Rapport de projet

Sujet :

Classification de données multimédias et étude cross-modale d'un corpus

Réalisé par :

Théotime Dmitrašinović

Valentin Lafargue

Léa Fabriol

Formation : M2 SID

Tuteurs académique :

Julien Pinquié

Jérome Farinas

Linda Lechany

Octobre 2023 - Février 2024

Table des matières

1 Introduction

1.1 Présentation du projet

Ce rapport présente le travail réalisé dans le cadre d'un projet académique concernant la classification de vidéos et une étude cross-modale. Ce projet s'inscrit dans la continuité du cours "Données Multimédia" et a pour objectif principal de mettre en pratique les concepts et les méthodes abordés dans ce cours mais aussi découvrir des méthodes non abordées dans ce dernier. Nous avons exploré les diverses modalités de données multimédia, notamment le texte, l'audio et la vidéo, en utilisant majoritairement des modèles pré entraînés. Notre tâche consistait à classifier ces données en utilisant des techniques de fine-tuning, si nécessaire, pour les modalités de vidéo, texte et audio.

Dans les sections suivantes de ce rapport, nous détaillerons les étapes que nous avons suivies et les méthodes employées (2). Nous présenterons également les résultats de nos travaux (3).

1.2 Présentation des données

Pour notre projet, nous avons exploité QuerYD [**?**], un jeu de données spécialement conçu pour la recherche et la localisation d'événements dans les vidéos. Ce jeu de données se caractérise par la présence de descriptions accompagnant chaque vidéo. En effet, pour chaque vidéo, nous avons une description écrite des évènements visuels de la vidéo pour les mal voyants ; de plus, toujours pour ces derniers, un grand nombre de vidéos est accompagné d'une deuxième piste audio : nous avons évidemment l'audio d'origine, mais aussi une description verbale détaillée du contenu visuel, créant ainsi un contexte riche pour notre analyse. Cette description auditive est généralement différente de la description écrite.

1.2.1 Limites des données

Il est important de noter que les classes attribuées à chaque vidéo sont définies par les créateurs de contenu sur YouTube. Nous avons identifié que la répartition des classes était inégale, avec la catégorie 'éducation' représentant presque 30% des données, ce qui pouvait potentiellement biaiser nos résultats. C'est un fait que nous avons pris en compte par la suite en adaptant le traitement des données afin de réduire les disparités entre les effectifs des classes et l'impact sur les résultats.

1.3 Stratégie générale

Dans notre but premier qu'est la classification, ainsi que notre but secondaire que nous aborderons brièvement qu'est la cross-modalité ; nous avons besoin d'un pré-traitement des données avant de pouvoir appliquer des méthodes de classification.

Notre stratégie est donc la suivante, commencer par avoir des espaces vectoriels représentants au mieux les modalités textuelles, auditive et visuelle que nous appellerons embeddings. Et par la suite, une fois que nous avons ces modalités sous des formes exploitables, nous appliquerons des méthodes de machine learning classiques ainsi que du deep learning sur ces derniers.

2 Présentation des différentes approches

2.1 Prétraitement des données

2.1.1 Formatage des données

Après avoir récupéré les données dans un drive partagé, nous avons créé des fichiers distincts pour l'ensemble d'entraînement, l'ensemble de test et les métadonnées. Lors de l'exploration des métadonnées, nous avons remarqué que certaines données sous forme d'embeddings étaient destinées à des tâches spécifiques, mais nous n'avions pas suffisamment de contexte pour déterminer s'ils étaient pertinents pour notre analyse ou s'ils étaient destinés à la partie générative de l'article. Nous avons donc choisi de ne pas les inclure dans notre analyse principale.

Pour garantir la cohérence des données, nous avons opté pour le format CSV. Nous avons utilisé la fonction enumerate pour traiter les données de manière itérative, en veillant à ce que chaque élément soit correctement aligné à son label.

Nous avons exclu les vidéos qui ne possédaient pas de labels et celles qui n'étaient pas en anglais. Cela nous a permis de travailler avec un ensemble de données plus ciblé et d'améliorer la qualité de nos résultats. Le dataset final contient 2280 vidéos exploitables.

2.1.2 Gestion des fichiers audio

En ce qui concerne les fichiers audio, nous avons choisi le format MP3 en raison de sa taille plus compacte, réduisant ainsi la quantité de données à manipuler. Cependant, cela a posé un problème lors de l'utilisation de la bibliothèque librosa pour lire les fichiers, en raison de la compression spécifique au format MP3. Pour résoudre ce problème, nous avons dû utiliser une autre bibliothèque pour lire les fichiers MP3, les diviser en morceaux appropriés, puis les relire avec librosa.

2.1.3 Rééquilibrage des classes

2.1.3.1 Sur/sub sampling

En explorant le jeu de données, nous avons constaté que les catégories n'étaient pas équilibrées, avec un écart significatif entre le nombre d'individus dans différentes classes, allant de 732 individus pour la catégorie la plus représentée à seulement 15 pour la catégorie la moins représentée.

Pour remédier à ce déséquilibre, nous avons adopté différentes stratégies de gestion des classes sous-représentées. Tout d'abord, nous avons décidé de dupliquer des individus des classes sous-représentées jusqu'à un seuil spécifique, calculé en divisant la taille totale de l'ensemble de données par le nombre de catégories.

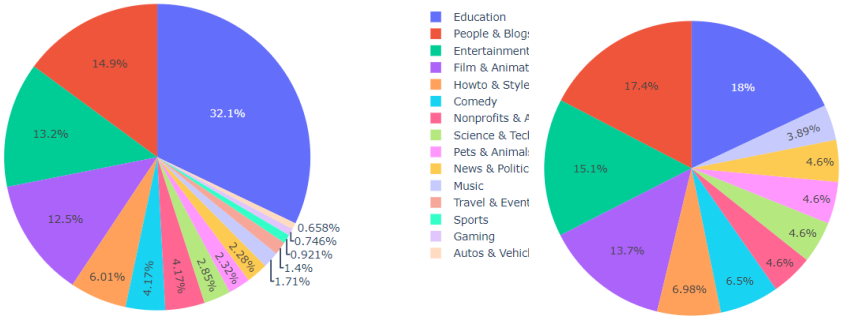


FIGURE 2.1 – Labels avant et après sur et sub-sampling

Nous avons combiné le suréchantillonnage avec la suppression d'individus dans certaines catégories afin d'atteindre un équilibre optimal.

Nous avons réexaminé les vidéos des catégories comme "auto et véhicules", et nous avons constaté que les classifications initiales étaient incorrectes. Presque aucune vidéo ne correspondait réellement à cette catégorie, nous avons donc dans certains jeu de donnée redistribué les individus vers d'autres classes plus appropriées.

Une autre approche que nous avons tentée était la fusion de classes minoritaires pouvant être englobées dans d'autres catégories.

2.1.3.2 ZeroShot Classification

Comme nous le disons ci-dessus, lors de l'équilibrage des données, nous nous sommes demandés si par exemple les classes très minoritaires comme les vidéos avec comme label Auto & Vehicule, nous pouvions les classer dans d'autre classe, cela rendrait notre tâche plus facile, que ce soit lors de l'apprentissage ou lors des analyses des résultats. Nous nous sommes confrontés à un problème que nous n'imaginions pas : les labels Youtube n'étaient pas exactes, en effet parmis cette vingtaine de vidéo ayant comme label Auto & Vehicule, moins de 5 n'avaient qu'un semblant de rapport avec la catégorie youtube donnée.

Confronté à ce problème et se rappelant du diagramme circulaire du papier présentant le jeu de donnée QuerYD, nous avons voulu regarder s'il ne serait pas possible de re-labéliser les données, avec la modalité Textuel où chaque label a un sens sémantique.

Nous avons justement trouvé une telle méthode très prometteuse : ZeroShot Classification ; ZeroShot learning est généralement une méthode qui permettrait classer des éléments dans des classes pré-existantes et potentiellement de classer des éléments dans une nouvelle classe si le modèle le trouverait pertinent. Ici, dans une interprétation purement textuelle, où les labels ont donc un sens sémantique comme expliqué ci-dessus, nous avons trouvé sur Hugging Face une implémentation qui répondait à tous nos critères.

En effet, nous avons pu associer à chaque vidéo, en utilisant la description écrite, un nouveau label parmi lequel nous avons permis le modèle de choisir : les labels présentés dans le papier qui avaient l'air équilibré.

Le résultat fut anti-climatique, nous avons obtenu une classe majoritaire à presque mille individus sur deux mille vidéos, la classe 'accessibility', qui nous avions hésité de mettre dû à notre incompréhension devant ce thème.

En stockant la probabilité d'apparition des autres classes, nous avons été capables de faire différentes approches pour re-équilibrer les classes, telles qu'enlever le label entièrement, ou le garder seulement quand la deuxième classe la plus probable n'était que très peu probable. Cependant, nous avons peur que le classement des labels suivants était affecté par cette classe 'accessibility'. C'est-à-dire qu'enlevé cette classe en post-processing n'est pas équivalent à ne pas la mettre dans les labels à trouver pour le model ZeroShot.

Finalement, nous testerons ultérieurement des modèles avec ces labels pour voir si ces nouveaux labels sont plus pertinents que ceux donnés par Youtube.

2.2 Texte

2.2.1 TF-IDF

Le principe du TF-IDF (Term Frequency-Inverse Document Frequency) est une technique fondamentale en traitement automatique du langage naturel et en recherche d'information. L'intérêt du TF-IDF réside dans sa capacité à extraire les mots clés essentiels d'un document tout en réduisant le poids des mots fréquents et non spécifiques tels que les articles ou les prépositions. Cela permet de représenter les documents sous forme de vecteurs numériques où chaque dimension du vecteur correspond à un terme et sa valeur TF-IDF associée, facilitant ainsi leur traitement par des algorithmes d'apprentissage automatique.

Nous avons effectué un premier test en utilisant TF-IDF avec CountVectorizer sur nos données textuelles fournissant ainsi des embeddings numériques pour chaque document. Ces embeddings ont ensuite été utilisés comme entrées dans nos algorithmes de classification.

2.2.2 DistillBERT

La méthode DistilBERT est une technique d'apprentissage automatique qui le modèle de langage BERT (Bidirectional Encoder Representations from Transformers). Plutôt que d'utiliser le modèle complexe de BERT avec ses nombreux paramètres, DistilBERT utilise un processus appelé distillation pour transférer la connaissance du modèle BERT vers un modèle plus léger et plus rapide.

Ce processus oblige le modèle plus léger à apprendre à imiter les sorties du modèle BERT tout en étant plus efficace en termes de calcul et de mémoire. Ainsi, DistilBERT parvient à conserver une grande partie de la puissance de BERT tout en réduisant considérablement la complexité

du modèle, ce qui le rend plus adapté aux applications nécessitant des ressources informatiques limitées.

DistilBERT, tout comme le modèle BERT dont il est dérivé, est un modèle de traitement du langage naturel utilisé pour effectuer des prédictions sur diverses tâches liées au langage.

Plus précisément, DistilBERT est généralement utilisé pour effectuer des prédictions sur des tâches de classification de texte, où le modèle est entraîné à classer un morceau de texte dans une ou plusieurs catégories prédéfinies.

Nous avons utilisé DistilBERT sur nos différentes informations textuelles : les descriptions écrites ou transcrite des descriptions auditive et les sous-titres de l'audio original de la vidéo. Nous avons fait des approches de manière individuelle, duale ainsi que complète en concatenant les textes quand nous en prenions plusieurs à la fois. Ces combinaisons visaient à tirer parti des informations contextuelles complètes disponibles en espérant obtenir des représentations textuelles plus riches et des prédictions plus précises.

Il s'est avéré que prendre les informations textuelles de la description écrite seule était le plus pertinent en terme d'accuracy globale, ces informations de description semblaient plus efficace que la transcription de l'audio que nous avons obtenu avec Whisper Small, nous vous en parlerons par là suite. Le problème majoritaire de la transcription des descriptions auditive était le fait qu'ils ne couvrait que entre 60 et 70% du corpus de vidéo.

La classification des sorties de DistilBERT à été effectué avec un classifieur spécifiquement à entrainer pour ce type d'embeddings (DistillBERT for sentence classification).

Cette approche combinant la puissance de DistilBERT avec un classifieur dédié, a permis d'obtenir des résultats prometteurs dans notre tâche de classification de texte.

2.2.3 Sentence-BERT

Nous avons appliqué Sentence-BERT à nos données textuelles. En utilisant ce modèle, nous avons généré des embeddings pour nos phrases qui ont ensuite été utilisées comme entrées dans nos algorithmes de classification.

Le principe de Sentence-BERT (SBERT) repose sur une extension du modèle BERT, adaptée spécifiquement pour le traitement de partie de phrases ou de phrases complètes plutôt que de simples mots. Contrairement à BERT classique, SBERT utilise une stratégie appelée "Siamese Network" pour entraîner le modèle à apprendre des représentations sémantiques similaires pour des phrases équivalentes ou sémantiquement proches. Ce processus permet à SBERT de générer des embeddings (représentations vectorielles) de phrases qui captent la sémantique et la similarité entre les phrases de manière plus efficace.

La différence principale entre DistilBERT et Sentence-BERT réside dans leur objectif et leur architecture. DistilBERT, comme son nom l'indique, est une version distillée de BERT, visant à réduire le nombre de paramètres tout en préservant les performances. Sentence-BERT, en

revanche, est spécifiquement conçu pour apprendre des représentations sémantiques de phrases entières.

Nous avons en particulier testé deux implémentations différentes de SentenceBert, en effet l'implémentation paraphrase distilroberta ainsi que l'implémentation MPnet. Les meilleurs résultats de classification avec SentenceBert viendront sûrement de token CLS d'embeddings de SBERT créée spécialement pour la classification, c'est la raison pour laquelle nous avons choisi paraphrase distilroberta ; MPnet lui était jugé comme l'implémentation la plus efficace dans une tâche similaire dans un classement officiel d'hugging face.

Pour réaliser des prédictions à partir de ces embeddings, nous avons choisi d'utiliser des classifieurs tels que le Support Vector Classifier (SVC) et les Perceptrons. Ces classifieurs ont été entraînés sur les embeddings générés par Sentence-BERT, permettant ainsi de tirer parti de la structure sémantique des phrases.

Nous avons obtenu des résultats proches pour ces deux embeddings, avec la classification de MPnet fonctionnant légèrement mieux quand nous regardions l'accuracy globale, c'est pour cela que nous avons choisi cette représentation textuelle lors des études multimodales, quand nous n'avons pas eu l'opportunité de tester avec plusieurs représentations.

2.3 Vidéo

2.3.1 Descripteurs Vidéo

Nous avons entrepris l'extraction de certains descripteurs fondamentaux et faciles à obtenir à partir des vidéos que nous avons analysées. À l'aide des Travaux Pratiques Multimédia-Images, nous avons réussi à extraire deux descripteurs clés : le nombre de coupes (cuts) dans chaque vidéo et la durée totale de la vidéo. Ces deux descripteurs sont non seulement simples à extraire, mais ils sont également intrinsèquement liés, ce qui en fait une paire d'informations intéressante à utiliser conjointement dans notre analyse.

Bien que nous ayons envisagé l'extraction d'images clés pour enrichir notre ensemble de données, nous avons pris la décision de mettre cette approche de côté. À la place, nous avons opté pour l'application de l'algorithme YOLO (You Only Look Once) sur l'ensemble des images de chaque vidéo.

2.3.2 Détection d'objet avec YOLOv7

L'une de nos propositions pour la classification des vidéos consiste à appliquer un algorithme de détection d'objets à chaque vidéo. Ce processus permettrait d'obtenir une liste d'objet par vidéo que nous transformerions ensuite en embeddings. Ces embeddings seraient ensuite introduits dans un modèle de classification pour catégoriser les vidéos.

2.3.2.1 Utilisation du modèle préentrainé :

Nous avons utilisé YOLOv7 (You Only Look Once version 7) pour prédire les objets présents dans les vidéos, exploitant ainsi sa capacité à effectuer des détections d'objets en temps réel.

YOLOv7 est une architecture de réseau de neurones convolutifs (CNN) basée sur un modèle de détection d'objets en une seule étape, ce qui signifie qu'elle peut identifier et localiser les objets dans une seule passe à travers le réseau.

Le fonctionnement de YOLOv7 est basé sur la division de l'image en une grille régulière, où chaque cellule de la grille est responsable de prédire les boîtes englobantes (bounding boxes) et les classes d'objets. Le modèle est capable de prédire plusieurs boîtes englobantes par cellule, chacune caractérisée par les coordonnées du centre, la largeur, la hauteur et la confiance de la prédiction. Ces prédictions sont ensuite post-traitées pour éliminer les boîtes englobantes redondantes et maintenir les prédictions les plus fiables.

Il nous a fallut modifier le code des fichiers permettant de formater les sorties du modèle afin de récupérer les objets détectés dans chaque frame et la probabilité de détection de l'objet associée dans un même fichier.

Nous avons calculé que l'exécution complète de l'algorithme YOLO sur l'ensemble du corpus de vidéos prendrait environ 50 heures. Pour optimiser l'utilisation de nos ressources disponibles, nous avons ensuite divisé cette tâche en plusieurs sous-tâches que nous avons exécutées en parallèle sur sept ordinateurs à notre disposition.

Ces résultats ont ensuite été rassemblés dans un dictionnaire regroupant les prédictions de toutes les vidéos. Ainsi, une fois toutes les vidéos traitées, on dispose d'un dictionnaire contenant pour chaque frame de chaque vidéo, tous les objets détectés et leur probabilité de détection. Notons qu'une seule vidéo n'a pas pu être traité avec l'algorithme yolo (vidéo de 40 minutes et de trop haute qualité).

C'est à partir des objets présents dans les videos nous avons tenté de prédire les catégories des vidéos.

2.3.2.2 Traitement des résultats pour la classification :

Afin d'exploiter les resultats obtenus à partir de l'algorithme YOLOv7 nous les avons transformés en embeddings. Pour ce faire, nous avons exploré deux approches distinctes :

La première méthode que nous avons considérée impliquait l'utilisation de Sentence-BERT. Cependant, cette approche ne prend pas en compte les probabilités d'apparition des objets détectés.

Nous avons donc élaboré notre propre système d'embeddings en créant des vecteurs spécifiques pour chaque vidéo. Chaque vecteur a une taille correspondant à celle du vocabulaire utilisé, comprenant tous les objets détectés par YOLOv7 dans notre corpus de vidéos (le vocabulaire est de taille 80 dans notre cas). Dans ce système, chaque indice du vecteur est associé à un mot représentant un objet détecté, et la valeur à cet indice représente la probabilité maximale d'apparition de cet objet dans la vidéo correspondante.

2.3.3 ViMAE

Le modèle ViMAE est un modèle appliqué sur les vidéos qui nous a fortement intéressé. En effet il rendait possible deux choses que nous voulions, avoir un embedding des images de la video mais aussi avoir un espace commun entre un texte et une image. En plus de cela, ce modèle avait été entrainé sur de la classification, on aurait donc pu avoir de très bon résultat avec ce modèle.

Nonobstant, pour utiliser ce modèle, et le re-entrainé sur nos vidéos, nous avions besoin d'avoir nos vidéos placé dans un format particulier : dans des dossiers train-valid-test et dans ces dossiers il fallait que la vidéo soit dans un sous dossiers dans suivant son label. Une tâche gargantuesque à effectuer à la main, nous avons directement voulu l'automatiser avec des raccourcis drive.

Après des recherches conséquentes, et un tutoriel pour s'inscrire et récupérer des tokens d'identification par le cloud, puis un script python qui utilise ces tokens pour s'identifier en ligne. Le problème final fut le suivant : il fut impossible de lancer le navigateur pour s'identifier dans colab. Avec aucun forum parlant de ce problème nous avons abandonnée cette piste, et nous sommes tourner vers le model ViViT.

2.3.4 ViViT

Video Vision Transformer (ViViT) est un modèle qui extrait des informations spatiales et temporelles d'une vidéo qui serait l'entrée. Plus précissément, ViViT dans l'implémentation que nous avons utilisé, celle présentée sur hugging face, demande 32 frames de la vidéo dans un ordre chronologique.

Une particularité de ViViT qui a été souligné par les auteurs dans leur papier de présentation de ViViT est le suivant : quand les modèles transformers ont généralement besoin d'un très grand corpus de données pour être performant, ce ne fut pas le cas ici. En effet, le modèle atteint des très bon résultats avec un corpus de taille modeste voir faible pour ce genre de modèle.

Ce modèle avait pour but original de classifier des vidéos dans le pré-entraînement choisie, nous avons fait ce choix logique puisque nous avions un objectif similaire.

Nou n'avions jusque là, pas fait de pré-traitement sur les images des vidéos : en effet le modèle yolo prenait directement les fichiers vidéos mp4 en entrée. Ici ViViT demandait explicitement 32 images, dans l'exemple d'hugging face, les images étaient prises aléatoirement dans la vidéo, puis il y avait une classification dessus. Cela pouvait être liée au fait qu'ils avait dans leur corpus d'entrainement des vidéos relatiement courtes, donc cet aléatoire n'était pas génant pour eux. Pour nous ayant des vidéos avec plus de 20k frames, cela aurait été un problème.

Nous avons donc choisie de prendre uniformément 400 images dans chaque vidéo : uniformément c'est à dire à distances égales entre chaque indice de frame. Et on prenait 400 images puisque plus n'était pas suporté par la RAM pour la suite de cette stratégie. Avec ces 400 images, nous avons pour en sélectionner que 32 nous avons fait du clustering avec k-means (k=32) sur ces

400 images, avec une distance euclidienne (faire une quantification puis du clustering avec une distance de Manhattan aurait été tout à fait possible aussi). Une fois ces clusters crées, nous avons sélectionner pour chaque cluster, l'image la plus proche du centre du cluster. Pour les vidéos à moins de 32 frames, nous prenions les indices aléatoirements.

Une fois tous les inputs du modèle choisis, il ne restait plus qu'à lancer avec toutes les images. nous avions le choix de récupérer les derniers couches du modèle ou la dernières couche, pour des raisons de RAM on a choisit que la dernière couche, qui était de dimension 3137 x 768 (à la place de 32 x 3137 x 768).

J'avais donc besoin pour stocker l'ensemble des embeddings, d'une matrice numpy qui capable de stocker 2280 x 3137 x 768, cela représente donc une matrice de 42 Go (première fois où le disque de Colab a faillit planter). En faisant tourner deux fois le modèle sur deux images (l'une puis l'autre), notre RAM plantait, la seul solution trouvée fut de supprimer l'outputs du model entre deux itérations et de stocker seulement la partie qui nous intéressait de l'output. Nous avons fait ça en utilisant la librairie gc (puisque la suppression simple : del ne libérait pas l'espace RAM). En dépit de stocker que les outputs souhaités, nous pouvions au final, puisque le processus d'appel du model pour récupérer les outputs prenait beaucoup de RAM, récupérer au maximum 15 vidéos une par une en moyenne avant de faire saturer la RAM de Colab. On a donc lancer le code 175 fois en prenant 13 vidéos à chaque fois et en sauvegardant une matrice 13 x 3137 x 768. Même en utilisant un format float16 (un format de float plus bas n'est pas implémenté), nous n'avons pas réussi à entrainer des modèles avec toutes ces données, que ce soit des modèles de ML tel que SVC ou des modèles de deep où on esperait stocker la RAM dans la RAM du GPU plus que la RAM classique. En réduisant fortement les données (passer de 1900 individus dans le train à 1000) cela ne suffisait pas.

Ainsi on a malheuresement été obligé pour exploiter ces embeddings de faire des moyennes ou des maximums sur chaque axes pour chaque vidéo, ce qui a forcément réduit la capacité d'apprentissage, et donc les résultats.

2.4 Audio

2.4.1 Embedding de l'audio avec VGGish

VGGish est un modèle de réseaux de neurones développé par Google pour l'extraction de caractéristiques audio à partir de fichiers audio. Il est particulièrement conçu pour extraire des embeddings audio, le fichier audio est transformé par le preprocess suivant puis traité dans un réseau de neurones convolutifs (inspiré du modèle VGG) pour obtenir des embeddings.

Voici le pipeline du preprocessing des fichiers audio du modèle VGGish :

- Les données brutes de modulation par impulsions codées (PCM) du fichier WAV sont converties en nombres décimaux (floats) sur une échelle de [-1,0, +1,0].

- S'il y a deux canaux, les éléments sont moyennés pour produire un seul canal.

- Les données sont échantillonnées à 16 000 échantillons par seconde.

- Les données sont découpées en plusieurs fenêtres se chevauchant.

- Une fenêtre de Hamming est appliquée à chaque fenêtre.

- Le spectre de puissance est calculé à l'aide d'une transformation de Fourier rapide.

- Les fréquences au-dessus et en dessous de certains seuils sont supprimées.

- Des filtres de fréquence de type Mel sont appliqués.

- Enfin, le logarithme naturel est pris de toutes les valeurs.

2.4.1.1 Utilisation du modèle

L'un des problèmes majeurs entrainé par la taille importante des fichiers audio était un dépassement de la capacité de la RAM lors de la récupération du signal à l'aide de la bibliothèque librosa.load(). Nous avons opté pour une approche de découpage des fichiers en parties égales si leur durée dépassait un certain seuil. Ensuite, pour chaque segment, nous avons appliqué le même processus de post-traitement des embeddings afin de générer un seul vecteur 128-D représentatif.

Le processus d'extraction des embeddings a pris plusieurs heures pour être complété. Pour assurer un suivi efficace de l'exécution et pouvoir reprendre en cas de bugs imprévus, nous avons enregistré non seulement les résultats des descripteurs audio, mais aussi le nom des fichiers audio traités dans un fichier log au fur et à mesure de l'exécution. Ce fichier log s'est avéré extrêmement utile pour reprendre le processus là où il s'était arrêté, malgré les interruptions imprévues.

De plus, notre travail était réalisé sur Google Colab, avec l'utilisation de Google Drive comme espace de stockage. L'espace disponible sur le disque était limité. Nous avons pris la décision d'enregistrer tous les embeddings pour chaque fenêtre de 0.96 seconde de chaque vidéo afin de d'éviter de refaire tourner le code sur une vidéo déja entamée en cas d'interruption.

Pour chaque fenêtre temporelle de 0,96 seconde d'un fichier audio, VGGish génère un vecteur de 128 dimensions, créant ainsi une liste d'embeddings pour chaque fichier audio.

2.4.1.2 Traitement des résultats de VGGish pour la classification :

Pour obtenir un vecteur de même taille pour tous les fichiers audio, plusieurs méthodes ont été explorées :

- moyenne de chaque élément des vecteurs sur l'axe du temps,
- max-pooling : on conserve la valeur maximale de chaque dimension du vecteur,
- mécanique d'attention : on réalise une moyenne pondérée des embeddings sur l'axe temporel, offrant ainsi une approche intermédiaire entre les deux techniques précédemment mentionnées

2.4.2 Détection de son avec YAMNet

Pour prédire les sons d'environnement présents dans chaque vidéo nous avons utilisé YAMNet, un modèle d'apprentissage profond entraîné sur l'ensemble de données AudioSet de Google. Ce modèle est composé de 521 étiquettes sonores couvrant une gamme diversifiée d'éléments tels que des animaux, des instruments, des événements météorologiques, et des localisations spécifiques comme 'dans une petite pièce'.

Le temps d'execution sur les donnée de YAMNet est d'environ 20 heures. L'algorithme renvoie pour chaque video une liste d'embedding correspondant à la découpe de l'audio en plage de 0.96 seconde de manière analogue à VGGish.

Afin d'obtenir un embedding par video, on conserve le vecteur des moyennes pour chaque dimension de chaque plage de 0.96 seconde contenue dans la liste prédite par YAMNet pour chaque vidéo, créant ainsi un vecteur de longueur 521 avec des poids associés. Cette méthode capture l'essence des sons présents dans la vidéo, reflétant la présence ou l'absence des différentes étiquettes sonores.

Ces représentations ont été utilisées comme entrées dans nos classifieurs pour identifier la catégorie de chaque vidéo. Nous avons également tenté différentes combinaisons avec les résultats de VGGish en concatenant les embeddings obtenus dans les deux modèles.

2.4.3 Whisper Small

Meilleur compromis entre temps de calcul et transcription, on a testé d'autres modèles tel que le *speechbrain/asr-wav2vec2-commonvoice-en* ou le modèle de Facebook *s2t-small-librispeech-asr* qui donnaient des transcriptions trop loin de ce que nous espérions.

Le modèle whisper de openai lui marchait bien, nous avons comparer les résultats du large, du small et du tiny, et nous avons conclue que la transcriptions était bonne à partir du small. Pour éviter des temps de calcul trop long, nous somme donc parti sur le modèle Whisper Small.

2.5 Classification

A partir des embeddings obtenus en traitant les différentes modalités, plusieurs méthodes de classification ont été testées, nous présenterons dans cette section les principales méthodes employées.

2.5.1 MLP

Nous avons tenté de prédire les catégories des données en utilisant un réseau de neurones simple fait à la main, plus précisément un perceptron multicouche (MLP). Nous avons effectué plusieurs tests en variant le nombre de neurones par couche et le nombre de couches du réseau.

En analysant les résultats pour les différentes modalités, nous avons constaté que le modèle ne semblait pas apprendre car même en mettant très peu de neurones l'accuracy était toujours de 30% . Au lieu de capturer des motifs et des caractéristiques significatives, le modèle prédisait

systématiquement la classe "éducation" pour la plupart des exemples, cette classe représentant plus de 30% de nos données c'est donc pour cette raison que l'on obtient 30% de bonne réponses. En d'autres termes, le modèle ne parvenait pas à généraliser correctement et à faire des prédictions précises pour les autres classes, se contentant de choisir la classe majoritaire.

Les performances du modèle s'amélioraient légèrement lorsque nous modifiions les classes de notre problème avec les classe prédites par 0-shots. Nous avons réussi à augmenter le taux de bonnes prédictions de 2%.

2.5.2 SVC

Le Support Vector Classifier (SVC) est un algorithme de machine learning utilisé pour la classification binaire et multiclasse.

Le principe de base du SVC est de trouver un hyperplan dans un espace de caractéristiques (représentation des données) qui maximise la marge entre les différentes classes tout en minimisant l'erreur de classification. L'avantage majeur du SVC réside dans sa capacité à trouver des frontières de décision complexes et non linéaires.

Cet algorithme s'est avéré être très rapide à faire tourner et nous avons souvent obtenu des résultats légèrement supérieurs à ceux des réseaux de neurones.

2.5.3 One-Against-All SVC

Le classifieur One-Against-All (OvA) est souvent utilisé en combinaison avec des algorithmes de classification binaire tels que le Support Vector Classifier (SVC). Dans ce contexte, le SVC OvA crée plusieurs classifieurs binaires SVC, un pour chaque classe, et choisit la classe avec la prédiction la plus confiante comme la classe finale pour un exemple donné. Cela permet d'effectuer des prédictions précises dans des problèmes de classification multiclasse complexes.

L'algorithme est très rapide et les résultats obtenus par cette méthode sont souvent sensiblement meilleurs que ceux de SVC simple.

2.5.4 Gradient Boosting

- très lent

- pour des résultats bof

2.5.5 Naive Bayes

- particulièrement pour le texte car c'est là que c'est le plus intéressant

2.6 Cross-modalité

2.6.1 Dalle-mini

- utilisation de la métrique CLIP pour comparer le texte à l'image retournée

- comparaison des images récupérées avec Dalle et des 32 images les plus distantes les unes des autres de la vidéo

2.6.2 CLIP

- CLIP : modèle d'openai permettant d'evaluer la correspondance d'un un texte à une image

- résumer les textes avec bart car entrée de CLIP doit être de 77 caractères max

- associer les résumé des description ecrites aux images les plus représentatives de la vidéo

2.6.3 D'embeddings à embeddings

Dans cette approche, nous avons cherché à passer de la représentation d'une vidéo dans un espace d'embedding d'une modalité à une autre représentation dans un espace d'embedding d'une autre modalité.

En effet, nous avons entraîné des modèles pour apprendre le passage d'un embedding d'une des trois modalités, afin de passer à un embedding d'une autre modalité. Une fois la nouvelle représentation optenue (la sortie du modèle) nous regardons quel embedding dans cet espace est le plus proche de notre sortie et nous considérons ça comme notre vraie sortie du modèle (cette partie n'est pas apprise par le modèle).

Nous avons utilisé deux types de modèle pour passer de la représentation d'un embedding à un autre :

1. la méthode directe avec un Multi-layer perceptron comprenant des couches denses avec plus de neurones à chaque couche que la dimension de la sortie.
2. La méthode auto-encodeur, cette méthode a le même esprit que la méthode ci-dessus, cependant, dans cette dernière, on espérait obtenir des modèles plus robuste, et potentiellement de meilleurs résultats (sur le test, l'apprentissage est forcément plus difficile).

Malheuresement, les résultats n'étaient que très peu concluants. Nos modèles, pour minimiser la loss entre leur sortie et la vérité terrain (embeddings dans l'autre modalité), avaient tendance à produire une sorte de moyenne des embeddings voulues. Ainsi, on manquait cruellement pour certains modèles de diversité, jusqu'à après observation de l'embeddings final, pour 150 différents embeddings en entrée, on arrivait à moins de 10 embeddings en entrée.

Parlons maintenant plus concrètement des résultats, la meilleure cross-modalité que nous avons eu et de loin fut le passage de vgg-ish average et max-pool concaténés (128+128 de dimension) allant vers MPnet (dim 768) en passage direct. À noter que prendre seulement vgg-ish seul vers MPnet donnait aussi de bons résultats, ce cross-modalité par un auto-encodeur, même en ayant un espace latent de dimension 64, ne donnait pas de bons résultats. Quand nous explicitons que ce modèle était le plus encouragent, c'est en termes de diversité produite, en effet, c'est le modèle offrant en sortie le plus de diversité après observation de l'embedding le plus proche (+120 embeddings différents trouvés pour 469 embeddings en entrée).

Nous pensions que le chemin inverse, c'est-à-dire, d'une modalité avec un embedding de plus grande dimension vers une autre modalité avec un embedding plus petit serait le plus facile, mais ce ne fut pas le cas en pratique, dans notre exemple peu poussé en tout cas.

2.7 Multi-modalité

Il existe plusieurs méthodes de réunification des modalités, voici les trois techniques que nous avons mises en place. A noter que nous nous sommes concentrés sur la réunion des trois modalités ensemble par temporaire, mais des approches bi-modales auraient été très intéressantes.

2.7.1 Réunion précoce

Il s'agit de réunir les caractéristiques des trois modalités avant la classification. Comme nous avons plusieurs types d'embeddings par modalité, ils existent plusieurs combinaisons possibles. Nous souhaitons toutes les tester. Pour cette partie, une structure de donnée avec classes python a été développée pour faciliter les combinaisons d'embedddings. Pour ce faire plusieurs concaténations des vecteurs des 3 modalités sont réalisées.

2.7.2 Réunion intermédiaire

Avec le même début que la réunion précoce, nous donnons dans notre modèle de Deep learning les entrées concaténées, cependant, dans cette partie, avant de vouloir directement classifier, nous séparions chaque modalitée dans notre modèle. cette séparation avait deux buts majeurs, réduire la dimension des embeddings pour rendre l'apprentissage plus facile (nous n'avons toujours qu'un peu plus de deux milles vidéos) mais aussi d'uniformiser les embeddings entre eux.

En effet, un embeddings avec une moins grande dimension pourrait se retrouver écrasée. Nous sommes particulièrement fan de cette méthode pour la raison suivante : la réduction de dimension est faite dans le réseau effectuant la classification, ainsi cette réduction a pour but de garder les informations les plus essentiels en particulier pour la classification.Lorsqu'un aurait pu se contenter de prendre l'espace latent sortant d'un auto-encodeur, nous sommes heureux de nous être formé et d'avoir appris comment séparer et re-concaténer des éléments dans un réseau de neurone.

Nous n'avons eu le temps de tester seulement sur les labels sur et sub samplés cet approche, mais sur ce dernier nous avons obtenues un des meilleurs résultats avec 48% d'accuracy sur ce jeu de donnée équilibré, cela vient donc égaler ce que nous avons trouvé sur le jeu de donnée équilibré avec DistilBert, notre meilleur modèle de classification jusque là. On aurait pu espérer de meilleur résultats si nous avions pu utiliser les embeddings de ViViT à la place de ceux de Yolo. Ce 48% fut obtenu avec comme entrée les embeddings de Yolo (SetenceBert sur phrase crée avec les objets), de VGG-ish average et max pooling concatenés ainsi que ceux de MPnet.

2.7.3 Réunion tardive

Il s'agit de réunir les probabilités de prédictions de chaque modalité. Il faut donc pour tout type d'embeddings pour chaque modalité calculer les probabilités de prédictions pour tous les individus au travers de trois modèles de classification choisies. Cependant à cause d'un problème de structure de donnée, nous n'avons pas eu le temps d'incorporer les probabilités de prédictions de distillBERT ni de ViVit.

3 Résultats

3.1 Texte

Classes utilisées	**Méthode embeddings**	**Méthode de classification**	**Temps de calcul**	**Accuracy**
Youtube	s-bert mpnet	MLP	2h	54 %
Youtube	DistillBERT	Model spécialisé	10 min	60 %
Sur+Sub Sampling	DistillBERT	Model spécialisé	10 min	48 %

TABLE 3.1 – Résultats des différents tests de classification à partir des données textuelles

3.2 Video

Classes utilisées	**Méthode embeddings**	**Méthode de classification**	**Temps de calcul**	**Accuracy**
Youtube	YOLOv7+SBERT	SVC	61 h	0.35
zero-shot	YOLOv7+SBERT	SVC	61 h	0.38
Youtube	YOLOv7+SBERT	MLP	61 h	0.32
zero-shot	YOLOv7+SBERT	MLP	61 h	0.34
Youtube	YOLOv7 +Proba_Avg	SVC	61 h	0.28
Youtube	YOLOv7 +Proba_Avg	MLP	61 h	0.27
Youtube	YOLOv7 +Proba_Max	MLP	61 h	0.28
Youtube	YOLOv7 +Proba_Max	SVC	61 h	0.29
zero-shot	YOLOv7 +Proba_Max	SVC	61 h	0.30
Youtube	ViViT	MLP (mean)	23 h	41 %

TABLE 3.2 – Résultats des différents tests de classification à partir des données textuelles

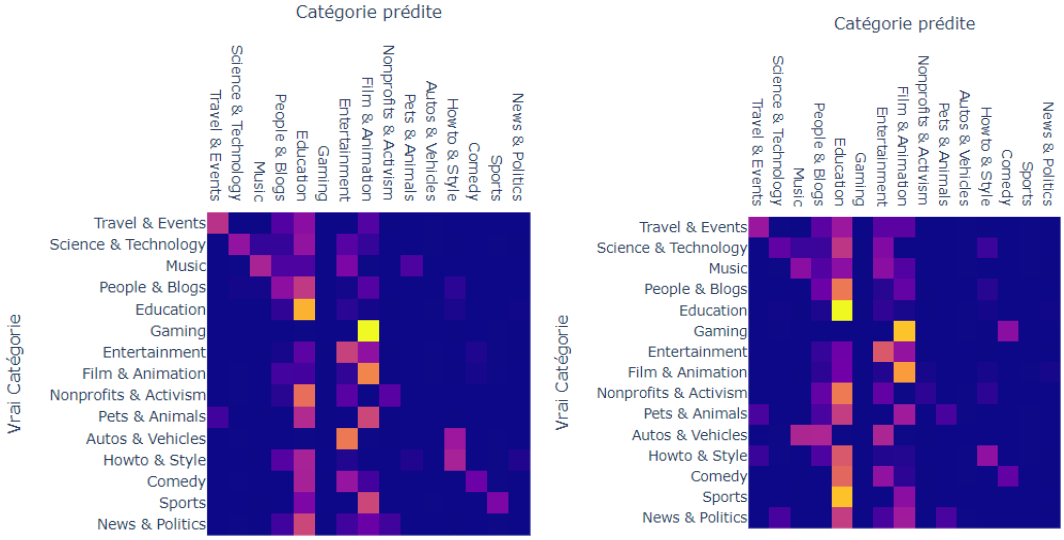
3.3 Audio

Classes utilisées	Méthode embeddings	Méthode de classification	Temps calcul	Accuracy
Labels Youtube	Average VGGish et Max pool VGGish	1vsRestSVC	<1min	0.524
Labels Youtube	VGGish et YAMNet	1vsRestSVC	<1min	0.491

TABLE 3.3 – Résultats des différents tests de classification à partir des données textuelles

Voici les meilleurs résultats (table 3.3), ils sont obtenus avec les labels Youtube normaux (pas de sur ni de sous sampling ni de redistribution). Il faut alors prendre le score d'accuracy encore plus avec des pincettes car les classes majoritaires faussent le score de bonne prédiction. Il faut alors regarder les matrices de confusions (Figure 3.2) pour une analyse plus fine des prédictions.

FIGURE 3.1 – Matrices de confusions des prédictions à partir des embeddings Audio



(a) Average VGGish et Max pool VGGish

(b) VGGish et YAMNet

On souhaite observer une diagonale propre et nette, or on observe sur les Figures 3.2 une diagonale très approximative avec des colonnes bien présentes où l'on retrouve les labels majoritaires.

3.4 Multi-Modalités Précoce

Classes utilisées	**Méthode embeddings**	**Méthode de classification**	**Temps calcul**	**Accuracy**
Labels Youtube	A :Average VGGish V :YOLO avgProba T :NPNet	1vsRestSVC	<1min	0.539
Labels Youtube	A :Max pool VGGish V :YOLO avgProba T :NPNet	1vsRestSVC	<1min	0.526

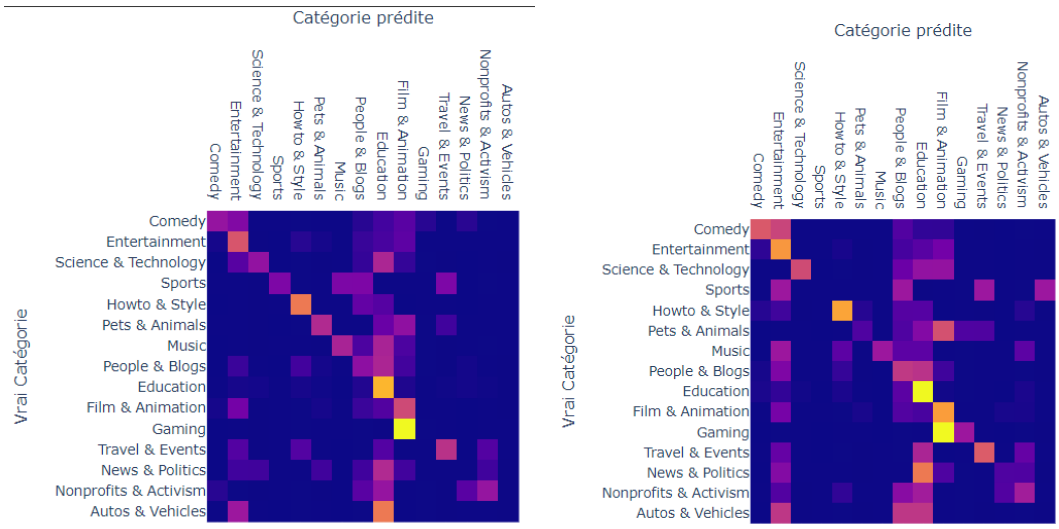
TABLE 3.4 – Résultats des différents tests de classification à partir des différentes concaténation d'embeddings

3.5 Multi-Modalités Tardive

Classes utilisées	**Méthode embeddings**	**Méthode de classification**	**Temps calcul**	**Accuracy**
Labels Youtube	A :Average VGGish IN 1vsRestSVC V :YOLO maxProba IN SVC T :NPNet IN 1vsRestSVC	1vsRestSVC	<3min	0.548
Labels Youtube	A :Max pool VGGish IN SVC V :YOLO avgProba IN SVC T :NPNet IN SVC	1vsRestSVC	<3min	0.530

TABLE 3.5 – Résultats des différents tests de classification à partir des différentes concaténation de probabilités de classification

FIGURE 3.2 – Matrices de confusions résultant d'un modèle à partir des prédictions antérieures



(a) Classification avec accuracy 0.548

(b) Classification avec accuracy 0.530

4 Gestion de projet

4.1 Diagramme de Gant

Au cours de ce projet, nous avons travaillé de manière collaborative, en tenant compte des objectifs du projet ainsi que du calendrier fixé. Nous avons tout au long du projet partagé nos pistes de réflexion. Le travail s'est réparti assez naturellement en fonction des envies de chacun. Dans la partie classification, Valentin à travaillé essentiellement sur les modalités texte et vidéo, Théotime quant à lui s'est concentré surtout sur le traitement de l'audio, Léa à travaillé également sur les modalités vidéo et texte. Si nous nous sommes partagé le travail, nous sommes restés au courant du travail des un des autres. Nous avons commencé à travailler sur la cross-modalité la dernière semaine.

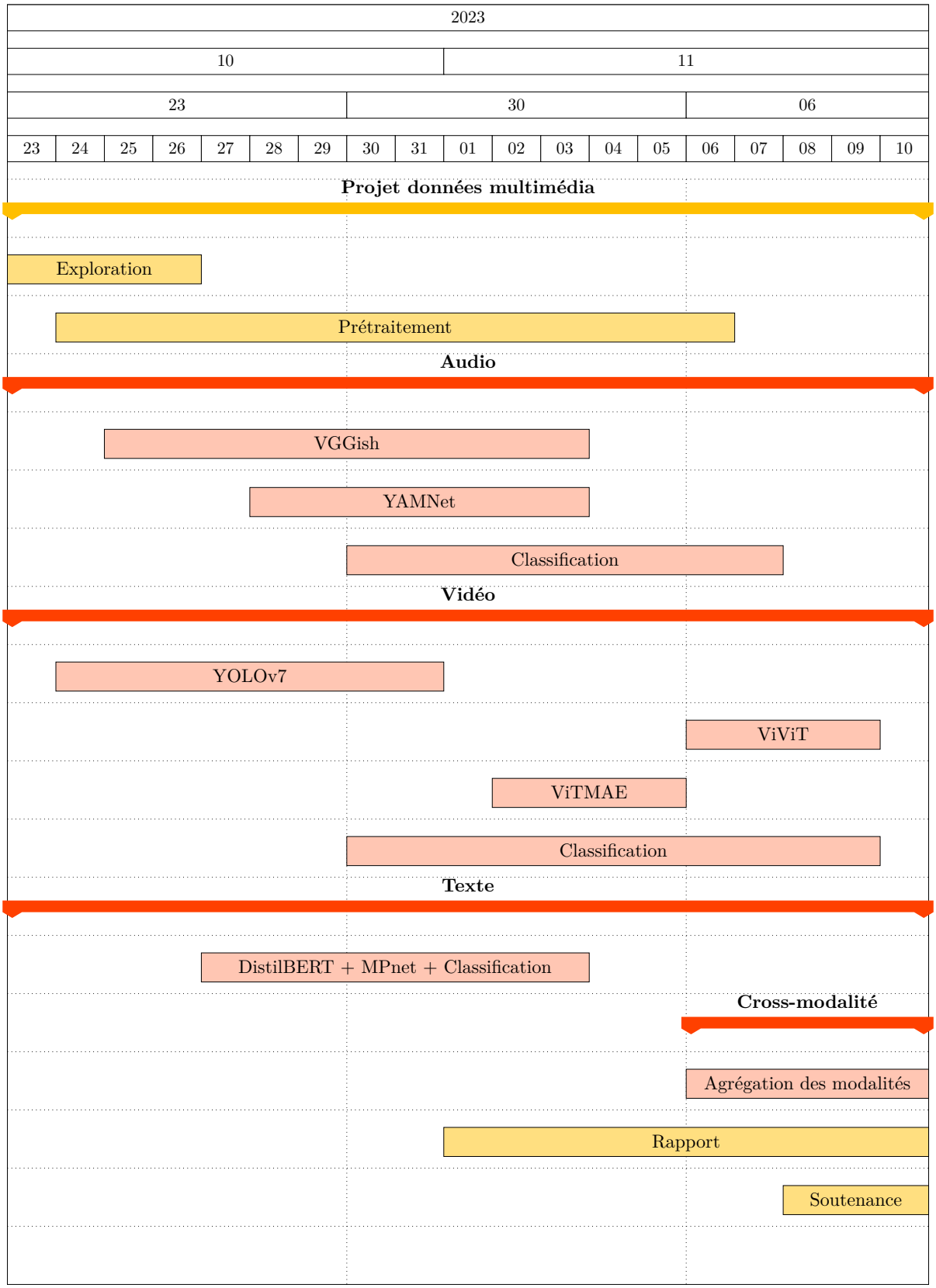


FIGURE 4.1 – Diagramme de Gantt : Gestion du projet

4.2 Difficultées rencontrés

Compte rendu et liste des problèmes rencontrés lors de ce projet :

1. Accès aux salles : Badges u3 pour le groupe du projet premièrement indisponible : badge prévu par le secretaire Clement Nicolas en milieu de première semaine n'ouvrait que les salles u2 et pas u3. Procédure de demande pour badge personnel étonnament accepté, badge à récupérer en 3R1, on y va dès qu'on a l'information mais la personne est déjà rentrée chez elle. Y retourner première heure le lendemain et une fois avoir récupérer le badge, se rendre compte que le badge ne marche pas, retourner en 3R1. Comprendre que la personne qui s'occupe de la sécurité sera présente en fin de matinée, finalement récupérer le badge en état, pour se rendre compte que le bâtiment est en travaux toute la semaine de vacance. Devoir re-chercher un badge u2 avec presque tous les secrétaires en vacance, en passant par récupérer un badge de thésard qui ne marchait pas non plus. Cela ne devrait pas être si compliqué.
2. Ressources matérielles : la RAM et la puissance de calcul manquaient cruellement, la méthode de paraléllisation et un peu moins d'une vingtaine de compte google crées ont permis de rendre une production final acceptable, cela dit quand nous pourrions prendre en compétence et vraiment comprendre les architectures des modèles pré-entrainés, nous passions du temps à lancer des notebooks en parallèle sur différents comptes. Plus de 4 journées entières de travailles en cumulée ont été passés à simplement lancer du code et faire des capchats, cela est problématique sur un projet de 3 semaines. De même nous avons rencontré des problèmes au niveau de la RAM, simplement avoir en tête par exemple tous les problèmes rencontrés lors de l'implementation de ViViT pour la RAM, pour au final ne même pas pouvoir utiliser pleinement les résultats est particulièrement frustrant.

5 Documentation

Ce projet fut basé sur le corpus QuerYD, vous pouvez trouver la page officielle ici : https ://www.robots.ox.ac.uk/ vgg/data/queryd/

Liens Hugging Face pour acceder aux modèles

- https ://huggingface.co/openai/whisper-small
- https ://huggingface.co/docs/transformers/v4.31.0/model_doc/vivit
- https ://huggingface.co/speechbrain/asr-wav2vec2-commonvoice-en
- https ://huggingface.co/facebook/s2t-small-librispeech-asr
- https ://huggingface.co/openai/whisper-large, https ://huggingface.co/openai/whisper-small, https ://huggingface.co/openai/whisper-tiny
- https ://huggingface.co/docs/transformers/model_doc/vit_mae
- https ://developers.google.com/drive/api/quickstart/python
- https ://huggingface.co/tasks/zero-shot-classification
- https ://huggingface.co/docs/transformers/model_doc/distilbert
- https ://huggingface.co/docs/transformers/model_doc/mpnet
- https ://huggingface.co/sentence-transformers/paraphrase-distilroberta-base-v1

Liens Github pour acceder aux modèles

- https ://github.com/tensorflow/models/blob/master/research/audioset/vggish/README.md
- https ://github.com/tensorflow/models/tree/master/research/audioset/yamnet