# Classification Multi-Classe à sorties multiples à l'aide de Transformers

## 1. Introduction

La tâche de « classification multi classe à sorties multiples » (Multi-output multi-class classification en anglais) est un problème d'apprentissage automatique qui vise à prédire simultanément plusieurs étiquettes de classe pour chaque instance d'un ensemble de données (Ma & Álvarez, 2023). Contrairement à la classification à étiquette unique, la classification de texte à étiquettes multiples a un ensemble d'étiquettes plus large et chaque texte a un ensemble d'étiquettes différent, ce qui apporte un certain degré de difficulté à la tâche de classification(Yan et al., 2022). Un exemple typique de classification de texte à étiquettes multiples est qu'un article d'actualité sur un site web d'actualités peut appartenir au domaine économique et au domaine sportif, voire à plusieurs domaines. Cette classification permet de mieux refléter la signification réelle du texte (Yan et al., 2022).

La classification multi-output est utilisée dans de nombreux domaines et applications. Des applications comme : l'étiquetage d'images ; où on peut prédire plusieurs étiquettes pour une seule image, comme la reconnaissance de différents objets présents dans l'image. L'étiquetage de texte ; on peut prédire plusieurs catégories ou étiquettes pour un seul document ou paragraphe. La reconnaissance vocale ; on peut prédire plusieurs mots ou phonèmes à partir d'un enregistrement audio. Cela est utile dans des applications telles que la transcription automatique de la parole ou la traduction vocale. Les prévisions météorologiques ; prédire plusieurs variables météorologiques simultanément, telles que la température, l'humidité, la pression atmosphérique, etc.

Parmi les projets auxquels j'ai travaillé tout au long de ma formation, le projet 5 'Catégorisez automatiquement des questions' a suscité un intérêt particulier de ma part en raison de son objectif de classification multi-output. Dans le projet 5, le modèle le plus performant que j'ai obtenu consistait à utiliser le Transformer Sentence-BERT pour l'extraction des caractéristiques (features), suivi d'un apprentissage avec la régression logistique. Dans ce projet 7, j'ai été motivé par l'opportunité d'explorer des méthodes prometteuses en utilisant le transfert d'apprentissage. J'ai souhaité tirer parti des modèles pré-entraînés déjà performants sur des tâches similaires pour améliorer les performances de la classification multi-output. J'ai donc décidé d'expérimenter le fine-tuning, qui consiste à adapter ces modèles pré-entraînés à ma tâche spécifique en ajustant leurs poids lors de l'apprentissage. Cette approche m'a semblé très prometteuse, car elle me permettait de bénéficier des connaissances et de la capacité de généralisation des modèles pré-entraînés, tout en les adaptant spécifiquement à ma tâche de classification multi-output.

## 2. Etat de l'art

Dans cette section, je vais approfondir ma compréhension de la tache de la classification multi-output multi-class en examinant les méthodes et les approches les plus récentes utilisées pour résoudre cette problématique.

- **Les approches traditionnelles :**

Les premières approches de classification multi-label multi-classe reposaient souvent sur la transformation du problème en une série de problèmes de classification binaire ou multi-classe. Les méthodes les plus couramment utilisées étaient la méthode "One-vs-All" (OVA) et la méthode "One-vs-One" (OVO). Dans la méthode OVA, chaque classe est traitée indépendamment comme une classe positive et toutes les autres classes sont considérées comme négatives. La méthode OVO, quant à elle, construit un classifieur binaire pour chaque paire de classes possibles (Bogatinovski et al., 2022).

- **Approches basées sur l'apprentissage ensembliste :**

L'apprentissage ensembliste a été largement utilisé dans la classification multi-label multi-classe pour améliorer les performances des modèles. Les techniques populaires incluent l'approche du Bagging, qui combine plusieurs classifieurs indépendants pour réduire la variance, et l'approche du Boosting, qui construit un classifieur en série en accordant plus d'importance aux exemples mal classés.

- **Approches basées sur les réseaux de neurones :**

Les réseaux de neurones, en particulier les réseaux de neurones profonds, ont été très efficaces pour la classification multi-label multi-classe. Des architectures comme les réseaux de neurones convolutifs (CNN) et les réseaux de neurones récurrents (RNN) ont été adaptées pour résoudre ce problème. Des techniques telles que l'encodage binaire et l'encodage à étiquettes hiérarchiques ont également été utilisées pour améliorer les performances des modèles de réseaux de neurones.

- **Approches basées sur l'apprentissage profond :**

Récemment, les réseaux d'apprentissage profond ont connu un développement rapide dans la classification de texte multi-output et ont obtenu des résultats remarquables. Des architectures telles que les réseaux de neurones convolutifs profonds (Deep CNN) et les réseaux de neurones récurrents profonds (Deep RNN) ont été proposées. En outre, des modèles tels que les réseaux de neurones à attention, qui permettent au modèle de se concentrer sur des parties spécifiques des données d'entrée, ont été appliqués avec succès.

Les transformeurs ont également été appliqués avec succès à la classification multi-label multi-classe. Des architectures telles que le modèle BERT (Bidirectional Encoder Representations from Transformers) ont montré d'excellentes performances en apprenant des représentations contextuelles des mots et en capturant les relations complexes entre les différentes étiquettes. L'utilisation des transformeurs permet d'exploiter l'attention pour donner plus de poids à certaines parties de l'entrée lors de la prédiction des étiquettes. Les modèles basés sur les transformeurs peuvent prendre en compte les interactions complexes entre les différentes étiquettes et capturer les dépendances entre celles-ci. En outre, les transformeurs permettent d'intégrer des informations contextuelles plus larges dans le processus de classification. Ils peuvent prendre en compte le contexte global de l'instance et utiliser des informations provenant de l'ensemble du corpus d'entraînement.

Les transformeurs ont également été utilisés en combinaison avec d'autres techniques, telles que l'encodage hiérarchique des étiquettes ou l'apprentissage par transfert, pour améliorer davantage les performances de classification multi-label multi-classe. Dans ce travail, j'ai utilisé les Transformers avec l'apprentissage par transfert en utilisant la méthode de fine-tuning, pour créer un modèle capable d'affecter un ou plusieurs tags à des questions posé sur le site StackOverFlow.

J'ai essayé d'appliquer le fine-tuning avec le modèle oBERT tel que décrit dans l'article de (Kurtic et al., 2022). oBERT est une méthode de taille de poids basée sur des informations d'ordre secondaire approximatives. Il étend le travail existant sur la taille non structurée d'ordre secondaire en permettant la taille des blocs de poids et est applicable à l'échelle du modèle BERT. De plus, les auteurs du papier étudient l'impact de cette méthode de taille lors de la combinaison d'approches de compression pour obtenir des modèles hautement compressés mais précis. Cependant, en utilisant le checkpoint disponible sur Hugging Face "neuralmagic/oBERT-12-upstream-pruned-unstructured-97" que j'ai utilisé, le nombre de paramètres entraînables est de 109 millions et l'apprentissage a pris plus de 11 heures. Par conséquent, j'ai décidé de chercher un autre modèle malgré l'amélioration des performances que j'ai obtenue par rapport à mon modèle de référence (Baseline).

| | Optimal BERT Surgeon |
|---|---|
| eval_loss | 0.054705 |
| eval_f1 | 0.698087 |
| eval_accuracy | 0.386213 |
| eval_runtime | 156.071400 |
| eval_samples_per_second | 28.070000 |
| eval_steps_per_second | 1.756000 |
| epoch | 12.000000 |

Ensuite, j'ai essayé d'appliquer un fine tuning avec le modèle SetFit décrit dans (Tunstall et al., 2022). SetFit est un cadre sans prompts pour le réglage fin des modèles de Sentence Transformers dans le contexte de l'apprentissage avec peu d'exemples. Le cadre SetFit fonctionne en deux étapes principales. Tout d'abord, un modèle Sentence Transformer pré-entraîné est réglé fin sur un petit nombre de paires de textes dans une approche contrastive Siamese. Cette étape vise à apprendre des embeddings de texte riches et discriminatifs. Ensuite, ces embeddings sont utilisés pour entraîner une tête de classification qui permet d'effectuer des tâches de classification sur des données avec peu d'exemples. Les résultats d'évaluation sur le dataset de test sont les suivant :

```
metrics = trainer.evaluate()
metrics
```

```
Applying column mapping to evaluation dataset
***** Running evaluation *****
{'f1': 0.6608429115372394, 'accuracy': 0.38538812785388127}
```

Comparé à mon modèle de référence, SetFit améliore l'accuracy de 5,6 % mais présente une diminution de 1 % du score F1. De plus, SetFit est particulièrement adapté lorsque les données sont limitées en quantité. Par la suite, j'ai découvert le modèle CANINE que je vais décrire dans la section 4 de la modélisation.

## 3. Jeu de données

Dans ce travail, j'ai effectué mes tests en utilisant le jeu de données nettoyé que j'ai collecté pour mon projet 5 intitulé 'Catégorisez automatiquement des questions'. Ces données ont été obtenues à partir de la plateforme[1] 'StackExchange Data Explorer' proposée par 'StackOverFlow'. J'ai utilisé la requête SQL suivante pour extraire les données :

```sql
SELECT TOP 50000 Title, Body, Tags, Id, Score, ViewCount, AnswerCount
FROM Posts
WHERE PostTypeId = 1 AND ViewCount > 10
AND Score > 5 AND AnswerCount > 0 AND LEN(Tags) - LEN(REPLACE(Tags, '<','')) >= 5
```

Les données extraites comprennent les colonnes suivantes :

- Title: titre de la question
- Body : le contenu principal de la question
- Tags : étiquettes associées à la question
- Id : Identifiant unique de la question
- Score : Le nombre de votes positifs sur une question/réponse moins le nombre de votes négatifs.
- ViewCount : Le nombre de vue de la question
- AnswerCount : Le nombre des réponses à la question

Dans le projet 5, après avoir collecté les 50 000 questions brutes à partir de la requête mentionnée précédemment, j'ai effectué un processus de nettoyage des données pour garantir leur qualité et leur

[1] https://data.stackexchange.com/stackoverflow/query/new

cohérence. Ce processus de nettoyage a permis d'éliminer les questions qui ne répondaient pas à certains critères ou qui étaient mal formatées. Après avoir appliqué les étapes de nettoyage appropriées, il me reste un total de 43 801 questions qui sont prêtes à être utilisées dans mon projet.

- ❏ Filtrer le corps suivant la langue **anglaise**.
- ❏ Suppression des balises **HTML** avec **BeautifulSoup**.
- ❏ Nettoyage des chevrons des tags.
- ❏ Récupérer les 50 tags les plus populaires dans le jeu de données.
- ❏ Garder uniquement les questions qui ont au moins 1 tag parmis les 50 tags les plus populaires.

```
New size of dataset : 43801 questions.
```

- ❏ Passage au **miniscule**.
- ❏ Supprimer la ponctuation, l'extra-espace et les caractères spéciaux , sauf + et #.
- ❏ **Tokenization**
- ❏ Supprimer les mots vides (**stop words from NLTK**).

Figure 1. Le processus de nettoyage du jeu de données.

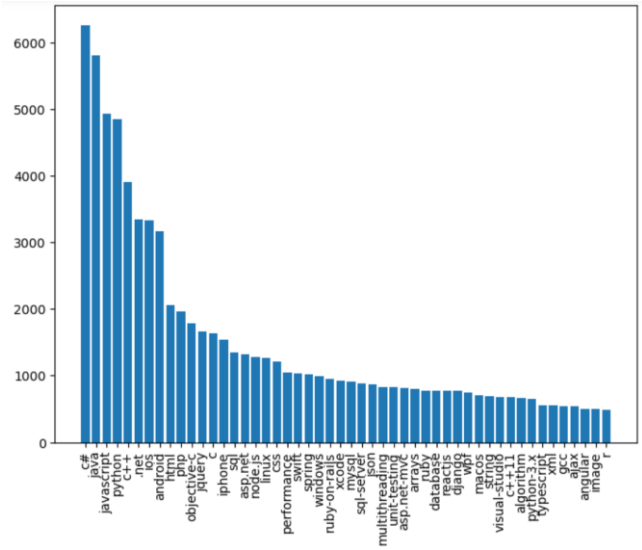


Figure 2. La distribution des 50 tags les plus populaire dans le jeu de données.

Dans le cadre de mon projet actuel, mon attention s'est portée sur trois colonnes essentielles : " Title_cleaned", "Body_cleaned" et "Tags_list". Ces colonnes ont été soumises à diverses manipulations afin de préparer les données pour les analyses ultérieures et les tâches de classification multi-output multi-classe. Tout d'abord, j'ai réalisé une concaténation des phrases présentes dans les colonnes "Title_cleaned" et "Body_cleaned". Cette étape permet de regrouper le titre et le corps des questions en une seule séquence de texte. Par la suite, j'ai procédé à un encodage one-hot de la colonne "Tags_list", qui contient les tags associés à chaque question. Cette opération a abouti à la création de 50 nouvelles colonnes, chacune représentant une étiquette spécifique. Chaque colonne est un indicateur binaire qui indique la présence ou l'absence de l'étiquette correspondante dans la question. Ces manipulations des colonnes du jeu de données me permettent d'avoir des données préparées et structurées pour la suite de mon projet, facilitant ainsi les analyses et les tâches du Transfer Learning par la suite.

- **Répartition des données**

Pour gérer mes données de manière efficace et pratique, j'ai utilisé la puissante bibliothèque Datasets de HuggingFace. Datasets est une bibliothèque open-source conçue pour faciliter le chargement, la préparation et la gestion de jeux de données dans le domaine du traitement du langage naturel.

Grâce à Datasets, j'ai pu facilement charger mon jeu de données et le préparer pour l'entraînement de mon modèle. J'ai utilisé les fonctionnalités fournies par Datasets pour diviser mes données en ensembles d'entraînement, de validation et de test. J'ai opté pour une répartition de 80% des données pour l'ensemble d'entraînement, 10% pour l'ensemble de validation et 10% pour l'ensemble de test. Cette répartition équilibrée m'a permis d'entraîner mon modèle sur une grande partie des données tout en disposant de données de validation et de test distinctes pour évaluer ses performances.

Datasets fournit également une interface intuitive pour explorer, filtrer et transformer les jeux de données. Elle offre un large éventail de fonctionnalités, telles que le chargement à partir de sources diverses (dans mon travail, chargement à partir de pandas), la normalisation des données, le filtrage basé sur des critères spécifiques, etc. De plus, la bibliothèque Datasets est accompagnée d'une documentation détaillée et d'une communauté active, ce qui facilite son utilisation et permet de résoudre rapidement d'éventuels problèmes.

# 4. Modélisation

- **Présentation du modèle choisis (CANINE-s)**

Les modèles neuronaux d'End-to-End ont largement remplacé le pipeline traditionnel du TALN en raison de leur capacité à générer automatiquement leurs propres représentations sophistiquées. J'ai utilisé le modèle CANINE-s[2] de Google qui a était introduit dans le papier de « CANINE : Pre-training an Efficient Tokenization-Free Encoder for Language Representation » (Clark et al., 2022) . J'ai effectué un fine-tuning en utilisant mes propres données annotées pour affiner le modèle et l'adapter spécifiquement à ma problématique.

CANINE est un modèle Transformers pré-entraîné sur un large corpus de données multilingues de manière auto-supervisée, similaire à BERT. Cela signifie qu'il a été pré-entraîné uniquement sur les textes bruts, sans intervention humaine pour les annoter de quelque manière que ce soit (ce qui lui permet d'utiliser de nombreuses données disponibles publiquement), en utilisant un processus automatique pour générer des entrées et des étiquettes à partir de ces textes.

CANINE est un encodeur neuronal qui fonctionne directement sur des séquences de caractères sans tokenisation explicite ou vocabulaire. Le modèle est conçu pour surmonter les limites des méthodes de tokenisation traditionnelles, qui ne sont pas également adaptées à toutes les langues et peuvent limiter la capacité d'un modèle à s'adapter.

Voici les principaux composants du modèle CANINE :

- **Incorporations de Hash de Caractères :** CANINE utilise des plusieurs fonctions de hachage pour chaque caractère Unicode afin de générer des embeddings de caractères. Ces embeddings sont concaténés pour représenter chaque caractère dans la séquence. Cela permet à CANINE de gérer l'ensemble de l'espace des caractères Unicode avec un nombre réduit de paramètres.
- **Stratégie de Downsampling :** Pour rendre CANINE plus efficace, les auteurs utilisent une stratégie de downsampling en plusieurs étapes. Ils utilisent d'abord un transformer d'attention locale pour encoder les caractères de la séquence. Ensuite, ils appliquent une convolution stridée pour réduire le nombre de positions de séquence, similaire à un modèle de tokenisation.
- **Empilement Profond de Transformateurs :** Après le downsampling, CANINE utilise un empilement profond de transformateurs pour capturer les informations contextuelles et les relations entre les caractères. Ce composant est similaire au cœur des modèles BERT et dérivés.

[2] https://huggingface.co/google/canine-s

- **Stratégie de Upsampling :** Pour les tâches de prédiction de séquence, CANINE a besoin de produire une représentation de sortie avec la même longueur que la séquence d'entrée. Les auteurs proposent une stratégie de upsampling pour reconstruire une représentation de sortie caractère par caractère en concaténant la sortie du transformateur de caractères avec la représentation downsampling.
- **Pré-entraînement :** La procédure de pré-entraînement de CANINE conserve la tâche de Modèle de Langage Masqué (MLM) et propose deux stratégies distinctes pour calculer la perte MLM - prédiction de caractères autorégressive (CANINE-c) vs. Prédiction de sous-mots (CANINE-s).

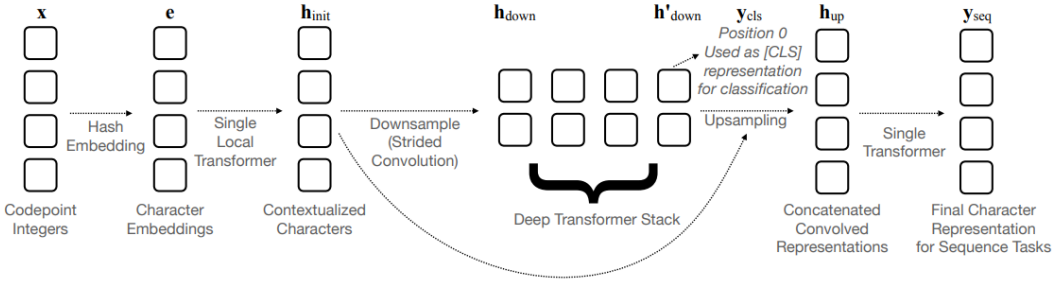


**Figure 1.** L'architecture de CANINE.

CANINE a été pré-entraîné avec deux objectifs :

- **Modélisation du langage masqué (MLM) :** une partie des entrées est aléatoirement masquée, que le modèle doit prédire. Ce modèle (CANINE-s) est entraîné avec une perte de sous-mots (subword), ce qui signifie que le modèle doit prédire les identités des sous-tokens, tout en prenant les caractères en entrée. En lisant les caractères et en prédisant les sous-tokens, la contrainte de frontière de jeton difficile présente dans d'autres modèles tels que BERT devient un biais inductif souple dans CANINE.

- **Prédiction de la phrase suivante (NSP) :** le modèle concatène deux phrases en tant qu'entrées lors de la pré-entraînement. Parfois, ces phrases correspondent à des phrases qui étaient voisines dans le texte original, parfois non. Le modèle doit ensuite prédire si les deux phrases se suivent ou non.

- **Fine-tuning CANINE-s**

J'ai utilisé la bibliothèque HuggingFace pour effectuer le fine-tuning de mon modèle. HuggingFace est une bibliothèque de premier plan dans le domaine du traitement du langage naturel, offrant une vaste gamme de modèles pré-entraînés et d'outils pour la manipulation et l'entraînement de ces modèles. Plus précisément, j'ai utilisé la bibliothèque Transformers de HuggingFace, qui fournit une interface conviviale pour l'entraînement des modèles de langage basés sur des réseaux de neurones, tels que BERT, GPT, RoBERTa, CANINE et bien d'autres. J'ai exploité les fonctionnalités avancées de la bibliothèque, notamment le Trainer, qui m'a permis de gérer facilement les étapes d'entraînement, de validation et d'évaluation de mon modèle. L'utilisation de HuggingFace dans mon projet a été cruciale pour sa réussite. La bibliothèque a simplifié le processus de fine-tuning en fournissant une API intuitive et des fonctionnalités puissantes.

Comme les modèles tels que BERT et CANINE n'attendent pas de texte en entrée directe, mais plutôt des inputs_ids, etc., je tokenise le texte en utilisant le tokenizer. Dans ce travail, j'ai réutilisé le code d'un tutoriel présenter dans le lien[3][4]. L'auteur du tutoriel a utilisé le dataset « sem_eval_2018_task_1 ». "SemEval-2018 Task 1 : Affect in Tweets" est un ensemble de données utilisé dans la compétition SemEval-2018. Ce dataset est spécifiquement conçu pour la tâche de classification des émotions dans les tweets.

Pour commencer, J'ai utilisé l'API AutoTokenizer, qui chargera automatiquement le tokenizer approprié pour CANINE-s. CANINE tokenize les phrases en les divisant en caractères individuels.

Je dois également fournir des étiquettes (labels) au modèle. Pour une classification multi-label, il s'agit d'une matrice de forme (taille_du_batch, nombre_d'étiquettes). Il est également important de noter que cela doit être un tensor de nombres flottants plutôt que d'entiers.

Ensuite, j'ai défini un modèle qui inclut une base pré-entraînée (c'est-à-dire les poids de Canine-s) qui est chargée, avec une tête de classification initialisée aléatoirement (une couche linéaire) au-dessus. On devrait affiner cette tête, ainsi que la base pré-entraînée, sur mon jeu de données. J'ai spécifié le type de problème comme étant "multi_label_classification" dans la fonction de chargement du modèle, ce qui garantit l'utilisation de la fonction de perte appropriée (BCEWithLogitsLoss). J'ai également veillé à ce que la couche de sortie dispose de 50 neurones, et j'ai défini les mappings id2label et label2id.

```
model = AutoModelForSequenceClassification.from_pretrained("google/canine-s",
                                                           problem_type="multi_label_classification",
                                                           num_labels=len(labels),
                                                           id2label=id2label,
                                                           label2id=label2id)
```

Pour apprendre le modèle avec HuggingFace's Trainer API, j'ai besoin de définir deux choses :

- **TrainingArguments :** Les TrainingArguments spécifient les hyperparamètres d'entraînement. Toutes les options peuvent être trouvées dans la documentation[5]. Dans ce projet, j'ai spécifié que je voulais évaluer après chaque époque d'entraînement, enregistrer le modèle à chaque époque, définir le taux d'apprentissage, la taille du batch à utiliser pour l'entraînement/validation, le nombre d'époques d'entraînement, etc.
- **Trainer :** L'objet trainer fournit une interface simplifiée pour gérer les différentes étapes de l'apprentissage, telles que l'optimisation des hyperparamètres, le chargement des données, la boucle d'entraînement, la validation croisée, l'évaluation des performances, etc.

## 5. Evaluation

Le modèle Baseline que j'ai utilisé dans ce projet est le même modèle que j'ai obtenu dans le projet 5. Dans ce modèle, j'ai utilisé la régression logistique pour l'apprentissage ainsi que le transformer sentence-BERT pour l'extraction des caractéristiques. Je compare principalement les métriques de précision (Accuracy) et de score F1 (F1-score) pour évaluer et comparer le modèle Baseline avec le modèle CANINE fine-tuned.

[3] https://github.com/NielsRogge/Transformers-Tutorials/blob/master/BERT/Fine_tuning_BERT_(and_friends)_for_multi_label_text_classification.ipynb
[4] https://colab.research.google.com/github/NielsRogge/Transformers-Tutorials/blob/master/CANINE/Fine_tune_CANINE_on_IMDb_(movie_review_binary_classification).ipynb
[5] https://huggingface.co/docs/transformers/main_classes/trainer#trainingarguments

Pour le modèle Baseline, les résultats appliqués sur le dataset de test est :

| | logitR_BERT |
|---|---|
| **Accuracy** | 0.327824 |
| **F1** | 0.670116 |
| **Jaccard** | 0.496880 |
| **Recall** | 0.587398 |
| **Precision** | 0.733199 |

Accuracy = 0.328

F1-score = 0.670

Pour le modèle CANINE fine-tuned, les résultats d'évaluation avec le dataset de test sont :

Accuracy = 0.386

F1-score = 0.675

```
trainer.evaluate(eval_dataset=encoded_dataset['test'])
```

```
***** Running Evaluation *****
  Num examples = 4381
  Batch size = 16
```

[274/274 02:07]

```
{'eval_loss': 0.05945124477148056,
 'eval_f1': 0.6747147534241212,
 'eval_accuracy': 0.3855284181693677,
 'eval_runtime': 63.8668,
 'eval_samples_per_second': 68.596,
 'eval_steps_per_second': 4.29}
```

```
***** Running training *****
  Num examples = 35,040
  Num Epochs = 10
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 21,900
  Number of trainable parameters = 132,121,394
```

[21900/21900 3:39:34, Epoch 10/10]

| Epoch | Training Loss | Validation Loss | F1 | Accuracy |
|---|---|---|---|---|
| 1 | 0.109200 | 0.096995 | 0.316997 | 0.196804 |
| 2 | 0.078600 | 0.075512 | 0.464224 | 0.271918 |
| 3 | 0.066500 | 0.067618 | 0.567790 | 0.310731 |
| 4 | 0.059300 | 0.063008 | 0.613154 | 0.340639 |
| 5 | 0.052400 | 0.061332 | 0.639522 | 0.360959 |
| 6 | 0.047400 | 0.059763 | 0.657507 | 0.368950 |
| 7 | 0.043300 | 0.059195 | 0.671227 | 0.383790 |
| 8 | 0.039900 | 0.059406 | 0.676050 | 0.383790 |
| 9 | 0.037200 | 0.059382 | 0.675956 | 0.384018 |
| 10 | 0.035400 | 0.059195 | 0.677206 | 0.387443 |

D'après les résultats d'évaluation, le modèle CANINE surpasse le modèle Baseline pour les deux métriques évaluées, à savoir l'accuracy et le F1-score. L'accuracy est supérieure de 5,7% dans le modèle CANINE, tandis que le F1-score affiche une amélioration de 0,5%.

De plus, étant donné que le modèle du projet 5 a été déployé sur le web, le temps d'inférence est un critère essentiel pour comparer les modèles. J'ai réalisé une comparaison du temps d'inférence entre le modèle Baseline et le modèle CANINE fine-tuned en utilisant trois phrases différentes. Les résultats (Tableau 1) montrent que CANINE nécessite seulement 0,1 seconde pour l'inférence des trois phrases, tandis que le modèle Baseline prend respectivement 0,17 seconde, 0,11 seconde et 0,8 seconde pour les mêmes phrases.

**Tableau 1.** Temps d'inférence pour les deux modèles (Baseline vs CANINE fine-tuned)

| | Modèle Baseline | Modèle CANINE fine-tuned |
|---|---|---|
| **Phrase 1** | 0.0 hours, 0.0 minutes,0.17 seconds | 0.0 hours, 0.0 minutes,0.01 seconds |
| **Phrase 2** | 0.0 hours, 0.0 minutes,0.11 seconds | 0.0 hours, 0.0 minutes,0.01 seconds |
| **Phrase 3** | 0.0 hours, 0.0 minutes,0.08 seconds | 0.0 hours, 0.0 minutes,0.01 seconds |

Ces résultats démontrent clairement que le modèle CANINE offre une meilleure performance en termes de précision et de temps d'inférence par rapport au modèle Baseline. L'amélioration significative de l'accuracy et du F1-score, combinée à un temps d'inférence plus rapide, fait de CANINE un choix supérieur pour la tâche de classification multi-label et pour le déploiement sur le web.

Cependant, malgré les avantages mentionnés ci-dessus, il convient de noter quelques limitations de CANINE. Tout d'abord, en raison de sa complexité et de son architecture spécifique, CANINE peut nécessiter des ressources informatiques plus importantes, notamment en termes de mémoire et de puissance de calcul. De plus, bien que CANINE soit conçu pour fonctionner avec différentes langues, ses performances peuvent varier selon la langue spécifique et la disponibilité des ressources linguistiques. En outre, l'interprétation des décisions prises par CANINE peut être plus difficile en raison de sa complexité. Malgré ces limitations, les performances améliorées de CANINE en termes de précision et de temps d'inférence en font un choix supérieur pour la tâche de classification multi-label et pour le déploiement sur le web.

## 6. Conclusion

J'ai présenté dans ce rapport la problématique de la classification multi-classe à sorties multiples, ainsi que les différentes approches proposées dans la littérature pour résoudre cette problématique. J'ai notamment présenté CANINE, un modèle pré-entraîné basé sur la free-tokenisation, et j'ai expliqué comment j'ai appliqué un fine-tuning sur ce modèle afin de l'adapter à ma tâche de classification multi-label.

En analysant les résultats d'inférence et en comparant les métriques d'évaluation, il est clair que l'utilisation de modèles entraînés avec des techniques de Transfer Learning est préférable, en particulier pour les modèles déployés sur le web. Ces modèles pré-entraînés permettent d'exploiter les connaissances acquises lors de l'entraînement sur de vastes corpus de données et offrent de bonnes performances, même sur des tâches spécifiques avec des ensembles de données plus petits.

En conclusion, l'utilisation du Transfer Learning avec des modèles pré-entraînés comme CANINE s'avère être une approche efficace pour résoudre la problématique de la classification multi-label. Cette méthode permet de tirer parti des avantages du pré-entraînement tout en adaptant le modèle à la tâche spécifique. Les résultats obtenus renforcent l'idée que les techniques de Transfer Learning sont une option recommandée pour obtenir de bonnes performances dans des scénarios réels de déploiement sur le web.

Pour améliorer davantage les performances de notre modèle de classification multi-label, plusieurs axes d'amélioration peuvent être envisagés. Premièrement, il serait intéressant d'explorer différentes architectures de modèles pré-entraînés tels que BERT, RoBERTa ou d'autres variantes de transformers pour évaluer si elles peuvent offrir des performances supérieures. Deuxièmement, l'augmentation des données peut aider à accroître la diversité des exemples d'entraînement et à améliorer la généralisation du modèle. Enfin, une optimisation plus poussée des hyperparamètres, telle que l'utilisation de techniques d'optimisation automatique, peut permettre de trouver les meilleures combinaisons de paramètres pour maximiser les performances du modèle.

**Reference**

Bogatinovski, J., Todorovski, L., Džeroski, S., & Kocev, D. (2022). Comprehensive comparative study of multi-label classification methods. *Expert Systems with Applications*, *203*, 117215. https://doi.org/https://doi.org/10.1016/j.eswa.2022.117215

Clark, J. H., Garrette, D., Turc, I., & Wieting, J. (2022). *CANINE: Pre-training an Efficient Tokenization-Free Encoder for Language Representation*. https://doi.org/10.1162/tacl_a_00448

Kurtic, E., Campos, D., Nguyen, T., Frantar, E., Kurtz, M., Fineran, B., Goin, M., & Alistarh, D. (2022). *The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models*. http://arxiv.org/abs/2203.07259

Ma, C., & Álvarez, M. (2023). Large scale multi-output multi-class classification using Gaussian processes. *Machine Learning*, *112*, 1–30. https://doi.org/10.1007/s10994-022-06289-3

Tunstall, L., Reimers, N., Jo, U. E. S., Bates, L., Korat, D., Wasserblat, M., & Pereg, O. (2022). *Efficient Few-Shot Learning Without Prompts*. http://arxiv.org/abs/2209.11055

Yan, Y., Liu, F., Zhuang, X., & Ju, J. (2022). An R-Transformer_BiLSTM Model Based on Attention for Multi-Label Text Classification. *Neural Process. Lett.*, *55*(2), 1293–1316. https://doi.org/10.1007/s11063-022-10938-y