# Analyse du TP de classification de genre en vue de son amélioration

Nous nous sommes réparties les tâches de la façon suivante. Axel et Samia se sont occupés du modèle 1: Classification de genre avec CNN. Nous avons analysé le modèle que nous avions vu en cours avec M.Faye.

## Réduction de la taille d’image

# Function for resizing image

import tensorflow as tf

def preprocess(image, label, size=(64,64),mode='train'):

if mode == 'train':

img\_crop = tf.image.random\_crop(image,size=(178,178,3))

img\_resized = tf.image.resize(img\_crop,size=size)

img\_flip = tf.image.random\_flip\_left\_right(img\_resized)

return img\_flip/255.0 , tf.cast(label,tf.int32)

else:

img\_crop = tf.image.crop\_to\_bounding\_box(image,

offset\_height=20,

offset\_width=0,

target\_height=178,

target\_width=178)

img\_resized = tf.image.resize(img\_crop,size=size)

return img\_resized / 255.0 , tf.cast(label,tf.int32)

Cette fonction de preprocessing permet de standardiser les images d’entrées en fonction du mode (train ou test). Elle prend en entrée l’image à traiter, le label associé à l’image, la taille de l’image et le mode puis prépare les images pour le modèle.

On entre une image en 256x256 et si le mode est ‘train’ on coupe une partie aléatoire de l’image (de taille 178x178 avec 3 canaux de couleurs. L'image recadrée est ensuite redimensionnée pour atteindre la taille souhaitée (par exemple, 64x64). Pour ajouter plus de précision et de diversité sur les données, on utilise random\_flip\_left\_right pour transformer l’image sous un autre angle. L’image est retournée avec des valeurs normalisées, ainsi que les labels convertis en entier sous 32 bits.

Si le mode est ‘test’, on effectue un cadrage fixe pour assurer que chaque image de validation est traitée de manière uniforme. On la recadre et on retourne de même la valeur de l’image normalisée.

# Resize image and create batch

import numpy as np

image\_size = (64, 64)

ds\_train = train\_ds.map(lambda x, y:preprocess(image=x, label=y, size=image\_size,mode='train'))

ds\_valid = val\_ds.map(lambda x, y:preprocess(image=x, label=y, size=image\_size,mode='eval'))

La fonction de preprocessing va s’appliquer aux datasets de train et de validation et on va mapper les valeurs pour chaque image (et son label). Elle permet donc de faire du parallélisme, ce qui rend plus rapide le traitement des données..

## La création du model

import tensorflow as tf

from tensorflow.keras import layers, models

def create\_model():

inputs = tf.keras.Input(shape=(64, 64, 3)) # Input shape for 64x64 RGB images

# First convolutional block

x = layers.Conv2D(64, (3, 3), padding='same', activation='relu')(inputs)

x = layers.BatchNormalization()(x) # Batch Normalization

x = layers.MaxPooling2D(pool\_size=(2, 2))(x)

# Second convolutional block

x = layers.Conv2D(128, (3, 3), padding='same', activation='relu')(x)

x = layers.BatchNormalization()(x)

x = layers.MaxPooling2D(pool\_size=(2, 2))(x)

# Third convolutional block

x = layers.Conv2D(256, (3, 3), padding='same', activation='relu')(x)

x = layers.BatchNormalization()(x)

x = layers.MaxPooling2D(pool\_size=(2, 2))(x)

# Global Average Pooling instead of Flatten

x = layers.GlobalAveragePooling2D()(x)

# Fully connected layer with fewer units than the Global Average dimension

x = layers.Dense(128, activation='relu')(x)

x = layers.Dropout(0.5)(x) # Dropout to prevent overfitting

# Output layer for binary classification (2 classes)

outputs = layers.Dense(1, activation='sigmoid')(x) # Sigmoid for binary classification

# Create the model

model = models.Model(inputs=inputs, outputs=outputs)

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

return model

# Create and summarize the model

model = create\_model()

model.summary()

Ce code prend en entrée des images de taille 64x64. On a trois blocs de convolution avec des filtres de plus en plus grands (64,128,256). Chaque bloc de convolution utilise des convolutions 3x3 avec une activation relu pour bloquer les valeurs négatives et laisse passer les valeurs positives dans les couches suivantes. On utilise BatchNormalization dans chaque bloc pour normaliser en réduisant les valeurs. On a aussi MaxPooling2D qui permet de réduire la taille des dimensions ???

Ensuite on a un GlobalAveragePoooling2D, qui réduit le nombre de dimensions tout en conservant l’information. C’est mieux que d’utiliser Flatten. On va faire un Dense (avec 128 units) suivi d’un Dropout (avec 0.5 de taux) pour réduire et éviter l’overfitting (surapprentissage). On utilise un Dense, cette fois avec l’activation sigmoid car on a 2 classes (classification binaire). On compile ensuite le modèle avec Adam comme optimizer, et binary\_crossentropy comme loss (approprié pour la classification binaire).

En utilisant ce modèle, on a une accuracy finale de 0.9461 avec 15 epochs. On va tenter de l’améliorer.

# Choix et mise en place d’une nouvelle structure pour la classification de genre

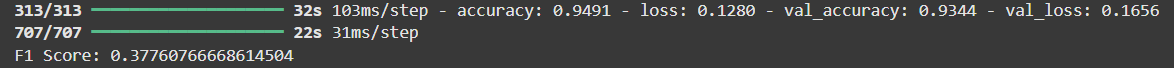
Tableau présentant les différentes métriques et de leurs cas d’utilisation en vue de l’amélioration de la mesure de performance de nos modèles

|  | Description | Avantages | Inconvénients |
| --- | --- | --- | --- |
| F1-Score | Le F1-Score est une moyenne harmonique de la précision et du recall | Regroupe 2 métriques (Precision et Recall) en une seule métrique | * Complexité en termes de calcul et de mémoire |
| Accuracy | L’accuracy permet de décrire la performance du modèle sur les individus positifs et négatifs de façon symétrique. Elle mesure le taux de prédictions correctes sur l’ensemble des individus | Simple d’utilisation  Valeurs entre 0 et 1 | * elle évalue mal les performances d’un modèle basé sur des données déséquilibrées |
| Recall | Le recall permet de savoir le pourcentage de positifs bien prédit par notre modèle.  En d’autres termes c’est le nombre de positifs bien prédit (Vrai Positif) divisé par l’ensemble des positifs (Vrai Positif + Faux Négatif). | Capture les vrais positifs | Ne tient pas compte des faux positifs  Ne mesure pas si les prédictions positives sont fiables ou pertinentes  Besoin d'être complété par d'autres métriques comme la précision pour évaluer la qualité globale. |
| Precision | Taux de vrais positifs parmi tout ce qui a été prédit par le modèle | Moins de paramètres  Convient bien aux tâches nécessitant un peu de ressources | * Un peu plus d’entraînement à faire à cause de la profondeur * Moins de performance avec des données très petites |
| MAE | Moyenne des écarts absolus entre les valeurs réelles et prédites. | Moins sensible aux valeurs aberrantes que MSE.  Facilement interprétable en termes d'unité des données. | Peut ignorer les grandes erreurs par rapport au MSE. |
| MSE | Moyenne des carrés des écarts entre les valeurs réelles et prédites. | Met l'accent sur les grandes erreurs grâce à la mise au carré.  Couramment utilisée et facile à optimiser. | Sensible aux outliers (valeurs aberrantes). |

<https://datascientest.com/metriques-en-machine-learning>

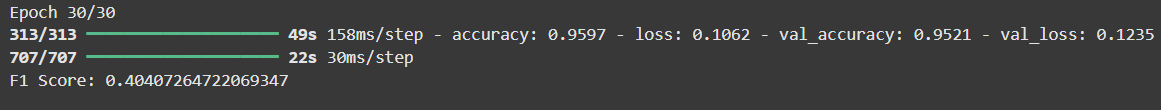
Dans notre cas, nous avons 2 classes possibles en sortie: mâle ou femelle. Il ne devrait pas y avoir de classe qui prime sur l’autre. Dans l’ensemble, ces classes sont équilibrées.

Utilisation de F1 Score:

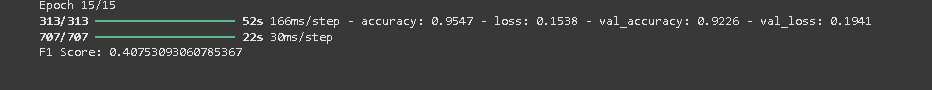


Un F1 Score bas et une accuracy haute comme dans notre test, indique un problème de déséquilibre des classes.

Pourtant, nos données sont bien “mélangées” pour éviter que le modèle ne prenne en compte qu’une certaine partie des données qui créerait donc un déséquilibre. Cette solution pourrait ne pas être suffisante.

On va donc entraîner de nouveau le modèle avec 30 epochs au lieu de 15 pour voir s' il y a un changement concernant l’accuracy ou le résultat du f1-score.

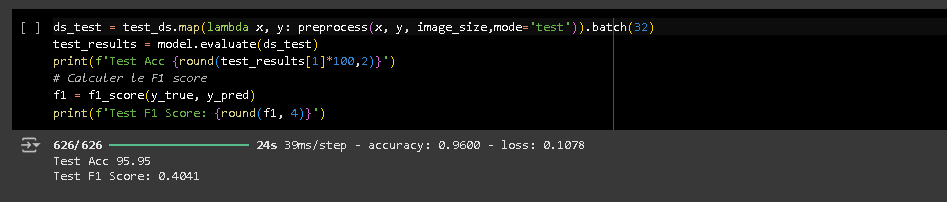
En utilisant une pondération des classes, on obtient ces scores sur 15 epochs :



On voit bien que le F1 Score a bien augmenté même si il n’est pas suffisant.

Donc il faut réduire les faux positifs (précision) et les faux négatifs (recall).

Screen de ton score



J’ai rajouté dans le model-genre(2) le callback(early stopping) et les poids des classes

On va alors regarder dans le dataset s' il y a plus d’images d’un certain genre ou de l’autre, qui pourrait expliquer notre F1 score bas:

|  | Female | Male | Différence |
| --- | --- | --- | --- |
| Test | 11 542 | 8 459 | 3 083 |
| Train | 92 845 | 67 155 | 25 690 |
| Validation | 13 778 | 8 820 | 4 958 |

On voit donc bien qu’il y a beaucoup plus d’images de femelles à chaque fois. La différence est énorme au niveau du train. Cela pourrait justifier les résultats aussi bas dans le f1-score: la classe femelle prime sur la classe mâle.

En termes de pourcentage sur la globalité, les femmes représentent 58% du dataset et les hommes seulement 42%.

En termes de métrique, l’accuracy est potentiellement biaisée car si le modèle décide de ne prédire que des filles, il aura une accuracy de 0.58 environ et il n’aura rien appris.

On dira donc que la classe représentant les mâles est dite minoritaire et la classe représentant les femelles est dite majoritaire.

Pour s’assurer que le problème vient de là, on va faire en sorte de donner le même nombre d’images pour chaque classe. C’est une méthode qui existe déjà, appelée undersampling.

Evidemment, elle a son lot d’avantages et d’inconvénients: on va respectivement avoir des classes équilibrées, en contrepartie on pourrait perdre de l’information et notre modèle serait moins bon.

# Amélioration avec des modèles pré-entraînés

Tableau présentant des suggestions de structures (pré-entraînés) afin d’améliorer notre modèle

|  | Description | Avantages | Inconvénients |
| --- | --- | --- | --- |
| ResNet | Peut créer des modèles sans compromettre la performance grâce à des “skip connections” | * Meilleure performance * Plus stable * Peut construire des réseaux très profonds (plusieurs centaines de couches) | * Complexité en termes de calcul et de mémoire |
| DenseNet | Connecte chaque couche aux suivantes | * Moins de paramètres * Meilleure utilisation des caractéristiques | * Plus gourmand en mémoire et en temps d’entraînement * Plus complexe |
| Inception Networks | Utilisent des blocs multi-filtres dans chaque couche | * Moins de paramètres qu’un réseau profond standard donc plus rapide | * Plus complexe à comprendre * Pas aussi performant que EfficientNet |
| EfficientNet | Approche de ‘scaling’  Maximise la précision en minimisant les paramètres et la charge de calcul | * Moins de paramètres * Convient bien aux tâches nécessitant un peu de ressources | * Un peu plus d’entraînement à faire à cause de la profondeur * Moins de performance avec des données très petites |
| MobileNet | Conçu pour les appareils mobiles ou les systèmes embarqués | * Très léger et rapide pour l’inférence sur des appareils mobiles | * Moins précis que les modèles comme ResNet ou DenseNet |