POLYTECH TOURS
Département
Aménagement et Environnement

citeres
UMR 7324

université de TOURS

Projet de Fin d'Études (PFE) 2022-2023

Détermination du potentiel solaire en milieu urbain

Mindjid Maizia (sous la direction de)

Alexandre Bouzaires

AVERTISSEMENT

Cette recherche a fait appel à des lectures, enquêtes et interviews. Tout emprunt à des contenus d'interviews, des écrits autres que strictement personnel, toute reproduction et citation, font systématiquement l'objet d'un référencement.

L'auteur (les auteurs) de cette recherche a (ont) signé une attestation sur l'honneur de non-plagiat.

Formation par la recherche, Projet de Fin d'Etudes en génie de l'Aménagement et de l'Environnement

La formation au génie de l'aménagement et de l'environnement, assurée par le département aménagement et environnement de l'Ecole Polytechnique de l'Université de Tours, associe dans le champ de l'urbanisme, de l'aménagement des espaces fortement à faiblement anthropisés, l'acquisition de connaissances fondamentales, l'acquisition de techniques et de savoir-faire, la formation à la pratique professionnelle et la formation par la recherche. Cette dernière ne vise pas à former les seuls futurs élèves désireux de prolonger leur formation par les études doctorales, mais tout en ouvrant à cette voie, elle vise tout d'abord à favoriser la capacité des futurs ingénieurs à :

- Accroître leurs compétences en matière de pratique professionnelle par la mobilisation de connaissances et de techniques, dont les fondements et contenus ont été explorés le plus finement possible afin d'en assurer une bonne maîtrise intellectuelle et pratique,
- Accroître la capacité des ingénieurs en génie de l'aménagement et de l'environnement à innover tant en matière de méthodes que d'outils, mobilisables pour affronter et résoudre les problèmes complexes posés par l'organisation et la gestion des espaces.

La formation par la recherche inclut un exercice individuel de recherche, le projet de fin d'études (P.F.E.), situé en dernière année de formation des élèves ingénieurs. Cet exercice correspond à un stage d'une durée minimum de trois mois, en laboratoire de recherche, principalement au sein de l'équipe Dynamiques et Actions Territoriales et Environnementales de l'UMR 7324 CITERES à laquelle appartiennent les enseignants-chercheurs du département aménagement.

Le travail de recherche, dont l'objectif de base est d'acquérir une compétence méthodologique en matière de recherche, doit répondre à l'un des deux grands objectifs :

- Développer toute ou partie d'une méthode ou d'un outil nouveau permettant le traitement innovant d'un problème d'aménagement
- Approfondir les connaissances de base pour mieux affronter une question complexe en matière d'aménagement.

Afin de valoriser ce travail de recherche nous avons décidé de mettre en ligne sur la base du Système Universitaire de Documentation (SUDOC), les mémoires à partir de la mention bien.

REMERCIEMENTS

Je tiens à remercier les personnes qui m'ont soutenu au cours de mes études et de ce projet.

Dans un premier temps je voudrais remercier mon directeur de projet de fin d'étude monsieur Mindjid Maizia, enseignant chercheur à l'université de Tours. Sa disponibilité, ses conseils et le suivi qu'il a exercé sur l'ensemble du projet m'ont permis d'arriver à mes objectifs.

Je remercie également monsieur Paul Vola, avec lequel nous avons aborder le même sujet avec des approches différentes. Ses conseils et sa vision du projet m'ont permis de me remettre en question et d'avancer.

Je suis également reconnaissant à monsieur Luka Chavoit, qui m'a aidé lors de ce projet et plus généralement sur l'ensemble de mes études.

Merci à mes amis de Polytech Tours et d'ailleurs, mademoiselle Mary Paryseck, monsieur Dorian Rumeau, monsieur Arthur Villeneuve et bien d'autres qui mériteraient également d'être cités.

Enfin, je remercie ma famille qui m'a permis de réaliser les études que je souhaitais en m'apportant un soutien indéfectible durant toutes ces années.

Table des matières

Table des figures

1 Introduction

1.1 Estimer les gisements d'énergies renouvelables afin de développer leurs modes de production

Le Ministère de la transition écologique et de la Cohésion des territoires et ministère de la Transition énergétique fixe pour objectif une production d'électricité provenant à 40% des énergies renouvelables à l'horizon 2030. (Anne, 2022) Pour ce faire, il est primordial de déterminer les ressources énergétiques renouvelables afin de développer leurs modes de production. Or, il est difficile de quantifier ces ressources, leurs disponibilités étant inégales dans le temps et dans l'espace. Il est donc important de mettre au point des technologies capables d'effectuer cette tâche. Cela passe par l'amélioration des techniques existantes. Les données nécessaires à leurs fonctionnements (données 3D, 2D, relevés météorologiques...) ainsi que le temps d'exécution de ces méthodes sont des critères indicateurs de leurs performances.

1.2 Un réseau neuronal convolutif, une technologie rapide permettant de déterminer le potentiel solaire urbain

Nous allons vérifier si l'utilisation d'un réseau neuronal convolutif permet de quantifier d'une part plus rapidement et d'autre part, avec moins de données de type géométrique le potentiel solaire urbain de la ville de Tours que ne le ferait une méthode par ray-tracing.

1.3 Définitions

1.3.1 Ray-Tracing

Le Ray-Tracing est une technique de rendu basé sur un rayon lumineux avec des propriétés géométriques transportant une certaine quantité d'énergie. Ce rayon vient rencontrer les différents objets 3D constituant la scène et imite le comportement d'un rayon lumineux réel. (Jakica, 2018)

1.3.2 Deep-learning

En 1959, Arthur Samuel définit le « machine-learning » comme un champ d'étude qui donne la capacité au programme d'apprendre sans être explicitement codé pour. Le Deep-Learning applique le même principe, à la différence que le deep-learning apprend lui-même à générer une partie de ses données contrairement au machine-learning qui doit être pourvu en données présélectionnées. (Chassagnon et al., 2019)

1.3.3 Potentiel solaire

Le potentiel solaire est la quantité de radiation par unité de surface reçu par une surface donnée. (Huang et al., 2019) Cette définition peut être complétée par une quantité de radiations par unité de surface pour une surface horizontale (Zhong et al., 2021) Nous prendrons ici, comme définition du potentiel solaire, l'ensemble du rayonnement solaire parvenant sur une surface horizontale au cours d'une année.

2 Le Ray-Tracing et le deep learning dans l'estimation dupotentiel solaire urbain

2.1 Le Ray-Tracing, un modèle historique dans la simulation de la distribution de la lumière

Le Ray-Tracing est une technique de rendu basé sur la projection de rayons aux propriétés géométriques sur des éléments en 3D. Cette technologie est déployée pour la première fois en 1986 dans le milieu cinématographique. Elle est aussi majoritairement utilisée pour simuler les effetsde lumières dans des jeux vidéo et dans le domaine du cinéma. Elle est dérivée d'un algorithme de jetée de rayons. (Jakica, 2018) Cet algorithme a pour but de simuler les interactions entre un rayon lumineux et une surface. Cependant, cette méthode n'est capable d'assigner qu'un unique rayon par pixel, alors que dans la réalité un petit élément de surface (assimilé à un pixel ou un groupement de pixels) peut être frappé par plusieurs rayons. De plus, cet algorithme ne prend pas en compte les phénomènes de réflexion et réfraction de la lumière. A l'inverse, le Ray-Tracing prend en compte ces deux paramètres : un pixel peut recevoir plusieurs rayons et la réflexion ainsi que la réfraction sont bien prises en compte. L'ajout de ces deux paramètres fait du Ray-Tracing la seule solution informatique basée sur une réalité physique.

3 Catégories de Ray-Tracing se distinguent : le mode « Forward », « Backward » et « bidirectionnel ».

1. Le Ray-Tracing « Forward » simule un rayon de lumière étant émis de la source (Figure 1). C'est le modèle « idéal » de Ray-Tracing car il est le plus proche du phénomène physique réel. Cependant, c'est aussi le plus coûteux en ressource informatique, le modèle devant tracer un nombre de rayons supposés infinis pour arriver à un résultat net. (Jakica, 2018)
2. Le Ray-Tracing « Backward » quant à lui simule les rayons lumineux depuis la surface de l'objet éclairé (Figure 1). Cette simulation est bien plus rapide que son homologue « Forward » car il ne trace que les rayons étant certains d'atteindre un élément de surface et revenant à l'observateur. Toutefois, ce modèle est moins fiable quant à la reproduction des phénomènes de réflexion et réfraction et reste peu fiable avec des sources de lumières complexes ou la reproduction d'ombres. (Jakica, 2018)
3. Enfin, le Ray-Tracing « bidirectionnel » est la combinaison des deux méthodes précédentes. Une partie des rayons tracés proviennent de la source lumineuse vers l'objet, et l'autre part de la surface de l'objet vers la source lumineuse. Ce modèle est utilisable dans des situations complexes, avec un coût informatique correct. Il souffre tout de même de problème de «bruit », avec des rayons inexistants tracés. C'est pour cela que la méthode de Metropolis Light Transport (MLT) vient souvent accompagner le Ray-Racing « bidirectionnel » : elle assigne à l'objet lumineux une valeur finie d'énergie, que cet objet redistribue ensuite parmi les rayons émis. (Jakica, 2018)

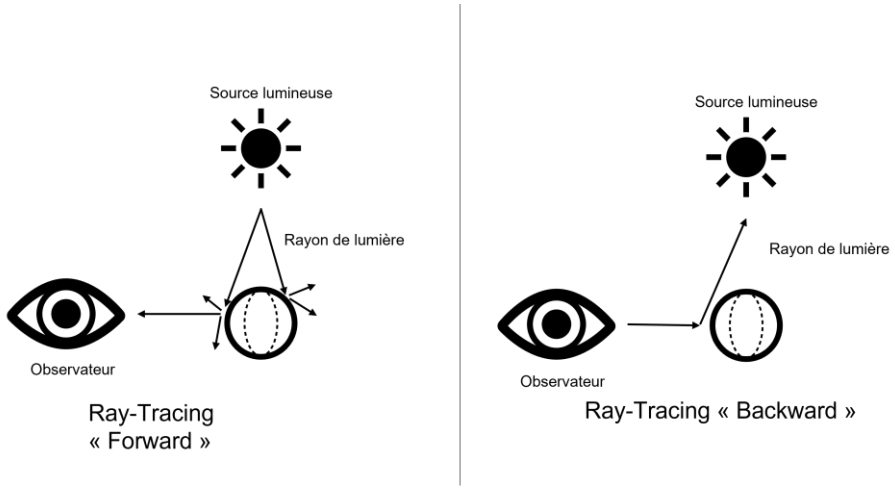


Figure 1 – Ray-Tracing "Forward" et "Backward". (Bouzaires, 2022)

Le Ray-Tracing « Forward » est utilisable pour simuler le comportement de la lumière en milieu urbain, comme l'ont démontré Manni et.al en 2020. Dans cet article, cette technologie couplée à une méthode statistique de Monte-Carlo est utilisée pour estimer le potentiel solaire dans des canyons urbains. Cette méthode statistique est répandue dans les études ayant pour objectif de déterminer les probabilités des résultats selon différents scénarios. Dans l'étude de Mani et.al, elle vient décider du nombre de réflexion qu'un unique rayon va effectuer avant de s'arrêter (entre 0 et 3 réflexions dans l'étude).

2.1.1 Une technologie rigide quant aux données d'entrées

Le Ray-Tracing n'est applicable qu'avec un contenu 3D. (Jakica, 2018) La géométrie en 3D est en général sous deux formes : un objet 3D représenté comme un ensemble de surfaces ou comme un volume. L'utilisation d'une représentation par surface permet d'obtenir une meilleure précision par l'intermédiaire du NURBS (Non-Uniforme Rational Bezier Splines) qui permet un haut niveau de précision sur les propriétés de l'élément. Les premiers modèles de Ray-Tracing ne sont pas compatibles avec la technologie NURBS, seuls les programmes récents le sont.

2.1.2 La vitesse : un problème inhérent au Ray-Tracing

Cependant, ce procédé est lourd avec un temps d'exécution important et nécessite une grande quantité de mémoire. (Jakica, 2018) En effet, la simulation du phénomène de réfraction et de réflexion engendre une grande complexité informatique, le rayon étant susceptible de ne s'arrêter qu'après de nombreuses réflexions. C'est pour cela que cette supposée réalité physique n'est pas atteinte : le rayon n'est plus réfléchi après un nombre fixé par des limites techniques. Cela explique l'utilisation de la méthode statistique de Monte-Carlo par Mani et.al qui rend le traitement informatique moins lourd et diminue ainsi le coût temporel.

Dans les années 80, la production du rendu prenait des heures voir des jours. (Jakica, 2018) L'application du Ray-Tracing passe alors par l'utilisation du CPU (Central Processor

Unit). Ce composant connaît jusqu'en 2016 une croissance sur le nombre de transistors sur la puce selon la Loi de Moore : tous les deux ans, le nombre de transistors sur la puce double. Ainsi, la cadence des CPU est de plus en plus élevée et permet une réduction du temps d'exécution du Ray-Tracing. Mais en 2005, les CPU atteignent une limite théorique sur la cadence de 4GHz pour cause de surchauffe. Afin de suivre la Loi de Moore, il est décidé de mettre les CPU en parallèle. Cependant cette solution est problématique pour le Ray-Tracing, l'investissement monétaire n'étant plus proportionnel à l'augmentation de la vitesse.

Les GPU (Graphics Processing Unit) connaissent à peu près à la même période un développement similaire. Les technologies de Ray-Tracing s'adaptent alors et deviennent utilisables parles GPU, profitant des progrès dans le domaine. Mais un autre seuil de vitesse est atteint à cause d'un problème de mémoire accessible par les composants.

Malgré l'amélioration du matériel informatique, la problématique de la vitesse reste un enjeu important quant au perfectionnement de la technologie du Ray-Tracing.

2.2 Les Réseaux Neuronaux Convolutifs, un outil qui a déjà fait ses preuves.

2.2.1 Un modèle adapté au traitement d'images.

Un réseau de neurones artificiels (RNA) est une méthode de deep learning ou de machine learning inspiré du fonctionnement du cerveau humain. Cet algorithme imite l'apprentissage en associant à un objet en entrée, une solution en sortie. Un enfant associe le mot « rouge » à la couleur rouge car un élément extérieur le lui a indiqué et il crée ensuite sa propre règle ou en emprunte une pour comprendre pourquoi. Un réseau de neurones artificiels fait la même chose. Il se présente sous la forme d'une couche d'entrée (input layer), de couches cachées (hidden layers) et d'une couche de sortie (output layer). Chaque couche est un vecteur. Si un modèle ne comporte qu'une seule hidden layer on parle d'un réseau de neurones simple (Figure 2), autrement on parle d'un réseau de neurones profond. (Dastre, 2021)

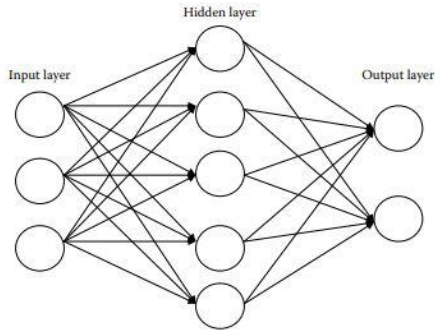


Figure 2 – Structure basique d'un réseau de neurones artificiels simple.
(Balamurugan, 2022)

Chaque couche et composée d'un à plusieurs éléments dont le fonctionnement est assimilable à celui d'un neurone. Une couche N est intégralement connectée à sa couche N-1, on parle de couches totalement connectées (fully-connected layers). Un neurone est constitué de 2 éléments : une fonction d'activation qui transforme l'information arrivant depuis la couche N-1 et un poids qui définit l'importance du résultat du neurone pour la couche N+1. (Figure 3) La connexion entre 2 neurones est régie par un seuil : si

la valeur du neurone à la couche N-1 ne remplit pas la condition du seuil, la liaison est inactive, autrement la liaison est effective. (Dastre, 2021)

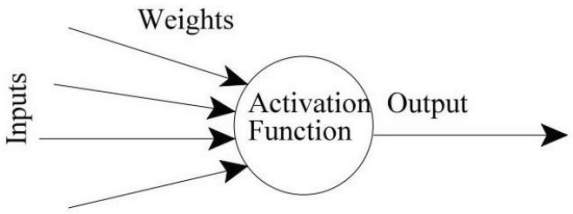


Figure 3. Weight of each element and input and output of the ANN system.

Figure 3 – Fonctionnement d'un neurone. (Dastre, 2021)

Dans la phase d'apprentissage, la donnée se présente en input layer et passe par des hidden layers avec des poids aléatoires. Le résultat produit en output layer est donc, à la première itération, aléatoire. Or, le principe de la phase d'apprentissage et de modifier le modèle selon le résultat que l'on cherche à obtenir. S'ensuit alors la phase inverse dite de « backward propagation » : si le résultat obtenu n'est pas le bon, le poids des neurones est modifié à chaque itération jusqu'à obtenir l'output attendue. (Dastre, 2021) Un des modèle dit MLP (Multilayer Percepetron) fait partie des algorithmes les plus utilisés appartenant à la classe des RNA. (Balamurugan, 2022)

Un réseau neuronal convolutif (Convolutional Neuronal Network ou CNN) prend en général comme input une matrice 3 dimensions représentant une image. Il reprend la structure précédemment évoquée en ajoutant deux couches : des couches convolutives (convolution layers) et des couches de groupement (pooling layers). (Yamashita et.al, 2019) (Figure 4)

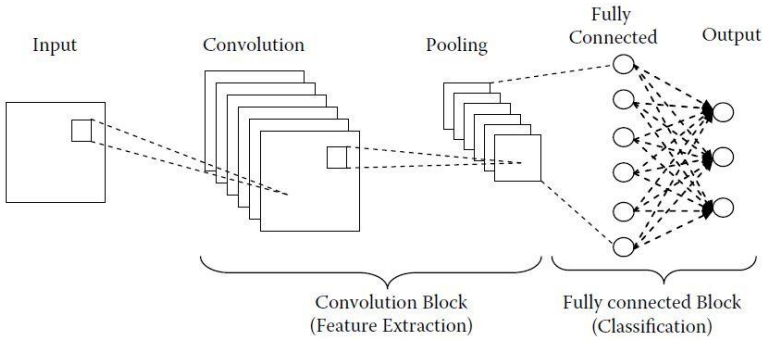


Figure 4 – Structure d'un réseau neuronal convolutif. (Balamurugan, 2022)

Une couche convolutive agit comme un filtre : un « kernel » de taille à définir parcourt l'image et extrait des éléments que l'algorithme juge important (des bordures d'objets par exemple). Cette étape constitue une première réduction du format des images. (Figure 5) Ces informations sont ensuite concaténées par la couche de pooling (Figure 6) afin de réduire une fois de plus le format de l'image en couche N-1. (Yamashita et.al, 2019)

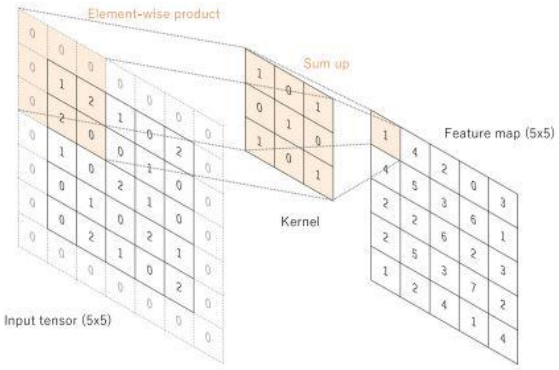


Figure 5 – Schéma du fonctionnement d'une couche convolutive.
(Yamashita et.al, 2019)

Après des convolutions et pooling successifs, il ne reste que les informations jugées importantes pour le modèle. Il faut ensuite connecter une couche de pooling à une couche dense qui va diminuer la dimension des éléments des images en un vecteur. Le reste du procédé est identique à celui d'un RNA. (Yamashita et.al, 2019)

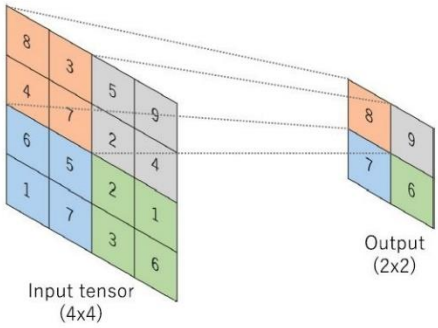


Figure 6 – Schéma du fonctionnement d'une couche de MaxPooling.
(Yamashita et.al, 2019)

Pour la classification d'image, un réseau CNN est plus adapté qu'un modèle MLP. En effet, un CNN peut traiter une image dans son format original (3D) et ainsi garder la répartition spatiale des éléments là ou un MLP nécessite « d'aplatir » l'image en passant de 3 dimensions à une seule. Il est aussi réputé comme convergeant plus vite et prenant moins de ressources informatiques qu'un MLP. (Balamurugan, 2022)

2.2.2 Application à un milieu urbain

Une méthode similaire de réseau neuronal convolutif est appliquée par Zong et.al pour prédire le potentiel solaire urbain de la ville de Nankin en Chine. Elle est ici utilisée dans un exercice de segmentation. L'objectif est d'isoler les surfaces de toitures pouvant accueillir des panneaux solaires en partant de photographies satellites de la ville.

L'étude se base sur des photographies satellites, elles constituent donc la donnée de départ pour le logiciel. Elles sont ici issues de Google Earth et font 256x256 pixels.

L'objectif étant de sélectionner les surfaces de toiture présentes sur les photographies satellites,il faut créer une base de données adéquate au modèle. Cette action s'effectue en 2 étapes.

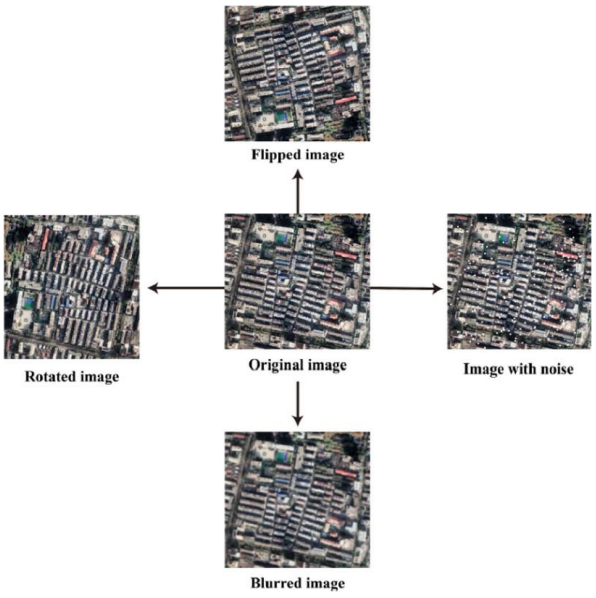
L'image issue de Google Earth (Figure 7, 1.) est d'abord traitée par ArcGIS (Figure 7, 2.) afin de marquer sur cette même image les surfaces de toitures. Un prélèvement

manuel est alors effectué pour obtenir une image « labeled » (Figure 7, 3.) qui correspond à l'image finale. L'image
« labeled » est ensuite associée à son échantillon image issue de Google Earth d'origine.

Figure 7 – Exemple de création de données. (Zhong et.al, 2021)

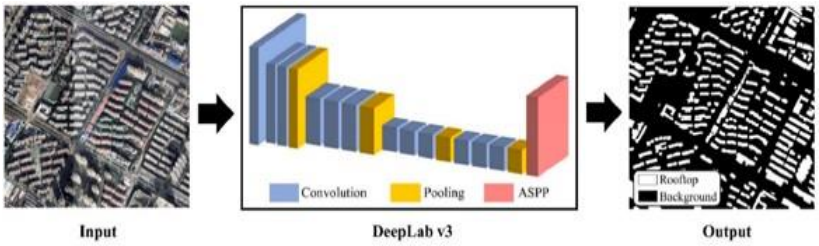
Cependant, l' « entraînement » d'un modèle de machine-learning ou de deep-learning demande beaucoup de données et la ville de Nankin étant un espace délimité la question du manque de données se pose. Pour pallier ce problème, chaque image Google Earth subit 4 opérations : l'image est tournée (à 90°), retournée, floutée et du « bruit » est ajouté. Ces actions permettent de dupliquer les données en entrées tout en prenant en compte des possibles imperfections des photographies satellites. Le bruit simule un manque ou une perturbation de l'image, ici symbolisé par les points blancs sur l'image. Ces images sont ensuite traitées selon la démarche précédente. (Figure 8)

Figure 8 – Modification de l'image Google Earth. (Zhong et.al, 2021)



Ces données sont ensuite rentrées dans le logiciel DeepLab v3 selon la méthode du « small-batch gradient descent method ». Cette méthode a ici pour but d'utiliser une partie de chaque échantillon pour mettre à jour des paramètres pendant l'apprentissage. Ainsi, le nombre de répétitions pour obtenir un modèle abouti est diminué, un même élément d'entrée étant utilisé plusieurs fois dans le processus d'apprentissage contrairement à d'autres méthodes. Le modèle est alors capable de déterminer les surfaces de toiture. (Figure 9).

Figure 9 – Résultat attendu de l'"entraînement". (Zhong et.al, 2021)



La vérification de la fiabilité et de la précision du modèle se base aussi sur la matrice de confusion M_{conf} :

$$M_{conf} = \begin{pmatrix} Vrai\ Positif\ (VP) & Faux\ Négatif\ (FN) \\ Faux\ Positif\ (FP) & Vrai\ Négatif\ (VN) \end{pmatrix}$$

Afin de mieux interpréter les résultats suivants et dans le cadre de cet article, les valeurs de la matrice ont été ramenées entre 0 et 1. Ainsi, il est possible de déterminer la fiabilité, la précision,le retour et le score FI :

La fiabilité (F) est le pourcentage de surface de toit et d'arrière-plan (« background » Figure 4) correctement extrait par le modèle sur la surface de toit et d'arrière-plan total.

$$F = \frac{VP + VN}{VP + VN + FP + FN}$$

La précision (P) se définit comme le pourcentage d'une surface de toit correctement extraite sur le total de la surface de toit extraite.

$$P = \frac{VP}{VP + FP}$$

Le retour (R) correspond au pourcentage d'une surface de toit correctement extraite sur lasurface totale réelle de toit.

$$F = \frac{VP}{VP + FN}$$

Enfin, le score F1 (F1) varie entre 0 et 1 et représente une moyenne pondérée entre la précision et le retour. Plus cette valeur est proche de 1, meilleur le modèle est. Un score F1 de 50% signifie qu'en moyenne pour un Vrai (Positif ou Négatif) on obtient deux Faux (Positif ou Négatif).

$$F = \frac{2 \times P \times R}{P + R}$$

En fonction des critères imposés par le cahier des charges pour chaque variable ci-dessus, le modèle est validé ou non.

3 Méthode de comparaison du Ray-Tracing et d'un réseau neuronal convolutif dans l'exercice de détermination du potentiel solaire urbain

3.1 Mise en place de l'approche Ray-Tracing

Comme expliqué en section [2.1], le Ray-Tracing consiste à projeter des rayons sur des objets géométriques. Ces éléments sont représentés par les bâtiments présents sur la ville de Tours. Ces données proviennent d'IGN France. Elles sont ensuite exportées dans Google SketchUp par l'intermédiaire de ToasterIntegral. Le type de donnée est géométrique (3D). Afin de simuler le potentiel solaire annuel, nous créons un diagramme solaire représentant la trajectoire du soleil sur l'année dans le ciel de Tours. Ce diagramme dépend de la latitude du lieu d'étude, ici la latitude est de 47,3833°. Il est dimensionné plus grand que l'espace urbain tourangeau afin de répartir les rayons de manière égale sur l'ensemble de la ville.

Une fois le rendu effectué par un Ray-Tracing bidirectionnel couplé à une technologie similaire à du Métro Light Transport sur une vue aérienne d'un ensemble de bâtiments, nous obtenons une image reprenant la distribution du potentiel solaire sur cet échantillon. (Figure 10) Le calcul de ces images par Ray-Tracing prend en moyenne 1h. Ce temps varie en fonction de la complexité du lieu sur lequel s'effectue le rendu : si beaucoup de bâtiments différents sont présents, le temps de rendu augmente.

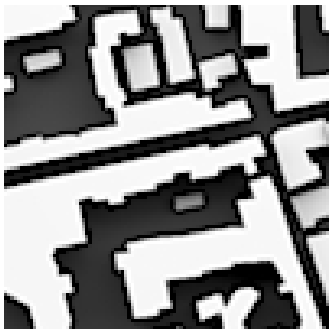


Figure 10 – Application du Ray-Tracing sur une parcelle de la ville de Tours. (Bouzaires, 2022)

Afin de connaître la quantité de rayonnement solaire sur l'ensemble de l'image, il faut appliquer un algorithme déterminant la quantité présente sur chaque pixel, puis sommer la valeur associée à chaque pixel. Nous supposons que l'information du potentiel contenu dans un pixel évolue linéairement par rapport à sa valeur en niveau de gris.

P_p correspond au potentiel de chaque pixel. L'image étant en niveau de gris, la C_p varie entre 0 et 255. Selon l'outil PVGIS crée par la commission européenne, la valeur annuelle du potentiel solaire à Tours en kW/m2 égale à 1502 kWh/m2/an. (European Commission, 2016)

$$P_p = \frac{C_p}{255} \times 1502$$

$P_i \ en \ kWh/m^2/an$ est la somme du potentiel de chaque pixel divisé par le nombre de pixels afin d'obtenir le potentiel moyen de l'image.

$$P_i = \frac{\sum_{p=1}^{10000} P_p}{10000}$$

Ainsi, nous sommes capables d'associer à chaque vue satellite de la ville de Tours un potentiel solaire annuel. En effectuant un rendu sur une zone situé en centre-ville de Tours de la taille 1500 pixels par 600 pixels ensuite séparé ensuite en 90 images de 100 pixels par 100 pixels, nous trouvons pour un potentiel solaire moyen de 696 kWh/m2/an avec un potentiel maximal de 995 kWh/m2/an et un potentiel minimal de 542 kWh/m2/an.

3.2 Réseau neuronal convolutif : une architecture adaptable à chaque problème

3.2.1 Base de l'apprentissage : création de la base de données.

La démarche suivante se base sur la création d'un modèle selon un « supervised machine learning » ou « apprentissage supervisé », les images qui lui sont fournies sont déjà labellisées : à chaque input est au préalable associé un output. (Touzet, 1992)

Afin d'entraîner un CNN, il faut constituer une base de données. Une base de données pour entraîner ce type de modèle est décomposé en 3 « set » différents : un set d'entraînement, un set de validation et un set de test ou de prédiction. Le set d'entraînement est le set contenant le plus de données, c'est ce dernier qui va permettre au modèle de s'adapter au problème. Le set de validation assure à chaque itération que le modèle « s'entraîne » bien en prenant une image de ce set et vérifiant la valeur qu'il produit en la comparant à celle qu'il est supposé trouver, le résultat de cette comparaison influence ensuite l'entraînement. Même s'il est fourni au modèle, ce set est « caché » à l'algorithme car ce dernier ne mémorise pas les inputs et output présents. Enfin le set de test ou de prédiction permet de s'assurer que le modèle se généralise bien sur des données inconnues. Il est préférable de ne pas tester le modèle sur le set de validation car ses données ont influencé l'apprentissage du modèle.

Nous utilisons ici un CNN dans un exercice de classification : nous associons à chaque image une classe. Le site d'étude est la ville de Tours. Comme vu en section 2.2.1, un CNN travaille avec des images en entrée. Ces données sont générées à partir de « photographies » aériennes des entités shapefile représentant la ville de Tours importé sur Google SketchUp. Chaque bâtiment est représenté par un volume dont la couleur dépend de la hauteur. Plus la construction est haute, plus sa couleur est rouge. Une image satellite en format 1500 pixels par 600 pixels RGB est ensuite produite. (Figure 11) L'idée de cette démarche de coloration des bâtiments et de permettre au modèle de créer plus facilement la règle définissant le potentiel solaire sur une image en milieu urbain en augmentant l'information disponible dans une image.

Ces images en format 1500x600 pixels sont ensuite décomposées en images de format 100x100 pixels RGB également. Chaque couche est divisée par 255 afin d'obtenir des valeurs RGB comprises entre 0 et 1, les RNA et CNN interprétant mal les entrées dont la valeur absolue est supérieure à 1. L'apprentissage nécessite une quantité importante de données (Yamashita et.al, 2019) et la ville de Tours ne dispose que d'un nombre fini d'images à fournir au modèle. Nous devons donc procéder à l'augmentation des données. Des opérations faciles à mettre en place telles que des rotations de 90°, 180° et 270° remplissent cette tâche. Ainsi, le set d'entraînement comporte 2 200 images, celui de validation 45 et enfin celui de test est constitué de 90 images.

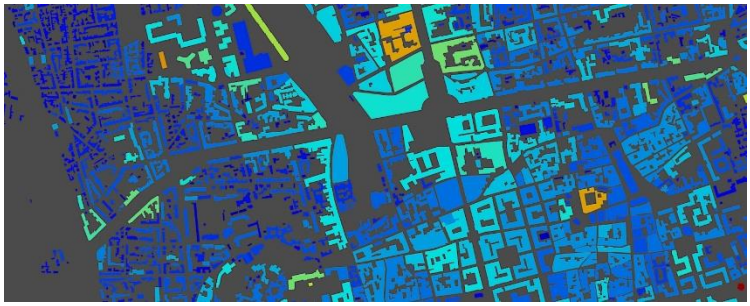


Figure 11 – Exemple d'une image input avant séparation. (Bouzaires, 2022)

Les résultats obtenus par le protocole en section 3.1 servent à construire nos données en sortie. Nous associons à chaque image la classe de potentiel correspondant. Mais cette valeur ne peut être la sortie finale de notre algorithme car si à chaque image est associé un potentiel unique, le modèle serait incapable de prédire correctement la valeur d'une image ne provenant pas du set d'entraînement. En effet, il y aurait autant de classes que d'images, traduisant un apprentissage complexe, avec une « règle » par image, et peu généralisable. Pour pallier ce problème, nous devons créer des classes. Les bornes de ces classes sont définies par un algorithme prenant en entrée les valeurs du potentiel obtenu depuis les images du set d'entraînement et le nombre de classe désiré. Les valeurs minimale et maximale constituent les bornes limites et forment deux classes : une classe inférieure au minimum et une classe supérieure au maximum. Le reste des classes est défini en suivant un pas (en kWh/m²/an) calculé selon le minimum, maximum et le nombre de classe souhaité (en excluant les bornes définis précédemment) par la formule suivante :

$$Pas = \frac{(\max(P_i - \min(P_i))}{n}$$

Nous prenons ici n=3 afin d'obtenir 5 classes. On incrémente ensuite ce pas depuis la valeur minimale jusqu'à atteindre la valeur maximale. Nous classons ensuite chaque image dans les intervalles formés par les bornes. Ainsi, ayant réduit les valeurs en sortie, nous nous plaçons alors dans un cas de classification d'images. Nos classes varient entre 0 et 4, et ces dernières sont associées à une plage de potentiel solaire moyen :

Classe 0 $\leq 542\ kWh/m^2/an$
Classe 1 $\in [542\ ;693]\ kWh/m^2/an$
Classe 2 $\in [693\ ;844]\ kWh/m^2/an$
Classe 3 $\in [844\ ;995]\ kWh/m^2/an$
Classe 4 $\geq 995\ kWh/m^2/an$

Le même protocole est suivi pour créer le set de validation et de prédiction, exception

faite de la création des bornes : les bornes doivent rester invariantes afin que le modèle associe les images à une sortie qu'il est capable de reconnaître. La proportion des catégories d'images dans chaque set est similaire, les espaces urbains étudiés étant proches spatialement et morphologiquement.

3.2.2 Choix d'une architecture et des paramètres

L'ensemble du code nécessaire à ce projet est fait sur Google Colab, un environnement basé sur le langage de programmation Python. Colab est hébergé sur le notebook Jupyter, ce qui permet d'accéder à une unité de traitement graphique (GPU) en ligne. Le modèle est produit à l'aide de la bibliothèque Keras, qui fait lien avec des algorithmes de RNA et CNN provenant de TensorFlow.

Le réseau neuronal convolutif est un modèle séquentiel (Sequential), adapté au traitement de couches successives (couches RGB) et base d'un CNN. Il est complété par une alternance de 4 couches de convolution et de MaxPooling. Le MaxPooling est une couche de pooling ne gardant que la valeur maximale afin de ne conserver que les paramètres les plus importants. La fonction d'activation de ces couches est la fonction Unité Linéaire Rectifiée (Rectified Linear Unit ou reLu) qui renvoi un 0 si la valeur d'entrée est négative, autrement elle produit la valeur d'entrée. On définit x comme un réel compris entre 0 et 1 et on désigne par f(x) la fonction reLu. (Ling et.al, 2018)

$$f(x) = \begin{cases} x \; si \; x > 0 \\ 0, sinon \end{cases}$$

Facile à calculer avec un temps de calcul moindre, comparé aux autres fonctions d'activations (source reLu), elle est souvent utilisée en paramètre des CNN, notamment par Zhang et.al au cours de leur étude. Enfin, nous connectons ces éléments à 2 couches denses de 256 neurones puis de 5 neurones. A l'image nous choisissons une fonction d'activation reLu pour la couche en N-1 de l'output, pour les mêmes raisons qu'évoquées précédemment, et une fonction exponentielle normalisée ou softmax pour la couche produisant l'output. Cette dernière produit l'exponentielle de la somme de ses entrées, le tout divisé par la somme des exponentielles de ses entrées. On définit z = (z1, …, zK) de K nombres réels avec la fonction softmax notée $\sigma(z)$.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_j}} \; pour \; tout \; j \in \{1, \dots, K\}$$

Cette dernière est la couche la plus utilisée et souvent la plus efficace pour produire la sortie des CNN : elle synthétise les valeurs des couches précédentes pour créer la sortie. (Cardarilli et.al 2021)

3.3 Evaluation des performances du réseau neuronal convolutif face au Ray-Tracing

3.3.1 Interprétation des résultats du réseau neuronal convolutif

Un CNN reste une approche statistique d'un problème de classification. Ainsi, chaque modèle est différent et ce malgré les mêmes sets d'entraînement et de validation. Sur les différentes itérations de l'architecture de modèle présentées précédemment seul

le modèle le plus performant est retenu.

La prédiction se fait sur le set de test : le modèle dispose en entrée d'images qu'il ne connaît pas et définit de lui-même leurs classes. L'opération prend 163 ms. Il faut désormais vérifier les performances du modèle. Pour se faire nous constituons la matrice de confusion de l'échantillon de prédiction.

Nous appelons cette matrice Mc :

$$M_c = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 3 & 9 & 2 & 0 \\ 0 & 0 & 10 & 22 & 3 \\ 0 & 0 & 0 & 20 & 9 \\ 0 & 0 & 0 & 2 & 8 \end{pmatrix}$$

Ce type de matrice se compose de la sorte : en ligne les prédictions du modèle, en colonne les résultats réels. Les éléments sur la diagonale sont les images bien prédites, le reste représente les erreurs de classification. Nous obtenons 41 images bien classées contre 49 classées dans la mauvaise catégorie, soit 45,56% de prédictions correctes.

Nous pouvons rentrer dans le détail dans chacune des classes en utilisant les formules présentées en section 2.2.2. En généralisant la matrice de confusion de l'étude de Zhang et.al à une matrice carrée de dimensions supérieures à 2 (i, j tel que i=j et i>2), nous pouvons calculer VP, FN, FP et VN pour une classe k :

VP_k = Valeur présente sur la diagonale à la ligne k, i_k, j_k.

FN_k = Somme des valeurs présentes sur la ligne k, la valeur en position i_k, j_k exclue.

FP_k = Somme des valeurs présentes sur la colonne k, la valeur en position i_k j_k exclue.

VN_k = Somme des valeurs de la matrices, valeurs sur la ligne i_k et sur la colonne j_k exclue.

Nous en déduisons les caractéristiques du modèle pour chaque classe (Figure 12) :

	Fiabilité F (%)	Précision P (%)	Retour R (%)	F1 score (%)
Classe 0	97,8	0,0	0,0	Non définit
Classe 1	85,6	75,0	20,0	31,6
Classe 2	62,2	28,6	52,6	37,0
Classe 3	61,1	69,0	43,5	53,3
Classe 4	84,4	80,0	40,0	53,3

Figure 12 - Evaluation multicritère de la performance du modèle.
(Bouzaires, 2022)

Dans notre cas, la fiabilité indique le nombre d'images bien classifiées sur l'intégralité des prédictions. La précision représente le rapport entre les prédictions correctes dans une classe sur l'ensemble des prédictions de cette classe. Le rappel est le rapport entre les prédictions correctes et le nombre réel d'images dans la classe.

Les résultats produits pour la classe 0 sont peu interprétables, elle constitue la classe minoritaire du set d'entraînement (1 seule image) et est également minoritaire dans les autres sets.

Quant aux autres classes, nous observons un déséquilibre entre la fiabilité et le F1-score, cet écart est le plus marqué pour la classe 1. Au regard de la structure de nos sets, il est plus intéressant de se fier au F1-score. En effet la proportion des classes n'est pas négligeable pour l'interprétation des résultats : la distribution n'est pas égale. Les classes 0 et 1 sont peu représentées dans l'échantillon de test, mais également dans les autres

sets. Cela s'explique par le choix du lieu d'étude : la ville de Tours est un espace urbain dense avec peu de zones non urbanisées. Ainsi, il est plus probable de tomber sur des espaces avec un potentiel élevé que l'inverse. Quant à la classe 2, la fiabilité est moyenne avec un F1-score faible malgré une bonne représentation dans les sets. En observant la matrice de confusion, nous pouvons voir que le modèle a tendance à « surclasser » les images : sur les 35 images prédites en classe 2, 25 appartiennent en réalité à une classe supérieure. Cette tendance s'efface pour les classes 3 et 4, où le F1-score augmente : l'algorithme surclasse peu la catégorie 3 et il ne peut pas dépasser une classe 4. En prenant en compte l'évolution précédemment décrite, nous pouvons affirmer que le modèle est plus performant sur des images à potentiel élevé et très élevé, performances se dégradant plus le potentiel diminue.

3.3.2 Un modèle peu performant : le problème du surapprentissage

Avec une précision globale de 45,56% sur l'ensemble du set d'entraînement et un F1-score ne dépassant pas 53,3%, il convient de s'interroger sur la ou les causes de ces résultats. Le problème dont souffre le modèle est une situation souvent rencontrée par les CNN : le surapprentissage ou « overfitting ». (Narasinga et.al, 2018) Le modèle mémorise les données du set d'entraînement, parvenant à produire les résultats attendus, mais n'apprend pas réellement : il ne parvient pas à généraliser. Ainsi une fois confrontée à un set inconnu il ne parvient pas à correctement classifier les éléments. Cette tendance est observable dès la phase d'entraînement. (Ying, 2019)

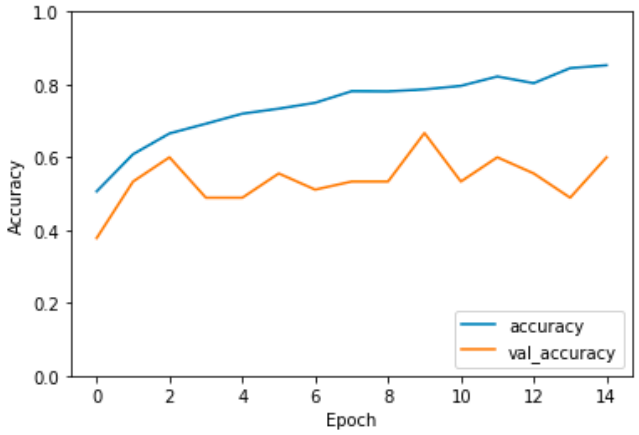


Figure 13 - Résumé de la phase d'apprentissage du modèle. (Bouzaires, 2022)

On observe dès le troisième epoch (à voir comme une itération) un décrochage entre la courbe d'accuracy et de val_accuracy. (Figure 13) L'accuracy est la performance du modèle sur le set d'entraînement et la val_accuracy sa performance sur le set de validation. Dès lors, la val_accuracy croit et décroit entre 0,49 et 0,67 (0,54 en moyenne) alors que l'accuracy augmente (à l'exception d'une légère baisse à l'epoch n°12). Le modèle apprend donc bien sur le set 1 mais ne parvient pas à apprendre davantage sur le set de validation. Une faible performance sur le set de validation se répercute sur le set de test et les données inconnues en général.

Plusieurs causes peuvent expliquer le surapprentissage, le manque de données d'entraînement ou avec du « bruit » (altération des pixels) venant dégrader la qualité des images. Cependant, à l'issue du protocole présenté en section 2.2.2 aucune image n'est concernée par l'apparition de « bruit ». Le mauvais équilibre des classes dans le set d'entraînement est aussi un élément dégradant la val_accuracy et menant à un

overfitting. Une architecture de modèle trop complexe par rapport au problème traité peut aussi être à la base de ce phénomène. Dès lors, le l'algorithme « créer » des hypothèses non-généralisables à des images inconnues afin d'augmenter sa précision sur le set d'entraînement. (Ying, 2019) Dans notre cas, la diminution ou l'augmentation de la complexité de l'architecture du modèle ne traduit qu'une baisse de ses performances sur le set d'entraînement : l'architecture n'est pas à mettre en cause. Ici, l'origine du surapprentissage provient donc d'un set d'entraînement trop peu fournit.

Afin de lutter contre cette tendance, plusieurs solutions sont applicables. La plus fréquente est l'augmentation des données du set d'entraînement. (Ying, 2019)) Toutefois, l'intégralité des images satellite de la ville de Tours est déjà répartie dans les différents sets. Malgré les opérations de rotations présentées en section 3.2.1, le set d'entraînement reste visiblement trop limité pour empêcher le surapprentissage.

La modification des hyperparamètres peut aussi diminuer l'impact de ce phénomène. La mise en place de couches de « Dropout » est un des facteurs limitant le surapprentissage. À chaque itération, cette couche désactive certains neurones et ses connexions aléatoirement. (Figure 14) Chaque neurone a la même probabilité d'être désactivé à chaque phase d'entraînement. L'intérêt est de perturber l'apprentissage pour empêcher le modèle de développer des hypothèses trop complexes entre chaque couche. Le modèle est contraint à exploiter ses neurones plus individuellement que dans un modèle totalement connecté, chaque neurone pouvant se trouver isolé de ses voisins. (Srivastava, 2014)

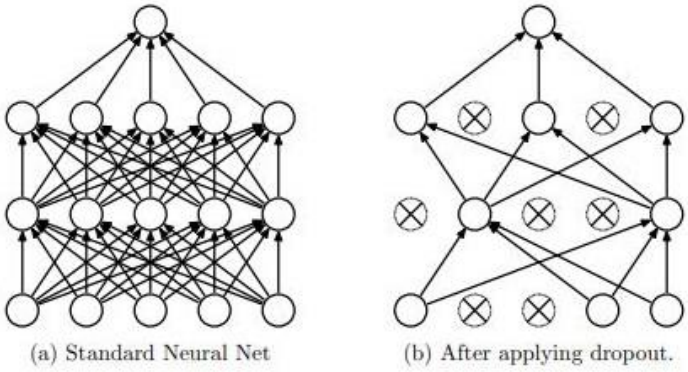


Figure 14 – Couches denses intégralement connectées (a) et couches denses avec Dropout (b). (Srivastava, 2014)

Une autre protection mise en place est l'application d'une régularisation sur l'apprentissage des neurones. (Ying, 2019) Pour empêcher chaque neurone d'avoir un poids trop élevé (pouvant mener à un surapprentissage) une fonction de régularisation contrôle leur croissance. Ici nous prenons la fonction l2-regularization. Cette dernière ajoute la somme des poids provenant de la couche N-1 au carré, le tout multiplié par un facteur alpha variant entre 0 et 1 au poids actuel. En cas de détection d'une croissance trop élevée d'un neurone sur une itération, le facteur αpondère cette augmentation en diminuant la valeur de la somme ($\alpha \approx 0$). (Demir-Kavuk, 2011)

On définit p comme le nombre de connexions provenant de la couche N-1 du neurone.

$$régulisationL2 = fonction\ d'activation + \ \alpha \sum_{j=1}^{p} {b_j}^2$$

Cependant, les opérations décrites ci-dessus ne viennent, dans notre cas, pas à bout du surapprentissage. Le modèle présenté en section 3.3.1 est déjà paramétré avec des couches de Dropout et la l2-regularization mais il présente encore le problème

d'overfitting. Ces protections permettent tout de même d'obtenir des performances supérieures aux modèles produits sans ces hyperparamètres.

La modification des hyperparamètres ne suffisent donc pas à compenser le manque de données d'entraînement et la distribution inégale des classes.

4 Conclusion

À la suite de cette étude, nous avons démontré que l'utilisation d'un réseau neuronal convolutif pour déterminer le potentiel solaire urbain présente un intérêt non négligeable au regard du gain de temps que cette méthode fournie comparé à une méthode de Ray-Tracing. Pour déterminer le potentiel de 90 images de 100 pixels par 100 pixels, nous passons d'une moyenne de 1h à 216 millisecondes. De plus, le modèle travaille avec des images aériennes de la ville de Tours, là où le modèle de Ray-Tracing utilisé nécessite d'avoir des données 3D.

Cependant, au-delà de ce paramètre, les performances du modèle sont peu convaincantes : une fiabilité de 45,5% n'est pas suffisante pour que les résultats du modèle justifient son utilisation dans une étude. De plus, ce dernier n'est pas capable de calculer avec précision le potentiel solaire moyen d'une image, il peut seulement l'approximer en le classant.

Néanmoins, il reste intéressant de poursuivre les études d'application d'un CNN à ce type de problème. En effet nous sommes parvenus à déterminer les problèmes à la base des performances du modèle présenté dans ce document. En augmentant la quantité des données pour l'entraînement et en équilibrant la distribution des classes dans chaque set, il est raisonnable de penser que le modèle puisse atteindre une fiabilité supérieure.

Il convient toutefois de préciser que la nature des méthodes que nous comparons sont différentes. Le Ray-Tracing est basé sur une simulation informatique de la physique de la lumière alors qu'un réseau neuronal convolutif est une approche statistique.

Enfin, l'utilisation du Ray-Tracing dans la création du réseau neuronal convolutif vient biaiser la comparaison effectuée, un modèle se basant sur l'autre. Nous n'avons pas abordé ici de point de comparaison entre le Ray-Tracing et la réalité.

Bibliographie

(1) Anne, B. (2022, September 20). Les énergies renouvelables. Ministères Écologie Énergie Territoires. https://www.ecologie.gouv.fr/energies-renouvelables

(2) Balamurugan, S. (2022). A Comprehensive Study on MLP and CNN, and the Implementation of Multi-Class Image Classification using Deep CNN. Machine Learning and Deep Learning Techniques for Medical Science, 1–25. https://doi.org/10.1201/9781003217497-1

(3) Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Nannarelli, A., Re, M., & Spanò, S. (2021). A pseudo-softmax function for hardware-based high speed image classification. Scientific Reports, 11(1). https://doi.org/10.1038/s41598-021-94691-7

(4) Chassagnon, G., Vakalopolou, M., Paragios, N., & Revel, M. P. (2019). Deep learning: definition and perspectives for thoracic imaging. European Radiology, 30(4), 2021–2030. https://doi.org/10.1007/s00330-019-06564-3

(5) Dastres, R. (2021, September 20). Artificial Neural Network Systems. Archive Ouverte HAL. https://hal.archives-ouvertes.fr/hal-03349542

(6) Demir-Kavuk, O., Kamada, M., Akutsu, T., & Knapp, E. W. (2011). Prediction using step-wise L1, L2 regularization and feature selection for small data sets with large number of features. BMC Bioinformatics, 12(1). https://doi.org/10.1186/1471-2105-12-412

(7) Huang, Z., Mendis, T., & Xu, S. (2019). Urban solar utilization potential mapping via deep learning technology: A case study of Wuhan, China. Applied Energy, 250, 283–291. https://doi.org/10.1016/j.apenergy.2019.04.113

(8) IGN : produire et diffuser les données géographiques et forestières en France - Portail IGN - IGN. (n.d.). https://www.ign.fr/

(9) Jakica, N. (2018). State-of-the-art review of solar design tools and methods for assessing daylighting and solar potential for building-integrated photovoltaics. Renewable and Sustainable Energy Reviews, 81, 1296–1328. https://doi.org/10.1016/j.rser.2017.05.080

(10) JRC Photovoltaic Geographical Information System (PVGIS) - European Commission. (2016, January 11). https://re.jrc.ec.europa.eu/pvg_tools/en/

(11) Lin, G., & Shen, W. (2018). Research on convolutional neural network based on improved Relu piecewise activation function. Procedia Computer Science, 131, 977–984. https://doi.org/10.1016/j.procs.2018.04.239

(12) Manni, M., Bonamente, E., Lobaccaro, G., Goia, F., Nicolini, A., Bozonnet, E., & Rossi, F. (2020). Development and validation of a Monte Carlo-based numerical model for solar analyses in urban canyon configurations. Building and Environment, 170, 106638. https://doi.org/10.1016/j.buildenv.2019.106638

(13) M.R.Narasinga Rao, D., Venkatesh Prasad, V., Sai Teja, P., Zindavali, M., & Phanindra Reddy, O. (2018). A Survey on Prevention of Overfitting in Convolution Neural Networks Using Machine Learning Techniques. International Journal of Engineering &Amp; Technology, 7(2.32), 177. https://doi.org/10.14419/ijet.v7i2.32.15399

(14) Srivastava, N. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Toronto.Edu. https://jmlr.org/papers/v15/srivastava14a.html

(15) Touzet, C. (1992). LES RESEAUX DE NEURONES ARTIFICIELS, INTRODUCTION AU CONNEXIONNISME. Aix Marseille Université. https://hal-amu.archives-ouvertes.fr/hal-01338010

(16) Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. Insights Into Imaging, 9(4), 611–629. https://doi.org/10.1007/s13244-018-0639-9

(17) Ying, X. (2019). An Overview of Overfitting and its Solutions. Journal of Physics: Conference Series, 1168, 022022. https://doi.org/10.1088/1742-6596/1168/2/022022

(18) Zhong, T., Zhang, Z., Chen, M., Zhang, K., Zhou, Z., Zhu, R., Wang, Y., Lü, G., & Yan, J. (2021). A city-scale estimation of rooftop solar photovoltaic potential based on deep learning. Applied Energy, 298, 117132. https://doi.org/10.1016/j.apenergy.2021.117132

Annexes

Code Python :

```
# -*- coding: utf-8 -*-
"""CNN propre.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1JKr8OzmGl4IRgMq9U8fvJyIdbG0rGONh
"""

#Faire le lien entre Drive et Colab, le '/content/gdrive/' représente la base du chemin d'accès aux données.

from google.colab import drive
drive.mount('/content/gdrive/', force_remount=True)

"""Tout d'abord nous procédons par importer l'ensemble des données d'entrées. Ici des images codées en RGB représentant une vue satellite de la ville de Tours avec à chaque hauteur de bâtiment une couleur associée. Plus la construction est haute plus la couleur passe du bleu au rouge.
Les images proviennent de différents répertoires. Nous important donc répertoire par répertoire.
"""

#Fonction afin d'assigner une image à une position dans la liste en fonction de son nom, ainsi l'image 00 se retrouve en position 0 de la liste.

def lire_indice_image(filename):
  i = 0
  indice = 0
  while (filename[-i-5] in ['0','1','2','3','4','5','6','7','8','9']):
    indice = int(filename[-i-5])*(10**i) + indice
    i = i + 1
  return indice

import os
import shutil
import glob
import numpy as np
import PIL
from PIL import Image
import os, sys
from scipy.io import loadmat
import matplotlib.pyplot as plt

# Création de la fonction permettant d'importer les images depuis un répertoire en fonction de leurs nombres, le chemin d'accès (ici le drive,
# à changer selon l'utilisateur) et la taille de l'image. On suppose ici que toutes les images sont carrées.

def load_data_x_train(nombre_images, data_path, image_size=100):
```

```
    x = np.zeros((nombre_images, image_size, image_size, 3))    #Ici, nous créons un 4D tableau constitué de 0 aux dimensions du nombre d'images,
# la taille et le format RGB
    # Chargement des images,
    for filename in glob.glob(data_path):
      # Ouverture de l'image
      img = Image.open(filename, 'r')
      # Conversion de l'image en RGB
      img = img.convert('RGB')
      # Ecriture dans la variable de retour x à l'emplacement dédié
      x[lire_indice_image(filename)] = np.asarray(img)
    return x

#Nous appliquons la définition à notre premier répertoire.

x_train = load_data_x_train(90, '/content/gdrive/MyDrive/PFE/InputTours100/*.png')
plt.imshow(x_train[5]/255)  #Vérification que la 5 ème image est bien importée et correspond bien à l'image n°5 du répertoire.

# Nous générons plusieurs images à partir de celle de base à partir d'opérations simples : rotation et rotation.
# Comme les répertoires sont susceptible de contenir un quantité d'image variable il est nécessaire de renseigner cette information.

def Transpose(x, nombre_images):
  x_t = np.zeros((nombre_images,100,100,3))  #Création d'un tableau de 0 pour accueillir les images transformées
  for i in range(nombre_images):
    x_t[i]=np.flip(x[i], axis=0)

  return(x_t)

def Rotation(x, nombre_images):
  x_t = np.zeros((nombre_images,100,100,3))
  for i in range(nombre_images):
    x_t[i] = np.flip(x[i], axis = 1)

  return(x_t)

# Application des transformations au premier set.

x_transpose = Transpose(x_train, 90)
x_180 = Rotation(x_train, 90)
x_90 = Rotation(x_transpose, 90)

# x_tot permet de récupérer les différentes opérations sur les images d'entrées. C'est cette variable qui va servir à entrainer le modèle
# il va récupérer l'ensemble des données d'entraînement et est réutilisé sur chaque répertoire.

x_tot = np.vstack((x_train, x_transpose, x_90, x_180))

# On répète les opérations précédentes sur chaque répertoire. On ne récupère pas toutes les données dans le répertoire de validation
# la moitié va servir à entrainer le modèle.
```

```
x_train_val = load_data_x_train(90,
'/content/gdrive/MyDrive/PFE/InputTours100Valid/*.png')
x_val_split1 = x_train_val[0:45]
x_val_split2 = x_train_val[45:90]

x_val_split_transpose = Transpose(x_val_split1, 45)
x_val_split_180 = Rotation(x_val_split1, 45)
x_val_split_90 = Transpose(x_val_split_180, 45)
x_tot = np.vstack((x_tot, x_val_split1, x_val_split_transpose, x_val_split_180,
x_val_split_90))

x_data = load_data_x_train(90, '/content/gdrive/MyDrive/PFE/InputTours100Data/*.png')
x_data_transpose = Transpose(x_data, 90)
x_data_180 = Rotation(x_data, 90)
x_data_90 = Transpose(x_data_180, 90)
x_tot = np.vstack((x_tot, x_data, x_data_transpose, x_data_180, x_data_90))

x_data2 = load_data_x_train(90, '/content/gdrive/MyDrive/PFE/InputTours100Data2/*.png')
x_data2_transpose = Transpose(x_data2, 90)
x_data2_180 = Rotation(x_data2, 90)
x_data2_90 = Transpose(x_data2_180, 90)
x_tot = np.vstack((x_tot, x_data2, x_data2_transpose, x_data2_180, x_data2_90))

x_1 = load_data_x_train(66, '/content/gdrive/MyDrive/PFE/X1/*.png')
x_1_transpose = Transpose(x_1, 66)
x_1_180 = Rotation(x_1, 66)
x_1_90 = Transpose(x_1_180, 66)
x_tot = np.vstack((x_tot, x_1, x_1_transpose, x_1_180, x_1_90))

x_2 = load_data_x_train(60, '/content/gdrive/MyDrive/PFE/X2/*.png')
x_2_transpose = Transpose(x_2, 60)
x_2_180 = Rotation(x_2, 60)
x_2_90 = Transpose(x_2_180, 60)
x_tot = np.vstack((x_tot, x_2, x_2_transpose, x_2_180, x_2_90))

x_3 = load_data_x_train(18, '/content/gdrive/MyDrive/PFE/X3/*.png')
x_3_transpose = Transpose(x_3, 18)
x_3_180 = Rotation(x_3, 18)
x_3_90 = Transpose(x_3_180, 18)
x_tot = np.vstack((x_tot, x_3, x_3_transpose, x_3_180, x_3_90))

x_4 = load_data_x_train(91, '/content/gdrive/MyDrive/PFE/X4/*.png')
x_4_transpose = Transpose(x_4, 91)
x_4_180 = Rotation(x_4, 91)
x_4_90 = Transpose(x_4_180, 91)
x_tot = np.vstack((x_tot, x_4, x_4_transpose, x_4_180, x_4_90))

#Enfin nous chargeons le set de test, ici de 90 images.
 x_pred = load_data_x_train(90, '/content/gdrive/MyDrive/PFE/InputTours100Pred/*.png')

"""Nous devons désormais importer les données de sortie, ce qui correspond à un scalaire
représentant une classe. Ces classes sont constituées en fonction du potentiel solaire contenue
dans chaque image.
Tout d'abord, nous prenons les images produites par le Ray-tracing puis calculons le potentiel
```

solaire de ces images.
Nous définissons ensuite nos classes en fonction du premier jeu du premier répertoire. Ainsi nous sommes susceptible d'obtenir plus d'image en classe 0 qu'en prenant le set total d'entraînement pour créer nos bornes et ainsi avoir une meilleure distribution des classes (il n'y a pas d'images vides dans le premier répertoire).
"""

```
def load_data_y_train(nombre_images, data_path, image_size=100):

    y = np.zeros((nombre_images, image_size, image_size))

    # Chargement des images,
    for filename in glob.glob(data_path):
      # Ouverture de l'image
      img = Image.open(filename, 'r')
      # Conversion de l'image en niveaux de gris
      img = img.convert('L')
      # Ecriture dans la variable de retour x
      y[lire_indice_image(filename)] = np.asarray(img)

    return y

# Nous importons le premier set.
y_train = load_data_y_train(90, '/content/gdrive/MyDrive/PFE/OutputTours100/*.png')
plt.imshow(y_train[5]/255, cmap="gray")

#Déterminer le potentiel de chacune des images du potentiel solaire par Ray-Tracing
def Potentiel(nb_images, y_train):

  Potentiel_Images = np.zeros((nb_images))  #Liste du potentiel solaire de toutes les images

  for elem in range(len(y_train)):  #On parcout chacun des tableaux dans la liste
    array = y_train[elem]
    Potentiel_Image = []          # On initialise une liste (vide initialement) qui va contenir le
potentiel d'une image
    for i in range(array.shape[0]):
      for j in range(array.shape[1]):
        pixel = int(array[i,j]*(1506/255))  #On détermine le potentiel de chaque pixel
        Potentiel_Image.append(pixel)
    Potentiel_Images[elem] = sum(Potentiel_Image)
  return Potentiel_Images

#Création des classes : on prend les données du potentiel (max_min)/n avec n nombre de classe
-2 (cart il y a inf à min et sup à max)

def Bornes(list_potentiel, nombre_classes_moins_deux):
  n = nombre_classes_moins_deux
  minl = min(list_potentiel)
  maxl = max(list_potentiel)
  pas = (maxl - minl)/n

  bornes=[]

  i = int(minl)
```

```
  while i<(maxl+pas):
    bornes.append(i)
    i = i + pas
  return(bornes)

def Classement(list_potentiel, bornes):

  label_class = np.zeros((len(list_potentiel), 1))

  for j in range(len(list_potentiel)):
    for i in range(len(bornes)-1):

      if float(bornes[i]) < float(list_potentiel[j]) < float(bornes[i+1]):
        label_class[j] = np.asarray([i+1])

      if float(bornes[0]) >= float(list_potentiel[j]):
        label_class[j] = np.asarray([0])

      if float(bornes[-1]) <= float(list_potentiel[j]):
        label_class[j] = np.asarray([len(bornes)])

  return label_class

Potentiel_Images_Tot = Potentiel(90, y_train)

n=3 #n=3 permettra à la fonction Bornes de créer 5 classes.

bornes = Bornes(Potentiel_Images_Tot, n)  #Création des bornes pour le reste du code
output_label = Classement(Potentiel_Images_Tot, bornes)
y_train = np.array(output_label)

print(y_train[77])
y_tot = np.vstack((y_train, y_train, y_train, y_train)) #En ayant augmenté les images, il faut
associer à chacune de ces nouvelles images le potentiel
                                        #correspondant. Une rotation de l'image n'affecte pas le
potentiel total, il nous suffit donc
                                        #d'assigner le même potentiel sans le recalculer.

#Les cellules suivantes reprennent le même principe que pour les données en entrées mais
appliqué sur les
#sorties : récupération des différents répertoires.

y_train_val = load_data_y_train(90,
'/content/gdrive/MyDrive/PFE/OutputTours100Valid/*.png')

Potentiel_Images_Tot_val = Potentiel(90, y_train_val)
output_label_val = Classement(Potentiel_Images_Tot_val, bornes)
y_train_val = np.array(output_label_val)
y_val_split1 = y_train_val[0:45]
y_val_split2 = y_train_val[45:90]
y_tot = np.vstack((y_tot, y_val_split1, y_val_split1, y_val_split1, y_val_split1))

y_data = load_data_y_train(90, '/content/gdrive/MyDrive/PFE/OutputTours100Data/*.png')

Potentiel_Images_Tot_val = Potentiel(90, y_data)
```

```
output_label_val = Classement(Potentiel_Images_Tot_val, bornes)
y_data = np.array(output_label_val)
y_tot = np.vstack((y_tot, y_data, y_data, y_data, y_data))

y_data2 = load_data_y_train(90, '/content/gdrive/MyDrive/PFE/OutputTours100Data2/*.png')

Potentiel_Images_Tot_val = Potentiel(90, y_data2)
output_label_val = Classement(Potentiel_Images_Tot_val, bornes)
y_data2 = np.array(output_label_val)
y_tot = np.vstack((y_tot, y_data2, y_data2, y_data2, y_data2))

y_1 = load_data_y_train(66, '/content/gdrive/MyDrive/PFE/Y1/*.png')

Potentiel_Images_Tot_val = Potentiel(66, y_1)
output_label_val = Classement(Potentiel_Images_Tot_val, bornes)
y_1 = np.array(output_label_val)
y_tot = np.vstack((y_tot, y_1, y_1, y_1, y_1))

y_2 = load_data_y_train(60, '/content/gdrive/MyDrive/PFE/Y2/*.png')

Potentiel_Images_Tot_val = Potentiel(60, y_2)
output_label_val = Classement(Potentiel_Images_Tot_val, bornes)
y_2 = np.array(output_label_val)
y_tot = np.vstack((y_tot, y_2, y_2, y_2, y_2))

y_3 = load_data_y_train(18, '/content/gdrive/MyDrive/PFE/Y3/*.png')

Potentiel_Images_Tot_val = Potentiel(18, y_3)
output_label_val = Classement(Potentiel_Images_Tot_val, bornes)
y_3 = np.array(output_label_val)
y_tot = np.vstack((y_tot, y_3, y_3, y_3, y_3))

y_4 = load_data_y_train(91, '/content/gdrive/MyDrive/PFE/Y4/*.png')

Potentiel_Images_Tot_val = Potentiel(91, y_4)
output_label_val = Classement(Potentiel_Images_Tot_val, bornes)
y_4 = np.array(output_label_val)
y_tot = np.vstack((y_tot, y_4, y_4, y_4, y_4))

#On finit par les classes du set de test
y_pred = load_data_y_train(90, '/content/gdrive/MyDrive/PFE/OutputTours100Pred/*.png')

Potentiel_Images_Tot_pred = Potentiel(90, y_pred)
output_label_pred = Classement(Potentiel_Images_Tot_pred, bornes)
y_pred = np.array(output_label_pred)

"""Nous pouvons désormais passer à la création du modèle."""

#Il faut tout d'abord importer les librairies de Keras.
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
from tensorflow.keras.layers import Dropout
from tensorflow.keras import regularizers
from tensorflow.keras import initializers
```

```
import matplotlib.pyplot as plt

# Définition du modèle, auquel on va ensuite ajouter les différentes couches, dans l'ordre

model = models.Sequential() #Le modèle sera un modèle de type séquentiel auquel nous ajouterons les couches dans les prochaines lignes


model.add(layers.Conv2D(16,3,activation ="relu", kernel_initializer='he_normal', input_shape = (100,100,3)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides = (2,2), padding="valid"))
model.add(Dropout(0.2))

model.add(layers.Conv2D(32,(3,3),activation ="relu",
kernel_regularizer=regularizers.l2(l=0.01)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides = (2,2), padding="valid"))
model.add(Dropout(0.2))

model.add(layers.Conv2D(64,(3,3),activation ="relu",
kernel_regularizer=regularizers.l2(l=0.01)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides = (2,2), padding="valid"))
model.add(Dropout(0.2))

model.add(layers.Conv2D(96,(3,3),activation ="relu",
kernel_regularizer=regularizers.l2(l=0.01)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides = (2,2), padding="valid"))
model.add(Dropout(0.2))

model.add(layers.Flatten())    # "Mise à plat" (vectorisation) du tenseur pour permettre de la connecter à une couche dense
model.add(layers.Dense(256, activation= "relu"))   # Couche dense, à 512 neurones
model.add(layers.Dense(5, activation= "softmax"))   # Couche de sortie

model.summary()

model.compile(optimizer='adam',
          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          metrics=['accuracy'])

history = model.fit(x_tot/255, y_tot, epochs=15, batch_size=40,
validation_data=(x_val_split2/255, y_val_split2))

#Création d'un graphique reprennant l'accuracy à chaque epoch.

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')

#Test de l'accuracy du modèle
test_loss, test_acc = model.evaluate(x_pred,  y_pred, verbose=2)
print(test_acc)
```

#Nous allons pousser la vérification des performances de notre modèle en utilisant une matrice de confusion

```
from sklearn.metrics import confusion_matrix

y_mat = model.predict(x_pred)
y_max = np.argmax(y_mat, axis=1)

labels = ['0', '1', '2', '3', '4']
cm = confusion_matrix(y_pred, y_max)

print(cm)
```

#Si les performances du modèle sont jugées satisfaisante, nous pouvons le sauvegarder en renseignant le lieu d'enregistrement et son nom.

```
#model.save('/content/gdrive/MyDrive/PFE/ModelsExempleh5', save_format='h5')
```