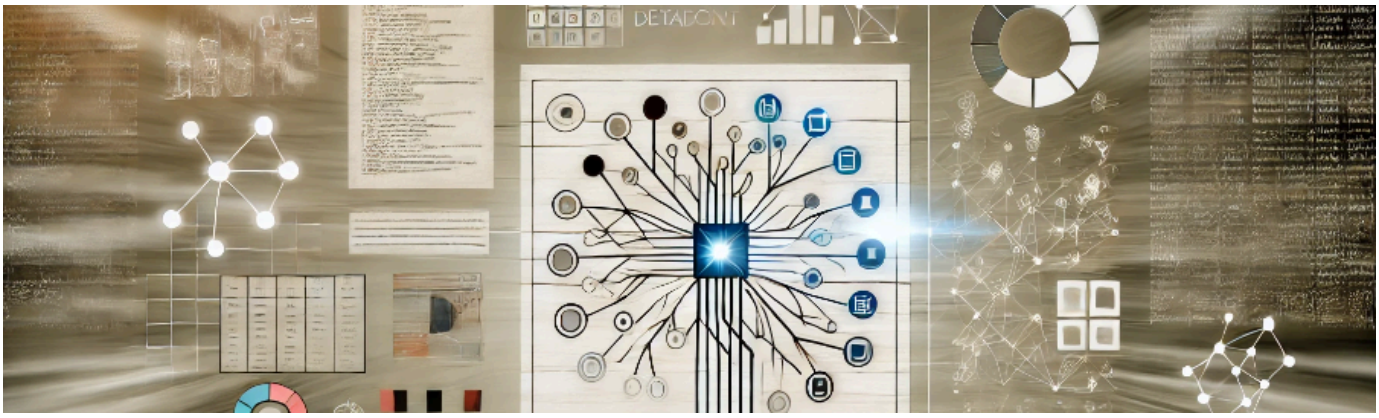
L'importance du jeu de données



Lorsqu'on démarre un problème en deep learning un reflexe est parfois de vouloir adapter une architecture existante à son problème, voire même de penser à faire sa propre architecture. Cette piste peut être coûteuse en temps pour des résultats ne dépassant pas les métriques d'évaluation des architectures déjà existantes.

Comme l'observe The 'it' in AI models is the dataset, le meilleur levier pour améliorer des modèles de deep learning réside davantage dans les données d'entraînement que dans l'architecture. En effet, avoir des données adaptées à notre cas d'application peut être une bonne solution pour améliorer les performances sur la tâche associée. On commence donc par se demander "*Sur quelle tâche dois-je entraîner mon modèle ?*" puis *Quels seront les cas d'applications ?*.

Par exemple en *Natural Language Processing* (NLP), il est central d'identifier la ou les langues utilisées. On peut voir depuis ces métriques que les modèles entraînés depuis l'anglais ne sont pas nécessairement les plus performants sur d'autres langues. Ceci est un exemple de l'adaptation de domaine décrit dans cet article.

Evaluer les performances d'un modèle entrainé en anglais sur un cas d'application en français donnera souvent de moins bons résultats que si l'entrainement avait directement été en français.

Par ailleurs, avoir des données de grandes qualités et donc du même domaine que notre cas d'application peut être utile pour spécialiser notre modèle avec du fine-tuning ou du LoRA.

Partant de ce constat, il est conseillé de commencer la résolution d'un problème par la définition claire des cas d'applications afin de construire son propre jeu de données.

Créer son propre jeu de données, n'est-ce pas trop long ?

Effectivement créer un jeu de données peut être fastidieux, coûteux en temps et en argent. Il est nécessaire souvent d'annoter manuellement les données par exemple. Pour palier à ce problème d'annotation il existe une solution. **Les données synthétiques.**

L'idée est d'utiliser des modèles externes qui s'occuperont de générer d'eux mêmes des données ou des labels.

Par conséquent, en combinant les outils fournis par Python (ou votre langage de programmation préféré) avec ceux des API de modèles déjà existant, il est possible d'automatiser la création de jeux de données adaptés à beaucoup de tâches différentes.

Afin d'illustrer cette pipeline, cet article propose l'automatisation de la création d'un jeu de données pour le *Visual Information Retrieval* et le *Document Visual Question Answering* (DocVQA) pour des diapositives en espagnol.

Définition des tâches

Avant de décrire précisement la pipeline, définissons précisément notre cas d'utilisation afin de sélectionner au mieux les données. On cherche à construire un jeu de données de diapositives en espagnol permettant d'entrainer et d'évaluer des modèles pour le Document Visual Question Answering et le Visual Information Retrieval.

Document Visual Question Answering (DocVQA)

Le Document Visual Question Answering c'est la tâche qui vise à répondre à une question posée en s'appuyant sur un document visuel (pdf, slide etc).

On a donc en entrée une image et une question puis en sortie une réponse. On imagine donc que notre jeu de données devra avoir une feature image de type slide ainsi qu'une feature question/réponse.

Visual Information Retrieval

Le Visual Information Retrieval c'est la tâche qui vise à déterminer parmi un grand nombre de documents et une question, quels sont les documents qui permettent de répondre à la question.

Construction du jeu de données

Pour construire un jeu de donnée adapté on propose la pipeline suivante :

1. Sélection d'une source de donnée de diapositive en espagnol et phase de récupération des données brutes.
2. Génération des questions/réponses grâce à l'API d'un modèle texte/image (Gemini, Claude, ChatGPT...)
3. Mise des données récupérées et synthétisées en un format partageable
4. Filtrage sur les données afin d'éviter des données de mauvaise qualités

On utilise un site de présentations diapositives libre de droit type Slideshare.net comme source de donnée. Il est possible de sélectionner la langue utilisée dans les présentations. Cette source est donc parfaitement adaptée à notre problème.

Enjeu des données synthétiques

L'idée qui rend possible cette automatisation est la combinaison de deux élements : la récupération automatique des données grâce aux méthodes de scraping et la génération automatique de données grâce à un VLM. Ces dernières données sont appelées **données synthétiques**.

L'utilisation des données synthétiques est un avantage pour pouvoir construire son propre jeu de données puisqu'elles apportent :

1. Gain de temps : l'API du modèle génère de manière automatique les questions/réponses ici
2. Réduction des coûts financiers : Ici pas besoin d'investir dans une annotation humaine des données.

Cet enjeu est souligné par le papier ChatGPT outperforms crowd-workers for text-annotation tasks.

Organisation du jeu de données

Une présentation diapositive est composée d'une ou plusieurs diapositive. Après analyse des données disponible sur la page de chaque présentation diapositive on profile notre jeu de donnée pour avoir les features suivantes :

1. Identifiant unique de la présentation
2. Liste d'images diapositives
3. Des méta-données sur chaque présentations (titre, slides les plus lues de la présentation...)
4. Liste de questions/réponses

Voici donc les features retenues :

Feature	Description
id	code d'identification de la présentation
presentation_url	url slideshare.net de la présentation.
title	titre de la présentation
author	username de l'auteur sur slideshare
date	format AAAA-MM-JJ
len	nombre de slides de la présentation
description	description de la présentation
lang	langue renseignée par l'auteur de la présentation
dim	dimension d'une slide HxW
likes	nombre de like sur la présentation
transcript	liste contenant la transcription du texte de chaque slide
mostRead	True pour les indices des listes les plues et False pour les autres.
images	il s'agit d'une liste de toutes les slides de la présentation.
questions/answers	il s'agit d'une liste de liste de dictionnaire de questions/réponses

Cette liste contient les paires questions/réponses de chaque diapositive (une diapositive peut avoir plusieurs paires questions/réponses)

Code

Pour l'exemple, nous allons collecter et annoter une centaine de documents, mais le code peut être très facilement mis à l'échelle sur des milliers de documents en le laissant tourner plus longtemps.

On utilise 3 notebooks python et un script python retrouvable à l'adresse suivante []. Voici l'organisation du code :

1. scrap.ipynb, ce notebook permet de récuperer dans un premier temps les liens de téléchargements des différentes présentations puis s'occupe de télécharger pour chaque présentation les diapositives une à une.
2. generate_qa.ipynb, ce notebook reprend les diapositives téléchargées et genère une ou plusieurs paires question/réponse pour chaque diapositive.
3. filter.ipynb, ce notebook s'occupe de charger les données au format Dataset en oubliant pas de supprimer les questions/réponses trop courtes ou les présentations diapositives avec un contenu trop court.

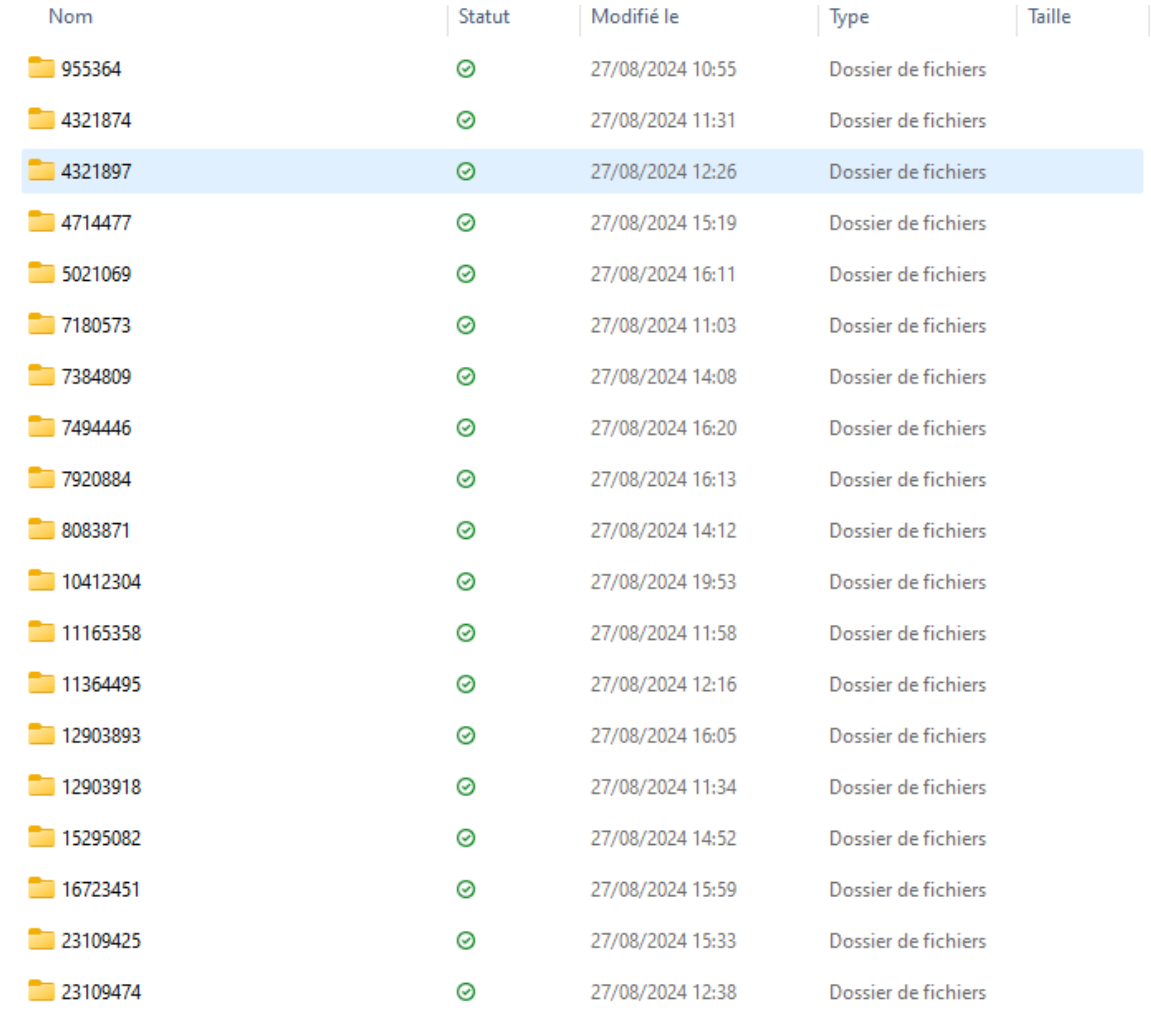
Et enfin dataset.py qui permet la génération du jeu de données au format Dataset HuggingFace.

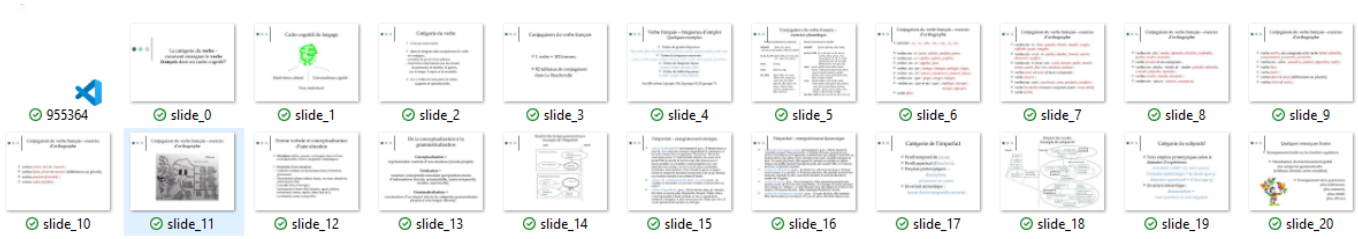
Scraping

A l'aide de module Python il est possible de faire une requête au site de présentations. On récupère ainsi les urls des différentes présentations de diapositives répondant au mot-clé "*Espagnol*".

Une fois ces urls récupérés, on lance les téléchargements des diapositives. Chaque présentation de diapositives est stockée dans un dossier dans lequel figure toutes les diapositives de la présentation donnée.

On enregistre également un fichier .json contenant les méta-données de la présentation (titre, auteur, id etc.)





Génération question/réponse

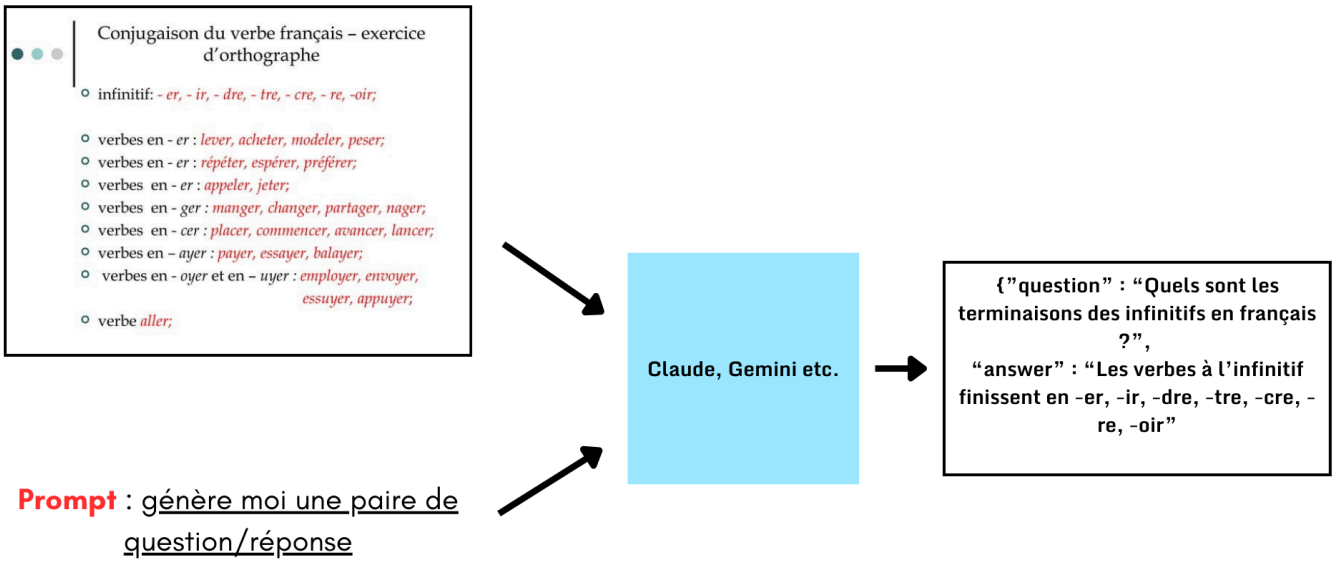
A présent on génère la partie donnée synthétiques, ici les paires de question/réponse pour chaque diapositive. A cette fin, on choisit un VLM muni d'une API avec accès gratuit.

Il est possible par exemple d'utiliser l'API de Claude ou celle de Gemini. Dans le cas d'un petit exemple voici un tableau recessant les limitations des versions gratuites.

	API Claude	API Gemini
Nombre requêtes/jour		
Nombre requêtes/minute		

Une fois le modèle choisi, il suffit de donner les diapositives accompagnées d'un prompt commme par exemple :

```
prompt = "Génère une paire de question réponse au format {question:, answer:}. La question doit être basée sur la diapositive et être instructive. Elle doit utiliser notamment les éléments visuels pour générer la question et la réponse."
```



Pour répondre aux besoins du *Visual Information Retrieval*, les questions/réponses générées (données synthétiques) doivent être sur des éléments précis, voire unique, à la diapositive.

On vérifie donc chaque paire de question/réponse permet d'identifier clairement la diapositive associée.

Pour chaque présentation, on stocke la liste de questions/réponses au format .json dans le dossier associé à la présentation.

Format dataset

Une fois les données constituées, il suffit de générer un jeu de données au format *Dataset*. Pour cela on utilise le template HuggingFace trouvable à cette adresse alt. Il est alors possible de générer notre jeu de données depuis la fonction *load_dataset* de HuggingFace et de l'utiliser pour l'évaluation de nos modèles par exemple.

Filtre sur les données

Afin de gagner en qualité de données, il peut être nécessaire de vérifier si les données synthétiques sont pertinentes. On vérifie manuellement quelques exemples de questions/réponses puis on vérifie leur cohérence vis à vis de la diapositive concernée.

Ensuite, on exclut toutes présentations qui :

1. A une réponse ou une question vide ou courte de moins de 10 caractères.
2. A un nombre de diapositives de moins de 3.

A présent notre jeu de données a gagné en qualité.

Performances

Discutons des performances de cette démarche et de la pertinence de l'utilisation des données synthétiques.

Temps de scraping

Récuperer les diapositives s'est fait grâce à des requêtes HTTP. Avec Python il est possible de récupérer 10 présentations de 20 diapositives ainsi que leurs méta-données rangées en json en **12 min**. Ce temps comprends les

temps de pauses que le programme effectue pour ne pas noyer le site de requêtes HTTP.

Un téléchargement manuel prendrait plus de temps sachant qu'il est nécessaire de ranger les méta-données. Ceci est donc le premier gain de temps de la démarche.

Temps de génération Q/A

La plus grande force de cette méthode est l'utilisation des données synthétiques. En effet en pratique, l'annotation des données passent par la rémunération de *crowd worker* pour les annotations les plus mécaniques.

Dans notre cas d'application, la génération des questions/réponses, la tâche est moins mécanique et demande une expertise plus grande. Cela est d'autant vrai qu'il est nécessaire que l'employé parle espagnol (on rappelle que dans notre exmple on construit un jeu de données en espagnol) et ceci sans faute de syntaxe.

En se basant sur des plateformes comme Amazon Mechanical Turk ou Appen on peut estimer une annotation humaine à minimum 0.5$. On a donc pour **10 présentations de 20 slides à 100$.** En comparaison l'annotation sythétique de 10 présentations à 20 slides avec Claude 3.5 Sonnet API coûte dans notre cas 0.5$.

Pour 200 paires de questions/réponses synthétiques, l'API Gemini met 4 minutes. Ce qui en toute vraissemblance est plus rapide qu'une annotation humaine.

Conclusion et mise à l'échelle

Nous avons donc réussi à génerer un jeu de données pour une application spécifique du Document Visual Question Answering et du Visual Information Retrieval à l'aide des données synthétiques.

Outre un gain de temps dû à l'automatisation, nous pouvons également noté le coût financier qui est en la faveur des données synthétiques. Bien qu'il faille tout de même filtrer les données synthétiques ainsi que vérifier au prélable la qualité des questions/réponses, une annotation manuelle nécessite également cette verification.

Ce jeu de données proche de l'application réel permettra d'optimiser l'évaluation de nos modèles.