Microsoft en France
Ecole IA

SIMPLON.CO

Amélioration d’une application d’IA

Amélioration d’une application de classification d’images avec PyTorch et Flask

Rédacteur : Quentin Angelot

Introduction

Pour le scénario présent, notre start-up MedTech vient de remporter un appel d'offre pour la réalisation d'un POC (Proof Of Concept) d'une solution IA capable de classer les images radios en six catégories. L'objectif étant de prouver les compétences techniques de notre start-up et de faire adhérer le corps médical au projet.

Une première version du modèle est construite avec Pytorch (MedNet by NVIDIA) et Flask. Il s'agit d'une interface simple où on sélectionne une image en local et qui lance le modèle pour prédire sa classe. Le nom de la classe prédite s'affiche ensuite à l'écran.

L'amélioration d'un projet d'IA peut-être avoir plusieurs axes :

- On améliore les performances de notre modèle d'IA a proprement parlé au regard de la métrique retenue
- On améliore son architecture fonctionnelle en implémentant une ou plusieurs fonctionnalités additionnelles de façon à maximiser sa valeur pour les équipes métier

Ici, notre mission est donc d'améliorer le modèle MedNet et d'ajouter une fonctionnalité qui permet de sélectionner un ensemble d'images et de les classer dans le dossier approprié en utilisant le modèle amélioré.

Concernant la charge de travail nécessaire, nous estimons le besoin en R&D pour l'amélioration de MedNet à 2 jours-hommes et le besoin en développement à 1 jour-homme.

Le notebook nécessaire à l'élaboration du modèle et le code de l'application Flask sont disponibles sur Github.

I. L’amélioration du modèle d’IA

Le modèle proposé par les équipes de Nvidia obtient déjà un bon score dans la classification de ces radiographies. Cependant, en optimisant la structure du modèle de base ou la distribution de notre jeu de données, nous pouvons peut-être limiter les erreurs de classification.

Le jeu de données

Il contient 58954 images dans 6 catégories distinctes : 'ChestCT', 'BreastMRI', 'CXR', 'AbdomenCT', 'HeadCT' et 'Hand'. Ces catégories contiennent respectivement 10000, 8954, 10000, 10000, 10000 et 10000 images. Ainsi, on peut noter que la classe BreastMRI est légèrement sous représentée. Enfin, notons que ces images sont de dimensions modestes : 64 x 64 pixels.

Le modèle de base

Le choix du CNN par les équipes NVIDIA tient au fait qu’il réalise d’excellentes performances en matière de classification d’images tout en restant relativement raisonnable dans ses temps d’entraînement et de computation. L’entrainement d'un CNN sur un ensemble d'images n'est pas si différent de la formation d'un perceptron multicouches sur des données numériques :

- Définition de l'architecture du modèle
- Chargement du jeu de données depuis le disque
- Implémentation de l’itération sur les epochs et batches
- Réalisation des prédictions et calcul de la perte
- Remise à zéro du gradient, rétropropagation et mise à jour des paramètres de notre modèle

La différence majeure entre un CNN et un MLP traditionnel tient à la couche de convolutions. Lorsqu'il interprète une image, l'œil identifie d'abord les bords et les limites. Ensuite, on peut distinguer des courbes, des formes et des

structures plus complexes à des niveaux d'abstraction plus élevés. En ne combinant d'abord que les informations des pixels voisins, une série de couches de convolutions imite ce processus organique. La taille de la convolution correspond au nombre de pixels adjacents pondérés et additionnés lors du passage à la couche suivante. De plus, ce processus permet de réduire largement le nombre de paramètres à apprendre lors de l'entraînement du modèle.

La structure initiale du modèle de NVIDIA est la suivante :

- numConvs1 = 5 : le nombre de convolutions dans le premier layer
- convSize1 = 7 : la taille du kernel utilisé
- numConvs2 = 10 : le nombre de convolutions dans le second layer
- convSize2 = 7 : la taille du kernel utilisé
- fcSize1 = 400 : le nombre de nœuds de sortie de la première couche entièrement connectée
- fcSize2 = 80 : le nombre de nœuds de sortie de la seconde couche entièrement connectée
- elu : la fonction d'activation utilisée
- learnRate = 0.01 : le learning rate
- maxEpochs = 20 : le nombre maximum d'epochs
- t2vRatio = 1.2 : le ratio maximal entre la perte de validation et la perte d'entraînement
- t2vEpochs = 3 : le nombre d'époques consécutives avant de s'arrêter si la perte de validation dépasse le ratio ci-dessus.
- batchSize = 300 : la taille du Batch

Les améliorations apportées et leur impact sur le modèle

Pour améliorer le modèle, il nous faut travailler de façon méthodique et envisager chaque piste d'amélioration séparément afin de pouvoir imputer le résultat de façon certaine.

Concernant le nombre maximum d'epochs, il faut savoir qu'un entraînement de plus en plus long peut entraîner une amélioration continue de la perte d'entraînement, mais la précision du modèle sur l'ensemble de données de test se stabilisera peu de temps après que la perte de validation et la perte d'entraînement auront commencé à diverger. Le paramètre t2vEpochs est donc très pertinent.

Les tailles de Batch réduites entraînent une amélioration de la vitesse d'entraînement car il y a plus d'itérations par époque. Cependant, cela ne se traduit pas nécessairement par une amélioration du modèle à long terme, car la mise à jour sur moins d'exemples induit plus de fluctuations dans les poids du modèle. L'augmentation du taux d'apprentissage peut également conduire au même type de comportement. Et inversement, un taux d'apprentissage trop faible peut également rendre l'apprentissage trop long.

Ici, après de nombreuses expérimentations, nous avons accru le learning rate de 0.01 à 0.05, tout en augmentant le t2vRatio de 1.2 à 1.5, cela permettant une certaine tolérance lors de la divergence entre la perte d'entraînement et de validation. Nous avons également réduit la batch size en essayant différentes valeurs, nous avons obtenu un bien meilleur résultat en passant de 300 à 64 images.

Un autre élément fondamental dans notre quête de résultats fut le passage de la fonction d'activation ELU (Exponential Linear Unit) à la fonction RELU (Rectified Linear Unit), ceci a permis un gain important en limitant les erreurs de classification entre "Chest" et "Breast" notamment.

Ensuite, nous avons choisi de travailler sur la complexité du modèle. Nous avons ainsi accru le nombre de convolutions en les passant de respectivement de 5 à 10 pour le premier layer et de 10 à 20 pour le second. De même, nous avons choisi de réduire la dimension du kernel de (7, 7) à (5, 5). Cette complexité du modèle a significativement contribué à améliorer la précision.

En particulier, augmenter le nombre de convolutions a permis de capturer davantage de patterns dans les images.

En revanche, concernant la taille des couches entièrement connectées, les changements apportés à la hausse n'ont pas porté leurs fruits. Nous sommes donc restés sur 400 pour le premier et 80 pour le second. De même l'ajout d'une troisième couche de convolutions n'a pas été pertinent.

La Batchnorm et le Dropout sont des techniques de régularisation parmi les plus populaires en computer vision. Elles peuvent également conduire à des améliorations marginales à long terme. La Batchnorm permet de rendre l'entraînement des réseaux de neurones plus rapides et plus stables grâce à la normalisation des entrées des différentes couches par recentrage et remise à l'échelle. Ici, deux Batchnorms successives ont permis d'améliorer légèrement le modèle. A contrario, un dropout à 0.2 puis à 0.5 a dégradé le modèle. En effet, la régularisation était trop forte et s'est traduite par une perte d'information pour le modèle.

De même, l'ajout de deux layers de pooling entre les convolutions n'a pas eu un impact significatif. L'opération pooling consiste à faire glisser un filtre sur chaque canal de la carte de features et à résumer les pixels se trouvant dans la région couverte par le filtre (en prenant la valeur maximale ou moyenne). Ici, on peut penser que la taille des images et la simplicité du modèle ne rendent pas cette étape prépondérante.

Pour conclure, nous avons essayé de modifier la quantité de certaines classes afin d'aider le modèle à se concentrer sur les classes problématiques (Chest, Breast et Hand notamment). De même, nous avons utilisé des pondérations personnalisées afin de mettre en avant une classe par rapport aux autres. Cependant, cela n'a pas permis d'éliminer les dernières erreurs de classification.

L'architecture du modèle final et ses résultats

L'architecture retenue pour notre version du CNN est la suivante :

- numConvs1 = 10 : le nombre de convolutions dans le premier layer
- convSize1 = 5 : la taille du kernel utilisé
- numConvs2 = 20 : le nombre de convolutions dans le second layer
- convSize2 = 5 : la taille du kernel utilisé
- fcSize1 = 400 : le nombre de nœuds de sortie de la première couche entièrement connectée
- fcSize2 = 80 : le nombre de nœuds de sortie de la seconde couche entièrement connectée
- relu : la fonction d'activation utilisée
- learnRate = 0.05 : le learning rate
- maxEpochs = 20 : le nombre maximum d'epochs
- t2vRatio = 1.5 : le ratio maximal entre la perte de validation et la perte d'entraînement
- t2vEpochs = 4 : le nombre d'époques consécutives avant de s'arrêter si la perte de validation dépasse le ratio ci-dessus.
- batchSize = 64 : la taille du Batch

Après ce processus itératif d'optimisation du modèle, notre classification a vraiment bien évolué, en termes de précision et de matrice de confusion. Nous sommes passés de 99,459% de précision avec le modèle proposé par Nvidia à 99.947% pour le nôtre. Soit une progression de près de 0.488 point.

```
Correct predictions:  5709 of 5740
Confusion Matrix:
[[966   0   5   0   0   1]
 [  1 997   0   4   0   1]
 [  8   0 977   0   0   0]
 [  0   3   1 954   0   1]
 [  0   0   0   4 947   0]
 [  2   0   0   0   0 868]]
['ChestCT', 'CXR', 'AbdomenCT', 'Hand', 'HeadCT', 'BreastMRI']
```

On constate que le modèle de base se trompe fréquemment en essayant de distinguer des ChestCT et des AbdomenCT. C'est un point sur lequel notre modèle semble avoir réellement progressé.

```
Correct predictions:  5737 of 5740
Confusion Matrix:
[[972   0   0   0   0   0]
 [  0 898   0   0   0   0]
 [  0   0 991   0   0   1]
 [  1   0   0 962   0   0]
 [  0   0   0   0 939   0]
 [  0   1   0   0   0 975]]
['ChestCT', 'BreastMRI', 'CXR', 'AbdomenCT', 'HeadCT', 'Hand']
```

II. L'amélioration des fonctionnalités

Flask est un framework Python de développement web. Il a pour objectif de garder un noyau simple mais extensible. Il n'intègre pas de système d'authentification, pas de couche d'abstraction de base de données, ni d'outil de validation de formulaires. Cependant, de nombreuses extensions permettent d'ajouter facilement ces fonctionnalités lorsqu'elles sont nécessaires. C'est donc l'outil parfait pour ce POC.

La prise en charge des prédictions multiples dans Flask et la classification automatique dans les sous-fichiers correspondants :

- Dans app.py, on définit la route qui va permettre d'importer plusieurs fichiers. La méthode request.files.getlist("file") permet de récupérer les fichiers postés par l'utilisateur sous forme d'une liste. Ensuite, nous pouvons itérer sur cette liste afin d'obtenir des prédictions pour chacune des images avec la méthode get_prediction(). Cette méthode retourne deux éléments : class_name et class_id. Avec l'élément class_name, on peut mettre en place une structure conditionnelle qui vient tester les différentes valeurs possibles de celui-ci. Un fois la condition remplie, on

enregistre l'image dans le bon répertoire. La fonctionne create_folder crée le répertoire s'il n'existe pas déjà.

- Dans inference.py, on charge le modèle entraîné et on définit la fonction get_prediction() utilisée dans app.py. Celle-ci réalise le prétraitement des images nécessaire à la prédiction par le modèle en faisant appel à d'autres fonctions utilitaires définies dans commons.py.
- Dans templates/result.html, nous utilisons une boucle for via Jinja pour pouvoir afficher plusieurs résultats au lieu d'un seul. Nous affichons ainsi le nom et l'id de la classe prédite ainsi que l'image correspondante.

III. Le contrôle de la non-régression

A chaque nouvelle version d'une application qui va être mise en production, il y a un risque de régression qui pèse sur les anciennes fonctionnalités. Les contrôles de non-régression sont donc essentiels pour garantir la qualité de notre application. Ici, elle reste très simple et il ne s'agit que d'un POC, nous pouvons donc contrôler la non-régression manuellement.

Pour se faire, il faut :

- Repérer une modification dans le code de l'application et trouvez les modules concernés. Ici, le module qui subit la modification majeure est app.py, avec le passage de la prédiction simple à la prédiction par lots et avec l'ajout de la fonctionnalité d'attribution de la bonne image au bon dossier. D'autre part, il faut veiller au bon affichage des résultats retournés.
- Réaliser le contrôle. Nous vérifions donc l'upload, la prédiction et la répartition des différentes images dans les dossiers, ainsi que la page d'affichage des résultats, manuellement à l'aide d'un ensemble aléatoire d'images issues du jeu de données de test.

- Déterminer le point de sortie avec les critères qui doivent être remplis avant de terminer le test. Ici, l'upload se déroule correctement, le modèle réalise des prédictions pertinentes, les différents dossiers sont créés et justement alimentés et la page d'affichage des résultats fonctionne également.
- S'assurer que le contrôle couvre toutes les variations et flux d'utilisation possibles. Ici, notre flux d'utilisation est très simple, on arrive sur la page d'accueil, on upload des images, on réalise des prédictions sur celles-ci, les sous-dossiers sont alimentés par les images correspondantes et une page de résultats rend compte du travail effectué. Il n'y a pas de flux d'utilisation alternatif.

Conclusion

Nous avons donc procédé à l'amélioration significative du modèle d'IA en passant de 31 à 3 erreurs sur le jeu de données de test. Nous avons également ajouté une évolution fonctionnelle majeure à l'application, afin de répondre aux besoins du corps médical. Celle-ci permet de sélectionner un ensemble d'images et de les classer dans le dossier approprié à l'aide d'un modèle d'IA hautement performant. Et enfin, nous nous sommes assurés de la non régression de l'application par rapport à la version initiale. Ces différentes exigences auxquelles nous nous sommes efforcés de répondre nous permettent d'obtenir une certaine crédibilité nécessaire à la confiance et à l'adhésion du personnel médical.

Afin de s'assurer de cette adhésion à l'outil développé, il est important de rappeler que l'IA ne remplacera jamais le personnel soignant. En effet, des technologies si avancées requièrent des personnels hautement qualifiés afin de les utiliser avec recul et d'en interpréter les résultats. De plus, il convient d'ajouter que notre produit n'est qu'une solution de classification d'images et qu'elle ne prend aucune décision d'ordre médicale au sens strict. Au contraire, cet outil doit permettre de libérer du temps au corps médical afin qu'il puisse être davantage au contact des patients.