture du modèle

Le modèle prend en entrée une image redimensionnée au format (100x100) (initialement redimensionnée en 128x128) Pour un total d'1.5M de paramètres, voici l'architecture du modèle :

Layer (type)	Output Shape	Param #
input_layer_8 (InputLayer)	(None, 100, 100, 3)	0
conv2d_30 (Conv2D)	(None, 98, 98, 64)	1,792
batch_normalization_28 (BatchNormalization)	(None, 98, 98, 64)	256
max_pooling2d_30 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_31 (Conv2D)	(None, 30, 30, 128)	73,856
max_pooling2d_31 (MaxPooling2D)	(None, 15, 15, 128)	0
conv2d_32 (Conv2D)	(None, 13, 13, 256)	295,168
max_pooling2d_32 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_33 (Conv2D)	(None, 4, 4, 512)	1,180,160
max_pooling2d_33 (MaxPooling2D)	(None, 2, 2, 512)	0
flatten_4 (Flatten)	(None, 2048)	0
dense_20 (Dense)	(None, 512)	1,049,088
dropout_16 (Dropout)	(None, 512)	0
gender_output (Dense)	(None, 1)	513

Total params: 2,600,833 (9.92 MB)
Trainable params: 2,600,705 (9.92 MB)
Non-trainable params: 128 (512.00 B)

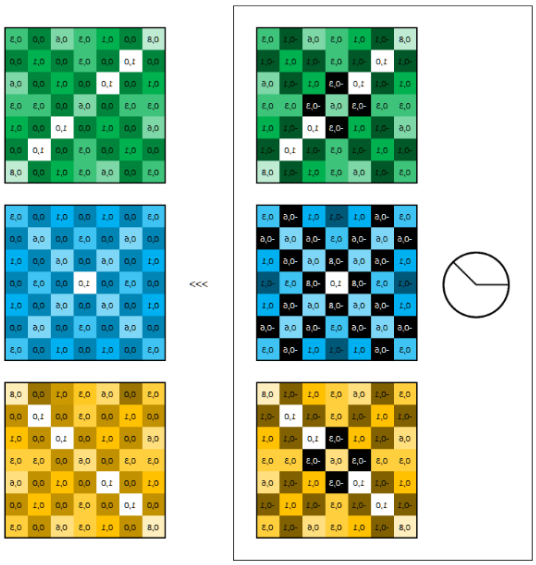
On prend en entrée des tailles d'images à 100x100.

Ce modèle enchaîne différentes couches de convolutions (64,128,256 et 512) en utilisant 4 MaxPooling2D pour extraire progressivement des caractéristiques visuelles.

Enfin, la sortie utilise une activation sigmoïde adaptée à la classification du genre, et la compilation en Adam avec un taux d'apprentissage de 0.0001 et une loss de type binary_crossentropy est cohérente pour ce type de tâche.

Comme le but de ce modèle est de seulement réaliser une classification du genre, nous produirons en sortie qu'un seul unit, qui est une valeur (entre 0 et 1) au format décimal de la vérification du genre.

La fonction d'activation utilisée est la fonction ReLU. Elle permet de transformer en *0* toutes les valeurs négatives et de conserver les valeurs positives. Un exemple de l'application d'une fonction ReLU sur 3 matrices (R,G,B) :



Nous appliquons la technique du max-pooling permettant de prendre la valeur maximale de chaque morceau de l'image.

Compilation et entraînement

Nous avons utilisé la loss est la fonction binary_crossentropy pour pénaliser les mauvaises prédictions incertaines, aidant le modèle à converger vers des résultats plus précis. Elle calcule donc la distribution prédite et la distribution cible qui permet d'ajuster les poids pendant l'entraînement.

L'optimiseur utilisé est Adam.

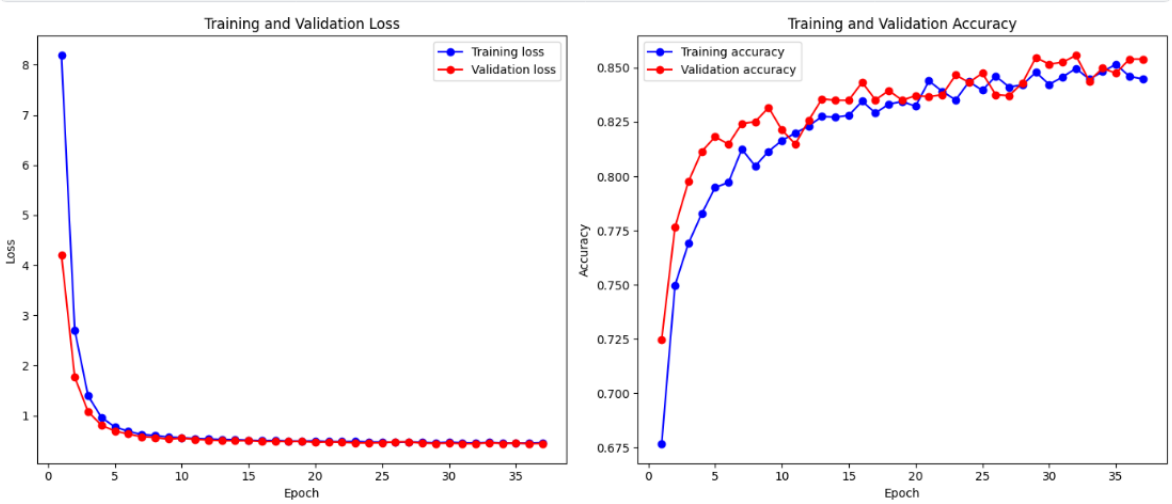
Les métriques utilisées sont l'accuracy et l'AUC qui représente la mesure de l'aire sous la courbe. Plus la métrique est élevée, plus le modèle est performant dans la séparation des classes (homme ou femme).

Évaluation et optimisation

Pour l'évaluation, on évalue la performance du modèle grâce à l'accuracy et également au F1_Score, qui se calcule via cette formule :

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

La Precision est le ratio des vrais positifs sur la somme des vrais et faux positifs, tandis que le Recall est le ratio des vrais positifs sur la somme des vrais positifs et des faux négatifs. La valeur du F1 score est entre 0 et 1. Plus la valeur se rapproche de 1, plus le modèle est performant. Cette métrique sert donc à calculer l'équilibre entre la Precision et le Recall.



Les résultats du premier graphique montrent les 2 courbes de loss qui se suivent. C'est un signe indiquant qu'il apprend sur les données d'entrainement mais qu'il généralise aussi sur les données de validation.

De plus, sur le deuxième graphique, On confirme les résultats du premier graphique où les courbes sont également très proches (Training accuracy et Validation Accuracy) et continue de progresser (légèrement) avec les epochs. Elle ne stagne plus à un niveau inférieur et ne fluctue pas de manière inquiétante. C'est un autre indicateur fort que le modèle généralise mieux.

Après analyse, on obtient les résultats suivants :

Model	Accuracy	F1_score	Precision	Recall	AUC
Genre (CNN)	86.33 %	0.8728	0.8627	0.8832	0.935

Prédictions

Le modèle donne de bons résultats à partir des images du dataset. Quelques petites erreurs sont remarquées chez un bébé, mais on considère que le genre d'un bébé est plus complexe à déterminer que le genre d'un enfant de plus de 5 ans par exemple.

Modèle 2. Age :

Architecture du modèle

Le modèle accepte des entrées d'images en 128x128.

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 128, 128, 3)	0
conv2d (Conv2D)	(None, 126, 126, 32)	896
batch_normalization (BatchNormalization)	(None, 126, 126, 32)	128
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 61, 61, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_3 (Conv2D)	(None, 12, 12, 256)	295,168
batch_normalization_3 (BatchNormalization)	(None, 12, 12, 256)	1,024
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 256)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 256)	0
dense (Dense)	(None, 128)	32,896
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Total params: 423,361 (1.61 MB)

Trainable params: 422,401 (1.61 MB)

Non-trainable params: 960 (3.75 KB)

Le modèle utilise 4 couches de Convolutions (32,64,128,256) avec une activation ReLU et régularisation L2 dans la première couche, suivie d'une BatchNormalization et d'une MaxPooling2D pour réduire la dimensionnalité. Après ces blocs, une couche de **GlobalAveragePooling2D** est utilisée pour extraire les caractéristiques globales de l'image.

Nous avons appliqué à la fin un Dropout à 50 % pour limiter le surapprentissage. Enfin, la couche de sortie est une couche dense à un seul neurone avec une activation linéaire, adaptée pour prédire une valeur continue (l'âge entre autres).

Compilation et entraînement

Pour la compilation, nous avons utilisés l'optimiseur Adam avec un learning rate initial de 0.001, la loss est MSE et la métrique est MAE. Nous avons mis en place des callbacks pour l'optimisation de l'entraînement, et un EarlyStopping avec une patience à 10 pour interrompre l'entrainement dès lors que le modèle commence à surapprendre. La patience égale à 10 permet d'équilibrer au mieux l'entraînement avec 50 epochs. Une fois fait, une diminution du learning rate s'applique quand les performances stagnent et le l'entrainement est lancé avec un indice sur le nombre de pas (steps_per_epoch) et de validation (validation_steps)

Évaluation et optimisation

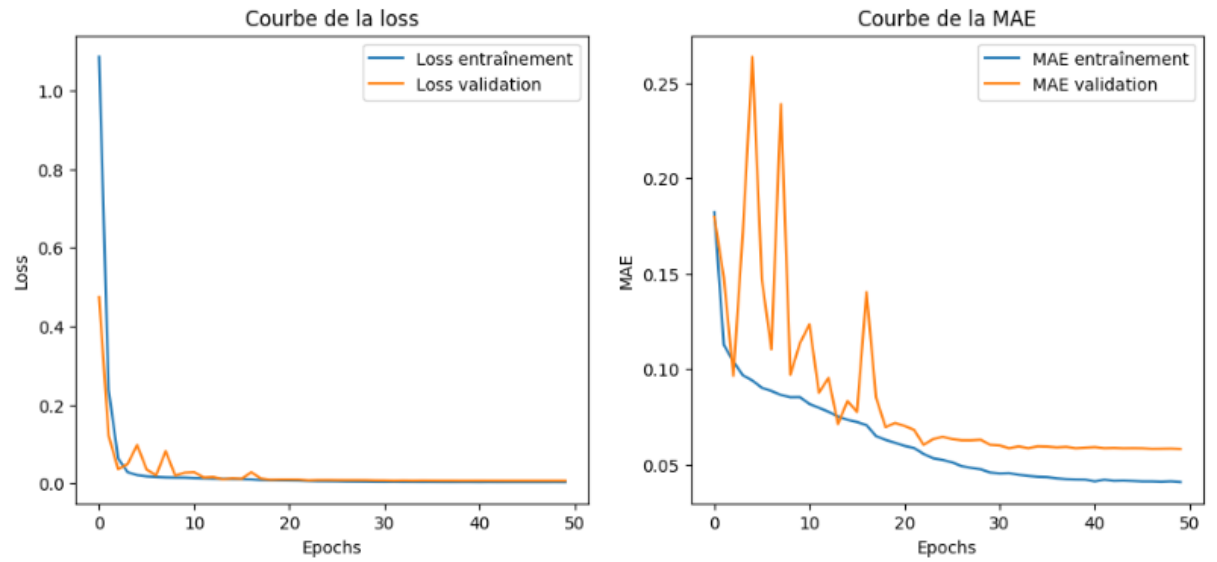
On a laissé l'opportunité de dévoiler l'âge, en sachant qu'il est possible que l'âge puisse aller aux frontières de la tranche d'âge : réel = 29ans prédit = 30 pour 30-39 ans par exemple, ce qui reste performant.

Après analyse, on obtient les résultats suivants :

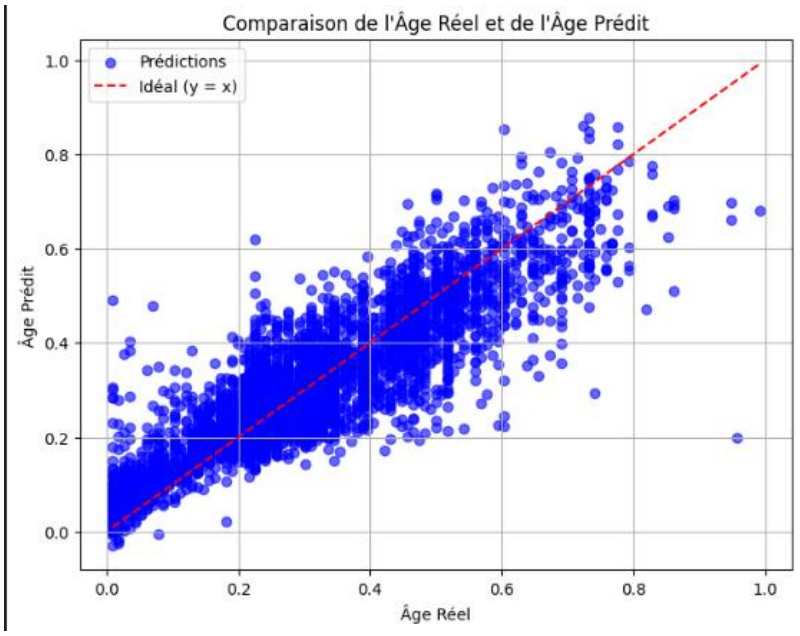
Model	MAE	MSE	RMSE
Age (CNN)	5.81	63.45	9.24

Le modèle se trompe de 5.81 ans (soit 6 ans une fois arrondi). Cependant, le fait que la MSE soit de 63.45 (et donc le RMSE de 9,24) confirme la présence de prédictions avec des erreurs plus élevées.

On imagine avant de prédire que ces erreurs de prédictions seront globalement centrées sur les personnes âgées ou bien les enfants.



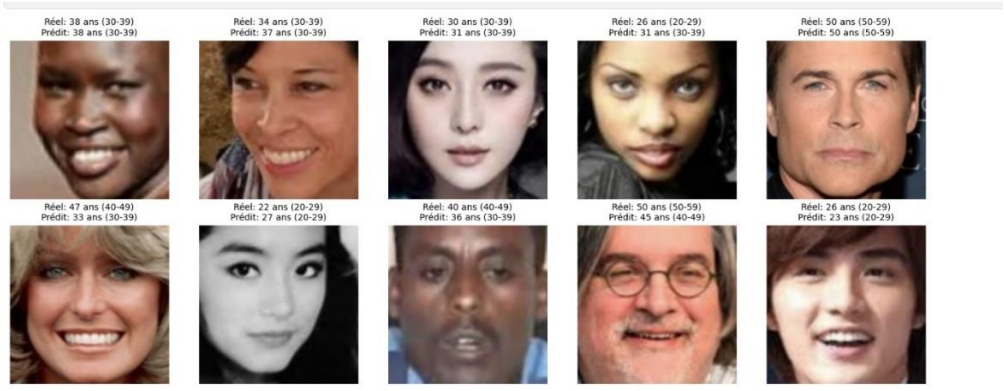
Globalement, Les courbes ne montrent pas de divergence marquée entre l'entraînement et la validation, signe que le modèle généralise plutôt bien. Les courbes de loss et de MAE sont plutôt encourageantes : Il atteint donc de bonnes performances et ne présente pas de signe d'overfitting mis à part entre 5 et 15 epochs pour la MAE où le modèle essaye de se stabiliser.



En somme, cela met en évidence que le modèle est probablement sensible à des écarts plus marqués sur certains échantillons. Cela peut s'améliorer par des stratégies d'augmentation de données ou de régularisation avec plus de Dropout, et d'optimisation du learning rate qui est à 0.001

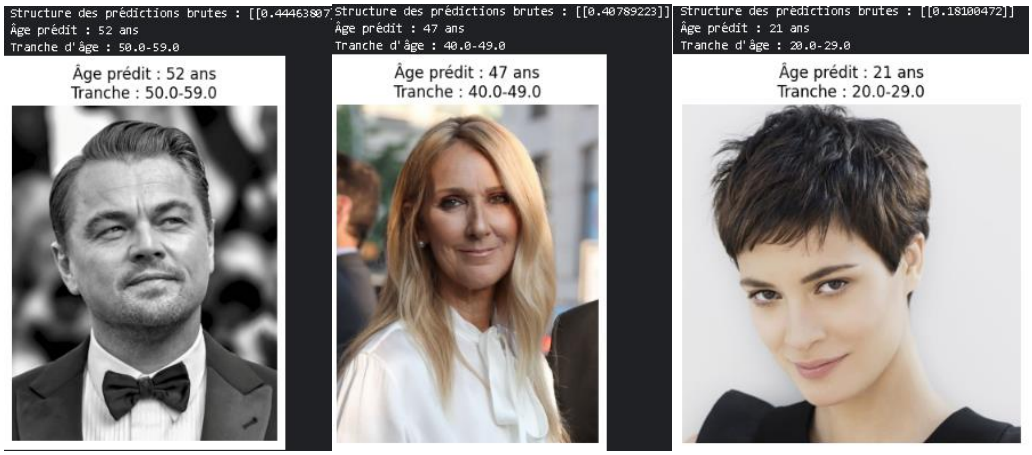
Prédictions

Globalement, le modèle donne de bonnes performances sur le Dataset.



L’image n°4 de la deuxième figure illustre bien cet exemple : De base la personne à 26 ans, mais le modèle a prédit 31, soit 5 ans de + et donc est dans une tranche d’âge différente.

3 exemples avec des images externes :



Le modèle généralise plutôt bien, même si pour les personnes les plus âgées, le modèle éprouve des difficultés.

Modèle 3. Age et Genre :

Architecture du modèle

Tout comme le modèle 2, nous faisons entrer des images de taille 128x128. Le modèle utilise 4 couches de Convolutions (32,64,128,256) avec une activation ReLU et un padding égale à « same ».

En mode « same », le padding prend 3 par 3 à chaque fois, et si les valeurs sont < 3 dans le filtre, il ajoute des 0. Cela permet de conserver les informations.

If you like ascii art:

- `"VALID"` = without padding:

```
inputs:         1  2  3  4  5  6  7  8  9  10 11 (12 13)
               |_________________|                dropped
                        |_________________|
```

- `"SAME"` = with zero padding:

```
            pad|                                      |pad
inputs:      0 |1  2  3  4  5  6  7  8  9  10 11 12 13|0  0
            |________________|
                     |_________________|
                              |_________________|
```

In this example:

Chaque bloc possède une régularisation L2 (0,0005) pour éviter le surapprentissage et possède également un BatchNormalization pour stabiliser et accélérer l'apprentissage, puis d'une couche MaxPooling2D qui réduit la dimension spatiale,

Ensuite on applique la régression pour l’âge avec une sortie possédant une activation « linear », et d’une classification avec 1 unit en sortie pour le genre, possédant une activation « sigmoid » (entre 0 et 1).

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 128, 128, 3)	0	-
conv2d (Conv2D)	(None, 128, 128, 32)	896	input_layer[0][0]
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0	batch_normalization[0…
conv2d_1 (Conv2D)	(None, 64, 64, 64)	18,496	max_pooling2d[0][0]
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 64)	256	conv2d_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 64)	0	batch_normalization_1…
conv2d_2 (Conv2D)	(None, 32, 32, 128)	73,856	max_pooling2d_1[0][0]
batch_normalization_2 (BatchNormalization)	(None, 32, 32, 128)	512	conv2d_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 128)	0	batch_normalization_2…
conv2d_3 (Conv2D)	(None, 16, 16, 256)	295,168	max_pooling2d_2[0][0]
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 256)	1,024	conv2d_3[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 256)	0	batch_normalization_3…
flatten (Flatten)	(None, 16384)	0	max_pooling2d_3[0][0]
dense (Dense)	(None, 256)	4,194,560	flatten[0][0]
dense_1 (Dense)	(None, 256)	4,194,560	flatten[0][0]
dropout (Dropout)	(None, 256)	0	dense[0][0]
dropout_1 (Dropout)	(None, 256)	0	dense_1[0][0]
age_output (Dense)	(None, 1)	257	dropout[0][0]
gender_output (Dense)	(None, 1)	257	dropout_1[0][0]

Total params: 8,779,970 (33.49 MB)

Trainable params: 8,779,010 (33.49 MB)

Non-trainable params: 960 (3.75 KB)

Compilation et entraînement

Pour la compilation, nous avons utilisés l'optimiseur Adam avec un learning rate initial de 1e^-4, qui permet une mise à jour équilibrée des poids. Pour la classification de genre, Binary_crossentropy est utilisée pour la loss. La MSE est la loss pour la régression

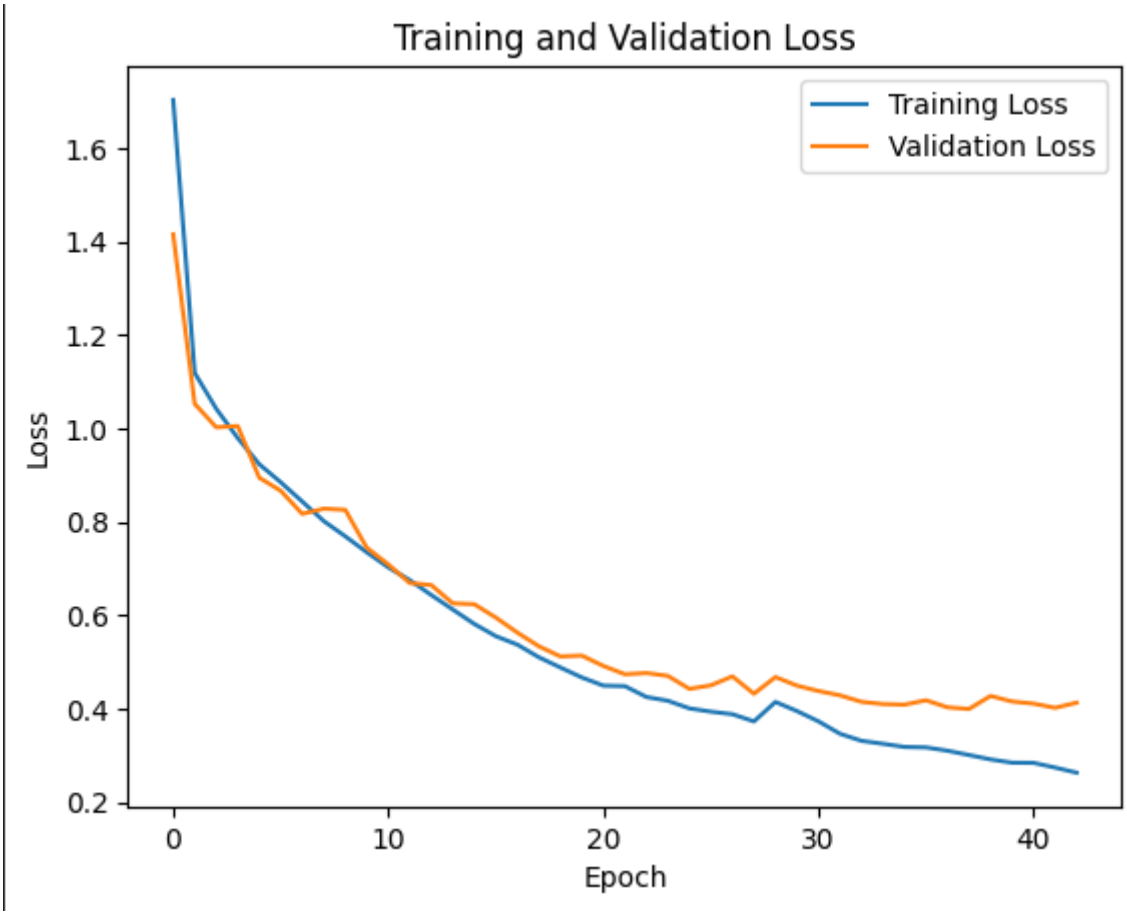
Les métriques sont :

- Genre : Accuracy pour mesurer la précision de la classification
- Age : MAE, MSE et RMSE pour évaluer la précision et la perte de manière plus ou moins précise.

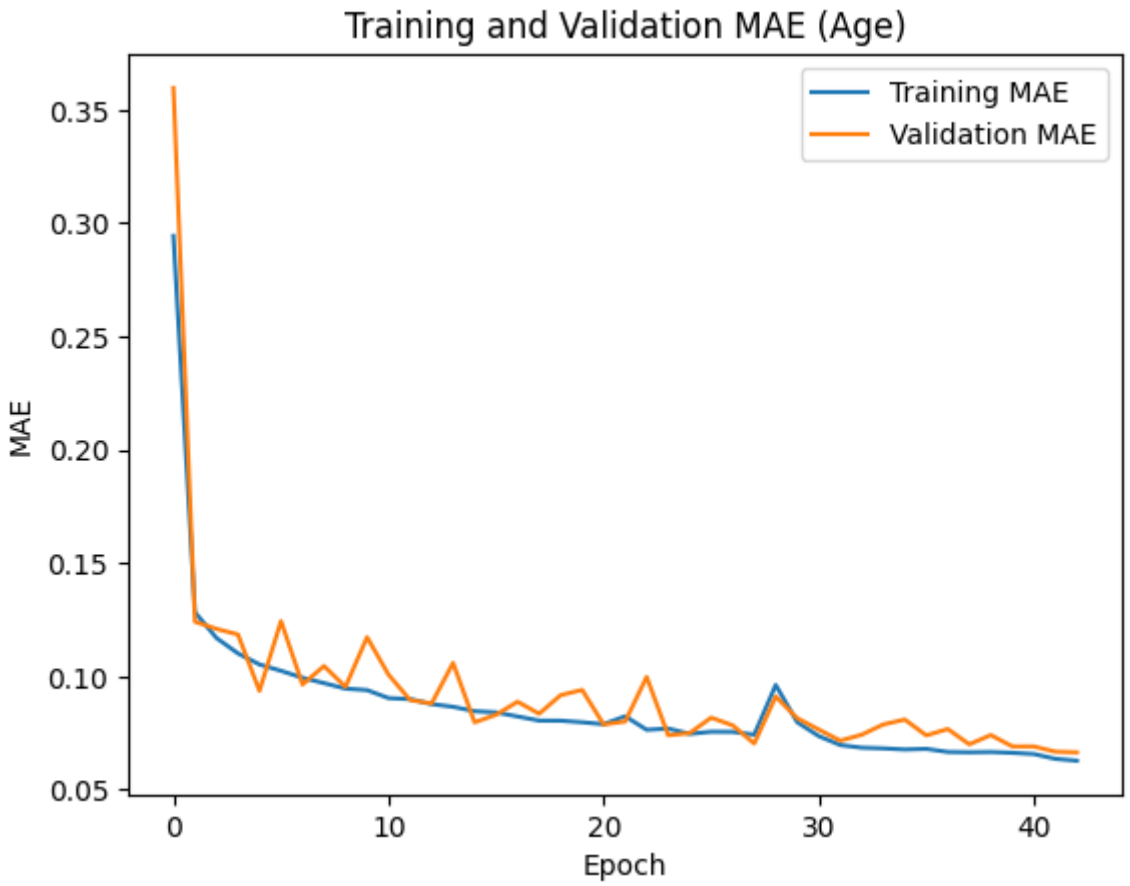
Nous avons mis en place des callbacks pour l'optimisation de l'entraînement, et un EarlyStopping avec une patience à 5 pour interrompre l'entrainement dès lors que le modèle commence à surapprendre.

Une fois fait, une diminution du learning rate s'applique quand les performances stagnent et l'entrainement est lancé avec un nombre d'epochs à 50.

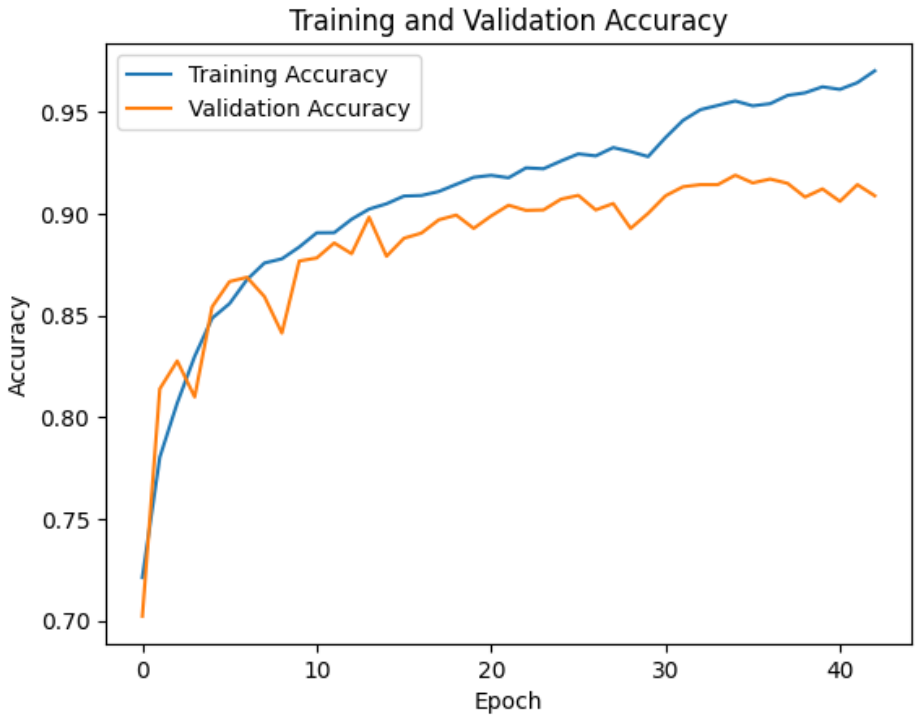
Évaluation et optimisation



Sur ce graphique, on aperçoit que les 2 loss diminuent drastiquement , ce qui est signe que le modèle apprend de manière efficace. Un léger surapprentissage apparaît autour de 0.4 epochs même si cela diminue et se stabilise par la suite.



Les courbes de loss baissent drastiquement et les courbes étant très proches les unes des autres, on considère qu'il n'y a pas de surapprentissage majeur.



Pour le dernier graphique, les courbes montent rapidement aux premiers epochs, la précision d'entraînement atteint presque 0,95 d'accuracy même plus tandis que la validation se stabilise à 0,90-

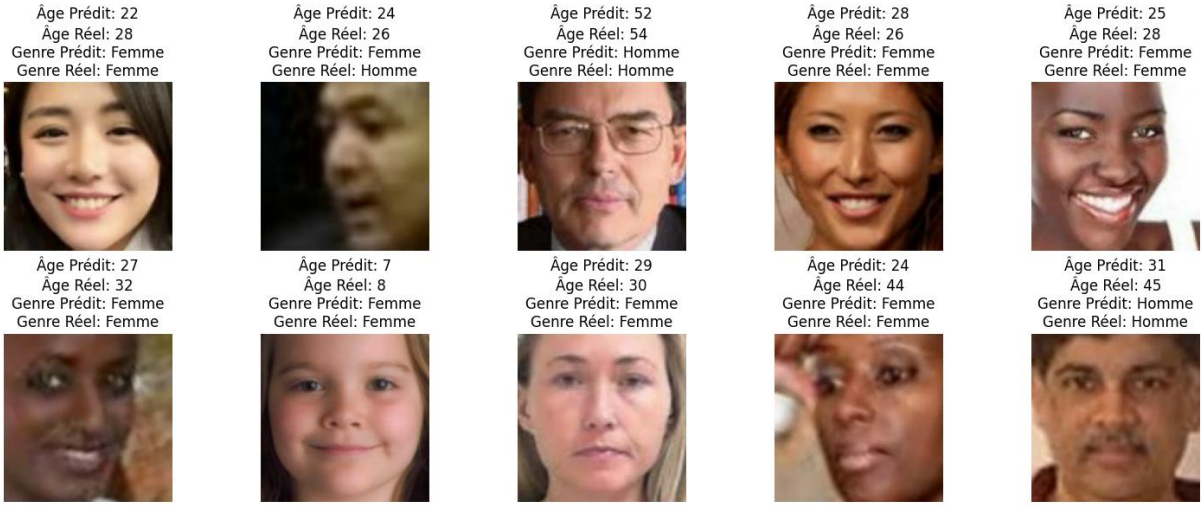
0,91. Cependant, l'écart entre les courbes indique un potentiel surapprentissage mais pas significatif. Le modèle peut être amélioré pour lui faciliter la généralisation, notamment en ajustant les hyperparamètres, avec un learning rate bien plus équilibré, ou ajouter bien plus de couches de Dropout.

Après analyse, on évalue le modèle suivant :

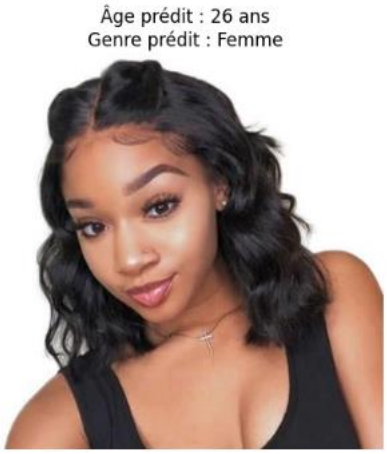
Model	Accuracy	MAE	MSE	RMSE
Age + Genre (CNN)	91 %	8.22	96	11.33

Prédictions

Globalement, le modèle donne de bonnes performances sur le Dataset si ce n'est 2-3 erreurs remarquées (Image 2 pour le genre et image 9 pour l'âge).



3 exemples avec des images externes au dataset :



On remarque que le modèle prédit absolument bien l'âge de la personne pour des images externes au dataset, ce qui confirme la généralisation pour l'âge. Cependant, pour le genre, quelques petits problèmes de généralisation ont étés repérés pour les personnes du genre Homme.

Modèle 4. Age et Genre (Pré-entraîné)

Architecture du meilleur modèle pré-entrainé

Pour le 4ème modèle, nous devions choisir le meilleur modèle en termes de performances en comparant différents modèles pré-entraînés. Voici la liste des modèles pré-entraînés :

EfficientNetB0	MobileNetV2
EfficientNetB1	VGG16
EfficientNetB2	ResNet50 (Caduque)

Parmi ces modèles, l'architecture de ceux-ci va être différent sur plusieurs points :

Dans un premier temps, les différences entre B0, B1 et B2 résident dans leur taille :

- EfficientNetB0 est la version la plus légère, avec moins de paramètres et une précision modérée.
- EfficientNetB1 est plus grand et plus précis, avec un compromis légèrement plus coûteux en calcul.
- EfficientNetB2 pousse encore plus loin la précision et la taille, tout en restant relativement efficace par rapport à des modèles plus anciens (7,7 Millions de paramètres)
- MobileNetV2 privilégie la légèreté et la vitesse (3,4 Millions de paramètres)
- VGG16 possède 16 couches. Il est composé de couches de convolutions, avec des couches de pooling, pour finir par des couches fully connected. Il est très gourmand en ressources (138 Millions de paramètres).
- ResNet50 est jugée caduque puisque nous l'avions trouvé inefficace par rapport à l'architecture qu'on a réalisé. Cependant, avec ses 5O couches, il introduit les couches résiduelles permettant de contourner certaines couches et faciliter l'entraînement de réseaux profonds.

Un exemple d'architecture pour EfficientNetB2 :

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 224, 224, 3)	0	-
efficientnetb2 (Functional)	(None, 7, 7, 1408)	7,768,569	input_layer[0][0]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1408)	0	efficientnetb2[0][0]
batch_normalization (BatchNormalization)	(None, 1408)	5,632	global_average_poolin…
dense (Dense)	(None, 512)	721,408	batch_normalization[0…
dropout (Dropout)	(None, 512)	0	dense[0][0]
dense_1 (Dense)	(None, 256)	131,328	dropout[0][0]
dropout_1 (Dropout)	(None, 256)	0	dense_1[0][0]
dense_2 (Dense)	(None, 256)	65,792	dropout_1[0][0]
dropout_2 (Dropout)	(None, 256)	0	dense_2[0][0]
dense_3 (Dense)	(None, 128)	32,896	dropout_2[0][0]
dense_4 (Dense)	(None, 128)	32,896	dropout_2[0][0]
dropout_3 (Dropout)	(None, 128)	0	dense_3[0][0]
dropout_4 (Dropout)	(None, 128)	0	dense_4[0][0]
age_output (Dense)	(None, 1)	129	dropout_3[0][0]
gender_output (Dense)	(None, 1)	129	dropout_4[0][0]

Total params: 8,758,779 (33.41 MB)
Trainable params: 8,598,246 (32.80 MB)
Non-trainable params: 160,533 (627.09 KB)

Afin de ne pas fausser les résultats, l'architecture du modèle sera similaire pour tous les modèles pé-entraînés, mis à part le modèle de base, ce qui signifie que les couches personnalisées ajoutées après le modèle de base, telles que les couches fully connected, les couches de dropout ou les sorties adaptées à la tâche spécifique, doivent rester identiques pour chaque modèle. Cela évite donc les biais qui peuvent influencer les résultats.

En testant de plusieurs manières les modèles, nous nous sommes rendu compte que le mieux était de « freezer » les 100 dernières couches du modèle de base. Pourquoi ? parce que les modèles pré-entraînés, comme EfficientNet ou ResNet, ont été entraînés sur d'énormes ensembles de données tels qu'ImageNet. Ils ont alors appris des caractéristiques utiles sur les premières et dernières couches. L'objectif de geler les 100 dernières couches va permettre de mieux s'adapter aux particularités de notre propre dataset, en plus de réduire le nombre de paramètres et par conséquent, accélérer l'entraînement et de réduire les surapprentissage.

```
base_model_efficientnet.trainable = True
for layer in base_model_efficientnet.layers[:100]:
    layer.trainable = False
```

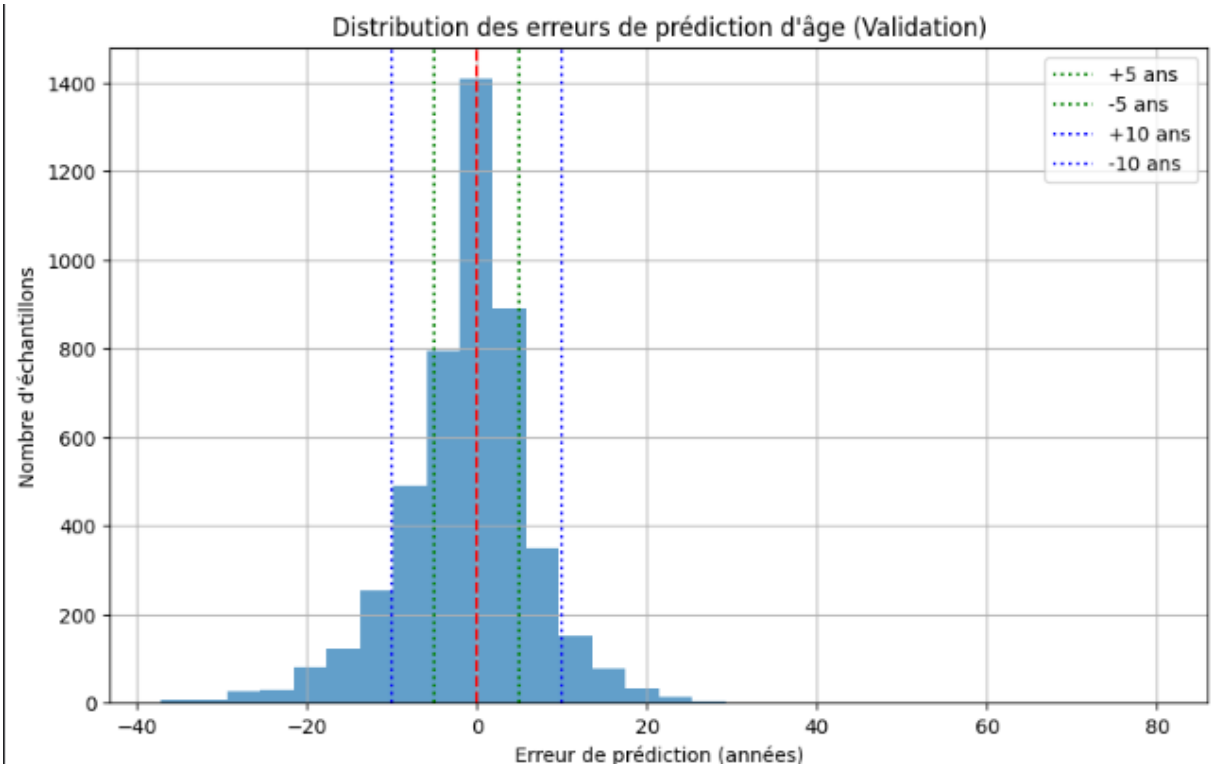
Compilation et entraînement

Pour la compilation, nous utilisons Adam adapté à un learning rate de 0.0001 pour éviter des réglages à haute intensité. Combiner les deux permettent une convergence rapide, et une stabilité grâce au learning rate.

De plus, comme nous combinons Classification et régression, la loss pour l'âge sera la MAE, tandis que pour le genre ce sera la fonction binary_crossentropy. Similairement pour les metriques, nous aurons la MAE, MSE, RMSE et l'accuracy (Pour 10 ans d'écart) pour l'âge, tandis que pour le genre on aura la binary_accuracy qui est utilisée de la même manière que l'accuracy, mais elle est spécialisée dans la classification binaire. On aura également le F1_score, accompagné du calcul du recall et de la precision. Les métriques comme la précision, le recall et le F1-score évaluent si la tranche prédite est correcte ou non.

Enfin pour l'entraînement, nous entraînons le modèle autour de 50 epochs, avec une patience à 10 dans le EarlyStopping. Le paramètre ReduceLROnPlateau est aussi utilisée pour réduire le learning rate de manière dynamique lorsque la performance sur l'ensemble de validation cesse de s'améliorer. Cela permet d'ajuster plus finement les poids du modèle et d'améliorer sa convergence vers une solution optimale.

Évaluation et optimisation

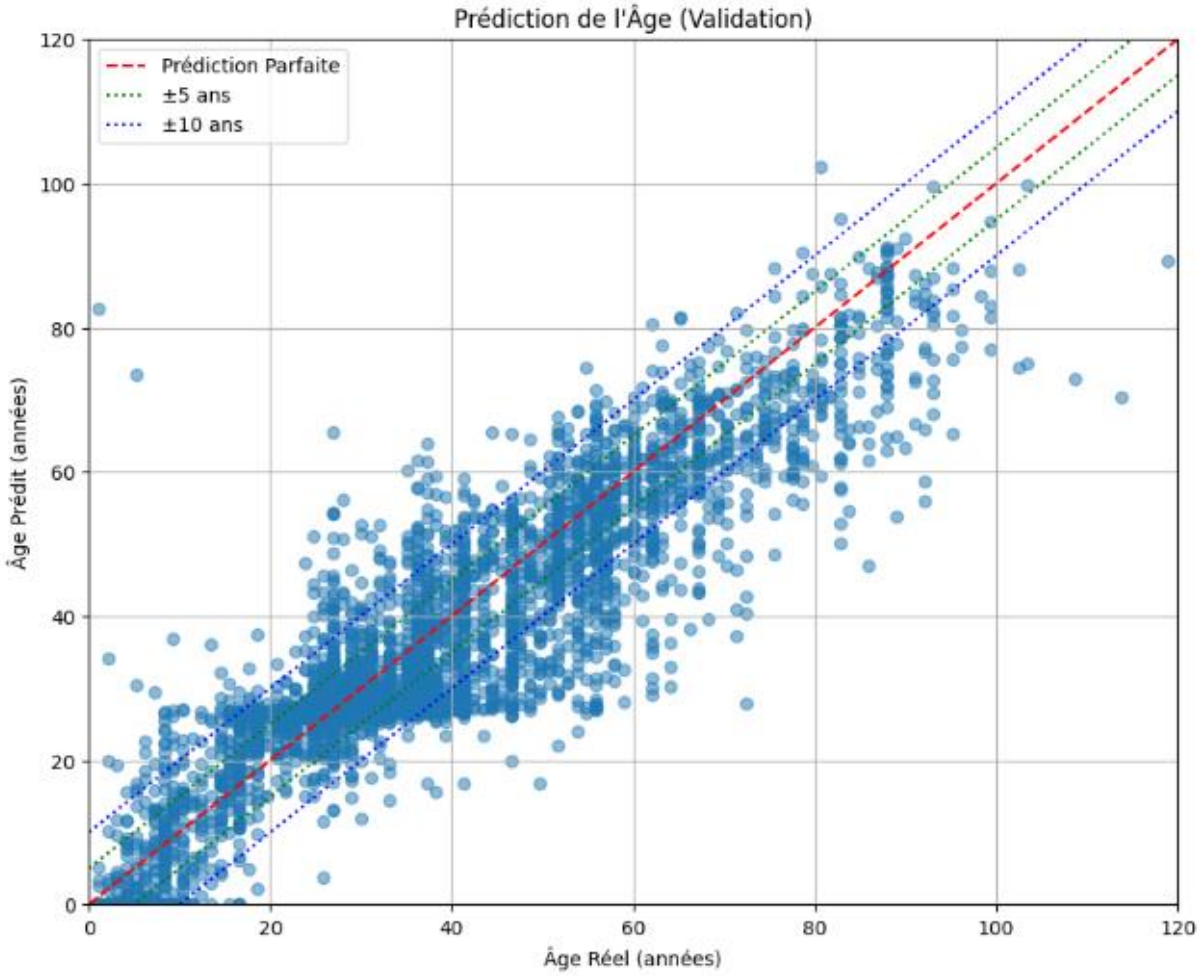


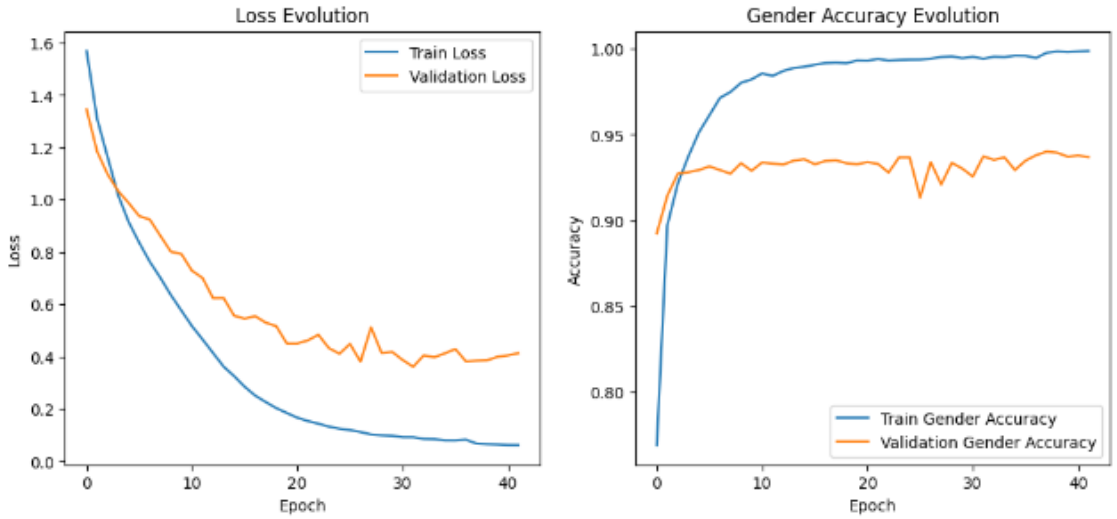
Ce graphique représente la distribution des erreurs de prédiction d'âge sur votre ensemble de validation. Il s'agit d'un histogramme qui montre comment les erreurs de prédiction sont réparties.

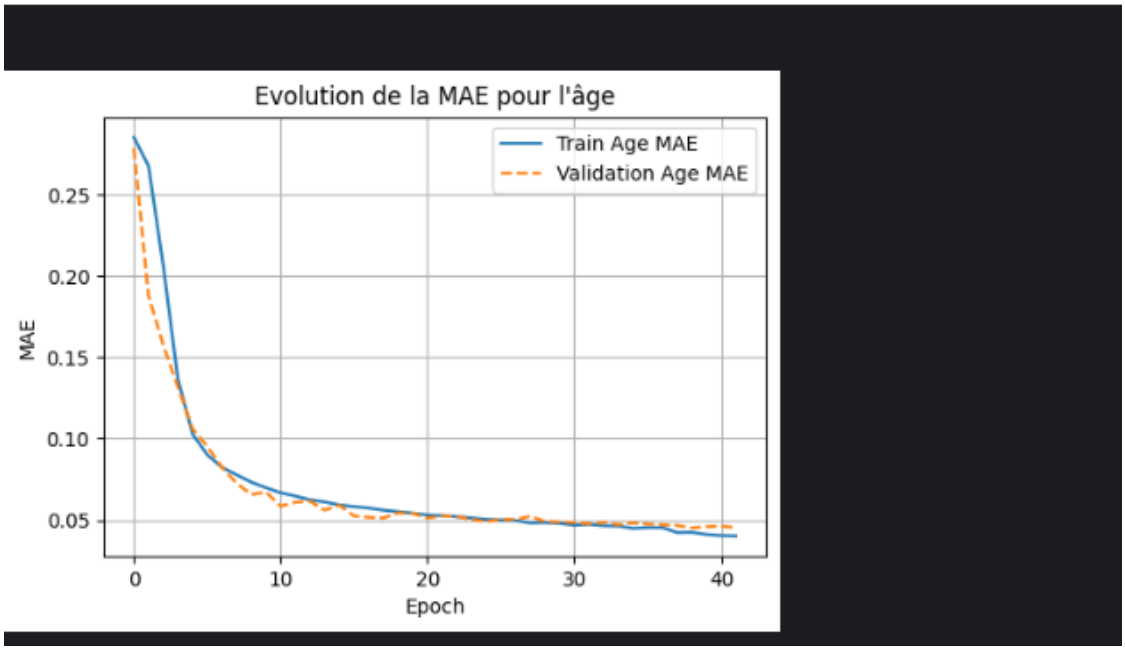
La majorité des prédictions se situent dans la plage de ±10 ans, avec une forte concentration dans la plage de ±5 ans, ce qui semble indiquer que le modèle est plutôt performant. On remarque également que beaucoup plus d'échantillons tombe sur une prédiction ± parfaite, ce qui renforce la qualité du modèle.

Cependant, La classe 100-119 ans n'a aucune prédiction correcte, mais avec seulement 5 exemples, cela reflète probablement un manque de données plutôt qu'une mauvaise qualité du modèle.

Les performances sur les âges plus élevés (70 ans et plus) sont faibles, avec des précisions et rappels bas. Cela peut être lié au faible nombre d'exemples dans ces classes.







De plus, les graphiques par rapport à la MAE indique une très bonne généralisation du modèle puisque le train et la validation se stabilisent à de bas niveaux avec un faible écart, ce qui justifie la généralisation.

Le graphique par rapport à l'accuracy montre une courbe qui se stabilise entre la deuxième et la 21eme epoch et montre des fluctuations entre la 22ème jusqu'à la 30 ème pour se stabiliser de nouveau par la suite. Cependant, cela reste une faible fluctuation puisqu'après le modèle arrive à se stabiliser autour des 93-94 % en Accuracy. Par conséquent, le modèle apprend bien tout en généralisant.

Enfin, la loss montre que les deux courbes semblent converger et se stabiliser vers la fin des epochs. La Train Loss est à un niveau très bas, et la Validation Loss s'est également stabilisée. Cependant, il y a des petites fluctuations (entre les epochs 25 et 30) mais rien d'alarmant.

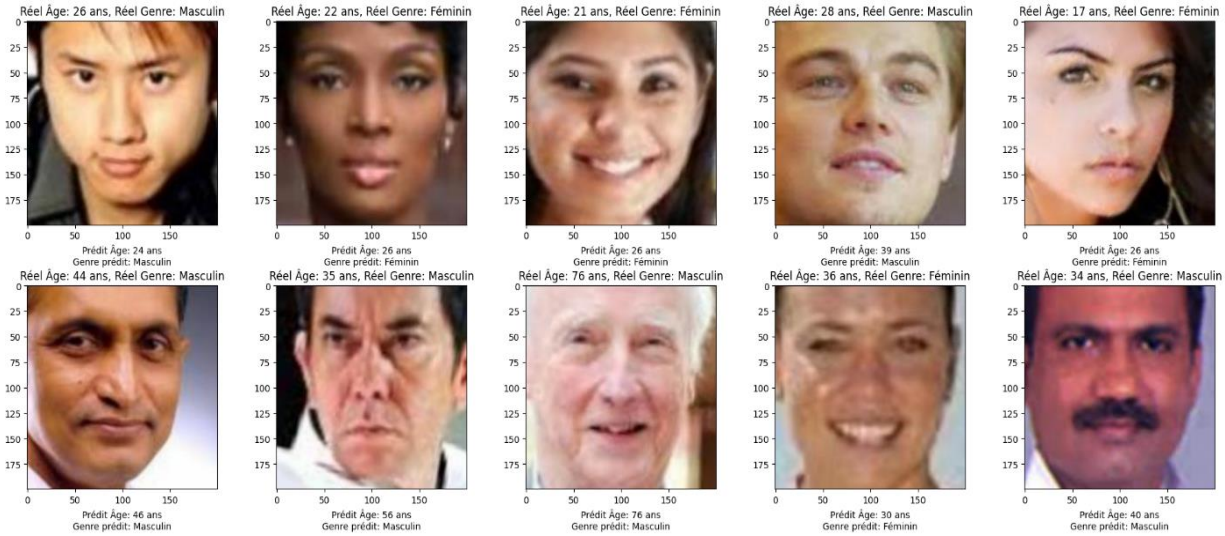
Après analyse, on obtient ces résultats suivants :

Model	Accuracy	F1_score	Precision	Recall	MAE	MSE	RMSE	Age Accuracy 10 years

EfficientNetB2 (CNN)	0.9373	0.94	0.95	0.93	5.52	62.79	7.92	83.64 %

Prédictions

De bonnes prédictions pour le dataset au niveau de l'âge ainsi que du genre.

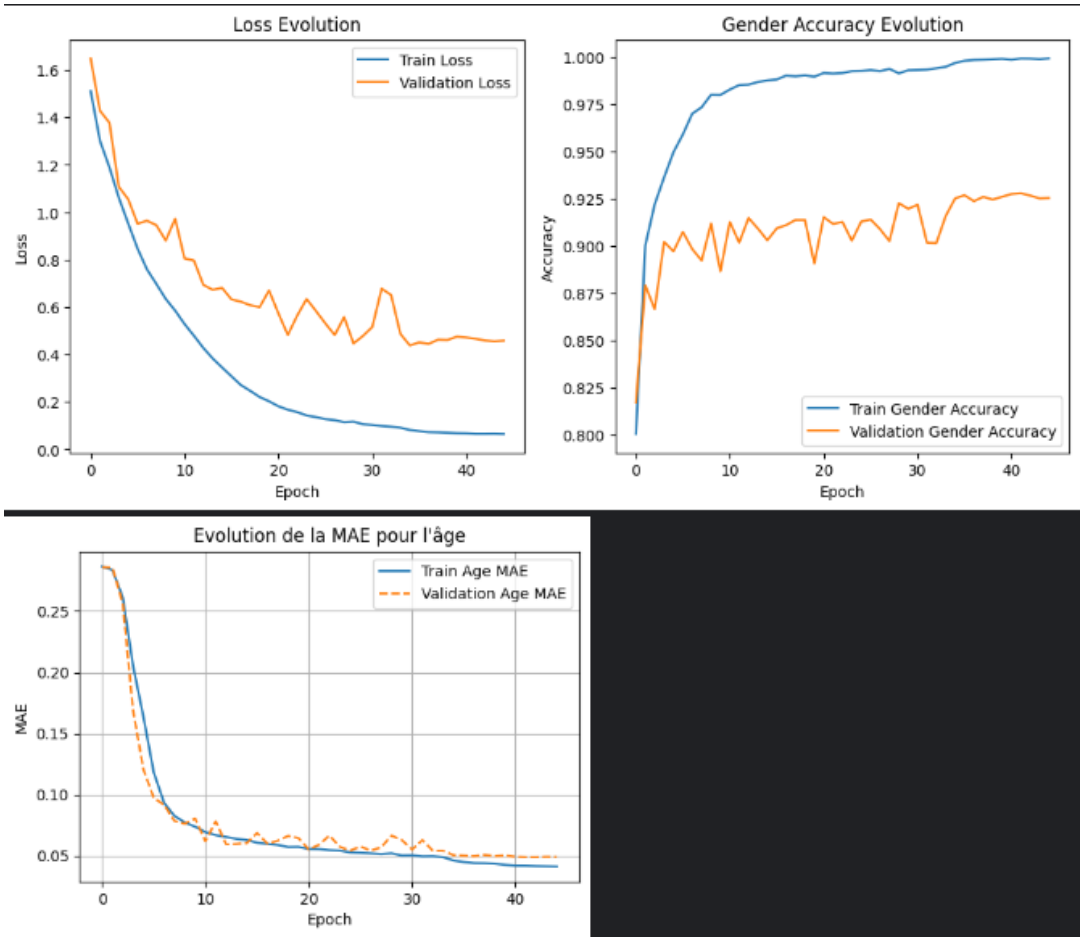


3 exemples d'images de célébrités publiques externes au dataset :

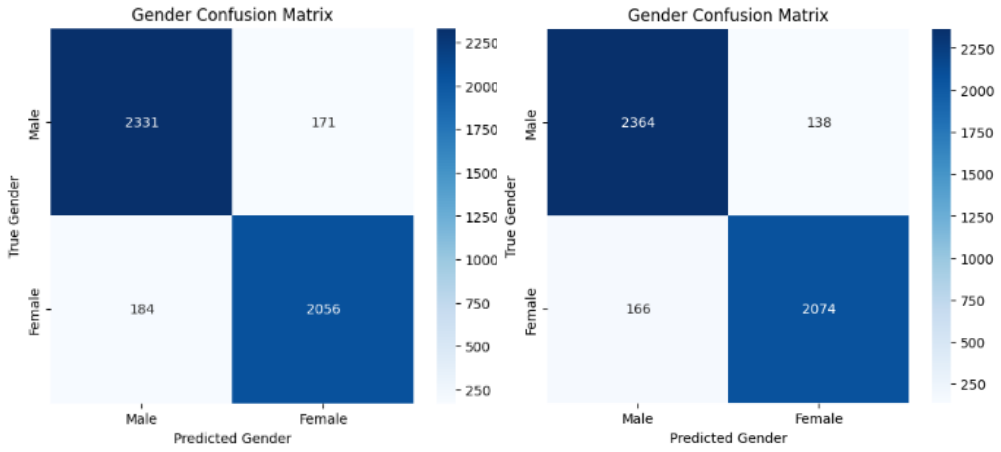
Comparatifs avec MobileNetV2

Nous pouvions comparer avec EfficientNetB0,B1 et VGG16, mais nous avions préférés comparer les résultats de manière plus détaillé avec une architecture différente :

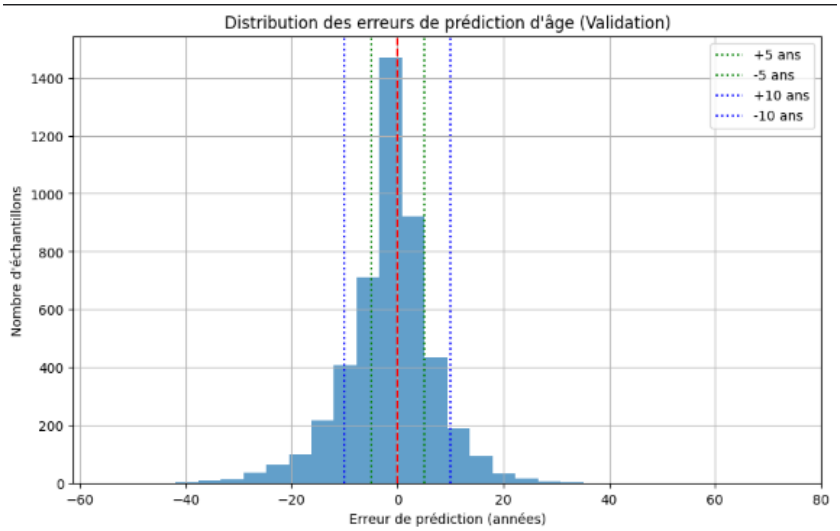
En gardant la même structure du modèle, on obtiens ces résultats :



On remarque alors de légères fluctuations par rapport à EfficientNetB2. Ce qui est signe que le modèle a un peu de mal à se stabiliser.



A gauche : MobileNetV2, à droite : EfficientNetB2. Pour le genre, EfficientNetB2 est légèrement au-dessus en ce qui concerne le faible taux de faux positifs/négatifs sur le genre homme et Femme.



On peut également remarquer que les erreurs de prédictions penchent plus d'un côté que l'autre par rapport à EfficientNetB2 : Le nombre d'échantillons a un écart par rapport à la prédiction parfaite.

Après analyse, on obtient les résultats suivants :

Model	Accuracy	F1_score	Precision	Recall	MAE	MSE	RMSE	Age Accuracy 10 years
MobileNetV2 (CNN)	0.93	0.93	0.93	0.93	6.07	76.52	8.75	80.64 %

En conclusion, bien que les métriques de MobileNetV2 indiquent une bonne performance, EfficientNetB2 possède la même accuracy que MobileNetV2, mais à une plus faible MAE que celle de MobilleNetV2.

Par conséquent, voici les résultats finaux de tous les modèles

Modèles	Accuracy - Genre	AUC	F1_Score	Precision	Recall	MAE	MSE	RMSE	Age Accuracy (10 ans d'écart en %)	#Params
Genre	86.33 %	0.935	0.8728	0.8627	0.8832	-	-	-	-	2,600 M
Age	-	-	-	-	-	5.81	63.45	9.24		423,361
Genre + Age	0.91	-				8.22	96	11.33		8,779,970
TA - EfficientNetB2	0.9373	-	0.94	0.95	0.93	5.52	62.79	7.92	83.64%	7,499 M
TA - EfficientNetB0	0.90	-	0.90	0.89	0.91	7.58	108.06	10.40	73.45%	4,383 M
TA - EfficientNetB1	0.90	-	0.91	0.91	0.91	6.86	92.93	9.64	77.01%	8,417 M
TA - VGG16	0.88	-	0.89	0.89	0.89	8.02	125.05	11.18	71.19%	15,7 M
TA - MobileNetV2	0.93	-	0.93	0.93	0.93	6.07	76.52	8.75	80.64	3.2M

On remarque que le modèle qui délivre le plus de performances sur toutes les métriques est le modèle EfficientNetB2. Cependant, nous serons sujet à améliorer l'ensemble des modèles car il ne fait aucun doute que la généralisation de ces modèles n'est pas parfaite, et qu'avec de la data augmentation, ainsi qu'un réglage des hyperparamètres, l'on pourra qualifier nos modèles de « modèles performants ».