Apprentissage Profond

Valentin Hesters 20201346
valentin.hesters@gmail.com
Sara Firoud 20235275
firoud.saraa@gmail.com

7 Décembre 2024

Sommaire

Le code se trouve à cette adresse:
https://github.com/suprawall/SpeechRecognition
Les 4 fichiers originaux contenant notre code ont tous le préfixe *"DL2425_"*. Ils sont 3 dans le dossier *moco-v3* et un à part. De plus, certains fichiers de Moco d'origine ont été aussi modifiés pour assurer la communication ou pour des résolutions de bugs détaillés dans ce rapport.
Pour lancer le code il faut executer *main_moco.py* avec les même commandes que pour MocoV3, sans utiliser les paramètres distribués et en spécifiant bien "–gpu 0".

1 Présentation de MoCo v3

1.1 Origine de MoCo V3

Les Vision Transformers (ViT) offrent une nouvelle approche pour la vision par ordinateur, en remplaçant les réseaux convolutionnels traditionnels comme ResNet par des modèles basés sur les Transformers. Bien que ces architectures aient démontré un fort potentiel, leur entraînement dans un cadre auto-supervisé pose des défis importants, notamment en termes de stabilité et d'efficacité.

L'apprentissage auto-supervisé, connu pour exploiter de grandes quantités de données non étiquetées, repose souvent sur des cadres d'apprentissage contrastif ou des réseaux siamois pour comparer des images similaires et différentes sans avoir besoin de reconstruire l'image d'origine.

Cependant ces méthodes se retrouve dans l'incapacité d'adopter ces techniques efficacement pour maximiser les performances des VIT.

C'est dans ce contexte que MoCo v3, une méthode basée les versions précédentes (MoCo v1 et v2) a été développé, offrant une solution optimisée pour entraîner efficacement les Vision Transformers dans un cadre auto-supervisé.

1.2 Son fonctionnement

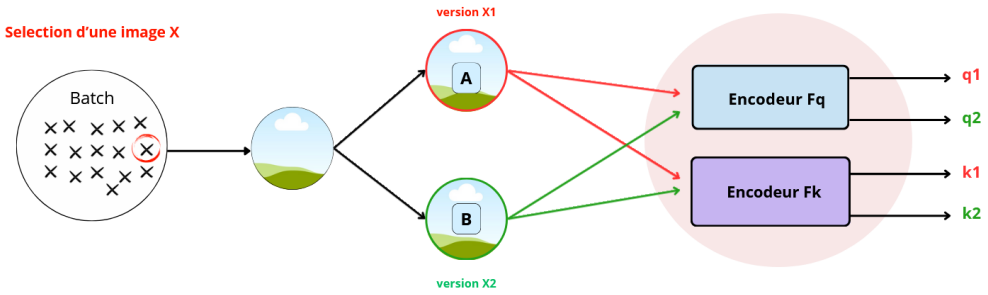

Figure 1: *Augmentation des données*

Moco V3 se décompose en trois étapes principales qui a pour but final une extraction efficace et robuste des caractéristiques des images.

- Premièrement, lors du chargement d'un lot de données, un mini-lot de N images est traité. Chaque image x est soumise à deux augmentations aléatoires, générant deux versions légèrement modifiées, x_1 et x_2. Ces variations servent à enrichir les données d'entraînement et à renforcer la robustesse du modèle. Les deux images augmentées sont ensuite transmises à deux encodeurs distincts.

- Deuxièmement, dans la phase d'encodage des images, deux encodeurs sont utilisés :

  L'encodeur f_q (requête), principal, basé sur des architectures performantes comme ResNet ou Vision Transformer (ViT). Cet encodeur extrait les caractéristiques des images et les transforme en vecteurs via une tête de projection et une tête de prédiction. L'encodeur f_k (clé), similaire à f_q mais dépourvu de tête de prédiction. Il est mis à jour progressivement pour suivre f_q, ce qui stabilise l'entraînement et améliore la cohérence des représentations.

  Chaque encodeur génère ainsi des vecteurs numériques appelés requêtes (q_1, q_2) et clés (k_1, k_2), qui condensent les informations pertinentes des images.

- Enfin, dans l'objectif d'apprentissage, le modèle apprend à associer correctement chaque requête q à sa clé correspondante k^+ (échantillon positif), tout en distinguant les clés des autres images k^- (échantillons négatifs). Cela repose sur une fonction mathématique de perte contrastive (*InfoNCE*), exprimée par la formule :

$$L_q = -\log \frac{\exp(q \cdot k^+/\tau)}{\exp(q \cdot k^+/\tau) + \sum_{k^-} \exp(q \cdot k^-/\tau)}$$

  où :

  - k^+ est la clé générée à partir de f_k pour la même image que q, appelée l'échantillon positif de q.
  - $\{k^-\}$ contient les clés des autres images disponible dans le lot, appelé échantillons négatifs of q.
  - τ (tau) est un hyperparamètre de température qui ajuste l'échelle des similarités calculées entre les vecteurs q et k, normalisés en L2.

Cette fonction maximise la similarité entre les vecteurs des échantillons positifs tout en minimisant celle avec les négatifs, favorisant ainsi des représentations compactes.

En abandonnant la mémoire en file utilisée dans les versions précédentes, MoCo V3 exploite efficacement de larges lots d'entraînement, ce qui améliore significativement les performances. Cette méthode permet d'apprendre à associer des versions similaires d'une même image tout en différenciant celles issues d'images différentes, contribuant à une meilleure généralisation des représentations dans diverses tâches en vision par ordinateur

2 Présentation de la base de données

2.1 Description générale

La base de données sur laquelle on travaille est EmoDB. c'est une base de données allemande librement accessible qui contient 535 enregistrements vocaux réalisés par 10 locuteurs (5 hommes et 5 femmes) et couvre sept émotions distinctes, chacune représentée par une lettre spécifique :

La fréquence d'échantillonnage f_s est de 16 000 Hz. Chaque énoncé dure entre 2 et 4 secondes.

Ainsi nous pouvons extraire et stocker ces données dans deux variables importantes que nous utilisons tout au fil du code: *audio_file_tab* qui est une liste contenant les chemins vers les fichiers audios et *emotion_tab* qui est la liste de leurs labels (émotions).

2.2 Distribution des émotions dans le dataset

A travers ce graphique, on illustre la répartition des différentes émotions dans la base de donnée Emo-DB, et grâce à laquelle on peut observer que l'émotion colère est la plus représentée avec 120 échantillons, à l'inverse du dégoût qui est la moins fréquente avec environ 50 échantillons. Les autres émotions sont réparties de manière relativement équilibrée.

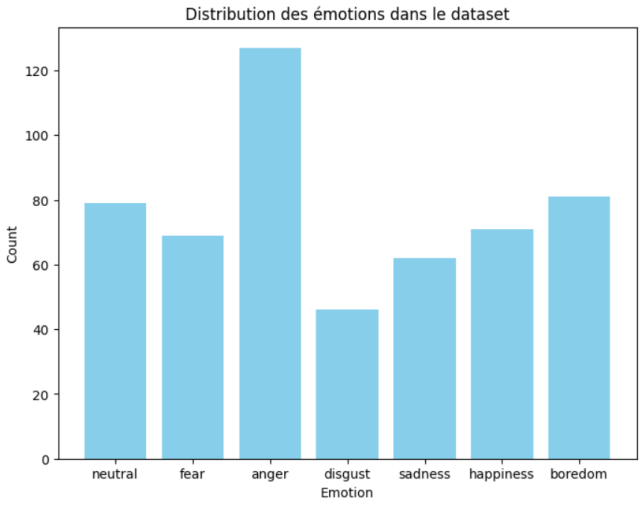

Figure 2: *Schéma de la dénomination des fichiers audios*

3 Traitement des données

3.1 Augmentation et Bruit

Une des clés de l'apprentissage par contraste de MocoV3 est d'utiliser des augmentations de données cohérentes et efficaces. Les auteurs s'appuyaient sur une méthode extérieure d'augmentation qui fonctionne particulièrement bien pour le traitement d'images. Dans nôtre cas, nous travaillons sur des données audios et il est donc plus judicieux d'abandonner cette méthode pour rajouter du bruit sur les fichiers audios directement avant de les convertir en un format d'images reconnu par MocoV3. Ainsi, nous avons décidé d'appliquer des méthodes d'augmentations connues pour fonctionner sur des sons: rajout de bruit, changement de vitesse, décalage temporel, et changement

de la tonalité du son.

```python
#BRUIT
def noise(data,noise_rate=0.01):
    noise_amp = 0.01*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
    return data

#CHANGEMENT DE VITESSE
def stretch(data, rate=0.8):
    return librosa.effects.time_stretch(data, rate=rate)

#DECALAGE TEMPOREL
def shift(data):
    shift_range = int(np.random.uniform(low=-5, high = 5)*1000)
    return np.roll(data, shift_range)

#CHANGER LA TONALITE
def pitch(data, sampling_rate, pitch_factor=0.7):
    return librosa.effects.pitch_shift(data, sr=sampling_rate, n_steps=pitch_factor)
```

Figure 3: *Augmentation des données*

Nous avons besoin de deux vues différentes venant de la même source, on combine donc ces augmentations pour obtenir: un fichier augmenté avec noise et stretch, qui correspond à un grésillement de fond et un ralentissement de la vitesse; et un autre fichier augmenté avec shift et pitch qui va couper le fichier audio en deux parties et échanger leurs ordres puis changer la tonalité du son.

MocoV3 contient déjà une technique d'augmentation pour les deux vues de l'image. La question est donc de savoir qu'elle est la plus efficace ou même si c'est intéressant de partir sur une fusion de ces deux techniques. Nous avons expérimenté cela et la conclusion est qu'il ne faut pas combiner ces techniques sinon Moco n'arrive pas à reconnaître les paires positives, qui sont trop éloignées.

Une autre idée que nous avons eu a été de faire dans un premier temps une augmentation du dataset EmoDB: on applique pour chaques fichiers originals les augmentations vues plus haut, et on les enregistre en tant que fichiers distincts puis on applique la méthode MocoV3 de base. Le but derrière cette démarche a été de pallier au problème de la base de données trop petite pour considérer les tailles de batchs les plus optimaux sur MocoV3; cependant, elle n'a pas marché car il va donc considérer les couples d'audio: originaux, bruité avec *"noise stretch"* et bruité avec *"shift pitch"* comme étant des paires négatives respectivement. Il a alors du mal à les différencier et à les séparer dans l'espace des vecteurs.

3.2 Conversion en spectrogramme

Nous avons lu plusieurs articles de recherche conventionnels parlant du traitement de fichiers audios dans le domaine de l'apprentissage. Il existe plusieurs métriques à extraire d'un son pour pouvoir capter ses caractéristiques et les faire apprendre à une machine. Celle que l'on a retenue est le Mel-spectrogramme. Pour ce faire on utilise la bibliothèque librosa en python. D'abord, on charge le fichier audio pour en récupérer un vecteur d'échantillons et la fréquence d'échantillonnage. Chacune des valeurs numériques dans le vecteur corresponds à sa position dans la représentation de l'onde sonore en amplitude dans le temps. La fréquence d'échantillonage est une valeur en Hertz qui correspond aux nombre de fois par seconde que le signal sonore a été convertis en valeur numérique. Ensuite on applique la transformée de Fourier à court terme (STFT) pour transformer le signal audio en une représentation temps-fréquence et amplitude. De manière visuelle on les observes celon 3 dimensions: temps en abscisse, fréquence en ordonné et l'amplitude en couleur.

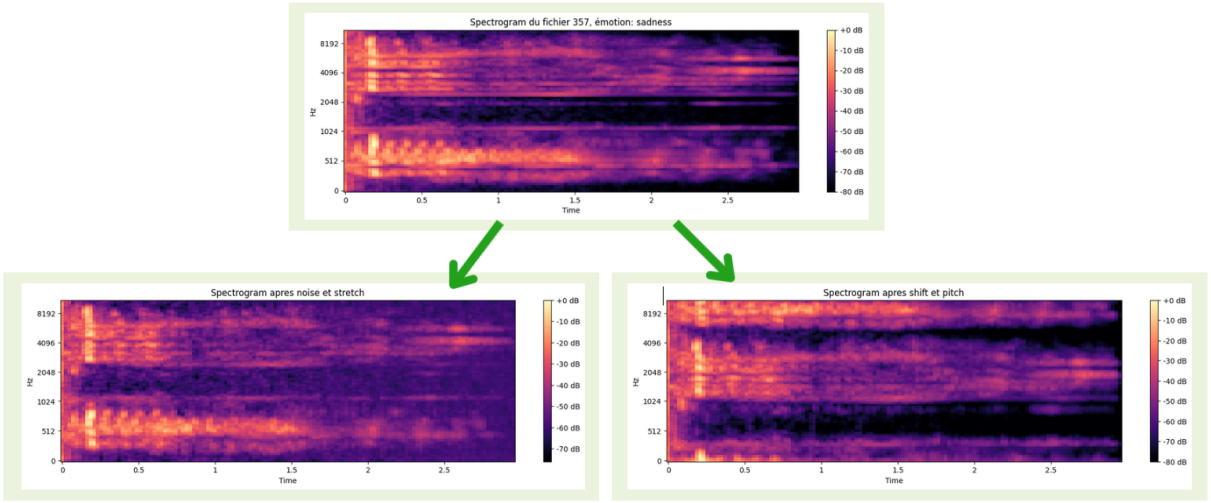

Figure 4: *Comparaison spectrogrammes originaux et augmentés*

Dans les spectrogrammes classiques les fréquences sont répartis linéairement, ce qui accorde autant d'importance aux basses fréquences qu'aux hautes. Cependant, les êtres humains sont plus sensibles aux basses fréquences quand ils parlent. Nous avons donc décider d'utiliser les mel-spectrogrammes pour représenter les fichiers audios car en plus d'utiliser une échelle logarithmique afin de privilégier les fréquences basses, ils sont souvent cité comme meilleur pour la reconnaissance vocal humaine et donc la classification d'émotion.

4 Erreur avec MocoV3

Une erreur que nous avons eu sur ce projet est le fait que le code original de MocoV3 interdit l'exécution sans distribution (multi-GPU). Nous avons d'abord pensé que le problème serait réglé grâce à l'environnement distant, donc nous avons attendu d'y avoir accès. Comme celui-ci ne nous donnait accès qu'à également un GPU, nous avons entrepris d'étudier le code plus en profondeur afin de savoir si nous pouvions le modifier et le faire tourner dans notre environnement. Il a donc fallu chercher dans tous les fichiers du code les méthodes et les appels à des fonctions distribuées pour les ré-écrire avec un comportement similaire qui marche sur une seule machine avec un GPU. Cette tâche nous a bloqué pendant longtemps car nous avons chercher beaucoups de solutions alternatives avant de vraiment modifier le code de MocoV3.

Une fois entraîné, nôtre modèle se sauvegarde à son état lors de la dernière epoch. Pour utiliser ces résultats et faire la classification à proprement parlé on doit passer par l'étape de linear probing. Cela est fait dans le fichier main_lincls.py de MocoV3. Cependant, malgré une analyse approfondie et plusieurs essais de modification, nous n'avons jamais réussi à utiliser les sauvegardes de nos différents modèles. A chaque fois le code échouait à geler les couches du backbone et à remplacer la couche de tête par une couche linéaire.
A partir de ce moment, nous avons alors essayé une autre solution: implémenter nous-même un code pour le linear probing. Ce code peut être retrouvé dans le fichier *"DL2425_linear_probing.py"*. De plus le chargement d'un modèle pré-entrainé pour le fine-tunning ne fonctionnait pas non plus comme nous allons le détailler dans la prochaine section.

5 Entrainement et Résultat pour MocoV3

Le plan que nous avions pour entrainer MocoV3 et obtenir un modèle robuste sur la nouvelle base de donnée était de faire du fine-tunning. Cela a pour objectif de pallier à deux problèmes principaux: le manque de données pour EmoDB et la limitation technique des ressources à notre disposition. D'abord, nous récupérons un modèle pré-entraîné mis à disposition par les créateurs

sur leurs GitHub. Son état est chargé et l'entrainement auto-supervisé est repris pour l'adapter à des nouveaux types d'images: les Mel-spectrogrammes. Le problème est que le code load des clés différentes que celle sauvegardées dans le fichier pré-entrainé. Notre idée est que cela est du aux différentes version de torch utilisé pour sauvegarder et pour charger. Les auteurs utilisaient une torch 1.9 seulement disponible avec Python 3.9, nous avons essayé de configurer cela sur le Docker distant mis à disposition sur ecampus, mais nous n'avons pas réussi.

Par manque de temps et par peur de ne pas avoir de résultats à montrer nous avons pris la décision d'entrainer un modèle de 0, directement sur EmoDB et d'effectuer le linear probing avec notre code modifier. Cette décision a eu des conséquences car nous ne pouvions dépasser un batch_size de 128 (du fait de la limitation de la mémoire RAM) pour l'entrainement auto-supervisé, ce qui est bien trop faible.
Ainsi les résultats obtenue ne sont pas satisfaisant mais pour étudier le comportement du modèle, on se concentre sur la forme et l'évolution des courbes plutôt que sur les résultats en eux mêmes.

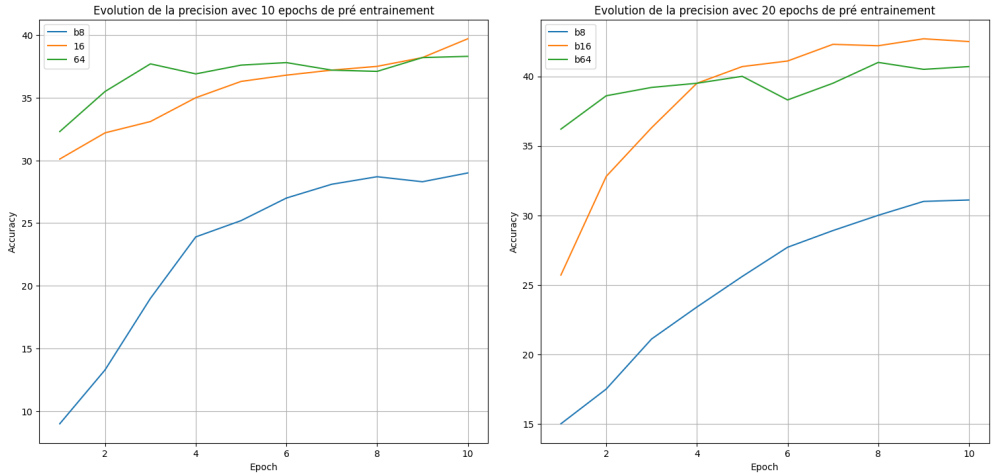

Figure 5: **Evolution de la précision par le nombre d'epochs d'entrainement**
modèle: Vit Small, optimizer: AdamW, lr: 0.01

Plusieurs points intéressant peuvent être observés ici. Tout d'abord, on remarque que la précision obtenue est meilleur en tous points pour les modèles pré-entrainés sur 20 epochs. De plus la meilleur précision obtenue a été faite sur 30 epochs de pré-entrainement. Cela montre déjà une certaine capacité du modèle à s'améliorer au fil du temps, et laisse alors suggérer que des résultats plus convainquant aurait pu être obtenue avec plus de temps.
On remarque également qu'un des points soulevé dans l'article de MocoV3 est également présent ici. C'est le difficlté que représente le faite de considérer de grandes tailles de batchs . Dans le deuxième graphique, le modèle entrainé avec un batch size de 16 dépasse celui de 64 en terme de précision. Cela a été montré dans le papier: plus les batchs sont grands plus l'instabilité augmente lors de l'entrainement.

Epochs auto-supervisé	**Epochs Linear Probing**	**Batch_size**	**Précisions**
30	20	32	51.246

Table 1: Meilleur précision obtenue

Les temps d'executions sont d'environ 1h45 pour les modèles pré-entrainés avec 20 epochs.

6 Classifieur audio

Par peur de ne pas obtenir des résultats pertinants et par besoin de confirmer que l'analyse et le traitement des données n'etaient pas inadéquate; nous avons décidé d'implémenter un modèle

de classifieur basé sur la convolution. Cela nous a aidé à détecter les meilleurs paramètres de traitement des sons. Nous l'avons construits en nous inspirant de plusieurs autres vu les articles consacrés à la raconnaissance d'émotion dans l'audio. Il a aussi été affiné lors des tests que nous avons effectués. Ce modèle reprend une structure très classique se composant de combinaisons de couches de convolutions et de pooling succesives. Les convolutions sont utilisées à différent niveaux pour capturer des motifs importants, ces couches sont reconnus comme particulièrement efficace dans la classification et ont été utilisé dans tous les papiers que nous avons lus. Les Batch normalisation standardise les données avant de passer par les fonctions d'activations pour rendre le modèle efficace contre les valeurs extrêmes qui peuvent survenir des augmentations. Enfin, il se termine sur des couche fully connected pour la classification.

Layer	**No. filters**	**Kernel Size**	**Stride**	**Padding**	**Dropout Rate**
Conv1D	256	5	1	same	-
ReLU	-	-	-	-	-
MaxPool1D	-	5	2	same	-
Conv1D	256	5	1	same	-
BatchNorm1D	-	-	-	-	-
ReLU	-	-	-	-	-
MaxPool1D	-	5	2	same	-
Conv1D	128	5	1	same	-
BatchNorm1D	-	-	-	-	-
ReLU	-	-	-	-	-
MaxPool1D	-	5	2	same	-
Dropout	-	-	-	-	0.2
Conv1D	64	5	1	same	-
BatchNorm1D	-	-	-	-	-
ReLU	-	-	-	-	-
MaxPool1D	-	5	2	same	-
Flatten	-	-	-	-	-
Dense	64	-	-	-	-
BatchNorm1D	-	-	-	-	-
ReLU	-	-	-	-	-
Dropout	-	-	-	-	0.25
Dense	7	-	-	-	-
Softmax	-	-	-	-	-

Table 2: Structure du modèle de réseau de neurones utilisé pour la classification des émotions audio.

Lors de l'entrainement, on utilise la CrossEntropy comme fonction de perte et l'optimizer Adam.

7 Evaluation du modèle

découpage (ms)	**précision**
128	61.26
256	64.74
512	71.82
1024	77.45
entier	65.30

Table 3: Précision celon le découpage

La table 3 décrit les résultats que nous avons obtenus celon le découpage des fichiers sonores en milisecondes. On remarque qu'un découpage trop petit ne rend pas service au modèle qui arrive moins bien à extraire des données intéressantes.

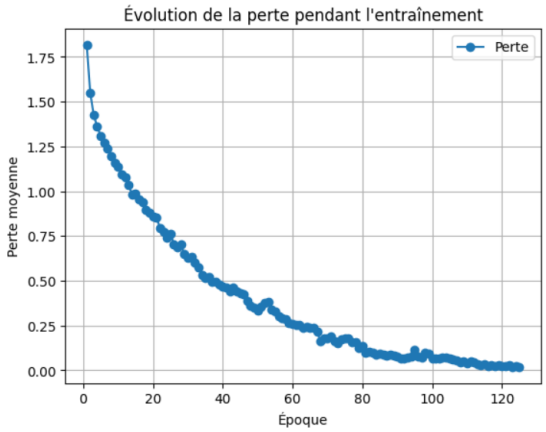

Figure 6: *Courbe Loss*

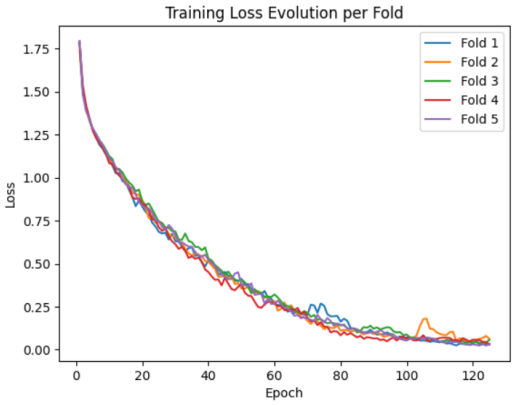

Figure 7: *Comparaison spectrogrammes originaux et augmentés*

Les courbes loss ont une forme qui suggère un bon apprentissage du modèle. De plus, la cross-validation effectuée montre bien l'absence de sur-apprentissage.

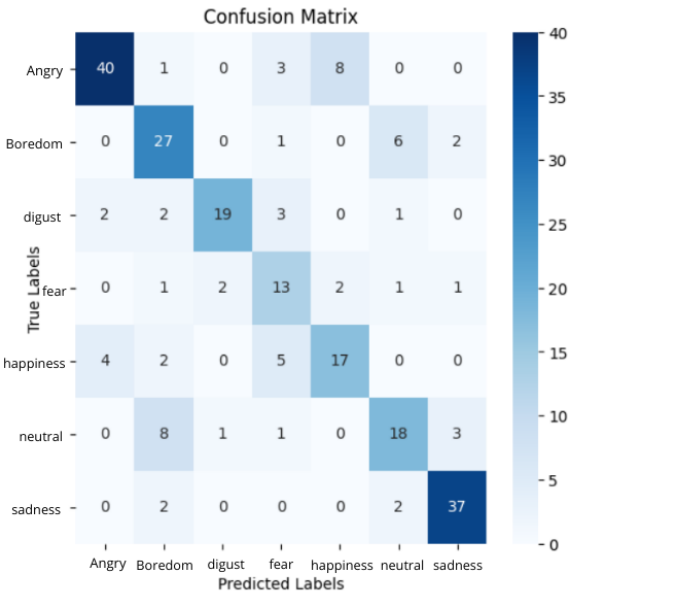

Figure 8: *Matrice de confusion*

Sur l'ensemble de test, on a donc ces résultats:

- Angry: 77%
- Boredom: 75%
- Disgust: 69%
- Fear: 66%
- Happiness: 61%
- Neutral: 59%
- Sadness: 90%

La matrice de confusion est intéressante dans le domaine de la classification car elle nous permet de visualiser qu'elles classes le modèle a tendance à confondre avec d'autres. Dans ces conditions, il a du mal à faire la distinction en Boredom et Neutral; de même avec Angry et Happiness. Dans les 2 cas, ces similitudes peuvent s'expliquer par le faîte que ces emotions sont proches en terme d'amplitude sonore.
Un point intéressant à constater aussi est la différence de précision pour chaque classe. Si la haute précision de l'émotion Angry peut s'expliquer par la prédominance de cette émotion dans EmoDB; les 90% d'accuracy pour Sadness est plus intéressante. Ce phénomène est dû à la basse fréquence utilisé généralement par les personnes tristes qui sont bien capturé par les Mel-spectrogramme.

8 Conclusion

En conclusion, plusieurs points prouvent que de bons résultats sont possibles pour classifier des émotions à partir de fichiers audios en utilisant l'apprentissage auto-supervisé de MocoV3. D'abord, le traitement des sons (segmentation, bruitage et conversion en Mel-spectrogramme) est bien effectué. Les bonnes performances du classifieur en sont la preuve. De plus, les tests expérimentaux sur MocoV3 sont encourageant de part l'augmentation de la précision par rapport aux nombres d'epochs utilisé dans l'apprentissage auto-supervisé.