Rapport final
Parcours IA : prédiction de tags musicaux

LE CHENADEC Elouarn
ARNAUD Axel
3A

Année universitaire 2024-2025

Présentation du sujet

Mewo est une plateforme de gestion de catalogues pour les bibliothèques de musique de production utilisée par de grands acteurs français des médias, de la publicité et de la musique. Mewo ont déjà un système de labellisation automatique qui prédit des scores numériques indiquant la pertinence d'un morceau pour différents tags qui sont organisés en trois grandes catégories : genres, instruments, humeurs ("mood"). Cependant, pour exploiter ces prédictions, il est nécessaire de les convertir en décisions binaires : associer ou non un tag à une musique.

L'approche présentée dans le challenge utilise un seuil par tag pour maximiser un F1-score individuel. Bien que performante, cette méthode ne prend pas en compte les relations entre tags, catégories ou classes. Bien sûr le challenge date de 2021 et leur processus de labellisation a certainement un meilleur benchmark aujourd'hui. L'objectif donné par le benchmark est un weighted F1-Score de **54%** sur le **train set** et **46%** sur le **test set**.

Le challenge consiste donc à concevoir une méthode plus avancée pour améliorer la qualité globale de la classification des labellisations.

Présentation du dataset

Nous avons à disposition un dataset important avec plus de **110 000** inputs, avec **290** features. Ces features décrivent la probabilité pour un tag de correspondre à une musique. Ces tags sont divisés en trois grandes classes : **instrument**, **genre** et **mood**. Il y a également 40 tags de **catégories** qui n'ont pas besoin d'être prédits et qui permettront ainsi de mieux prédire les tags des trois classes. Le fait que nous ayons un si grand nombre de features peut être un problème non négligeable étant donné qu'on devra ainsi traiter avec un espace de données complexe, amenant une difficulté computationnelle et un risque de surapprentissage. Nous ne pouvons en plus pas vraiment réduire le nombre de features car la plupart doivent être classifiées.

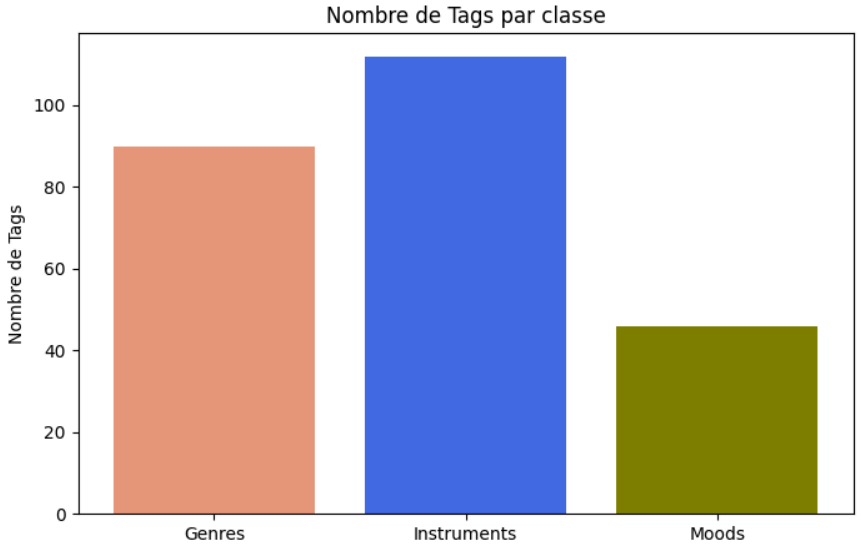
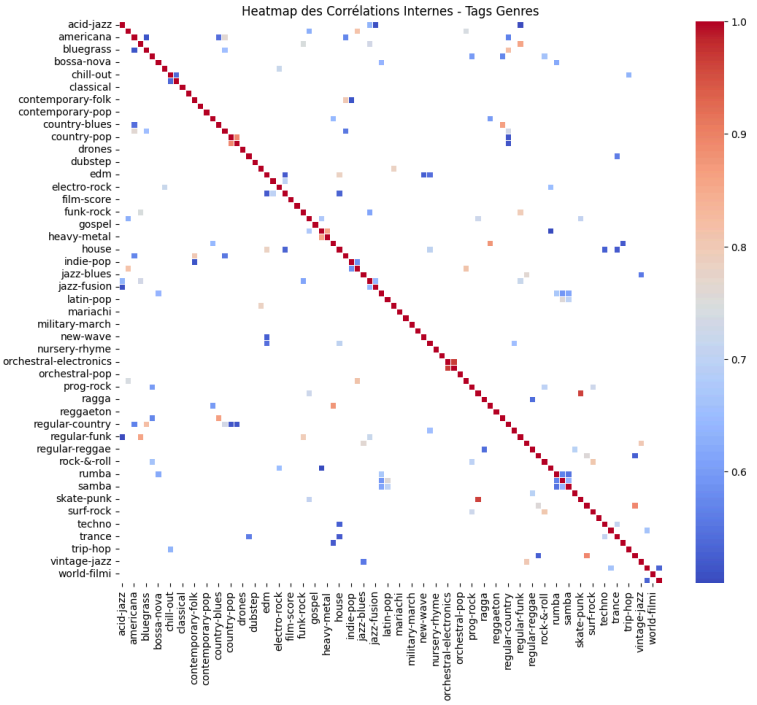
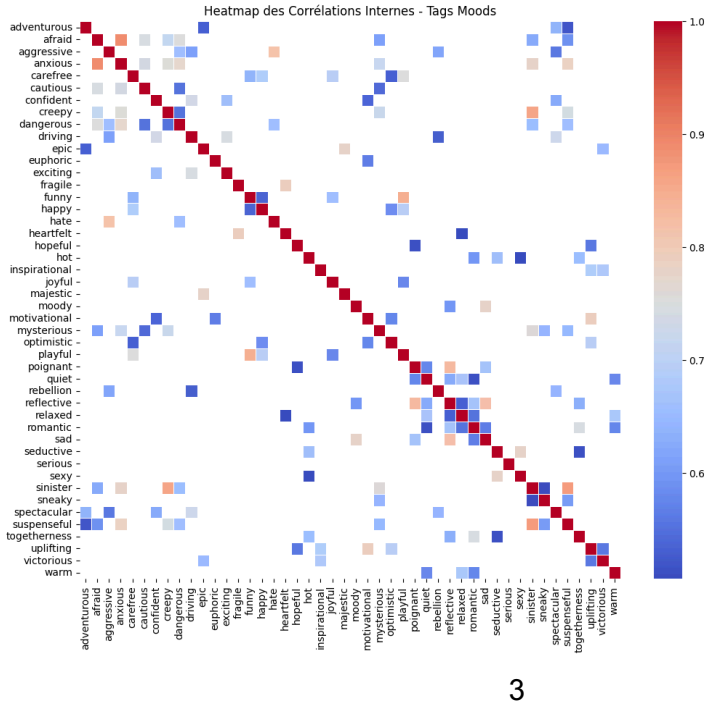


fig 1- Histogramme du nombre de tags par classe à prédire

Au niveau de la corrélation dans le jeu de données, on a ressorti quatre graphiques montrant les corrélations inter-classes et intra-classes :





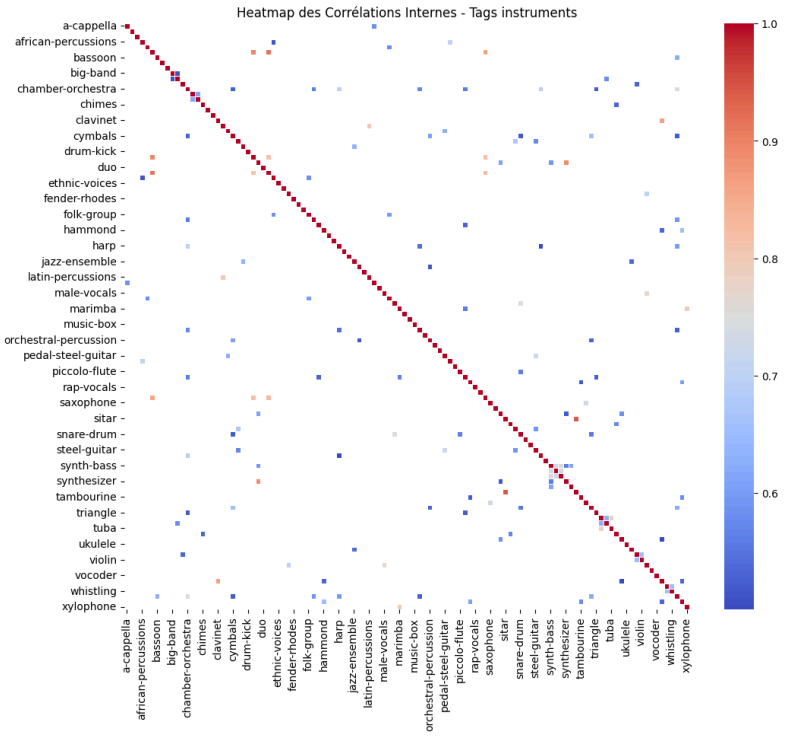


fig 2- Heatmaps des corrélations internes au sein des trois classes à prédire

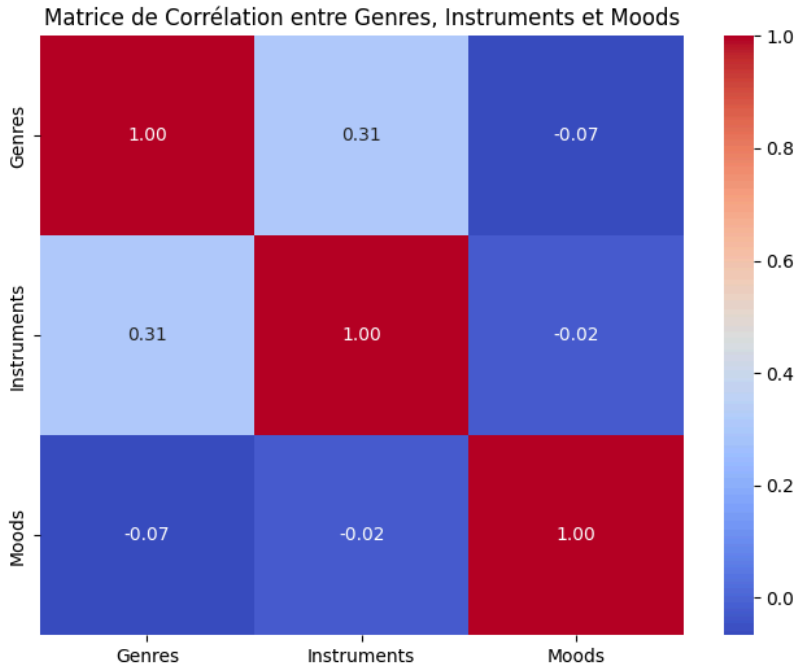


fig 3- Matrice de corrélation entre les trois classes à prédire

On remarque à première vue que les features et les classes de features sont peu corrélées entre elles. Un classifieur classique aura ainsi du mal à classifier notre jeu de données car c'est comme si on lui présentait en entrée trois jeux de données qui n'ont rien à voir entre eux. Il faudrait donc utiliser trois modèles en parallèle qui traitent chaque classe puis dont les sorties seraient traitées ensemble (sorte d'approche MoE) ou utiliser un modèle qui pourrait déceler des relations implicites entre et dans les classes.

Nous avons calculé l'indice de Gini, qui rend compte de l'inégalité de distribution au sein d'une population, et avons trouvé un **indice très proche de 1** (~0.99) pour chacune des classes, ce qui se traduit par une distribution très inégale. La courbe de Lorenz (cf ci-dessous) démontre également une distribution déséquilibrée des tags actifs : on peut remarquer qu'il y a environ 20% des tags qui représentent 60% de l'activité des tags.

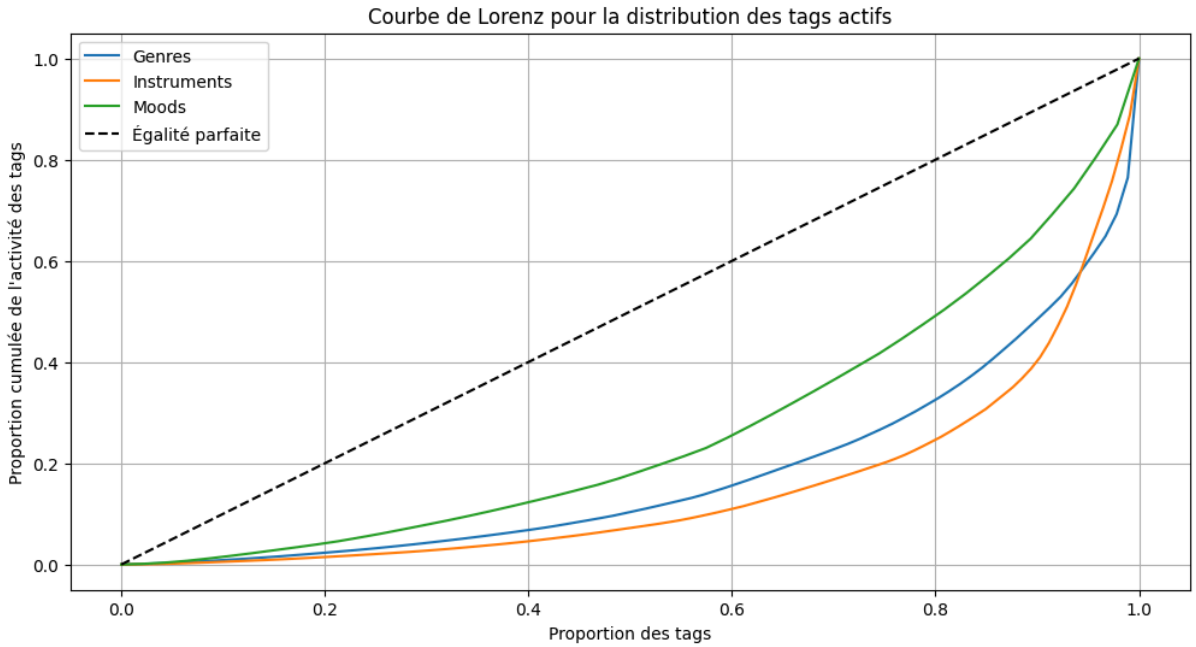


fig 4- Courbe de Lorenz pour la distribution des tags actifs

Comme on peut le remarquer sur le graphique ci-dessous, les moyennes des classes sont très basses. Ceci est dû au fait que la plupart des tags sont inactifs, tout du moins proches de 0, et les tags actifs sont minoritaires, de part l'inégale répartition de l'activité des tags au sein de notre jeu de données. Cela va représenter un défi lors de la classification car le modèle utilisé pourrait avoir une mauvaise performance pour les tags rares et un biais pour les tags prédominants.

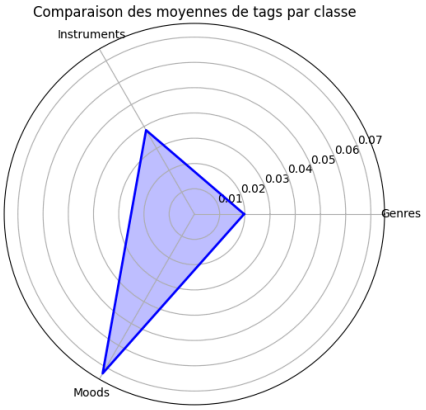


fig 5- Moyennes de tags par classe

Environnement technique

Langage de programmation : python

Dépendances utilisées :

- pytorch : gérer le modèle du transformer
- reservoir.py : gérer les modèles des ESNs
- pandas : préparation et sauvegarde des données
- numpy
- scikit-learn : permet de séparer les datasets et de créer le rapport avec le F1-score
- matplotlib et seaborn : génération des graphiques
- optuna : recherche des hyperparamètres

Evolution de la réflexion

Nous avons tenté plusieurs approches au début de notre projet. La plupart ont été infructueuses. Ainsi nous avons essayé de traiter le jeu de données en utilisant un réseau neuronal assez basique et un algorithme de random forest. Malheureusement ces deux options n'ont pas pu converger vers un résultat probant.
Nous avons alors eu l'idée de combiner deux architectures pourtant très différentes : le Transformer et les Echo State Networks (ESNs). Ces deux architectures neuronales présentent des avantages qui nous intéressent pour traiter notre jeu de données.
Le **Transformer** arrive à traiter des espaces de données complexes sans difficulté, et son attention multi-head non uniforme pourrait pallier aux problèmes de la distribution inégale de l'activité au sein des classes et de la quasi-division de notre jeu de données en trois. Cependant le transformer peut tout de même avoir tendance au sur apprentissage sur des données à la répartition inégale, ce qui sera à surveiller.
Les **ESNs** sont d'habitude utilisés pour prédire des séries temporelles, mais leur faculté à enrichir les corrélations au sein d'un jeu de données et à amener de la non-linéarité nous intéressaient, car ils pourraient permettre de faciliter le traitement du jeu de données par le transformer.
Bien sûr cette idée d'architecture est davantage basée sur une intuition/envie d'expérimenter qu'autre chose, mais étant donné que c'était la solution la plus efficace que nous avons essayé, et qu'elle battait le benchmark et les meilleurs scores obtenus au challenge sans recherche des hyper paramètres optimaux, nous nous sommes cantonnés à celle-ci.

Nous avons essayé plusieurs architectures avec les sorties des ESNs en entrée du transformer, combinées avec les inputs :

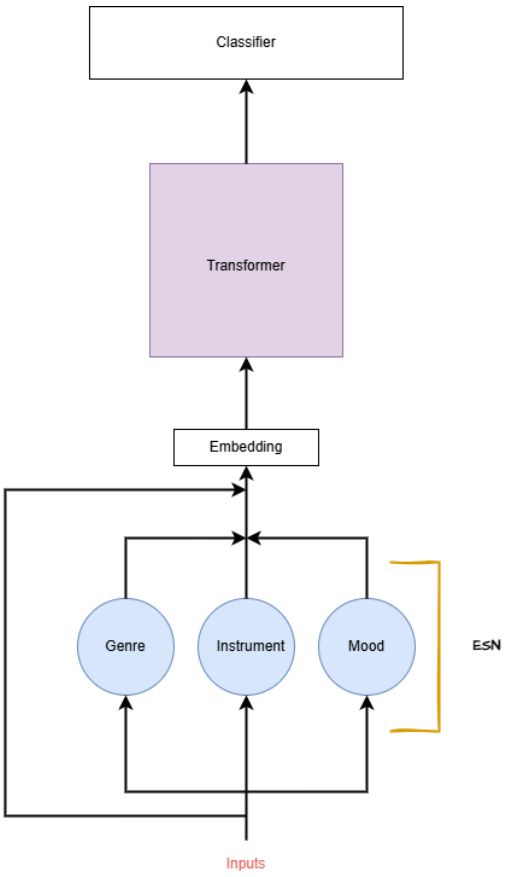


fig 6- Architecture avec un ESN pour chaque classe

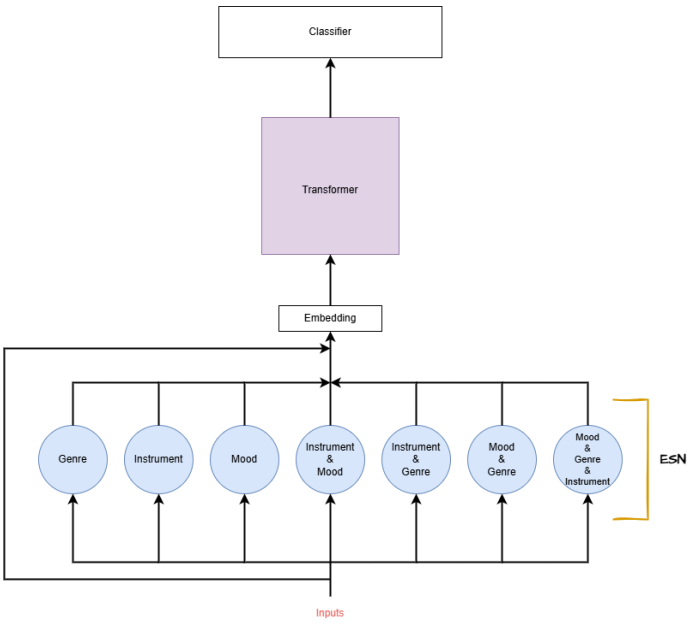


fig 7- Architecture avec un ESN pour chaque classe et pour chaque relation inter-classe

Nous avons ainsi essayé de combiner en entrée du transformer un ESN pour chaque classe, un ESN pour chaque relation inter-classe, un ESN pour les tags de type catégorie, pas d'ESN (juste un transformer)... et les meilleurs résultats obtenues l'ont été avec trois ESN en entrée du transformer : un pour chaque classe de tags à prédire.

Architecture du modèle

Notre architecture (*fig 6)* est paramétrée de façon particulière pour qu'elle fonctionne de façon optimale. Bizarrement, les deux architectures utilisées sont normalement utilisées dans un contexte "temporel", avec de la récurrence, ce dont on n'a a priori pas besoin. Nous avons paramétré les ESNs pour qu'ils se comportent comme des Extreme Learning Machines (ELMs). Nous avons ainsi paramétré le *spectral radius* à 0, enlevant toute récurrence dans le réservoir, qui se comporte ainsi comme une couche de neurones aux poids aléatoires avec une couche de neurones entraînée en sortie (readout). Nous avons déterminé le nombre de neurones par ESN à 100 et le *leaking rate* à 1.

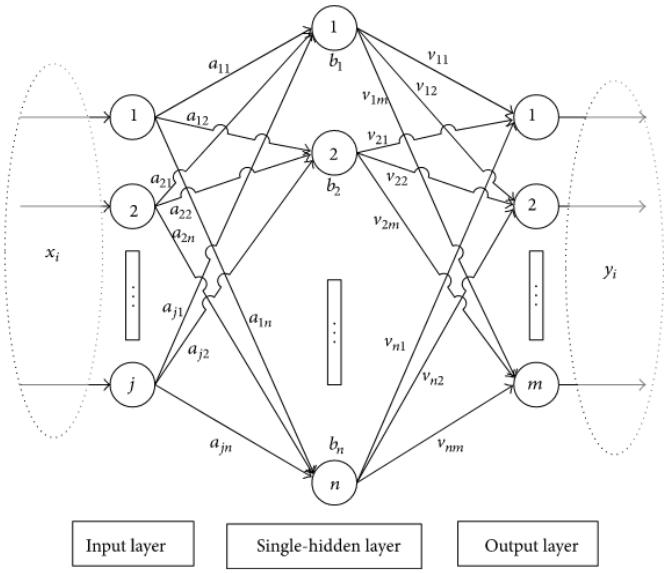


fig 8- Architecture d'une Extreme Learnig Machine

Les sorties des ESN sont toutes concaténées avec les inputs originaux lors de l'embedding. Le tout est ensuite introduit dans le transformer (nn.transformer de pytorch), qui est paramétré de façon basique.

```python
class MultiTaskTransformer(nn.Module):
    def __init__(
        self, input_size, embedding_size, num_heads, num_layers, num_labels, dropout
    ):
        super(MultiTaskTransformer, self).__init__()
        self.embedding = nn.Linear(input_size, embedding_size)  # Embedding Layer
        self.transformer = nn.Transformer(
            d_model=embedding_size,
            nhead=num_heads,
            num_encoder_layers=num_layers,
            num_decoder_layers=num_layers,
            dropout=dropout,
            batch_first=True,
        )
        self.classifier = nn.Linear(embedding_size, num_labels)  # Final Classifier
```

fig 9- Structure du transformer

Entraînement

Avant d'entraîner le modèle, nous avons séparé les données brutes en neuf fichiers CSV en nous basant sur une table de correspondance entre classes, catégories et tags fournie par Mewo :

- 3 fichiers contenant les prédictions des tags des musiques en fonction des trois grandes classes (instrument, mood ou genre).
- 3 fichiers contenant les prédictions des catégories des musiques en fonction des mêmes trois grandes classes.
- 3 fichiers contenant les prédictions binaires en sortie des tags des musiques en fonction des trois grandes classes.

Nous chargeons ensuite ces neuf fichiers dans des dataframes avant l'entraînement.
Pour constituer les jeux de données, nous répartissons les données de la manière suivante :

- 64 % pour l'entraînement,
- 20 % pour le test,
- 16 % pour la validation.

L'entraînement se déroule en deux temps en raison de l'utilisation de deux bibliothèques : ReservoirPy pour les ESNs (Echo State Networks) et PyTorch pour le Transformer.

Entraînement des ESNs

- Chaque ESN est entraîné à prédire un tag de sortie à partir des prédictions en entrée.
- Une fois entraînés, les ESNs génèrent les tags de sortie pour toutes les données d'entrée.

Préparation des données pour le Transformer

- Les prédictions des ESNs sont concaténées avec les données d'entrée pour constituer les jeux de données finaux.

Entraînement du Transformer

- Le Transformer est entraîné à classifier les tags de sortie (0 ou 1) à partir des prédictions de tags et de catégories fournies par Mewo, ainsi que des prédictions des ESNs.

Une fois une architecture optimale trouvée, nous avons procédé à une recherche d'hyper paramètres à l'aide du framework Optuna. Ce dernier permet :

- de sauvegarder la progression de la recherche,
- d'offrir un tableau de bord pour suivre l'entraînement,
- d'analyser l'importance relative des hyperparamètres.

L'objectif était de minimiser la perte moyenne sur le jeu de validation. Voici les meilleurs hyperparamètres trouvés :

Paramètre	Nombre de têtes	Nombre de couches	Taille d'embedding	Taux d'apprentissage	Dropout
Valeur	15	2	285	1.577e-4	0.262

On peut voir ci-dessous qu'on a atteint la limite de l'entraînement, le modèle converge correctement sans tomber dans le surapprentissage.

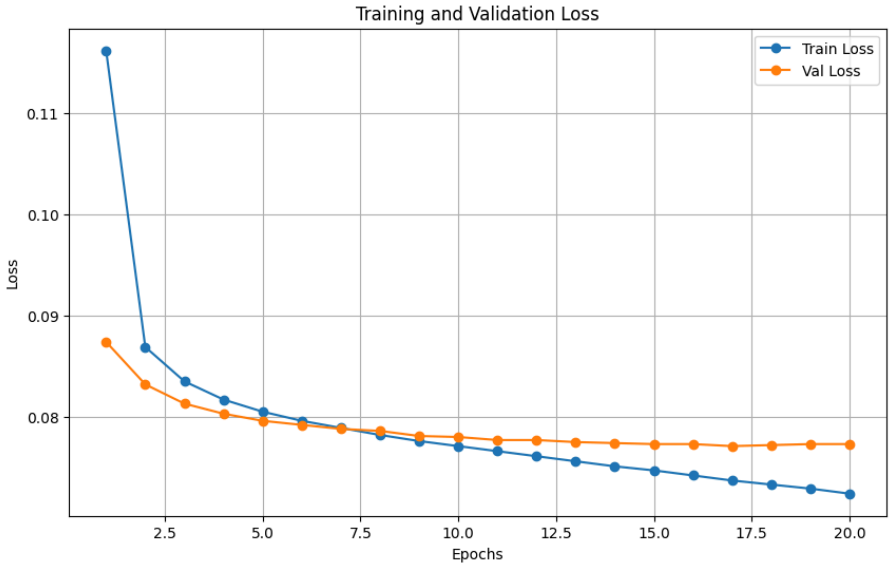


fig 10- Loss d'entraînement et de validation

Résultats

Finalement, l'utilisation de réservoirs améliore légèrement les résultats.
Résultat sur le dataset de test :

	Avec ESN	Sans ESN
Weighted avg F1 Score	**0.495**	0.484
Accuracy	0.97	0.97
Test Loss	0.0777	0.0784

Ces résultats sont meilleurs que ceux du classement du challenge mais le score est fait sur une partie du x_train qu'on a séparé en amont de l'entraînement du reste et qui n'a donc pas servi à entraîner le modèle. Les résultats sont tout de même prometteurs.

Rang	Date	Participant(s)	Score public
1	4 avril 2021 21:11	gcanat	0,4785
2	21 novembre 2021 21:30	aho	0,4651
3	12 avril 2021 03:17	huy217	0,4650
4	15 décembre 2021 20:53	vincent.bour	0,4646
5	28 janvier 2021 22:50	cakedev	0,4585
6	12 décembre 2021 09:40	luguedon & BaptisteBenard	0,4575
7	16 décembre 2021 12:35	ppavia & enthomas	0,4568
8	17 décembre 2021 16:55	mkouhou & Maelle_a	0,4563
9	7 avril 2021 16:42	yfe	0,4562
10	8 mars 2021 18:24	Dupin_Sylvio	0,4562
11	12 novembre 2021 10:23	serrabii & slebdaoui	0,4562
12	-	**benchmark**	**0,4561**

fig 11- Résultats au challenge data

Si on compare les fréquences du nombre de prédictions du modèle par rapport à tags réels on obtient le graphique suivant. Il manque encore une grande partie de tags à prédire.

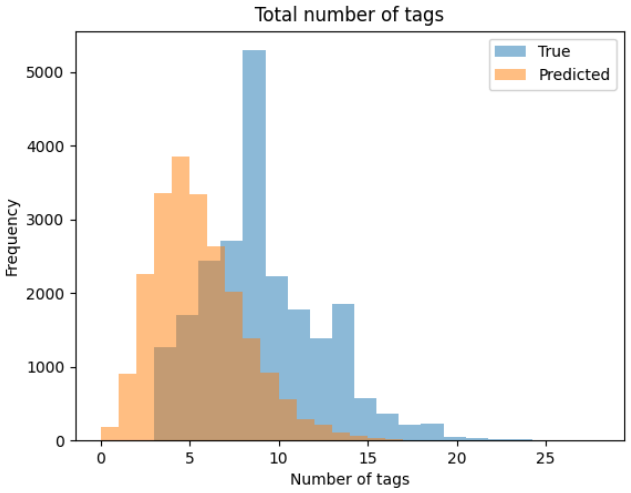


fig 12- Comparaison entre nos prédictions et les vrais tags

Cependant nous battons bien le modèle benchmark sur un autre jeu de test fourni par Mewo.

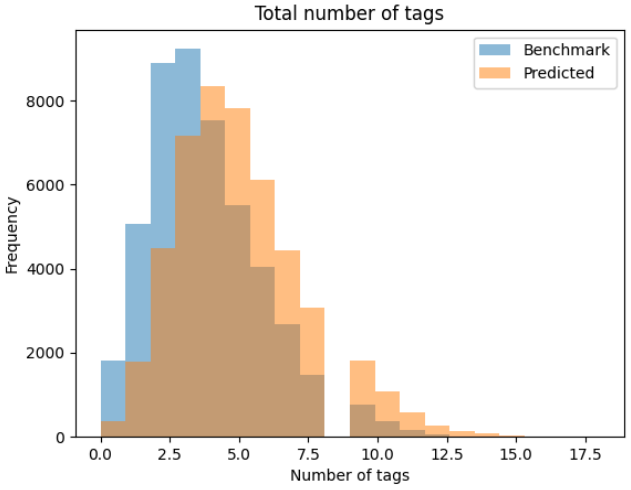


fig 12- Comparaison entre nos prédictions et le Benchmark

Répartition du travail

Elouarn	Axel
Code du transformer	Préparation des données
Code général	Analyse des données
Entraînement	Code des ESNs
Recherche d'hyper paramètres	

A deux
Réflexion sur les modèles, veille technologique, rédaction des rapports

Nous avons globalement travaillé en même temps, dans la salle IA et la plupart du temps lors des créneaux dédiés durant les semaines de spécialisation.

Retour sur le projet

Nous sommes satisfaits des résultats obtenus par notre modèle, étant donné que nous avons battu tous les scores du challenge original. Ce challenge date de 2021, mais le module pytorch de transformer que nous avons utilisé date de 2020, même s' il a pu avoir quelques modifications depuis. Nous avons donc utilisé des technologies existantes au moment du challenge, les ESN étant facilement programmables sans la bibliothèque reservoir.py qui est elle antérieure au challenge.
On peut néanmoins remarquer que l'utilisation des ESN (ou plutôt ELM) n'apporte qu'une légère amélioration du F1 score. Utiliser un transformer simple suffirait pour obtenir un score battant le challenge, ce qui porte à réfléchir sur les moyens à disposition à l'époque ainsi que sur le fait que nous n'ayons pas pu obtenir la partie "privée" du jeu de données, qui nous aurait permis de comparer sur un jeu de données différent.
Il aurait été tout autant intéressant d'essayer de résoudre ce challenge en utilisant plusieurs modèles qui traitent en parallèle différentes parties du jeu de données, ce qui aurait été une solution plus logique mais que nous n'avons pas exploré étant donné que notre solution "originale" était déjà efficace.