Travaux Pratiques sur l'IA Explicable (XAI)

Expérimentation et Analyse de Modèles de Classification d'Images

Initiation au Deep Learning - ENSICAEN

6 novembre 2025

Résumé

Cette séance de travaux pratiques d'une durée de 4 heures vise à vous faire expérimenter et analyser trois techniques d'explicabilité des modèles de deep learning appliqués à la classification d'images. L'objectif n'est pas de compléter un code existant, mais de vous amener à construire, comprendre et critiquer ces méthodes en vous basant sur la littérature scientifique. L'accent sera mis sur l'expérimentation et l'interprétation des résultats.

Introduction et Objectifs

Nous utiliserons un classifieur d'images pré-entraîné sur ImageNet, disponible dans la bibliothèque `torchvision`. Vous êtes libres de choisir le modèle de votre choix (ex : `AlexNet`, `ResNet50`, `VGG16`, etc.). Pour charger des images locales, vous pouvez utiliser la bibliothèque `Pillow` (PIL).

Les trois méthodes à explorer sont :

1. **Méthode globale par inversion de modèle :** Générer une image "prototype" qui maximise l'activation d'une classe donnée.
2. **Méthode locale par carte de saillance :** Identifier les pixels d'une image d'entrée qui influencent le plus la décision du modèle.
3. **Méthode locale par masquage aléatoire :** Évaluer l'importance des régions d'une image en mesurant l'impact de leur masquage sur le score de classification.

Quelques classes ImageNet intéressantes à explorer :

— 130 : Flamant rose (*flamingo*)
— 208 : Labrador retriever
— 288 : Léopard (*leopard*)
— 402 : Guitare acoustique (*acoustic guitar*)
— 620 : Ordinateur portable (*laptop*)
— 817 : Voiture de sport (*sports car*)
— 954 : Banane (*banana*)

1 Exercice 1 : Méthode Globale par Inversion du Modèle

Principe de la méthode

Cette approche, inspirée de l'article *"Understanding deep image representations by inverting them"* (Mahendran & Vedaldi, 2015), est une méthode **globale**. Elle ne cherche pas à expliquer la prédiction pour une image spécifique, mais plutôt à visualiser ce que le réseau a appris pour une classe en particulier.

Le principe est de partir d'une image (soit de bruit aléatoire, soit une image réelle) et de l'optimiser itérativement par **descente de gradient** pour maximiser le score de sortie d'une

classe cible. L'image résultante est une sorte de "rêve" du réseau, représentant son "idée" de la classe en question. Sans contraintes, ces images peuvent être très bruitées (exemples adverses). On y ajoute donc une **régularisation** pour obtenir des résultats plus interprétables.

Travail à faire

1. **Initialisation :**
   — Chargez un modèle pré-entraîné (`torchvision.models`).
   — Créez un tenseur PyTorch représentant une image de bruit aléatoire (ex : 224x224x3). Ce tenseur doit être un paramètre à optimiser (pensez à `requires_grad_()`).
   — Choisissez une classe cible parmi la liste fournie ou une autre de votre choix.
2. **Boucle d'optimisation :**
   — Définissez un optimiseur (ex : `torch.optim.SGD`) qui optimisera votre image.
   — Écrivez une boucle qui, à chaque itération :
   (a) Fait une passe avant (forward) de l'image à travers le modèle.
   (b) Calcule une "perte" (*loss*) qui est l'**opposé** du score de la classe cible (car les optimiseurs minimisent la perte).
   (c) Effectue la rétropropagation (`loss.backward()`) pour calculer le gradient de la perte par rapport aux pixels de l'image.
   (d) Met à jour l'image en utilisant l'optimiseur (`optimizer.step()`).
   (e) Applique une étape de régularisation sur l'image.
3. **Visualisation :**
   — Après un certain nombre d'itérations, l'image optimisée doit être post-traitée pour être affichée : dé-normalisation (inversion de la normalisation d'ImageNet), clipping des valeurs entre 0 et 1, et permutation des axes pour `matplotlib`.

Pistes d'expérimentation

— **Importance de la régularisation :**
   — Implémentez la méthode **sans régularisation** en partant du bruit. Observez l'image générée. Est-elle reconnaissable ? Ces images sont proches du concept d'*exemples adverses*.
   — Ajoutez une régularisation simple : à intervalles réguliers (ex : toutes les 5 itérations), appliquez un léger **flou Gaussien** à l'image (`torchvision.transforms.GaussianBlur`). Comparez les résultats.
— **Expérimentation "Deep Dream" :** Au lieu de partir d'un bruit aléatoire, chargez une image réelle (par exemple une photo de nuages, d'un paysage, ou même votre propre photo) et utilisez-la comme point de départ de votre optimisation. Le réseau va alors "rêver" et amplifier les motifs de la classe cible qu'il perçoit déjà dans l'image. Testez avec différentes classes sur la même image de départ.
— **Exploration des classes et modèles :**
   — Générez des images pour des classes variées : un animal, un objet, un véhicule. Le réseau a-t-il une représentation aussi claire pour toutes les classes ?
   — Remplacez AlexNet par un modèle plus moderne comme `ResNet50`. Les "rêves" de ce réseau sont-ils plus réalistes ?

2 Exercice 2 : Méthode Locale par Carte de Saillance

Principe de la méthode

Cette méthode locale, proposée dans l'article *"Deep inside convolutional networks : visualising image classification models and saliency maps"* (Simonyan et al., 2014), vise à expliquer

pourquoi une image d'entrée spécifique a été classée d'une certaine manière.

L'idée est simple : si un pixel est important pour la décision, une petite variation de ce pixel devrait entraîner une grande variation du score de la classe prédite. Mathématiquement, cela correspond à calculer le **gradient du score de la classe par rapport à chaque pixel de l'image d'entrée**. Les pixels ayant une forte valeur de gradient (en valeur absolue) sont considérés comme les plus "saillants".

Travail à faire

1. **Préparation de l'image :**
   - Chargez une image de votre choix (ex : une image de flamant rose que vous aurez téléchargée) avec `Pillow`.
   - Appliquez les transformations nécessaires pour la donner au modèle : conversion en tenseur, redimensionnement, et normalisation. Vous pouvez utiliser `torchvision.transforms`.
   - Assurez-vous que le tenseur de l'image autorise le calcul du gradient (`requires_grad_=True`).
2. **Calcul du gradient :**
   - Passez l'image à travers le modèle pour obtenir les scores de sortie.
   - Identifiez la classe prédite (celle avec le score le plus élevé, `torch.argmax`).
   - Prenez le score de cette classe et appelez la fonction `.backward()` dessus. PyTorch va alors stocker le gradient de ce score par rapport à l'image d'entrée.
3. **Visualisation de la carte de saillance :**
   - Récupérez le gradient qui a été stocké dans l'attribut `.grad` de votre tenseur d'image.
   - Ce gradient a la même taille que l'image (3 canaux). Pour le visualiser, vous pouvez :
     (a) Prendre la **valeur absolue** du gradient.
     (b) Calculer le **maximum** sur les 3 canaux de couleur pour obtenir une carte 2D.
     (c) Normaliser cette carte entre 0 et 1 pour l'affichage.
   - Affichez la carte de saillance (ex : `plt.imshow(saliency, cmap='hot')`).

Pistes d'expérimentation

- **Saillance pour différentes classes :** Pour la même image, calculez la carte de saillance non pas pour la classe prédite, mais pour une autre classe plausible ou erronée. Pour une image de chien, quelle est la carte de saillance pour la classe "chat" ? Le réseau se concentre-t-il sur les mêmes zones ?
- **Amélioration de la carte (SmoothGrad) :** La carte de saillance peut être bruitée. Pour la "lisser", implémentez **SmoothGrad** :
  1. Créez plusieurs copies (ex : 20) de votre image en y ajoutant un léger bruit Gaussien.
  2. Calculez la carte de saillance pour chacune de ces images bruitées.
  3. Faites la **moyenne** de toutes les cartes obtenues.

  Comparez avec le résultat original. L'explication est-elle plus claire ?
- **Grad-CAM vs Saliency :** Recherchez et implémentez une autre méthode basée sur les gradients, **Grad-CAM**. Comparez ses résultats avec ceux de la carte de saillance simple. Laquelle des deux méthodes vous semble la plus fidèle à la "logique" du modèle ?

3 Exercice 3 : Méthode Locale par Masquage Aléatoire (RISE)

Le troisième exercice consiste à implémenter une méthode d'explication par perturbation, inspirée de l'article suivant :

Petsiuk, V., Das, A., Saenko, K. (2018). ***RISE : Randomized Input Sampling for Explanation of Black-box Models.*** **In BMVC.** (Lien vers l'article sur arXiv)

Pour cet exercice, il vous est demandé de **lire attentivement la section 3 de l'article** pour comprendre le fonctionnement de la méthode RISE et de l'implémenter vous-même. C'est une approche "black-box" qui ne nécessite pas d'accéder aux gradients du modèle.

Pistes d'expérimentation

Une fois votre implémentation fonctionnelle, explorez l'impact des hyperparamètres de la méthode :

- **Nombre de masques (N) :** L'article utilise plusieurs milliers de masques. Comparez la carte d'importance générée avec $N = 100$, $N = 500$, et $N = 2000$. Comment la qualité de l'explication évolue-t-elle ?
- **Structure des masques :** L'article génère des masques de petite taille (ex : 8x8) qu'il interpole ensuite.
  - Variez la taille initiale du masque binaire (ex : 4x4, 16x16). Des masques plus fins ou plus grossiers donnent-ils de meilleures explications ?
  - Testez d'autres types de masques : au lieu de masques binaires interpolés, essayez des motifs différents comme des bandes verticales, des grilles, ou un simple bruit blanc Gaussien.
- **Combinaison image/masque :** La méthode RISE multiplie l'image par le masque. Essayez une autre approche : utilisez le masque pour "occulter" l'image, en remplaçant les pixels masqués non pas par du noir, mais par une couleur moyenne (le gris) ou par une version floutée de l'image.