HANOUZ Akram, JOUVIN Jules, JALAL Zakaria
DIA3

Projet final XAI

Technical Report

Ce projet vise à intégrer **deux systèmes d'explinability IA** (XAI) existants dans **une seule plateforme** interactive capable de traiter à la fois des données audio et des données image.

Conception et intégration spéciales effectuées

L'Application

Tout le code (sauf ce qui concerne Ollama) se trouve dans le fichier `app.py`. Cela comprend l'UI, la prédiction et les méthodes de XAI (la partie LLM est séparée). Les fonctions servant au LLM se trouvent dans `ollama_llm.py`.

Les deux tâches sont implémentées comme des classifieurs d'images.

- **Pour l'audio**:
    - l'utilisateur upload un fichier `.wav` d'un audio, l'audio est ensuite transformé en "mel-spectrogramme" de Mel (utilisant l'échelle des mels qui est une échelle de hauteurs de son). L'image est redimensionnée en 224×224 RGB puis transmise au modèle
    - si l'utilisateur upload une image de spectrogramme (aucune vérification n'est faite pour vérifier si c'est bien un spectrogramme), elle est simplement redimensionnée comme décrit ci-dessus puis transmise au modèle
- **Pour X-ray**: - L'utilisateur upload une image d'un x-ray thoracique (aucune vérification n'est faite pour vérifier si c'est bien un x-ray), de la même manière, l'image est redimensionnée puis transmise au modèle.

Peu importe la méthode, les deux pipelines renvoient une image redimensionnée, ce qui permet d'avoir un code assez flexible et scallable.

De plus, des modèles spécifiques sont définis pour chaque tâche.

Il est donc assez simple d'ajouter une nouvelle tâche de Computer Vision si elle suit ce schéma.

Classifieur standardisé en 1-logit

Tous les modèles sont des classifieurs binaires et ont pour sortie un seul logit.

Une fonction dédiée (`predict_proba()`) transforme ce logit grâce à l'aide de la fonction sigmoid en probabilités pour l'afficher dans l'UI.

Cette décision « 1 logit au niveau du modèle / 2 classes au niveau de l'API » simplifie l'entraînement tout en prenant en charge les XAI (LIME/SHAP) qui attendent une sortie de probabilité à 2 classes.

Pour l'instant, le code gère uniquement des modèles binaires et ne prend pas en charge la prédiction multi-classe.

Prétraitement

Lors de l'upload, comme évoqué précédement, l'image (ou le spectrogramme généré) est directement redimensionnée en 224×224.

On applique également la normalisation d'ImageNet. Cette normalisation est devenue un standard en CV. Nous l'avons donc utilisée lors de l'entraînement des modèles et elle doit donc être appliquée lors de l'inférence.

La pipeline pour la génération du spectrogramme audio est comme suivante : **matplotlib → PNG → reload**.

Le rendu visuel (axes, interpolation, palette de couleurs, remplissage) peut fortement dégrader le comportement du modèle ; il doit donc être enlevé en utilisant la pipeline décrite.

Grad-CAM pour les modèles binaires

Nous avons intégré Grad-CAM via la librairie `pytorch-grad-cam`.

1. Un `TwoClassWrapper` transforme le modèle binaire à logit unique en un modèle **2-logit** `[-logit, +logit]` afin que Grad-CAM puisse cibler une classe à l'aide du ciblage multiclasses standard utilisé par Grad-CAM.
2. Les couches cibles sont choisies via `get_gradcam_target_layers()` avec des heuristiques et des fall-back spécifiques au modèle, ce qui réduit les cas de défaillance « blank CAM ».

Performances Streamlit + choix de stabilité

Plusieurs choix d'optimisation ont été pris pour rendre l'utilisation de l'application agréable.

- Les poids des modèles sont mis en cache avec `st.cache_resource` (les modèles se chargent une fois par session). Lors de l'upload d'une image, la prédiction est presque instantané. Par contre, chargé le modèle, même si relativement léger prend quelques secondes. En mettant la fonction de load en cache, cela permet une meilleure expérience utilisateur et moins d'attente.
- Les calculs coûteux et répétés sont mis en cache dans `st.session_state` à l'aide d'une **clé de hachage d'entrée** :
  - `xai_cache` stocke les images d'explication calculées. Si la page est rechargé cela évite à devoir les re-exécuter (SHAP même si capé ici peut être assez long et donc frustrant)
  - `uri_cache` stocke les URI de données base64 pour le rendu en superposition (voir plus loin).
  - `xai_stats_cache` stocke des statistiques numériques extraitent des XAI utilisé pour le LLM
  - `llm_cache` stocke la réponse du LLM qui est la chose la plus longue a généré
- Les fichiers temporaires créés pendant le rendu audio vers spectrogramme sont nettoyés avec `try/finally`.

- Les dépendances facultatives (par exemple, OpenCV) ont un fallback : si `cv2` est manquant, le rendu de superposition Grad-CAM revient à un chemin non OpenCV.

Le cache sauf les poids est supprimé entièrement dès qu'un nouveau fichier est upload.

Explication grâce à un LLM locale facultative (Ollama)

L'application a un panneau **d'explication LLM** facultatif alimenté par un **serveur Ollama local** (par exemple, `gpt-oss:20b`).

Comme très demandant en ressource et nécessitant un set up supplémentaire nous avons décidé de le mettre facultatif

L'explication est **uniquement textuelle et fondée sur des valeurs numérique**.L'application n'envoie pas d'images/d'audio au modèle.

Elle fournit plutôt :

- les probabilités prédites + la marge de confiance
- des **résumés numériques** légers dérivés des sorties XAI (Grad-CAM toujours ; LIME/SHAP réutilisé lorsqu'il a déjà été calculé)

Le llm est aussi chargé de réalisé un petit disclaimer sur l'utilisation des résultats dans des décisions importantes.

Modèle sélectionnés et méthodes XAI

Modèles

Le repo prend en charge plusieurs modèles par tâche, tous récupérés à partir du dossier `./weights/{nom du modèle}.pt`.

Ces modèles sont:

- **Détection audio Deepfake**
  - `audio_vgg16`
  - `audio_resnet50`
  - `audio_mobilenetv2`
- **Tâche pour le cancer du poumon (« opacité pulmonaire »)**
  - `xray_alexnet`
  - `xray_densenet121`

Méthodes XAI

L'interface utilisateur propose quatres méthodes d'XAI pour les deux tâches :

- **Grad-CAM** : carte thermique produite par carte d'activation de classe pondérée par le gradient en utilisant les gradients de la cible de la couche convolutive finale
- **LIME (lime-image)** : explication du choix du modèle mais sur chaque valeur (dit local) - **SHAP (shap.Explainer + Image masker)** : attribution basée sur l'impact positif ou négatif des valeurs.
- **Local Ollama LLM (optional)** : réponse textuelle basé sur la prédiction et le XAI

Remarques supplémentaires sur l'intégration :

- Voir la section précédente pour l'intégration de Grad-Cam.
- LIME et SHAP appellent une fonction de classification qui renvoie des probabilités sous la forme `(N, 2)`.
- La visualisation SHAP utilise le rouge et vert pour distinguer les contributions positives (vert) des contributions négatives (rouge).

Améliorations apportées aux repos originels

Deux projets réunis et une meilleure ergonomie

- Les deux tâches sont réunit dans une seule application.
- Les méthodes XAI sont affichés côte à côte dans l'onglet « Comparer ». Ajout d'une option "overlay" qui permet d'afficher l'image originale à la place de l'image issu d'une méthode XAI. Les informations des méthodes XAI sont donc plus facile à accéder (pas besoin de scroll un seul click suffit) et a comparer.

Meilleurs prise en charge d'entrées audio

Le mode audio prend en charge à la fois :

- **des audios au format `.wav`** (convertis en images spectrogrammes mel)
- **les images de melspectrogrammes** (`png/jpg/jpeg`).

Lorsqu'une image melspectrogramme est uploadé, l'application essaye de **reconstitué le WAV** pour obtenir un aperçus audibles (un avertissement est affiché concernant la fidélité du rendu)

Mise en cache pour une appli plus interactive

Les résultats XAI sont mis en cache, ce qui évite tout recalcul inutile (qui peuvent être long et frustrant) lorsque Streamlit est relancé. Notamment, les données URI mises en cache permettent d'obtenir un comportement de survol stable et réactif sans avoir à recharger les images à plusieurs reprises.

Meilleure ergonomie d'exécution

Même si non recommandé car peut augmenter considérablement le temps d'exécution, si aucun GPU (cuda si disponible) est détecté, sélectionne automatiquement le CPU. Ceci est affiché sur l'UI.

Gestion des fichiers temporaires générés.

Utilisation falcultative d'Open-CV supporté.

Explications LLM Locales

Ajout d'une option permettant via un modèle LLM servit sur Ollama.

Voire le dernier paragraphe de la session "Conception et Intégration Spéciale Effectué" pour plus de détail.

Note additionnelle et contrainte

- L'application a besoin du dossier `./weights/` contenant les poids des modèles. Ces derniers sont soit à télécharger sur le lien drive partagé ou a regénérer via le notebook `training_models.ipynb`.
- PyTorch n'est pas inclu dans le fichier `requirements.txt`. Il est laissé à l'utilisateur le choix de la version de CUDA (ou sans) à téléchargé.
- SHAP métode est assez lent et nous avons du capé son nombre d'évaluation pour le garder utilisable facilement pour une démo live.
- l'explication via LLM est optionnel et demande à l'utilisateur de télécharger et servir un serveur Ollama. L'explication peut également être assez longue à générer dépendant du modèle.

Toutes ces informations sont décrites dans le `README`