Extraction de données multi-sources
Projet Final : Détection de fissures

Aurélien Blanco

21/12 Décembre 2025

Abstract

Nous nous intéressons ici à un problème de classification binaire concernant des fissures présentes sur des murs d'habitations. Nous allons traiter ce problème à l'aide d'algorithmes d'apprentissage automatique et nous allons donc faire face à la complexité du jeu de données, qui ne consiste pas uniquement à détecter des formes, mais à les détecter à travers différentes architectures de maisons. Grâce à un pré-traitement judicieux et à un fine-tuning finement pensé, nous proposons une méthode permettant d'atteindre 80% sur le jeu de données de test fourni.

1 Introduction

Les réseaux de neurones convolutionnels profonds (CNN) ont permis d'obtenir un gain de précision significatif dans la classification d'images [4]. Ce gain peut être très élevé par rapport aux approches traditionnelles, comme les réseaux de neurones profonds ou les machines à vecteurs de support (SVM) [6]. Ils se démarquent par les filtres qu'ils appliquent à toutes les parties d'une image afin d'en extraire des caractéristiques. Avant qu'une ou plusieurs couches de réseaux de neurones simples (aussi appelés " dense ") ne s'appliquent, les positions des éléments sont calculées et comparées entre elles pour prédire finalement une classe, c'est-à-dire un sous-ensemble de caractéristiques. Nous n'aborderons pas ici les transformers [9], qui ont fait une avancée remarquable dans la classification, et nous en resterons aux CNN.

Depuis plusieurs années, la communauté scientifique est confrontée à des jeux de données de plus en plus grands et complexes. On peut notamment citer le très célèbre ImageNet [8], qui, depuis 2010, intègre la majorité des nouveaux

articles abordant le thème de la vision par ordinateur. Ce jeu de données très populaire contient 14 197 122 images réparties dans 21 841 classes différentes, ce qui en fait l'un des plus importants disponibles gratuitement en ligne.

Nous nous intéressons ici uniquement à la détection de fissures sur des murs dans le but d'effectuer une classification binaire. Il s'agit d'une caractéristique assez peu fréquente dans les challenges de nos jours, alors que la plupart des articles se concentrent sur des sujets plus distincts (chat, chien, voiture, avion, etc.). C'est pourquoi nous avons fait preuve de créativité pour proposer un modèle atteignant 80% d'exactitude sur les quelques centaines données de test, malgré l'absence de jeu de données d'entraînement sur cette caractéristique précise.

2 Related work

Actuellement, les derniers modèles de réseaux de neurones convolutionnels atteignent environ 95% de précision lors de la classification du jeu de données ImageNet [8] [7]. Ces performances impressionnantes ne nous permettent en aucun cas d'établir des conclusions sur notre problématique.

3 Method

3.1 Architecture

Au début, nous avions décidé d'implémenter notre CNN en Python en utilisant uniquement CuPy (une alternative très similaire à Numpy permettant d'effectuer des opérations matricielles sur le GPU en utilisant CUDA), mais les boucles for se sont avérées très coûteuses, ce qui a rendu cette implémentation inutilisable.

Nous nous étions donc orientés vers PyTorch pour implémenter notre modèle. Voyant que les temps de calcul étaient beaucoup plus rapides, nous pensions pouvoir nous en tenir là pour notre modèle, mais comme les jeux de données d'entraînement étaient peu nombreux, nous avons dû abandonner cette option.

L'idée finale a donc été d'utiliser un modèle déjà existant sur PyTorch et de récupérer ses poids déjà entraînés afin d'avoir une base plus solide pour notre classification. Nous avons donc décidé de faire appel à ResNet34 [2]. L'architecture du modèle est la suivante :

- Convolution 7 x 7, avec 64 filtres, avec un stride à 2
- Max Pool 3 x 3, avec un stride à 2
- Residual * 3, avec Residual = Convolution 3 x 3, avec filtres = 64
- Residual * 4, avec Residual = Convolution 3 x 3, avec filtres = 128

- Residual * 6, avec Residual = Convolution 3 x 3, avec filtres = 256
- Residual * 3, avec Residual = Convolution 3 x 3, avec filtres = 512
- Softmax

L'architecture du modèle ResNet repose sur ses blocs "Residual" que l'on peut voir ci-dessous :

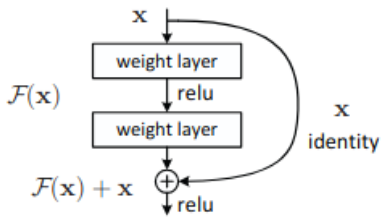

Figure 1: Représentation d'un block d'apprentissage résiduel

L'opération consiste à ajouter une couche x à une couche x + 2. Ce bloc permet de conserver une partie de l'information, même après avoir traversé une couche. Ce " raccourci " a été testé dans le papier selon différentes formules et comparé à une architecture équivalente sans résidus. Il a démontré de meilleurs résultats dans la majorité des cas, excepté lorsque le gradient explose, ce qui peut être corrigé.

De plus, ce modèle a été entraîné sur le jeu de données ImageNet1K (dans PyTorch), ce qui signifie qu'il a été entraîné à distinguer des images appartenant à 1 000 catégories différentes. C'est beaucoup trop pour nous.

On en déduit donc que cette méthode répond parfaitement à tous nos besoins. D'une part, le fait que les poids aient déjà été entraînés signifie que le modèle sait déjà détecter des coins, des séparations, etc. D'autre part, l'architecture du modèle lui confère une grande stabilité lors des différentes étapes d'apprentissage, ce qui évite qu'il néglige tout son apprentissage lorsqu'il en acquiert un nouveau. Enfin, en utilisant la version de ResNet à 34 couches [2], on s'assure que le modèle ne sur- ou sous-apprendra pas sur les nouvelles données.

3.2 Parameters

3.2.1 Optimizer

En ce qui concerne l'optimisation, AdamW [5] s'est avéré être bien meilleur que les autres méthodes Adam [3] et SGD [1]. En effet, cette variante permet de régulariser l'apprentissage et d'éviter la sur-apprentissage (phénomène qui peut se produire lorsque l'on utilise les autres méthodes). Cela s'explique par

le fonctionnement d'AdamW [5], qui repose sur le momentum, le RMS prop et la régularisation L2. Dans notre cas, nous n'avions pas de sur-apprentissage à première vue, mais cette régularisation est nécessaire pour obtenir la meilleure généralisation possible.

3.2.2 Loss function

Nous avons utilisé l'entropie croisée comme fonction de coût. Cette fonction est réputée et utilisée dans de nombreuses applications. Ici, elle répond tout à fait à nos besoins.

3.3 Hyperparameters

Voici donc toutes les valeurs que nous avons utilisées pour nos hyper-paramètres :

- Optimizer : AdamW
- Learning rate : 0.001
- Weight decay : 0.0001
- Loss function : Cross Entropy
- Batch size : 128
- Image resize : 224 * 224
- Image noramlization : True
- Base ResNet34 weights : IMAGENET1K_V1
- Precision : FP32 (FP16 était difficile à mettre en place mais aurait pu très bien fonctionner sans ajouter de grandes pertes de précision)

3.4 Fine-tuning

Afin de pouvoir tester notre modèle pour notre problème de classification binaire, quelques ajustements ont dû être effectués. Tout d'abord, la dernière couche du modèle a été modifiée de sorte qu'elle ne prédise plus une classe parmi 1 000, mais parmi 2 (fissures ou pas de fissures). Nous avons ensuite utilisé une fonction d'entraînement que nous aurions utilisée pour un modèle vierge, ce qui a suffi à modifier les paramètres pour obtenir un modèle de détection de fissures.

Tous ces choix ont été faits de manière à généraliser au maximum le modèle, car notre objectif principal n'est pas d'approximer correctement le nouveau jeu de données de fin de formation, mais d'avoir un modèle général qui approximera correctement le jeu de données de test.

4 Experiments

4.1 Cracks dataset

Pour affiner notre modèle et en faire un modèle de détection de fissures de murs, nous avons besoin d'un jeu de données de fin-tuning qui permettra d'établir un lien entre les connaissances du modèle et les données de test inconnues. Pour ce faire, nous avons trouvé un jeu de données étiquetées de fissures au sol (malheureusement plus disponible en ligne) qui nous a permis de réentraîner notre modèle afin de détecter la présence ou non d'une fissure sur une image. Voici un extrait de ce dernier :

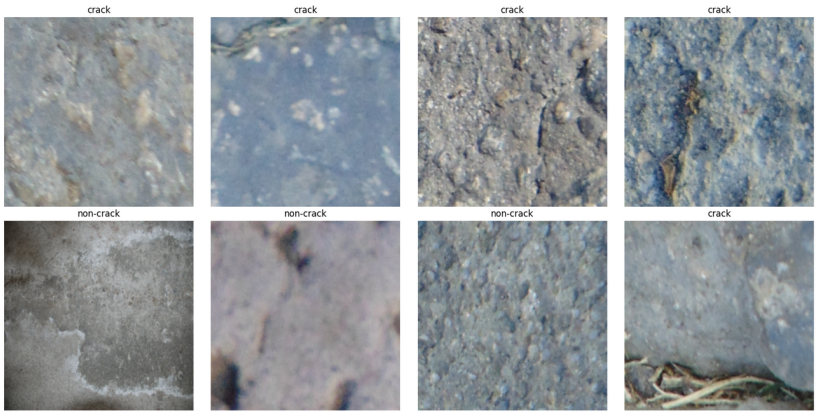

Figure 2: Extrait du dataset utilisé pour l'affinage

Le jeu de données a été augmenté en utilisant la normalisation. Cette augmentation a suffi à obtenir des résultats satisfaisants. Augmenter davantage le jeu de données n'a pas permis d'améliorer les performances, selon nos expérimentations. Les différentes méthodes d'augmentation testées étaient les suivantes : la rotation, la translation, la symétrie et la colorimétrie (changement de couleur + transformation en noir et blanc). À noter également que davantage de temps de calcul aurait permis de tester d'autres jeux de données plus grands.

4.2 Results

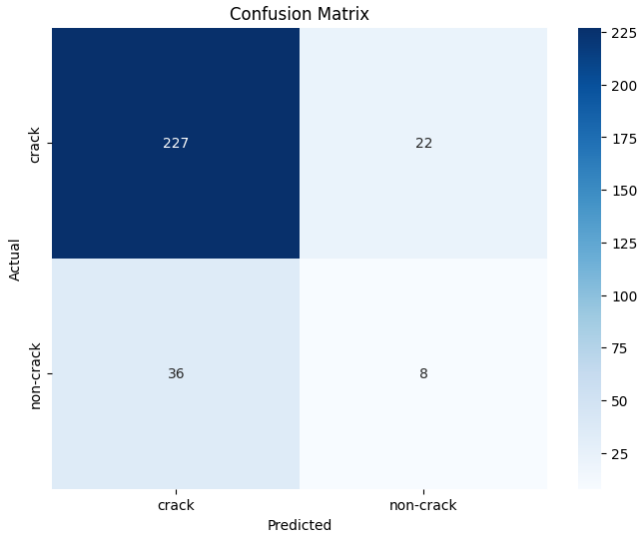

Figure 3: Matrice de confusion

Ce graphique pour notre classification binaire permet de tirer toutes les conclusions quant à notre implémentation. Nous constatons bien que la classe "non-crack", représentant les murs n'ayant pas de fissures, est très mal prédite par le modèle. Nous remarquons également que le modèle hésite équitablement entre les 36 faux positifs et les 22 faux négatifs : le modèle arrive bel et bien à identifier des caractéristiques différentes entre les classes mais n'arrivent pas à les classifier comme nous le voudrons.

Le jeu de données de test n'est pas équitablement répartie, il est composé à 85% de "crack" et de 15% de "non-crack". De ce fait il est intéressant de regarder la précision par classe : nous obtenons, avec ce modèle, une accuracy de 26% sur les "non-crack" et 86% sur les "crack". L'accuracy obtenue sur les "crack" est tout de même assez impressionnante, nous pouvons affirmer que la classification et le transfert de connaissance de ce côté s'est très bien déroulé.

Nous pouvons donc affirmer que la classification de fissures de murs représentent un challenge très difficile à classifier. L'architecture utilisé est pertinente mais peut-être pas autant que le dataset utilisé pour le fine-tuning. ResNet34 a été pré-entrainé de sorte à ce qu'il reconnaisse des coins ou autres formes géométriques, or les façades (de murs et de maisons) présentes dans le dataset de test a très certainement induit les prédictions du modèle en erreur.

4.3 Comparaison to others architectures

Nous avons expérimenté la classification sur deux algorithmes NN et CNN entrainé depuis le début avec ce même dataset et nous avons constater dans un premier temps une très grande difficulté de la part du NN pour la classification d'images avec une loss qui nécessite un grand nombre d'épochs pour diminuer et un énorme manque de connaissance de la part du CNN pour identifier des caractéristiques de fissures dans des images. Le CNN arrive a bien apprendre sur les données fourni, de l'ordre de 80 - 85% mais s'affole dès qu'une nouvelle donnée lui est présenté.

Ainsi, ResNet34, reste la meilleure architecture CNN a utilisé lorsque nous avons besoins de beaucoup de connaissances sur un domaine large et que nous avons également besoin d'affiner des poids.

5 Conclusion

En conclusion, nous avons montré ici, qu'en utilisant une architecture de réseaux de neurones convolutionnels basée sur des résidus, il est possible de faire un modèle assez général permettant de classifier un dataset encore inconnu. Nous avons utilisé les poids pré-entrainé du modèle pour transférer les connaissances vers ce que nous avions besoin et cela s'est traduit par une accuracy satisfaisante. Le rôle des paramètres joue également un rôle très important comme nous avons pu le voir dans nos expérimentations, ici les meilleurs d'entre eux vous sont montré.

Afin de pousser les résultats encore plus loin, une autre approche serait d'utiliser un jeu de données d'affinage bien plus grand ou encore d'introduire les transformers [9]. Cela nécessitera par contre beaucoup plus de puissances de calcul. La qualité des données est primordiale comme nous l'avons vu, notre modèle est capable d'apprendre sans problème de nouvelles données lorsqu'il existe des données d'entrainement en conséquences. Un dataset qui contiendrait un grand nombre d'images de fissures de murs (bien réparties), sur la même architecture, changerait drastiquement l'accuracy.

Enfin, tout ceci a été réalisé sur un ordinateur équipé de 32Gb de RAM ainsi que d'une carte graphique Nvidia P100 16Gb.

References

[1] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtarik. Sgd: General analysis and improved rates, 2019.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[5] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[6] Javier M. Moguerza and Alberto Muñoz. Rejoinder. *Statistical Science*, 21(3), August 2006.

[7] PyTorch. https://docs.pytorch.org/vision/main/models.html.

[8] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.