Détection et classification d'anomalies sur des images de panneaux solaires

Guillaume Quinquet
guillaume.quinquet@student-cs.fr
CentralesSupélec

Février 2021

Abstract

L'objectif de cette étude est de démontrer la faisabilité d'un système capable de détecter la présence d'anomalies et de dommages sur des panneaux photovoltaïques sur la base d'images. L'étude se décompose en trois parties : i) une exploration de l'état de l'art, ii) La réalisation d'une base de données exploitable, et iii) la réalisation et l'évaluation de différents modèles mettant en application différents concepts vus en cours. Nous tentons d'abord une première approche simple en utilisant une architecture convolutive préentrainée pour extraire des caractéristiques de nos images avant d'appliquer des méthodes de Machine Learning pour réaliser une classification de nos images par type d'anomalie. Nous utilisons ensuite des réseaux convolutifs suivis de couches denses afin de faire un décompte du nombre d'erreurs par image. Faces à l'intérêt pratique limité de telles approches, nous cherchons ensuite à détecter précisément le type et la localisation de chacun des défauts présents sur nos images. Pour cela, nous réglons différents modèles préentrainés d'architectures constituant l'état de l'art de la détection d'objet actuelle, notre choix étant malheureusement restreint par la quantité de mémoire des GPU à disposition.

1 Introduction

1.1 Contexte

Le domaine de l'exploitation photovoltaïque s'est formidablement développé ces dernières années en France et dans le monde suite à des politiques incitatives. Cette exploitation nécessite l'équipement de multiples terrains ensoleillés, qui sont généralement vastes et éloignés les uns des autres car situés dans différentes régions d'ensoleillement. De plus, cet équipement étant assez fragile et continuellement exposé à la nature et aux intempéries, les pannes sont fréquentes. Cependant, chaque panneau étant connecté à un même réseau, il n'est pas aisé de détecter la survenue et la localisation d'une panne. La maintenance de ces centrales photovoltaïques est donc malaisée, ce qui induit des pertes de revenu. Pour répondre à ce besoin, nous cherchons donc à mettre en place un protocole utilisant : i) des drones pour réaliser régulièrement une cartographie photographique des différentes centrales, et ii) du Deep Learning pour analyser les photographies et permettre une prise de décision de maintenance.

1.2 Problème

Le problème est donc le suivant : Nous disposons d'images aériennes (optiques et thermiques) de panneaux photovoltaïques pris en groupes dans des conditions variées, et figurant entre 0 et plusieurs défauts. Les dits défauts appartiennent à quatre catégories : Hotspot (surchauffe d'une cellule), Multihotspot (surchauffe de nombreuses cellules sur un même panneau), Hotspot Line (toute une ligne de cellules surchauffe), Ombrage (le panneau est ombragé, provoquant une baisse de productivité). Les images que nous avons à disposition ne sont pas labelisées, mais sont regroupées dans cinq dossiers correspondants à des panneaux sans défauts et à des panneaux présentant chacun des défauts susnommés. Cependant, comme nous allions nous en rendre compte, cette séparation est insuffisante car chacun de ces dossiers est composé d'images figurant de nombreux panneaux comportant souvent plusieurs défauts différents sur une même image. De plus, certaines erreurs figurent dans la catégorisation de la donnée de base.

Notre objectif est de trouver une méthode permettant de déclencher efficacement la décision de maintenance. Notre étude se décompose donc en trois étapes d'ambition croissante :

1. Dans un premier temps, nous cherchons à classifier les images comme contenant ou non des défauts. Nous tentons d'abord une approche non supervisée (clustering), puis expérimentons une approche de classification binaire et multiple.

2. Nous tentons ensuite d'inférer à partir de chaque

image le nombre de défauts présents. C'est donc un problème de régression.

3. Troisièmement, nous cherchons à détecter sur nos images chaque défaut, sa localisation et sa classe. C'est donc un problème de détection multiple.

Notre solution doit être capable de s'affranchir des conditions très variées de prise de photo. En effet, au delà des conditions météorologiques changeantes, le contrôle d'un drône n'est pas toujours aisé et ne permet pas la prise de photos homogènes. Par ailleurs, la solution doit aussi faire abstraction du contexte dans lequel les panneaux se trouvent, qui peut être varié. En effet, les centrales sont loin d'être centralisées et sont installées de façon variées.

2 État de l'art

Il existe une littérature assez importante en ce qui concerne les applications du Machine Learning et du Deep Learning à la détection de défauts de panneaux solaires. Cependant toutes les études que nous avons trouvé (e.g.) se concentrent sur la détection de défauts à partir de l'analyse des données numériques d'exploitation. Le défaut de cette approche est qu'elle permet au mieux d'identifier un array fautif de panneaux, mais pas directement le panneau comportant le défaut. En effet, ceux-ci sont installés en série et les données ne sont accessibles que par série. Les seuls travaux faits sur la base d'images se concentrent sur la détection des panneaux solaires dans des images satellite [2]. Nous avons trouvé un travail concernant la détection de défauts sur des panneaux solaires à partir d'images [3], dont les auteurs mettent même à disposition une base de données assez importante (+10000 images). Malheureusement, ce travail est réalisé dans des conditions très artificielles (prise de vues totalement identiques sur un panneau d'expérimentation, dégats et salissures réalisés à la main par les auteurs) qui ne permettent pas l'exploitation pour ce projet.

3 Réalisation d'une base de données exploitable

3.1 Donnée d'origine

Ayant le luxe de travailler avec un exploitant de centrales photovoltaïques et de disposer de drones équipés d'objectifs optiques et thermiques, nous avons pu obtenir 414 images de panneaux provenant de différentes centrales et pris sous différents angles. Chaque image comporte entre 20 et 150 panneaux dans des angles et des situations d'exposition variées. Elles ont ensuite été classées par l'exploitant dans 5 dossiers selon qu'elles comportaient ou non les défauts précédemment cités.

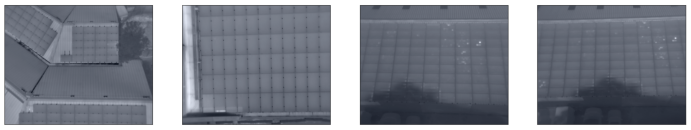

Figure 1: Quelques exemples des images reçues.

3.2 Labelisation de la donnée

Face à la complexité croissante de nos ambitions, nous choisissons de labeliser à la main 112 images à l'aide de la librairie open-source LabelImg [4], qui permet de tracer des cadres de délimitation associées à des labels sur des images, et de stocker ces informations de label dans des fichiers xml au format PASCAL VOC. Cette labelisation est réalisée sans l'aide d'expert et peut donc souffrir ses défauts.

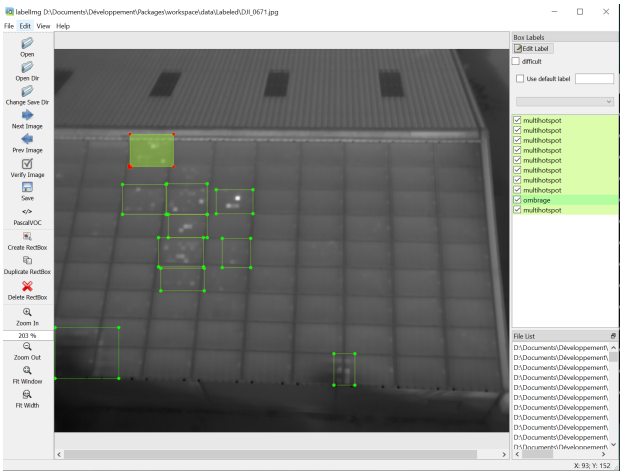

Figure 2: LabelImg nous a permis d'améliorer la précision des labels.

4 Première approche : clustering et classification par image

4.1 Protocole

Notre première expérience se décompose en quatre étapes :

1. Clustering binaire de la donnée pour tenter de séparer les images comportant des défauts
2. Classification binaire suivant le même objectif
3. Classification des images par type de défaut

Un ordinateur n'étant capable d'appréhender une image que comme une matrice de pixels, nous sommes face à un problème de complexité de taille : chaque échantillon comporte 640 x 500 soit 320 000 variables au minimum (en utilisant des images en échelles de gris) et 960 000 variables en RGB (les images sont donc représentées

comme des matrices de pixels de taille 640x500x3). Pour surmonter le fléau de la dimension, nous faisons le choix d'utiliser des réseaux convolutifs, qui constituent l'état de l'art dans le domaine de l'analyse automatique d'images. Ces réseaux permettent d'extraire différents concepts ou caractéristiques des images, et résument les images en termes de cartes d'activation de ces caractéristiques. Cela permet de fortement réduire la dimension des images, mais surtout d'en extraire des caractéristiques représentatives. Nous pouvons ensuite y appliquer un traitement soit par MLP, soit par diverses techniques de machine learning. Notre travail dans cette première expérimentation est donc découpé en deux parties : une phase d'extraction des caractéristiques, et une phase d'analyse de celles-ci.

4.1.1 Extraction des caractéristiques des images

Afin de capitaliser sur le travail de la communauté et d'avoir rapidement à disposition une architecture entraînée de haut niveau, nous faisons le choix de réutiliser une architecture VGG16 préentrainée mise à disposition par la librairie open-source Keras [5]. VGG16 est une architecture convolutive proposée par K. Simonyan et A. Zisserman [1] qui atteint une accuracy de 92.5% sur le jeu de données de référence ImageNet (1.2 millions d'images classées en 1000 catégories).

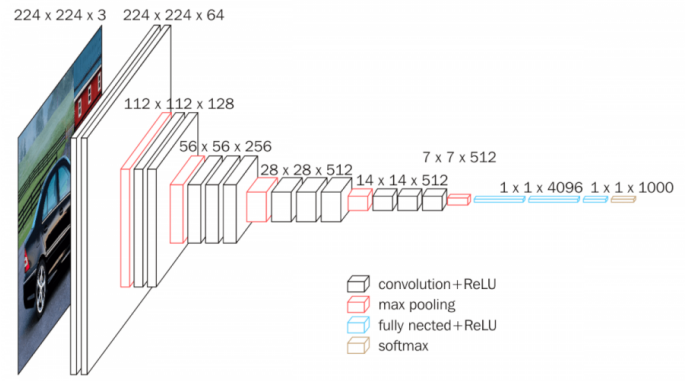

Figure 3: Visualisation de l'architecture VGG16 [6]

Plutôt que de tenter de réentrainer l'architecture, ce qui s'avèrerait trop lourd en termes de calcul, nous faisons le choix de réutiliser ses poids tels quels afin d'extraire les caractéristiques de nos images. Nous obtenons une sortie de taille 14x14x512 soit un vecteur de taille 100352 une fois aplatit.

4.1.2 Analyse des caractéristiques extraites

Pour chacune des trois sous-expériences précédemment citées, nous réutilisons les caractéristiques extraites à l'aide de VGG16, puis nous séparons ces caractéristiques et les labels associés en un jeu d'entraînement et un jeu de test. La validation des méthodes est faite par cross-validation en 5 folds sur le jeu d'entraînement.

4.1.2.1 Clustering binaire

Dans cette première expérience, nous cherchons à séparer de façon non supervisée les images comportant des défauts des autres. Pour cela, nous appliquons KMeans aux caractéristiques extraites par nos premières couches de VGG16, puis nous calculons le pourcentage d'images comportant des défauts dans chacun des deux clusters réalisés.

4.1.2.2 Classification binaire

Nous cherchons maintenant à mettre à profit le tri des images qui nous ont été fournies pour mettre en place une approche supervisée, avec le même objectif que précédemment. Nous utilisons donc comme labels une version binarisée des labels dont nous disposons : les classes correspondant à des défauts sont regroupées en un label qui s'oppose à la classe des panneaux sans défauts. Nous entraînons ensuite une forêt aléatoire dont nous recherchons les hyper-paramètres optimaux à l'aide d'une cross-validation avec recherche aléatoire. Nous évaluons ensuite la performance de cette approche en calculant l'accuracy sur les jeux de train et de test.

4.1.2.3 Classification multiple

Nous sommes maintenant plus ambitieux dans notre approche supervisée et cherchons à classifier les images par type de défaut. Nous appliquons le même protocole par forêt aléatoire que celui décrit précédemment, mais en utilisant cette fois-ci les 5 classes comme labels.

4.2 Réalisation, écueils et analyse des résultats

4.2.1 Clustering

Nous choisissons d'utiliser l'implémentation KMeans de scikit-learn, et obtenons après application de cette méthode deux clusters, l'un contenant 85% d'images comportant des défauts, l'autre 54%. Ces résultats ne sont pas concluants.

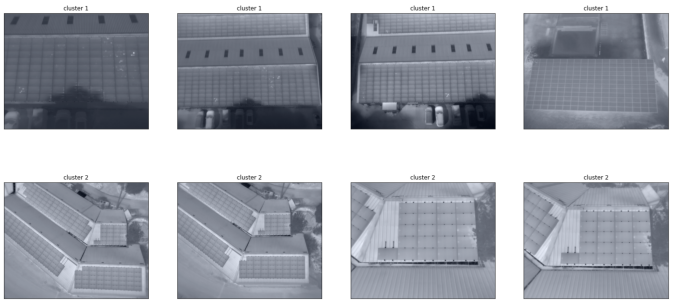

Figure 4: Échantillons du clustering

Une cause probable est que les images ont été prises dans des contextes et des orientations variées, qui peuvent être beaucoup plus différenciants que les défauts. Par exemple, toutes les images ayant un camion dans leur champ on été regroupées dans le premier cluster. Sans un travail important des images, il semble donc impossible en l'état d'appliquer avec succès une approche non supervisée.

4.2.2 Classification binaire

Nous utilisons la fonction RandomForestClassifier de scikit-learn pour l'implémentation de notre forêt aléatoire. Un premier entraînement sans réglage des hyper-paramètres donne des résultats à première vue beaucoup plus concluants : 99% d'accuracy en train et 75% en test. Cela est à mettre en regard du fait que les classes sont déséquilibrées : Nous avons 144 images sans défauts contre 270 images avec défauts soit 34% d'images sans défauts. Il serait donc possible d'obtenir un résultat similaire en test en prédisant que l'image contient des défauts dans 100% des cas. Cependant après vérification ce n'est pas le cas : Les classes prédites sont variées. Il semblerait que l'algorithme généralise bien.

4.2.3 Classification multiple

Sans surprise, l'application de cette méthode à l'ensemble des labels se révèle plus ardue : L'accuracy n'est plus que de 74% en train et 40% en test, ce qui n'est pas satisfaisant.

4.3 Conclusion intermédiaire

Cette approche a donné de meilleurs résultats que ce à quoi nous nous attendions. Notamment, elle a révélé qu'il était possible de classifier efficacement des images comme comportant ou non des défauts en utilisant une architecture VGG16 préentrainée sans réglage et en appliquant une classification par forêt aléatoire aux caractéristiques extraites. Cependant, cette approche montre ses limites lorsque l'on tente d'être plus précis et ne permet donc pas vraiment une application intéressante industriellement.

5 Seconde approche : Quantification du nombre de défauts par image

5.1 Protocole

Nous avons désormais à disposition des images avec des labels précis concernant les défauts individuels présents sur chaque image. Cela nous permet notamment de connaître le nombre de défauts présents dans chaque image. Nous allons donc chercher à construire un réseau de neurones capable de prédire à partir d'une image le nombre de défauts présents sur celle-ci. Pour cela nous :

1. Préparons le jeu de données en transformant les images en échelles de gris, analysant les fichiers xml des labels pour en déduire le nombre de défauts, puis en séparant ce jeu de données en un jeu d'entraînement et un jeu de validation.

2. Créons un pipeline d'augmentation des données à l'aide du module ImageDataGenerator de Tensorflow afin d'éviter le sur-apprentissage.

3. Construisons un réseau de neurones utilisant des couches convolutives pour l'extraction de caractéristiques et un MLP pour la régression du nombre de défauts.

4. Entraînons ce réseau.

5. Évaluons les résultats.

5.2 Réalisation, écueils et analyse des résultats

Lors de cette phase, après avoir trouvé une configuration dont la taille de filtres et la profondeur permettait un apprentissage du réseau sur le jeu de train, nous faisons face à un problème de sur-apprentissage.

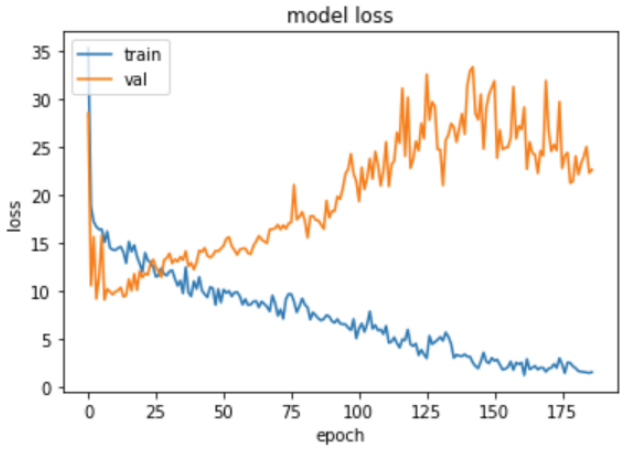

Figure 5: Le réseau rentre rapidement en surapprentissage.

Nous mettons en place un pipeline d'augmentation de la donnée opérant des rotations, des zooms et des variations d'éclairage pour contrer cela, ainsi qu'un certain nombre de couches de Dropout et de MaxPooling pour forcer le réseau à généraliser. Pour poursuivre dans cet axe et s'assurer que notre réseau de neurones est optimal pour le cas général, nous configurons un early stopping arrêtant l'entrainement lorsque la rmse de validation de notre réseau ne baisse plus pendant plus de 50 époques.

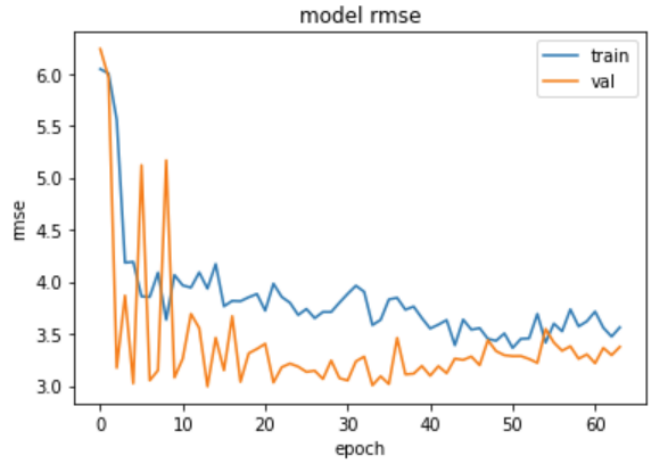

Figure 6: Le surapprentissage est corrigé, mais la performance globale en pâtit.

Cependant, nous ne parvenons pas à obtenir une rmse de moins de 3. Sachant que le nombre de défauts par image varie entre 0 et 15, cela constitue une erreur considérable.

6 Approche finale : détection des défauts individuels sur chaque image

6.1 Protocole

Face à la difficulté d'obtenir des résultats précis et exploitables avec un réseau entraîné de zéro avec si peu d'échantillons, nous faisons le choix de réutiliser un réseau de détection d'objet avec cadre de délimitation préentrainé, et de le régler sur notre jeu de données. Pour cela nous :

1. Obtenons les poids d'un modèle de détection d'objets avec cadre de délimitation préentrainé.
2. Préparons le jeu de données en le transformant en deux fichiers binarisés au format *tfrecord*, l'un pour le train et l'autre pour le test. Nous suivons pour cela les instructions fournies par Tensorflow.
3. Configurons l'application d'entraînement de façon qu'elle utilise les poids de notre modèle préentrainé et paramétrons le pipeline d'augmentation d'image intégré, ainsi que la taille des batches et le nombre de classes à détecter.
4. Ré-entraînons le réseau sur notre jeu de données.
5. Évaluons les résultats en observant la capacité prédictive du réseau sur le jeu de test.

6.2 Réalisation, écueils et analyse des résultats

Lors de cette dernière phase, nous décidons de totalement nous consacrer au réglage d'un réseau préentrainé qui gérerait notre problème de bout en bout. Nous utilisons pour cela l'API Tensorflow Object Detection [7], qui nous met à disposition des modèles préentrainés et des modules de chargement et d'entraînement. Nous nous inspirons pour cela du tutoriel mis à disposition par le site ReadtheDocs.io [8]. Ces modèles étant entraînés en utilisant des labelmaps et des fichiers binarisés, un grosse partie du travail consiste à trouver comment formaliser la donnée de façon adaptée. Nous cherchons ensuite à sélectionner des modèles présentant un bon compromis entre légèreté et performance de façon à pouvoir les entraîner avec les ressources à notre disposition (12Go de ram, deux processeurs et un GPU Tesla K80 gracieusement mis à disposition par Google Colab). Les modèles que nous sélectionnons finalement sont MobileNetV1, ResNet50 V1 et Faster R-CNN Inception ResNet V2. Nous modifions les fichiers de configuration de façon à adapter le réseau à notre cas et notre architecture de fichier, puis nous entrainons les réseaux. En utilisant les images à leur taille d'origine, nous ne

réussissions qu'à entraîner les réseaux MobileNetV1 et ResNet50 V1. Faster R-CNN Inception ResNet V2 surcharge la mémoire graphique et provoque des erreurs de mémoire.
Nous choisissons de façon arbitraire de stopper l'entraînement une fois la loss descendue en dessous de 1. Les résultats sont très satisfaisants pour MobileNet V1 : le réseau détecte avec précision les erreurs sur les images de test.

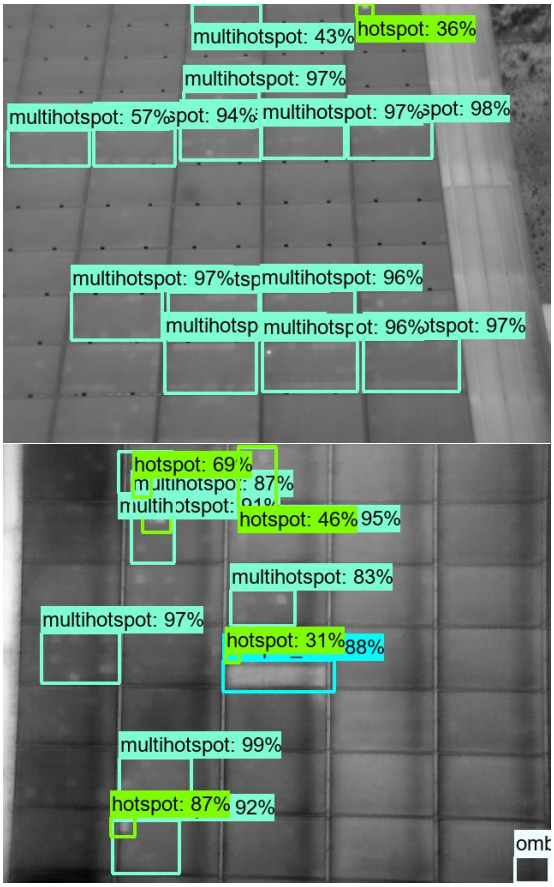

Figure 7: Les résultats en phase de test sont très convaincants.

7 Conclusion

Nous avons donc réussi à obtenir des résultats satisfaisants et pouvant véritablement avoir un intérêt pour le domaine en remplaçant la traditionnelle inspection de maintenance manuelle. L'approche retenue est donc une approche de réglage fin d'un réseau de détection d'objets préentrainé. Le principal enseignement que nous tirons de ce travail est l'efficacité du réglage fin de réseaux de neurones de référence : nous avons ici réussi à entraîner en moins d'une heure un réseau de détection d'objets efficace pour notre problème. Cela nous montre la grande qualité et généralité des caractéristiques extraites par ces réseaux. D'autre part, nous avons appris l'importance primordiale de la qualité d'un jeu de données : avant notre étape de labelisation précise de la donnée, toutes nos tentatives plus complexe qu'une simple classification binaire semblaient vouées à l'échec.

References

[1] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014

[2] Projet DeepSolar
`http://web.stanford.edu/group/deepsolar/home`

[3] Projet DeepSolarEye
`https://deep-solar-eye.github.io/posts/blog/`

[4] Tzutalin, labelimg
`https://github.com/tzutalin/labelImg`

[5] Keras Applications API:
`https://keras.io/api/applications/`

[6] source de l'image :
`https://neurohive.io/en/popular-networks/vgg16/`

[7] Repository github
`https://github.com/tensorflow/models/tree/master/research/object_detection`

[8] Tutoriel Tensorflow Object Detection API
`https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/`