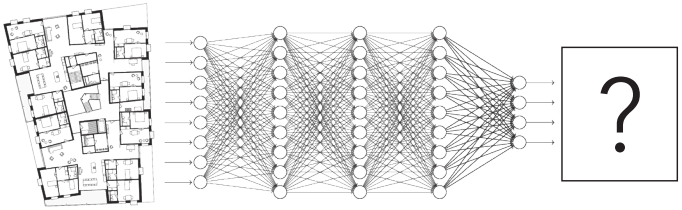
Mémoire de master par **Sophie Oros**
Séminaire : Activités et Instrumentation de la conception
Enseignants : François Guéna, Joaquim Silvestre, Anne Tüscher
ENSA de Paris La-Villette 2019

L'apprentissage machine au service de la conception architecturale ?

Une application : Extraire des informations à partir de plans



SOMMAIRE

INTRODUCTION

0| INTRODUCTION

Dans ce mémoire, nous allons explorer comment les techniques informatiques les plus récentes pourraient être intégrées à la palette d'outils de l'architecte concepteur. En d'autre termes, il s'agit de rechercher des applications possibles pour la conception architecturale de ce qu'on appelle abusivement *«l'intelligence artificielle»*, mais dont la branche qui se développe le plus aujourd'hui est plus précisément *l'apprentissage machine* (ou *machine learning*)[1]. Ce travail propose en effet d'esquisser une piste d'application qui permettrait d'améliorer certaines techniques d'aide à la conception déjà développées avec des modes de programmation plus «classiques», comme les grammaires de forme[2] ou d'autres algorithmes génératifs (automates cellulaires, algorithmes génétiques). Ce que toutes ces techniques ont en commun, est de chercher à automatiser le processus de conception architecturale, ce qui pose deux questions: pourquoi, et comment?

Pourquoi vouloir automatiser certaines tâches de la conception architecturale?

La production architecturale a cela de particulier, qu'elle ne produit que des prototypes, chaque projet répondant à une combinaison de contraintes objectives unique, auxquelles s'ajoutent des aspects subjectifs qui orientent implicitement les choix de conception. Le concepteur procède donc par tâtonnements, jusqu'à arriver à une solution convenable. Non seulement ce processus demande énormément de temps, mais il ne garantit même pas d'arriver dans le temps imparti à une solution optimale (impossibilité d'explorer toutes les combinaisons de paramètres intéressantes). Automatiser en partie la conception, c'est à dire déléguer à la machine, permettrait alors non seulement de gagner du temps pour d'autres tâches, mais encore d'aller plus loin dans la résolution des problèmes posés, grâce à la mise à profit de la capacité de la machine à effectuer massivement et rapidement des tâches numériques précises.

Comment automatiser certaines tâches du processus de conception?

Il a toujours été difficile et peu rentable de développer des outils informatiques sur mesure, répondants aux besoins toujours changeants de l'architecte concepteur. Alors qu'on sait aujourd'hui automatiser de manière extrêmement fiable des tâches complexes (comme jouer aux échecs, conduire une voiture, ou traduire un texte par exemple) notamment grâce à l'apprentissage automatique, on peine encore à soulager les concepteurs de tâches répétitives et laborieuses. Cette limite est due à la nature même des tâches à automatiser, ce qu'on peut appeler le «problème de la formulation explicite»[3]. En effet, pour déléguer des tâches à la machine, il faut pouvoir les expliciter sous la forme de paramètres quantitatifs à résoudre, ce qui est long et parfois impossible (beaucoup de paramètres étant

1. *Plus de détails dans la partie 1 sur l'histoire et les principes de base de l'apprentissage machine*
2. *Plus de détails dans la partie 2 sur l'état de l'art*
3. *Ce que Jean-Pierre Chupin expose aussi comme les «vilains problèmes mal formulés» in J.-P. Chupin, Analogie et théorie en architecture: de la vie, de la ville et de la conception, même. Gollion: Infolio, 2013, p.242*

de nature implicite ou subjective). Si de nombreux courants de pensée[4] ont tenté de produire une «syntaxe» qui permette de formaliser dans leur ensemble les paramètres à résoudre lors du processus de conception architecturale, ils se sont généralement heurté aux limites de l'exercice. C'est ici que l'idée d'utiliser l'apprentissage machine est intéressante: et si l'architecte pouvait transmettre ses connaissances à la machine, sans avoir besoin de les expliciter de manière fastidieuse? Notre hypothèse est que les plans (ainsi que d'autres modes de représentation), sont des supports de cette connaissance, en ce qu'ils sont utilisés par les architectes eux-même pour apprendre à partir de solutions existantes, mais aussi d'apprendre au fil du processus de conception en tant qu'ils permettent de mettre à l'épreuve les choix de conception. Il pourrait donc être intéressant d'explorer comment la machine pourrait être amenée à «comprendre intuitivement» les critères de l'architecte, sans que celui-ci ait besoin de passer par un mode d'expression qui n'est pas le sien, c'est à dire en lisant directement des plans.

Apprendre les machine à lire des plans...

Si les limites de temps et de moyens de ce travail n'ont pas permis de formaliser de manière exhaustive un outils opérationnel, l'approche choisie consiste donc à présenter une synthèse des connaissances acquises pour réaliser une expérimentation appelant à être approfondie. Il s'agissait donc de mettre en pratique ce type d'outils sur un problème simple : l'apprentissage machine de quelques «connaissances architecturales» à partir d'un corpus de plans. Le contenu du mémoire aborde donc diverses problématiques qui apparaissent lors d'une telle expérimentation. On peut citer entre autres : le choix du support de connaissances à soumettre à l'apprentissage[5], la formalisation précise des connaissances à apprendre ou encore le paramétrage de l'algorithme d'apprentissage adapté aux données soumises.

Dans la première partie, on reviendra donc sur le contexte technique et théorique dans lequel se situe notre sujet. Cela nous permettra en particulier d'introduire plus précisément les notions de base de l'apprentissage machine, mais aussi d'identifier certains besoins de l'architecte en matière d'outil d'assistance à la conception, auxquels pourraient répondre certaines applications cette technologie.

Dans la seconde partie, il s'agira de préciser notre problématique au regard de l'état de l'art plus spécialisé. L'analyse de certains travaux de recherche nous permettra en particulier d'identifier des absences de questionnement ou des pistes non explorées, pouvant faire l'objet de notre recherche. On définira ainsi les objectifs de notre expérimentation.

4. Entre autres on peut citer C.Alexander et son «Pattern Langage» ou Philippe Boudhon et son «Architecturologie», évoqués en partie 1

Dans la troisième partie, on exposera en détail les étapes de l'expérimentation. Chaque étape sera l'occasion d'aborder des problématiques de mise en œuvre plus générales. Si les résultats exposés sont encore peu convaincants quand à leur applicabilité, on pourra toutefois noter que la méthode mise en place constitue en elle même un résultat intéressant.

On évoquera finalement dans la dernière partie, des pistes d'approfondissement possibles à partir de l'analyse des résultats de notre expérimentation, mais aussi à partir des connaissances collectées pour sa réalisation. Au delà de leur aspect prospectif, ces pistes ont surtout pour vocation d'illustrer les intentions implicites de cette recherche, et les applications concrètes qui pourraient en découler.

Enfin, la conclusion permettra notamment de revenir sur les fantasmes qui entourent ces technologie dites d'intelligence artificielle, à la lumière des problématiques rencontrées dans notre mise en œuvre expérimentale. La question de l'automatisation totale des compétences de l'architecte n'est non seulement pas l'objet de notre exposé, mais encore moins une menace d'actualité au regard des différents paramètres évoqués au fil de ce mémoire.

CONTEXTE

Intelligence artificielle et rêves de concepteur

1| CONTEXTE

1.1 «L'intelligence artificielle», un sujet d'actualité

La notion «d'intelligence artificielle» (IA) est apparue dès l'invention des premiers ordinateurs, mais elle jouit aujourd'hui d'un essor dû à des succès technologiques très récents. Elle véhicule aujourd'hui beaucoup de fantasmes, notamment à cause de l'ambiguïté des termes employés pour la décrire au grand public. Pour illustrer les crispations qu'elle peut générer dans le domaine pourtant encore assez peu concerné de l'architecture, on peut citer une récente pétition lancée par le syndicat des architectes (*UNSFA*) à propos de la déclaration du patron d'Autodesk qui évoquait l'intégration d'une IA dans ces logiciels d'outils d'aide à la conception[1]. Si les architectes à l'origine de cette pétition s'inquiètent en particulier pour leur droit d'auteur (et c'est légitime), de nombreuses questions sont également soulevées par cette annonce: mais que peut vraiment faire une «intelligence artificielle» en terme de conception architecturale, même avec l'aide de données récoltées auprès des nombreux utilisateurs? Afin de mieux saisir les enjeux de ce sujet vaste, revenons donc rapidement sur quelques principes, et quelques techniques qui existent actuellement sur le marché, en terme d'IA.

1.1.1 Histoire d'un développement retardé

Si la notion «d'intelligence artificielle» apparaît pour la première fois lors d'une conférence en 1956, les premiers modèles théoriques de «réseaux de neurones artificiels» inspirés de la compréhension du cerveau, datent de 1943, soit avant même le premier ordinateur électronique. Inspirée au départ de la biologie, cette technologie s'est ensuite naturellement développée pour réaliser des tâches mettant à profit les capacités calculatoires et systématiques de l'ordinateur, pour assister l'humain dans des tâches laborieuses. Un débat continue d'ailleurs aujourd'hui d'animer les chercheurs sur l'ambiguïté des termes liés à une interprétation biologique, qui est non seulement porteuse d'ambiguïtés anthropomorphiques mais peut surtout induire en erreur les recherches.

«Définir l'intelligence artificielle n'est pas chose facile. (...) L'intelligence artificielle désigne en effet moins un champ de recherches bien défini qu'un programme, fondé autour d'un objectif ambitieux : comprendre comment fonctionne la cognition humaine et la reproduire ; créer des processus cognitifs comparables à ceux de l'être humain.» Cédric Villani[2]

« Même si les avions ont des oiseaux pour modèles, ils ne battent pas des ailes. De façon comparable, les RNA sont progressivement devenus assez différents de leurs cousins biologiques. Certains chercheurs soutiennent même qu'il faudrait éviter totalement l'analogie biologique, par exemple en disant unité au lieu de neurone, de peur que nous ne limitions notre créativité aux systèmes biologiquement plausibles. » Aurélien Géron[3]

Pour résumer son histoire récente, l'IA a donc connu un développement avec des périodes d'enthousiasme et de désillusions, repoussant toujours plus les limites de ce qu'on croyait pouvoir n'être fait que par les humains. Après les premières découvertes, dont le principe du «perceptron» (modèle permettant de réaliser des opérations logiques grâce à une

1. *« Je m'oppose au pillage des données stockées dans le cloud d'AUTODESK », Petitions24.net. Disponible sur: https://www.petitions24.net/je_moppose_au_pillage_des_donnees_stockees_dans_le_cloud_dautodesk.*
2. *C. Villani, Rapport « Donner un sens à l'intelligence artificielle, pour une stratégie nationale et européenne », 2018. Disponible sur: https://www.aiforhumanity.fr.*
3. *A. Géron, Deep Learning avec TensorFlow - Mise en œuvre et cas concrets. Dunod, 2017, p.73*

1943 Première formulation théorique des *"réseaux de neurones artificiels"* par le neurophysiologiste W.McCulloch et le mathématicien W.Pitts
1946 *ENIAC*, premier ordinateur entièrement électronique
1949 Règle d'apprentissage de Hebb "Les neurones qui s'activent entre même temps se lient entre eux"
1950 Création du *test de Turing*
Premières formulations théoriques du concept *d'apprentissage par renforcement*
1953 *IBM 650*, premier ordinateur commercial produit en série
1956 Conférence de Dartmouth, abordant pour la première fois la notion *d'intelligence artificielle* par John McCarthy Marvin (inventeur de LISP)
1957 Invention du principe de *perceptron* par F.Rosenblatt
1958 Premier *Modem* (transmission de données binaires sur ligne téléphonique)
1960 Fin des premierscespoirs démesurés sur les capacités immédiates de l'IA
1976 Apple I
1980 Début de "l'hiver de l'IA"
Prix Nobel de médecine attibué aux chercheurs D.H.Hubel et T.Wiesel pour leurs travaux sur le cortex visuel des chats et des singes
Concept du *néocognitron* inspiré par le cortex visuel
1981 IBM PC (personal computer)
1983 Naissance d'*internet* (protocole TCP/IP)
1985 Fin de "l'hiver de l'IA"
Découvertes pionnières en méthodes de machine learning
1986 Invention par D.Rumlhart et al. de *l'algorithme d'entraînement à rétro-propagation*
1991 Naissance du *Wold Wide Web* (protocole HTTP, langage HTML)
1994 *Yahoo*, premier moteur de recherche
1997 Le programme *"DeepBlue"* vainqueur contre le champion du monde en titre aux échecs Kasparov
1998 Publication par Y.LeCun, L.Bottou, Y.Bengio et P.Haffner de la célèbre architecture *LeNet-5* capable de reconnaître les numéros de chèques grâce à l'invention des *couches de convolutions*
2010 ImageNet Challenge
Découverte cruciale par X.Glorot et Y.Bengio de l'intérêt de *l'initialisation aléatoire*
2012 Reccord ImageNet battu par AlexNet
2013 La start-up DeepMind présente un système capable d'apprendre à jouer à n'importe quel jeu Attari à partir des seules règles du jeu, grâce à un apprentissage par renforcement
2014 Reccord ImageNet battu par GoogleNet
DeepMind racheté par Google pour plus de 500 millions de $
2015 Google DeepMind *"AlphaGo"* vainqueur face au champion du monde en titre de Go
Invention par S.Ioffre et C.Szegedy de la *technique de normalisation par lots*
Reccord ImageNet battu par ResNet
2016 "L'IA de Microsoft peut reconnaître un discours mieux que des sujets humains"

couche unique de «neurones artificiels» unitaires), l'IA (et plus précisément les «réseaux de neurones artificiels») a connu ensuite une «traversée du désert». En effet, les espoirs provoqués par ces découvertes ont été vite rattrapés par des limites infranchissables dues à l'immaturité des technologies. Si les techniques d'apprentissage continuent à progresser dans les années 80 , grâce à de nouvelles «architectures» des réseaux de neurones, les financements sont redirigés vers des techniques alternatives, telles que les *SVM* (machines à vecteurs de support), dont on comprend mieux les fondements théoriques. Dans les années 1990 un premier rebond apparaît, mais c'est surtout dans les années 2010 qu'on observe une véritable amélioration des performances des systèmes dits *«d'apprentissage machine»* (ou *apprentissage automatique* ou encore *machine learning*). Aujourd'hui on entend parler d'IA à longueur de journée, fort des succès récents et médiatiques de ces technologies. Pour résumer les atouts qui en font probablement une révolution technologique partie pour durer, voyons les arguments listés par l'auteur Aurélien Géron (dans son ouvrage qui a servi de référence très importante dans ce travail de mémoire):

«Finalement, nous assistons aujourd'hui à un regain d'intérêt pour les RNA (réseaux de neurones artificiels). Va-il s'évaporer comme les précédents ? Il y a quelques bonnes raisons de croire que celui-ci sera différent et qu'il aura un impact bien plus profond sur nos vie :

1. Il existe des données en quantité absolument gigantesques pour entraîner les RNA, et ils sont souvent bien meilleurs que les autres techniques d'apprentissage automatique sur les problèmes larges et complexes.

2. L'extraordinaire augmentation de la puissance de calcul depuis les années 1990 rend aujourd'hui possible l'entraînement de grands réseaux de neurones en un temps raisonnable. Cela est en partie dû à la loi de Moore, mais également à l'industrie du jeu qui a produit par millions (et donc à bas coût) des cartes graphiques équipées de GPU puissants.

3. Les algorithmes d'entraînement ont également été améliorés. Pour être honnête, ils ne sont que légèrement différents de ceux des années 1990, mais ces ajustements relativement limités ont eu un impact extrêmement positif.

4. Certaines limites théoriques des RNA se sont avérées plutôt bénignes dans la pratique. Par exemple, de nombreuses personnes pensaient que les algorithmes d'entraînement des RNA étaient condamnés car ils resteraient certainement bloqués dans un optimum local, alors que ces cas se sont révélés plutôt rares en pratique (et lorsqu'ils surviennent, ils sont en général assez proches de l'optimum global).

5. Les RNA semblent être entrés dans un cercle vertueux de financement et de progrès. Des produits incroyables fondés sur les RNA font régulièrement la une de l'actualité. Les RNA attirent ainsi de plus en plus d'attention, et donc les fonds. Cela conduit à de nouvelles avancées et encore plus de produits étonnants.» Aurélien Géron[4]

1.1.2 Quelques principes de l'apprentissage machine

Afin de pouvoir comprendre ce qui distingue les différentes branches qui constituent l'apprentissage automatique, on va d'abord évoquer très sommairement ses principes de bases. On favorisera ici les explications "qualitatives" sans alourdir l'exposé de considérations mathématiques (qui sont toutefois à la base de ces concepts). Voici donc tout d'abord trois définitions citées par A.Géron[5]:

Définition 1
"L'apprentissage automatique est la science ou l'art de programmer les ordinateurs pour qu'ils puissent apprendre à partir de données."

Définition 2
« L'apprentissage automatique est la discipline donnant aux ordinateurs la capacité d'apprendre sans qu'ils soient explicitement programmés. » Arthur Samuel, 1959

Définition 3
« Etant donné une tâche T et une mesure de performance P, on dit qu'un programme informatique apprend à partir d'une expérience E si les résultats obtenus sur T, mesurés par P, s'améliorent avec l'expérience E. » Tom Mitchell, 1997

Dans ces différentes définitions, la caractéristique principale du système mis en place est donc d'apprendre à partir d'exemples (appelés couramment *jeu d'entraînement* ou *training set*). Mais comment le système fait-il pour apprendre par lui même ? Voici encore une explication très claire donnée par A.Géron[6]:

«L'approche la plus fréquente consiste à créer un modèle prédictif et d'en régler les paramètres pour qu'il fonctionne au mieux aux données d'entraînement. Un modèle linéaire par exemple donne une prédiction qui est une somme pondérée des paramètres plus un terme constant. La phase d'entraînement du modèle consiste à trouver la valeur des coefficients de pondération (ou poids) à appliquer aux variables d'entrées pour minimiser l'erreur du modèle sur l'ensemble du jeu de données d'entraînement. Une fois les paramètres réglés, on peut utiliser le modèle pour faire des prédictions sur de nouvelles observations. L'espoir est que si le modèle fonctionne bien sur les données d'entraînement, il fonctionnera également bien sur de nouvelles observations. Si la performance est bien moindre, on dit que le modèle a « surajusté » le jeu de données d'entraînement. Cela arrive généralement quand le modèle possède trop de paramètres par rapport à la quantité de données d'entraînement disponible et à la complexité des tâches à réaliser.»

En réalité, deux principes de base permettent d'expliquer ce processus: le modèle de base du réseau neurone artificiel (ou perceptron), et le processus dit de rétropropagation. En effet, le modèle du réseau neurone artificiel a tout d'abord été inventé et utilisé dans sa composition la plus simple: une seule couche de neurones artificiels (chacun pouvant réaliser une classification binaire simple, basée sur une fonction de seuil), étant entraînée selon la règle de Hebb (définie en 1949), résumée historiquement ainsi: « les neurones qui s'activent en même temps se lient entre eux ». En d'autres termes, pour chaque neurone de sortie qui produit une prédiction erronée, il renforce donc les poids des connexions liées aux entrées qui auraient contribué à la prédiction juste. Ces perceptrons ont ensuite été enrichis d'une ou plusieurs couches cachées (devenant alors des *réseaux de neurones profonds*), et il a alors été bien plus difficile de les entraîner. C'est là que l'invention de l'algorithme de rétropropagation (1986) a joué un rôle fondamental, en permettant de décomposer l'entraînement en une succession de "passes en avant" et de "passes en arrière". L'explication qui suit pourra être complétée par la visualisation du schéma réalisé dans la partie 3.3.3 :

«Pour l'exprimer de façon concise : pour chaque instance d'entraînement, l'algorithme de rétropropagation commence par effectuer une prédiction (passe vers l'avant), mesure l'erreur, traverse chaque couche en arrière pour mesurer la contribution à l'erreur de chaque connexion (passe vers l'arrière) et termine en ajustant légèrement les poids des connexions de manière à réduire l'erreur (étape de descente du gradient).»[7]

4. *Ibid, p.74*
5. *Ibid, p.5*
6. *Ibid, p.7*
7. *Ibid, p.82*

Voyons maintenant les différents types de tâches pouvant être apprises, ainsi que les techniques d'entraînement qui permettent spécifiquement de les obtenir. C'est ici qu'apparaît l'importance des données d'entraînement, puisqu'elles conditionnent en grande partie ce qui peut être appris par le système (en effet les "connaissances" à acquérir doivent être lues par la machine dans ces données). On distingue trois types d'apprentissage:

L'apprentissage supervisé, qui requiert un jeu de données d'entraînement « étiqueté », c'est-à-dire pour lequel chaque observation est accompagnée de la réponse souhaitée, que l'on nomme étiquette ou cible (label ou target en anglais). Celui-ci peut être utilisé pour apprendre les deux types de tâches suivants:

- Tâche de classification: attribution d'une classe par mesure de similarité
- Tâche de régression : prédiction d'une valeur en fonction de divers paramètres

L'apprentissage non supervisé, pour lequel le jeu d'entraînement n'est pas étiqueté. Celui-ci peut être utilisé pour apprendre les trois types de tâches suivants:

- Tâche de détection d'anomalie : détection de différence par rapport à de nombreux exemples connus
- Tâche de partitionnement d'un jeu de données: classement des données par familles de similarité
- Tâche de réduction de la dimensionnalité: simplification selon certains paramètres

L'apprentissage par renforcement, qui est capable de générer ses propres données d'entraînement à partir de règles de bases (utilisé par exemple pour l'apprentissage de jeux de stratégie comme les échecs ou le Go). Ce dernier est de loin le plus intéressant, mais le plus complexe à mettre en œuvre.

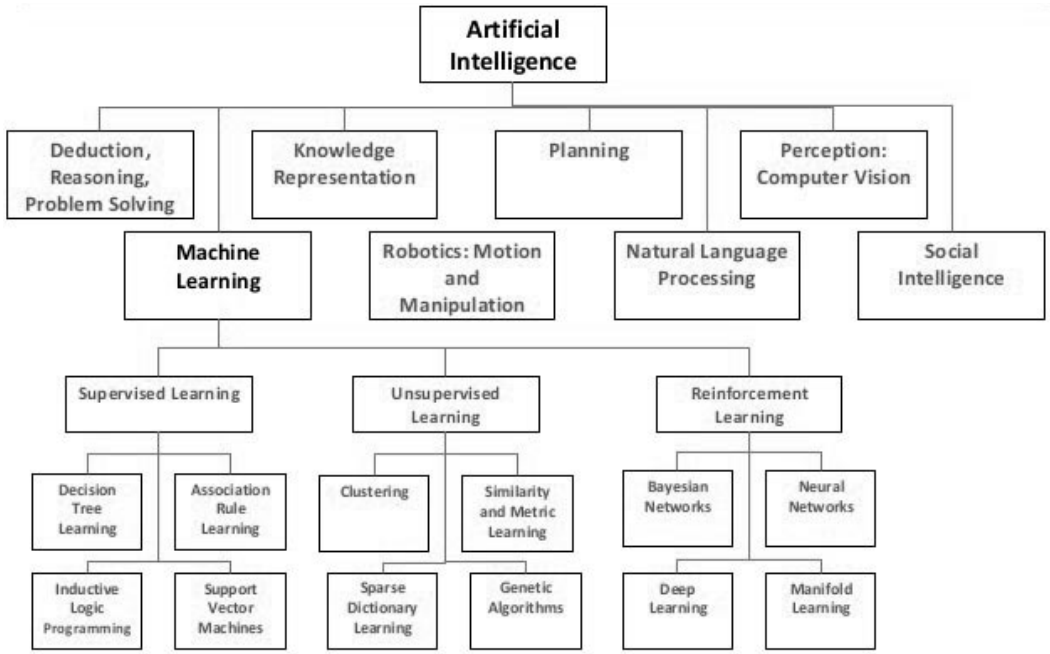


Schéma présentant une arborescence des différentes "branches" de l'IA.[8]

8. *ANZIEU, « S'amuser avec le Machine Learning ! Part2 », Alexis ANZIEU, 09-mars-2018 .*

On notera enfin que les applications de l'apprentissage machine sont très développées dans certains domaines, pour des raisons spécifiques, dues notamment à la nature des données à disposition. Par exemple, la reconnaissance automatique de photographies a bénéficié d'une grande quantité de données labellisées (grâce aux réseaux sociaux) et accessibles sur le web. On peut aussi évoquer ici les types de réseaux de neurones capables de réaliser ces différentes tâches, et dont l'invention a chaque fois permis de réaliser de nouvelles performances. En effet, après les réseaux de neurones profonds (RNP), on a inventé les réseaux de neurones convolutifs (RNC,dont le principe inspiré du cortex visuel a permis une nette amélioration des capacités "visuelles" de reconnaissances de motifs), puis les réseaux de neurones récurrents (RNR, capables de créer de la musique grâce à leur mémorisation de séries temporelles), et enfin les autoencodeurs qui sont capable d'apprentissage non supervisé. Si on voulait entrer dans le détail, on verrait que de nombreuses innovation techniques mais aussi méthodologiques on permis petit à petit d'améliorer la manière dont on entraîne ces réseaux, afin de dépasser les problèmes de divergence ou de sur-apprentissage qui ne cessent de réapparaître avec la complexification des architectures.

1.1.3 Quelques enjeux sociaux et économiques

Si l'IA pose encore beacoup de questions techniques, elle soulève aussi de nombreux enjeux sociaux et économiques. Parmi de nombreux auteurs s'essayant sur ces sujets, on peut citer le rapport officiel du député Cédric Villani, paru en 2018 et qui dresse un état des lieux à l'échelle française et européenne des évolutions de l'industrie liée à ces technologies. Si l'architecture et la construction ne font pas partie des secteurs prioritaires de développement qu'il préconise, ses recommandations d'ordre général s'appliquent à tous les domaines. Les problématiques qu'il pointe en particulier sont celle des emplois menacés par l'automatisation des tâches, ainsi que les problématiques de souveraineté de nos systèmes de connaissance.

«Dans ce monde-là, qui est désormais le nôtre, ces technologies représentent beaucoup plus qu'un programme de recherche : elles déterminent notre capacité à organiser les connaissances, à leur donner un sens, à augmenter nos facultés de prise de décision et de contrôle des systèmes. Et notamment à tirer de la valeur des données. L'intelligence artificielle est donc une des clés du pouvoir de demain dans un monde numérique.»

«L'intelligence artificielle est loin d'être une fin en soi et son développement doit prendre en compte plusieurs aspects. Tout d'abord la nécessité de penser les modes de complémentarité entre l'humain et les systèmes intelligents. Que ce soit au niveau individuel ou collectif, cette complémentarité peut prendre plusieurs formes et peut être aliénante comme libératrice. Au cœur du développement de l'IA doit résider la nécessité de mettre en œuvre une complémentarité qui soit capacitante, en ce qu'elle permet de dés-automatiser les tâches humaines.»[9]

Du point de vue de l'architecte, on a bien-sûr un prisme de lecture de ces enjeux assez spécifique. En effet, si les emplois de nature créative sont encore peu menacés par l'automatisation, on cherche pourtant à s'en approcher pour des raisons d'efficacité dans les tâches de plus en plus complexes à réaliser. Pourtant, notre métier ayant un fort impact sur la société, on ne peut pas déléguer à la machine des responsabilités trop importantes. On doit également veiller à ne pas supprimer aux concepteurs leur autonomie (ou souveraineté) de créa-

9. *C. Villani, Rapport « Donner un sens à l'intelligence artificielle, pour une stratégie nationale et européenne », 2018.*

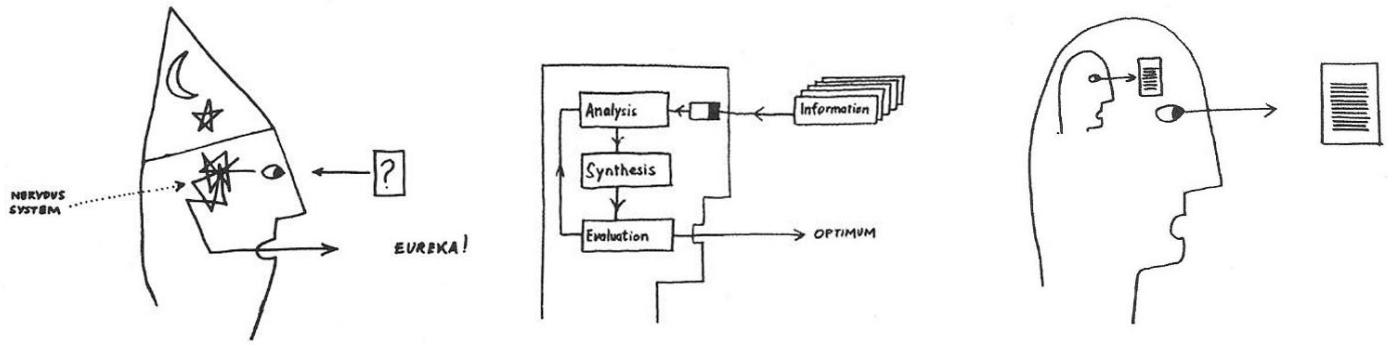
tion, par une domination des éditeurs de logiciels par exemple. Si l'IA peut devenir un outil de création très intéressant, c'est également une technologie très centralisatrice, dont ceux qui possèdent des données en nombre auront le monopole. On peut très bien imaginer des applications développées par des éditeurs de logiciels, mais aussi par des promoteurs dont les motivations essentiellement financières pourraient aveugler quand au pouvoir de standardisation et autres biais possibles de ces technologies.

1.2 Tentatives historiques d'automatisation de la conception architecturale

1.2.1 Le développement de l'informatique vécu par les architectes

Il serait bien trop long de détailler ici la longue liste histoire de l'appropriation par les architectes de l'outil informatique. En effet, depuis l'apparition des premiers ordinateurs, les concepteurs ont cherché à en faire un allié de travail dans leurs tâches besogneuses mais aussi dans leurs recherches les plus créatives. Si il existait déjà une infinité "d'écoles" de conception architecturale, la manière de se servir d'un ordinateur ajoute encore un paramètre à cette diversité des pratiques architecturales. Toutefois les théoriciens et historiens observent des "mouvements architecturaux", des tendances qui s'organisent autour de techniques computationnelles bien définies. C'est ainsi que Mario Carpo[10] évoque par exemple le rôle des "modeleurs de spline", développés à l'origine pour la conception automobile, dans l'émergence d'un mouvement architectural obnubilé par les courbes. Dans son ouvrage du même nom, l'auteur annonce par ailleurs ce qu'il appelle "the second digital turn" (un deuxième tournant digital), qui serait d'après lui en train d'apparaître du fait notamment de l'explosion de la quantité de données que l'on peut à présent stocker. Ce qu'il évoque par là est un certain essoufflement des mouvements "pionniers" de la conception numérique consistants uniquement à rechercher la complexité formelle à l'aide de processus complexes mais peu "intelligents" (au sens de "donnant du sens à leurs actions"). Leur manière d'utiliser la puissance de calcul de l'ordinateur étant très démonstrative, elle ne simplifie pourtant en rien, la tâche de l'architecte. Mais si ces écoles de conception démonstratives et formelles (qu'on désigne souvent comme les "déconstructivistes") sont celles dont on parle le plus souvent, en matière de digitalisation de la conception architecturale, d'autres plus discrètes se sont véritablement confrontées au "problème de la formulation explicite" (évoqué en introduction), qui semble être la véritable limite de la digitalisation du processus de conception, en ce qu'il bride la fluidité de la communication entre l'architecte et son outil qu'il aimerait pouvoir "éduquer". En effet, lorsqu'on tente de faire faire à l'ordinateur les tâches "intelligentes" du concepteur, on se heurte à un problème de traduction (ou de formulation) d'instructions claires et suffisantes pour la réalisation de cette tâche. Le concepteur doit donc choisir entre deux manières (incomplètes) d'utiliser la machine: soit il l'utilise pour optimiser un ensemble de paramètres qu'il a préalablement explicités (ce qui revient à un travail laborieux voir impossible selon le problème posé), soit il l'utilise pour générer des formes abstraites, ne résolvant que quelques paramètres géométriques du problème, qu'il doit ensuite «adapter» à la réalité du problème en terme d'usages et d'aménagements (ce qui revient à résoudre un problème dans le problème, avec de nouvelles contraintes géométriques).

10. M. Carpo, The second digital turn design beyond intelligence. Cambridge, MA: The MIT Press, 2017, p.55



Schémas présentant les "trois types de concepteurs" selon J.C.Jones 1969:

1. *"Designer as Magician"*
2. *"Designer as computer" (traitement systématique de l'information)*
3. *"Designer as self-organizing system" (le rôle de la reflexivité dans la conception)*

1.2.2 Des théories pour formaliser le processus de conception

En parallèle du développement de l'informatique, la quête de l'automatisation a poussé de nombreux concepteurs et théoriciens de la conception à un auto-analyse poussée, afin de pouvoir formuler une syntaxe communicable à la machine pour décrire les opérations du concepteur. Plusieurs écoles de pensée sont alors apparues, se confrontant majoritairement autour de deux positions: l'une voulant conserver le mystère de la "boîte noire" du concepteur (de peur d'en perturber le bon fonctionnement), et l'autre défendant une approche purement rationnelle consistant à décortiquer de manière exhaustive le processus de conception. Dans son ouvrage "Analogie et théorie en architecture"[11], J.P. Chupin procède à une analyse très documentée de ces différents courants de pensée et évoque notamment le cas de C. Alexander et son "Pattern Langage", la première tentative assez caricaturale de décomposition exhaustive des choix du concepteur. Si Alexander a fini par lui même désavouer sa propre méthode, d'autres tentatives plus élaborées ont été elles même confrontées à des limites: c'est le cas de Philippe Boudon[12] et sa théorie de "l'architecturologie" qui devait décrire intégralement sous forme "d'échelles architecturologiques" les paramètres à déterminer par le concepteur, et dont le nombre ne cesse d'augmenter au gré des besoins toujours plus divers de la conception. Pour résumer, cette école du «problem solving» qui propose de réduire la question posée en un ensemble de paramètres quantitatifs à faire optimiser par la machine, pose la question de la complexité d'une telle résolution (paramètres nombreux et parfois contradictoires). Mais sa limite est surtout l'impossibilité à expliciter complètement le problème en terme de paramètres explicites (sans compter le caractère unique de chaque problème).

1.2.3 Une troisième voie

Dans son livre "Permutation Design: Buildings, Texts ans Contextes", Kostas Terzidis[13] tente de proposer une alternative à la guerre de position entre les défenseurs de la boîte noire, et les adeptes du "problem-solving" à l'infinité de paramètres. Il souligne d'ailleurs l'apport selon lui majeur de C. Alexander passé plutôt inaperçu qui fût de proposer une définition de ce qui, dans la conception était "systématisable" et ce qui ne l'était pas, ainsi que la théorie selon laquelle une bonne modélisation du processus de conception devait inclure des changements de contraintes (changement d'avis du concepteur). Après une analyse similaire à celle présentée ici, Terzidis déduit un type d'usage qui devrait être fait de l'ordinateur, dans le but de résoudre aussi bien les limites de formulation explicite des "problem-solver" que les limites des concepteurs "à la boîte noire", qui ne parviennent pas tou-

11. *J.-P. Chupin, Analogie et théorie en architecture: de la vie, de la ville et de la conception, même. Gollion: Infolio, 2013.*
12. *P. Boudon, Sur l'espace architectural: essai d'épistémologie de l'architecture. Marseille: Parenthèses, 2003.*
13. *K. Terzidis, Permutation Design: Buildings, Texts, and Contexts. London ; New York: Routledge, 2014.*

jours à la solution optimale avec leur méthode de tâtonnement. D'après lui, la "conception automatique devrait permettre de garantir ces deux résultats: explorer une grande quantité d'options, et sélectionner la meilleure. Après avoir rappelé l'efficacité des méthodes d'allocation spatiale pour les problèmes très fonctionnalistes (comme les hôpitaux), il évoque même la nécessité de développer une méthode plus adaptative (proche du processus de tâtonnement), au cours duquel la machine apprendrait par elle même. Finalement le principe défendu par Terzidis est ce qu'il appelle les "permutation", dont il donne ici la définition:

«In design, the problems that designers are called upon to solve can be regarded as problems of permutations. A permutation is an ordered arrangement of elements in a set. In our case, the set is design and the elements are design components, such as lines, shapes, forms, or spaces.» Kostas Terzidis

Cette approche n'est pas sans rappeler une théorie moins récente: celle des *grammaires de formes (ou shape grammars)*, proposée pour la première fois par G. Stiny[14] en 1980, mais qui depuis continue à être l'objet de recherches. Si cette approche semble tout aussi laborieuse que les précédentes (celles des "problem-solving"), elle présente tout de même un intérêt majeur: elle se concentre sur la question spatiale, au lieu de vouloir intégrer tous les aspects du problème posé. Une telle approche permet non seulement de tirer parti des compétences de l'ordinateur sur une tâche très bien définie, mais elle permet également de puiser dans le champs des références déjà réalisées afin d'en extraire des connaissances (elle ne consiste donc pas à partir de rien). Pour illustrer ces considération, nous citerons un travail faisant désormais autorité dans cette branche de recherche : L'étude des maisons de Malagueira de l'architecte Alvaro Siza, par José Pinto Duarte[15].Dans Cette étude, le chercheur s'appuie sur un corpus de plans réels et sur les conseils de l'architecte lui même, afin de formuler une grammaire permettant non seulement de décrire les plans existants, mais encore d'en générer de nouveaux. Cet exemple bien qu'on ne peut plus unique, nous donne une piste d'exploration: les grammaires de forme comme moyen d'apprendre à partir d'exemples existants.

1.3 L'apprentissage machine, un piste de résolution?

Dans cette section, on produira quelques hypothèses sur la manière dont l'apprentissage machine semble pouvoir répondre aux besoins spécifiques du concepteur évoqués précédemment, et ce d'après ce qu'on comprend "intuitivement" des possibilités d'usages de ces systèmes. Ces hypothèses se basent sur une caractéristique récurrente des réseaux de neurones: la reconnaissance de motifs. Ces hypothèses nous guideront ensuite pour naviguer dans l'état de l'art, qui est vaste et touche à des champs disciplinaires très variés. Si l'expérimentation présentée en partie 3 ne permettra que partiellement d'en évaluer la validité, il est intéressant de les exposer ici, afin de donner à voir le cheminement vers notre problématique et les ambitions de départ, forcément trop larges, du travail de recherche.

Comme on peut déduire des théories de K. Terzidis notamment, l'apprentissage de l'architecture se fait en intégrant un vocabulaire de formes qui permettent une certaine abstraction de la réalité de l'objet à concevoir : la conception consiste en effet à manipuler ces abstractions pour les agencer et ensuite les retranscrire en une réalité constructible. L'une des fonctions que sont capables d'apprendre certains réseaux de neurones (les réseaux convo-

14. *G. Stiny, « Introduction to shape and shape grammars », Environment and Planning B: Planning and Design, vol. 7, n° 3, p. 343-351, 1980.*
15. *J. P. Duarte, « Towards the Mass Customization of Housing: The Grammar of Siza's Houses at Malagueira », Environment and Planning B: Planning and Design, vol. 32, n° 3, p. 347-380, juin 2005.*

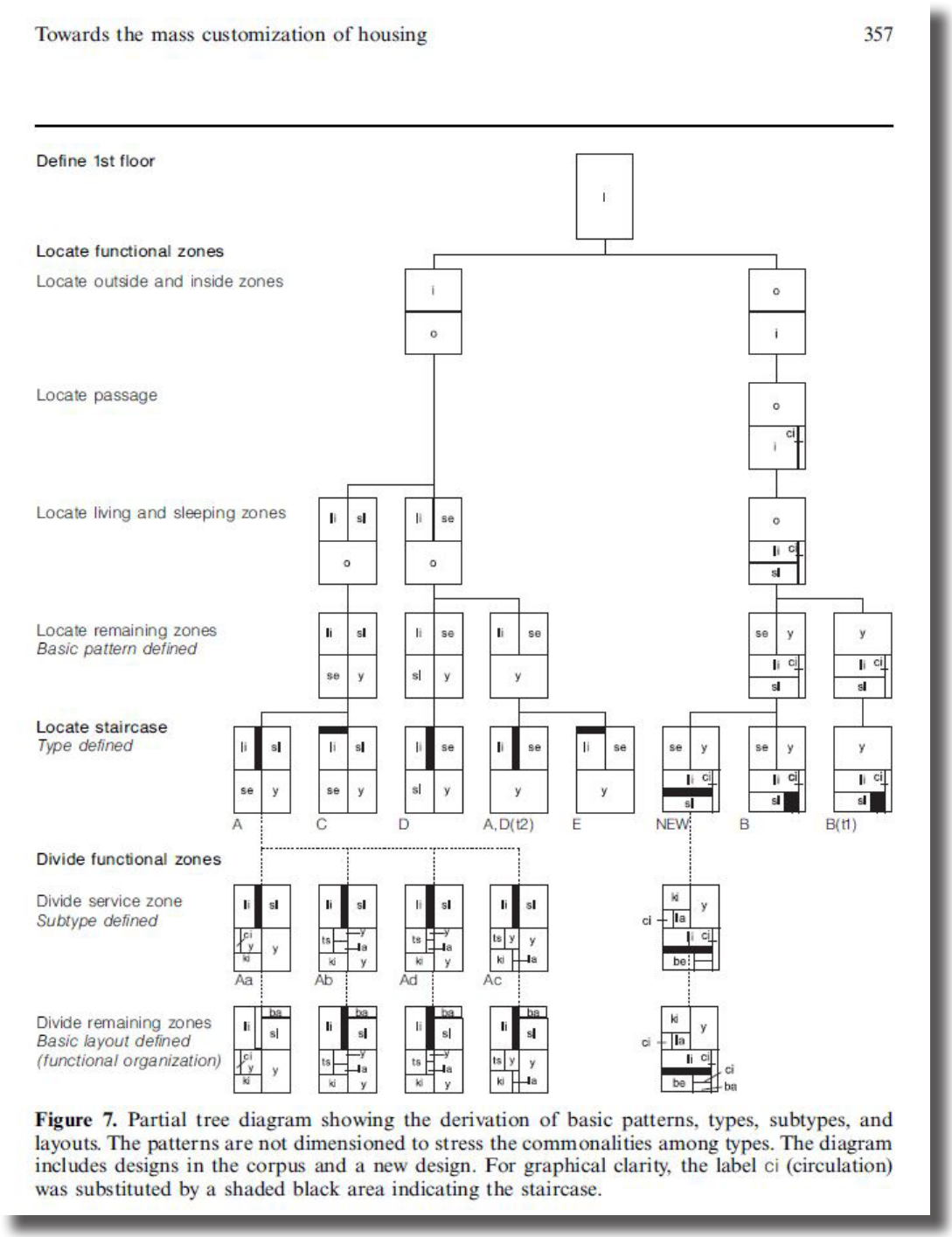


Figure 7. Partial tree diagram showing the derivation of basic patterns, types, subtypes, and layouts. The patterns are not dimensioned to stress the commonalities among types. The diagram includes designs in the corpus and a new design. For graphical clarity, the label ci (circulation) was substituted by a shaded black area indicating the staircase.

Diagramme d'arborescence exposant les règles de composition des différents types de plans, conformément à la "grammaire de forme" mise au point à l'aide d'un corpus de références, et des avis de l'architecte.

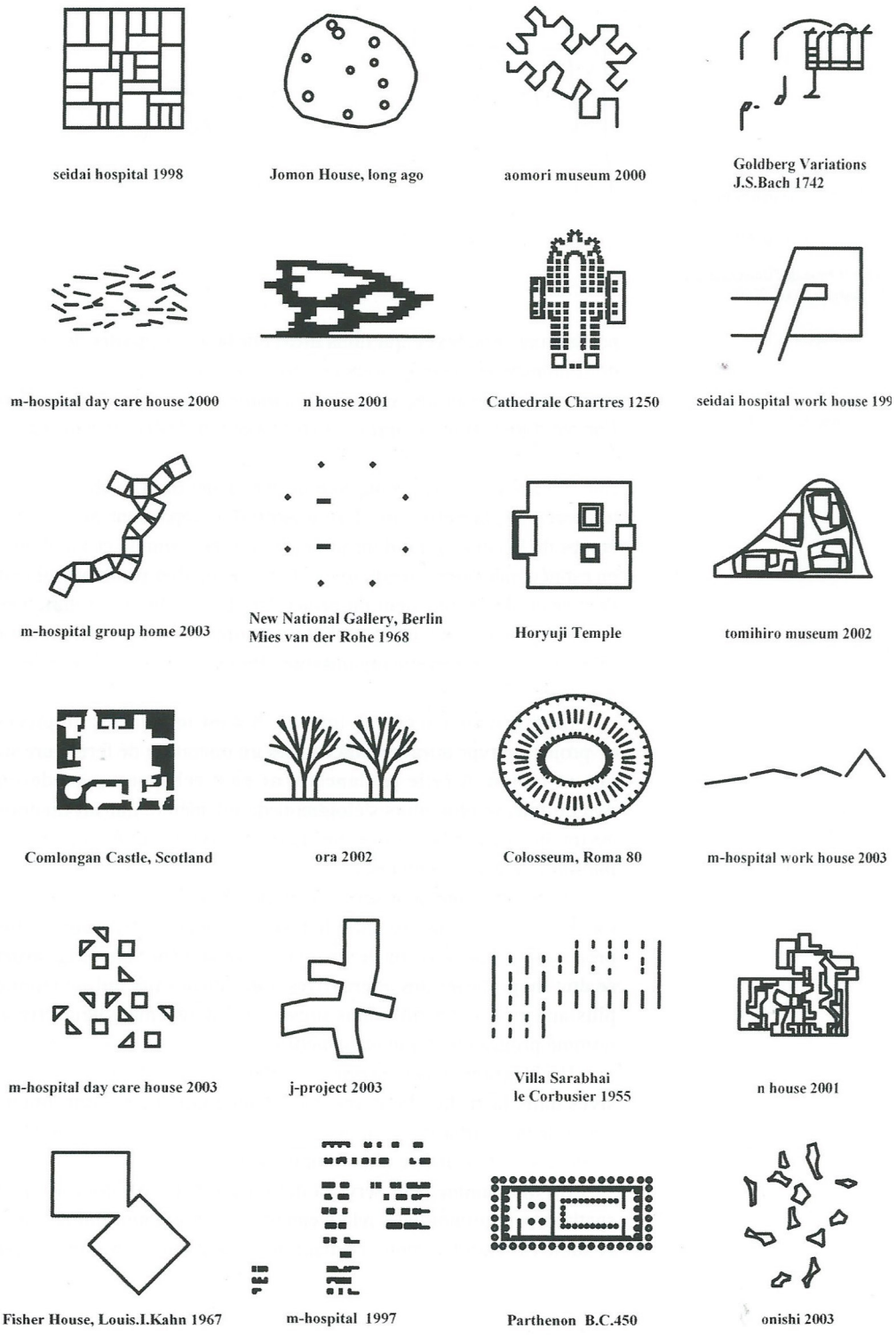
lutifs en particulier), est justement la reconnaissances de motifs (ou patterns). On pourrait alors imaginer "enseigner" à un réseau des connaissances architecturales sous la forme de motifs. Deux possibilités s'offrent alors: l'apprentissage supervisé de motifs identifiés au préalable, ou bien l'apprentissage de motifs non supervisé sur des plans réels non étiquetés. Chacune des deux options peut avoir un intérêt, même si la deuxième option semble la plus prometteuse, et la première la plus réaliste. Pour illustrer qualitativement la capacité des réseaux de neurones à apprendre à reconnaître de motifs de manière non-supervisée, on citera encore une fois l'ouvrage de A.Géron, à propos de la capacité de ceux-ci à produire en interne une représentation efficace des données d'entraînement:

« Prenons une série de nombres à mémoriser : une série plus longue mais obéissant à quelques règles simples est beaucoup plus simple à mémoriser (ex : suite de Syracuse). Si nous avions la capacité de mémoriser rapidement et facilement de très longues séries, nous n'aurions pas besoin de nous préoccuper de l'existence d'un motif dans la seconde suite et nous pourrions simplement apprendre chaque nombre par cœur. C'est cette difficulté de mémorisation des longues suites qui donne tout son intérêt à la reconnaissance de motifs. Comme le cerveau humain, si on contraint l'entraînement d'un réseau de neurones artificiel on le pousse à découvrir et à exploiter les motifs présents dans les données. »

«La relation entre mémoire, perception et correspondance de motifs a été étudiée sur les joueurs d'échec par William Chase et Herbert Simon dès le début des années 1970. Ils ont étudié comment les joueurs pouvaient mémoriser en très peu de temps la configuration de l'échiquier. Ils ont montré que les joueurs n'avaient pas une mémoire exceptionnelle, mais que c'était bien leur expérience du jeu qui leur permettait de reconnaître plus facilement des motifs de placement. A l'instar des joueurs d'échecs, un réseau de neurones artificiel examine les entrées, les convertit en une représentation interne efficace et produit en sortie quelque chose qui ressemble énormément à l'entrée.»[16]

A partir d'un entraînement sur des plans réels, on pourrait donc imaginer de produire un corpus de motifs, décrivant d'une certaine manière ces plans. Pourrait-on alors mettre en évidence un aspect "diagrammatique" de ces motifs? Seraient-ils intelligibles si on arrivait à les lire? Comment pourrait-on les utiliser? Le "mandala" de Sou Fujimoto[17] représenté ci-contre illustre une manière manuelle "d'encoder" des motifs, afin de les réinterpréter lors de la conception. Cette pratique permet à l'architecte de saisir l'essentiel d'une référence avant de la réinterpréter. Ce genre de mode de représentation pourrait-il devenir le langage de communication directe avec la machine si recherché par les architectes?

16. *A. Géron, Deep Learning avec TensorFlow - Mise en œuvre et cas concrets. Dunod, 2017, p.50*
17. *J. Lucan, Précisions sur un état présent de l'architecture. Lausanne: Presses Polytechniques et Universitaires Romandes, 2015.*



Diagrammes de l'architecte Sou Fujimoto, qu'il appelle ses "Madala" et qui sont pour lui un outil de conception.[17]

PROBLÉMATIQUE

État de l’art quand à l’automatisation de la conception

2| PROBLÉMATIQUE

Dans cette partie, il s'agit d'énoncer notre problématique, en s'appuyant sur une analyse (succincte) de l'état de l'art. Précédemment, on a donc évoqué les objectifs de cette recherche: extraire des "connaissances architecturales" à partir de plans, en tentant d'exploiter les capacités de "reconnaissance de motifs" propres aux réseaux de neurones (convolutifs en particulier). Afin de préciser nos objectifs, en terme de faisabilité mais aussi de méthode, nous allons donc rassembler les éléments qui pourraient nous orienter.

2.1 Structure de l'état de l'art

Tout d'abord, il s'agit d'avoir une vision claire de la structure de notre état de l'art. En effet, les domaines de recherche qui touchent à notre sujet sont variés et peuvent être répartis en cinq grandes familles thématiques:

1. Les grammaires de formes et autres algorithmes génératifs

   Ces recherches[1] ont déjà été évoquées, mais elles ont servi ici plus spécifiquement à nous aiguiller pour la réalisation de notre générateur de plans. Ceci dit, celui-ci reste très sommaire, et un intérêt de la recherche serait justement de pouvoir se dispenser de l'usage laborieux de ces grammaires de formes.

2. Le machine learning pour l'architecture

   Bien qu'assez éloignées de l'application spécifique de lecture des plan, ces recherches nous ont donné des pistes sur le type de données dont dispose d'architecte pour entraîner un réseau de neurones, mais aussi sur différentes applications possibles. On a par exemple pu voir des réseaux entraînés sur des "décisions implicites" d"un concepteur[2] (ce qui pose un gros problème de proportion entre la quantité d'opérations nécessaires à l'apprentissage, et la quantité d'opérations effectivement automatisées). Cette recherche présentait par contre une méthode intéressante, qui consistait à utiliser le réseau de neurone comme fonction d'évaluation d'un algorithme génétique, ce qui permettait de l'intégrer directement dans le processus de conception. On a également pu voir l'usage de données générées grâce à une maquette numérique, permettant un apprentissage supervisé de reconnaissance de types d'espaces à partir d'images perspectives [3]. Une autre étude proposait quand à elle de produire des "rêves architecturaux", en utilisant un réseau déjà entraîné sur une banque d'images d'animaux, afin de tester l'adaptabilité d'un tel réseau pour un usage à petite échelle[4]. Ces recherches donnent des pistes d'exploration intéressantes, et sont surtout la preuve d'un intérêt grandissant des architectes pour ces technologies, bien que les applications soient encore loin d'être opérationnelles.

---

1. *M. Barros, J. P. Duarte, et B. M. Chaparro, « A Grammar-Based Model for the Mass Customisation of Chairs: Modelling the Optimisation Part », Nexus Netw J Nexus Network Journal : Architecture and Mathematics, vol. 17, n° 3, p. 875-898, 2015.*
2. C. Sjoberg, C. Beorkrem, et J. Ellinger, « Emergent Syntax: Machine Learning for the Curation of Design Solution Space », in ACADIA 2017: DISCIPLINES & DISRUPTION [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 552- 561, 2017.
3. W. Peng, F. Zhang, et T. Nagakura, « Machines' Perception of Space: Employing 3D Isovist Methods and a Convolutional Neural Network in Architectural Space Classification », in ACADIA 2017: DISCIPLINES & DISRUPTION [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 474- 481, 2017.
4. J. Silvestre et Y. Ikeda, « ARTIFICIAL IMAGINATION OF ARCHITECTURE WITH DEEP CONVOLUTIONAL NEURAL NETWORK », p. 10.« Image-to

3. Le machine learning pour la création en général

   La créativité ou le design sont traités essentiellement de deux points de vue par la recherche en apprentissage automatique: l'application de motifs (ou de textures) [5], et la suggestion temporelle de séquence de croquis [6]. Il est intéressant de constater que ces application moins contraintes que celles en architecture, semblent permettre d'élaborer des solutions transposables à l'avenir à l'architecture. Il y a aussi des applications très proches de ce qu'on souhaite faire ici, comme l'apprentissage de motifs pour classer des peintures d'archives [7].

4. Le machine learning pour d'autres applications (techniques intéressantes)

   Cette partie de l'état de l'art est plutôt technique. En effet, pour comprendre ce qui est faisable et dans quelles conditions pour répondre à un problème donné, on peut être amené à consulter ce qui se fait dans des domaines annexes où les recherches ont de l'avance pour des raisons souvent d'enjeux économiques ou de meilleures conditions (comme des donnée abondantes par exemple). Dans notre cas, on a pu s'inspirer de recherches allant de la détection d'objets en 3D à l'aide d'objets modélisés synthétiquement[8], jusqu'à la classification de façade, qui se fait à partir de photos mais consiste aussi en la détection de motifs[9].

5. La lecture automatique des plans

   Enfin, une branche de l'état de l'art sur laquelle on ne pouvait pas faire l'impasse était celle qui concernait précisément notre sujet. Étonnamment, les applications proposées se focalisaient essentiellement sur la reconstitution de maquettes en 3D à partir de la détection des murs[10,11,12]. Cette approche conduit à occulter tout autre paramètre qu'on pourrait tenter de lire dans les plans, et procède souvent d'une définition très précise de ce qu'est un mur. On remarque aussi que ce sont essentiellement des méthodes d'apprentissage supervisé qui sont mises en œuvre, sur la base de bases de données de plans de promoteurs accessibles en ligne.

---

5. Image Translation with Conditional Adversarial Networks ». [En ligne]. Disponible sur: https://phillipi.github.io/pix2pix/.
6. D. Ha et D. Eck, « A Neural Representation of Sketch Drawings », arXiv:1704.03477 [cs, stat], avr. 2017.
7. B. L. A. Seguin, « Making large art historical photo archives searchable », p. 169.
8. P. S. Rajpura, H. Bojinov, et R. S. Hegde, « Object Detection Using Deep CNNs Trained on Synthetic Images », arXiv:1706.06782 [cs], juin 2017.
9. A. Martinović, M. Mathias, J. Weissenberg, et L. Van Gool, « A Three-Layered Approach to Facade Parsing », in Computer Vision – ECCV 2012, vol. 7578, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, et C. Schmid, Éd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, p. 416-429.
10. S. Ahmed, M. Weber, M. Liwicki, C. Langenhan, A. Dengel, et F. Petzold, « Automatic analysis and sketch-based retrieval of architectural floor plans », Pattern Recognition Letters, vol. 35, p. 91-100, janv. 2014.
11. L.-P. de las Heras, S. Ahmed, M. Liwicki, E. Valveny, et G. Sánchez, « Statistical segmentation and structural recognition for floor plan interpretation », IJDAR, vol. 17, n° 3, p. 221-237, sept. 2014.
12. S. Or, K.-H. Wong, Y. Yu, et M. M. Chang, « Highly Automatic Approach to Architectural Floorplan Image Understanding & Model Generation », p. 9.

2.2 Positionnement vis à vis de l'état de l'art

Si l'état de l'art spécifique sur la lecture automatique de plans nous a renseigné sur l'avancée des méthodes spécifiques pour la détection des murs, nous sommes obligés de nous en distancer un peu, étant donné que notre sujet est plus élargi. En effet, on cherche à mettre au point, non pas un système de détection de murs, mais d'un système qui pourrait potentiellement détecter d'autres "caractéristiques architecturales", telles que les proportions, les trames ou autres. Il nous fallait bien un prétexte pour démarrer, mais nous ne perdrons pas de vue nos objectifs. On aura donc un positionnement plus élargi sur l'état de l'art, afin de pouvoir envisager les perspectives plus larges de notre travail, quand à ses applications possibles pour assister la conception.

2.3 Définition des objectifs d'expérimentation

Au cours de cette expérience, nous chercherons donc à nous approcher le plus possible d'un résultat consistant à faire déchiffrer à un modèle entraîné, une "caractéristique architecturale" contenue dans des plans réels. Dans les limites de l'expérience, on travaillera avec des plans étiquetés qui seront générés par nos soins, mais toujours dans le but de comprendre comment on pourrait atteindre notre objectif, sans passer par une étape trop laborieuse (c'est à dire ni en étiquetant des centaines de plans, ni en ayant besoin de produire un générateur sans cesse plus complexe). Il s'agit donc de réaliser un entraînement supervisé pour une tâche de classification de plans d'appartement, qui prédise le nombre de pièces dans celui-ci.

EXPÉRIMENTATION

Construire un système d’apprentissage destiné à la lecture de plans

3| EXPÉRIMENTATION

Dans cette partie, on exposera en détail les étapes de l'expérimentation mise en place pour répondre à la problématique élaborée dans la partie précédente. Pour résumer, cette expérimentation consiste à produire un réseau de neurone et à l'entraîner sur un jeu de données, afin qu'il soit utilisable pour extraire un paramètre à partir d'un plan: le nombre de pièces. Il s'agit donc de réaliser une tâche de classification, où chaque classe correspond à un nombre de pièces possible. Le réseau utilisé est un réseau de neurone profond (c'est à dire avec plusieurs couches cachées), et convolutif (qui comprend une couche dite de convolution). Les données sur lesquelles est entraîné le réseau de neurone sont des plans générés, ce qui permet d'en produire en quantité illimitée sans avoir à les labelliser manuellement. La recherche consistera donc à évaluer l'influence sur la performance du réseau de quelques paramètres: la taille de l'échantillon d'entraînement, la proportion entre l'échantillon d'entraînement et l'échantillon de test (ou de validation), ou encore la diversité (ou complexité) des échantillons. Finalement, on analysera les limites de l'expérience, notamment quand à la possibilité d'utiliser le réseau pour lire de vrais plans.

3.1 Choisir l'outil de programmation

Pour réaliser cette expérience, il fallait tout d'abord choisir le support et le langage de programmation à utiliser. En effet, si certains logiciels permettent déjà d'utiliser des réseaux de neurone de manière intégrée, comme le *plugin LunchBoxML*[1], il reste néanmoins plus simple d'utiliser un outils plus commun pour le *Machine Learning*: la librairie TensorFlow du langage python. Cet outil a plusieurs avantages:

- Le langage python

  Outre le fait qu'il soit très majoritairement utilisé dans les applications de Machine Learning, les internautes sont généralement unanimes pour vanter les qualités synthétiques de ce langage. Aujourd'hui il est d'ailleurs devenu le premier langage enseigné dans les écoles d'ingénieur, et est compatible avec de nombreux logiciels ayant une interface de programmation comme Grasshopper ou Dynamo. La citation ci-dessous résume bien les arguments en faveur de cet outil:

  «Pourquoi préférer Python aux autres langages? Python est un langage facile à apprendre et son code est plus lisible, il est donc plus facile à maintenir. Il est parfois jusqu'à 5 fois plus concis que le langage Java par exemple, ce qui augmente la productivité du développeur et réduit mécaniquement le nombre de bugs. L'environnement python est riche en librairies. Vous trouverez toujours des projets open source qui vous faciliteront la vie.»[2]

- Les bibliothèques python, dont TensorFlow

  Les nombreuses bibliothèques qui enrichissent le langage python, permettent de réaliser avec le même langage des tâches spécialisées de manière compacte et modulable. Pour notre expérience, on utilise par exemple les librairies matplotlib (pour le tracé

Logo de python, langage de programmation

Logo de matplotlib, une bibliothèque spécialisée pour les tracés 2D, utilisée par exemple en infographie.

Logo de TensorFlow, une bibliothèque spécialisée pour le calcul numérique.

Logo de Keras, une bibliothèque intégrée à Tensorflow permettant une mise en oeuvre simplifiée de réseaux de neurones.

1. *Plugin développé par le développeur Proving Ground, compatible avec les logiciels Rhino ou Revit à travers leur interface de programmation respectives Grasshopper et Dynamo. Cet outils développé pour les concepteurs, pourrait faire l'objet d'une recherche plus approfondie. Source: https://provingground.io/tools/lunchbox/*
2. *Source: http://apprendre-python.com/ A. Géron, Deep Learning avec TensorFlow - Mise en œuvre et cas concrets. Dunod, 2017.*

géométrique des plans), numpy (pour les manipulations matricielles) ou encore os (pour les imports et exports de documents). La librairie spécifiquement utilisée pour mettre en place le réseau de neurone s'appelle TensorFlow.

«Tensorflow est une bibliothèque logicielle open source puissante destinée au calcul numérique. Elle est particulièrement bien adaptée et optimisée pour l'apprentissage automatique à grande échelle. Conçue pour être flexible, utilisable à toutes les échelles et prête à l'emploi en production.»[3]

- La communauté python et TensoFlow sur GitHub

  Le choix d'un langage de programmation se fait aussi par rapport à la dimension de la communauté qui participe à l'enrichir et qui garantit notamment les mises à jour nécessaires. De ce point de vue, python semblait également être un choix judicieux. Sur la plateforme de partage GitHub, on trouve notamment beaucoup d'extraits de codes sur des problèmes très similaires à celui qui nous concerne. On y trouve par exemple le code du projet *Pix2Pix*[4], une application open source qui permet de convertir une image d'entrée en une autre image (par exemple une façade dessinée en imitation de photographie).

- La bibliothèque Keras

  Keras est une interface de programmation applicative (API) de haut niveau, permettant une implémentation rapide de réseaux de neurones, grâce à une optimisation intégrée des hyperparamètres. Pour une recherche plus fine, ce type d'outil peut être limité car optimisé de manière automatique. Toutefois, dans le cadre de notre expérimentation, il s'est avérée largement suffisant et a permit d'obtenir rapidement des résultats encourageants, sans nécessiter une grande expérience.

La configuration choisie pour réaliser l'expérience est donc une configuration tout à fait accessible et gratuite, qui permet de se familiariser avec les problématiques propres à la conception d'un système d'apprentissage automatique. Cette expérience de mise en pratique nous a tout particulièrement permit de constater qu'au delà des résultats «techniques» limités obtenus, la mise en place d'une méthode est en réalité la difficulté majeure de l'exercice (et non pas l'apprentissage du langage de programmation, qui fait souvent peur aux non initiés).

3.2 Constituer un jeu de données d'entraînement

La question des données d'entraînement est cruciale pour l'apprentissage machine. Comme précisé précédemment, les réseaux de neurones ne sont devenus une technologie intéressante que très récemment, et ce en grande partie au regard de la quantité de données exponentielle partagée sur le web. On observe d'ailleurs que les domaines dans lesquels l'utilisation de cette technologie s'est le plus développée correspondent aux domaines possédant le plus de données disponibles: la classification de photos ou de vidéos, la traduction de textes, la reconnaissance vocale... La recherche dans ces domaines bénéficie d'ailleurs de nombreuses contributions, dont la mise en place de bases de données étiquetées. Pour notre expérience, plusieurs stratégies s'offraient à nous: rechercher une base de donnée accessible déjà étiquetée (il en existe déjà quelques unes pour les plans d'architecture),

3. *A. Géron, Deep Learning avec TensorFlow - Mise en œuvre et cas concrets. Dunod, 2017, p.47*
4. *https://phillipi.github.io/pix2pix/*

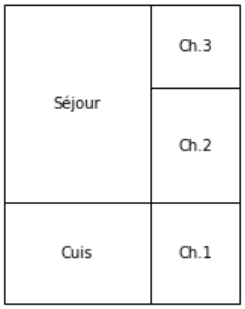
produire notre propre base de donnée manuellement (ce qui implique de sélectionner, puis labelliser chaque plan), ou enfin de générer directement une base de donnée correspondant à nos besoins. L'avantage majeur de cette dernière option, est de pouvoir attribuer automatiquement une étiquette à chaque élément généré, puisqu'il suffit d'extraire l'un des paramètres génératif qui est alors connu. Un autre avantage est de pouvoir à moindre coût obtenir un très large échantillon, et ainsi ne pas être limité par la quantité. Enfin, le fait de générer permet de choisir les paramètres pertinents à faire varier. Par exemple, il n'est pas nécessaire de se préoccuper de la pertinence architecturale des plans, tant qu'on veut apprendre à lire des paramètres géométrique comme le nombre de pièces.

Pour notre expérience, nous avons donc choisi de générer nos données d'entraînement selon un modèle simple. Afin de tester l'effet de différents paramètres sur la performance du réseau, nous avons d'ailleurs généré plusieurs versions de l'échantillon d'entraînement :

- Une version «Basique»,
  qui nous a permit de rapidement tester le réseau, mais dont la faible variété des paramètres implique des biais d'apprentissages importants (distribution des angles discrète limitée, orientation unique des pièces, textes aléatoires).

- Une version «Évoluée»,
  qui nous permet de corriger quelques biais d'apprentissages immédiatement observés avec la version basique, mais qui ne permet toujours pas d'atteindre des résultats sur des plans réels ou trop détaillés (avec des portes, des meubles ou d'autres épaisseurs de trait).

- Une version «Mixte»,
  qui tente d'influencer à la marge le réseau par l'ajout d'un mélange de quelques plans complexifiés manuellement (par un traitement sur Photoshop) et de quelques plans réels, étiquetés manuellement. Si cette dernière base de donnée ne permet pas encore d'obtenir des résultats probants, on verra néanmoins dans quelles conditions elle permettrait de s'en approcher.



plan234_3.png

Exemple type d'un élément issu de la base de donnée «Basique» : chaque élément est constitué par un plan, et une étiquette qui se trouve encodée dans le nom de fichier. Ici, il s'agit du 234ème plan généré dont la classe est la classe 3 (pour 3 chambres).

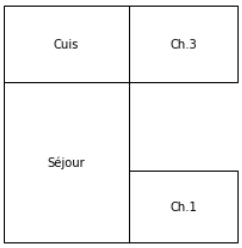
3.2.1 La version «Basique» de la base de données (version B)

Pour générer rapidement des figures géométriques ayant les caractéristiques de base d'un plan, on utilise des règles très simples (une sorte de grammaire de forme[5]), qui consistent à fixer certaines valeurs aléatoires (inclinaison, distance la plus grande, nombre de pièces) puis à déduire par des opérations simples (division, addition, transformation) les sous-paramètres permettant de construire la figure. Ainsi, chacun des plans générés est une version du plan de base (illustré ci-contre), qui lui même est réalisé selon les étapes suivantes:

1. Détermination aléatoire du couple (h0,w0), qui seront les coordonnées du point P0 le plus éloigné. Ce paramètre délimite l'enveloppe dans laquelle se situe le plan.

2. Détermination par déduction d'une liste de point P1, P2, ... P11 qui permettent de tracer le plan «maximal» (un plan qui aurait 3 chambres). Ce découpage se fait avec un degré d'aléa dans les proportions entre les distances, mais selon certains paramètres fixes (qui assurent par exemples que toutes les pièces sont des rectangles).

5. *Cette notion fait l'objet d'une autre partie plus détaillée, et nous a permit d'avoir une méthode de base pour arriver rapidement à un résultat.*



plan34_2.png

Exemple d'un élément issu de la base de donnée «Basique» : Ce cas particulier ne se retrouvera pas dans la version «Évoluée», dans laquelle on s'assurera de nommer les chambres selon le nombre total de chambres présentes. Ici la chambre appelée «Ch.3» devra s'appeler «Ch.2».

3. Détermination aléatoire des paramètres c1, c2 et c3 qui indiquent l'existence ou non de chacune des 3 chambres selon qu'ils prennent la valeur 1 ou 0. On attribuent ensuite aux chambres qui sont effectivement présentes un nom allant de «Ch.1» à «Ch.3».

4. Enfin, détermination aléatoire d'un angle de rotation parmi une liste d'angles possibles, et rotation du plan autour du point de base (0,0).

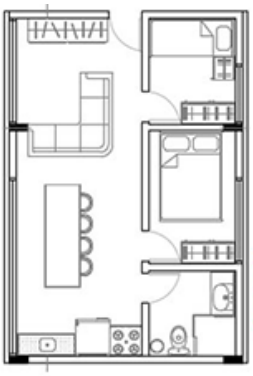
A partir de ces 4 règles ou étapes, on génère donc un nombre souhaité d'éléments, qu'on exporte chacun dans un fichier au format PNG de taille 300x300 pixels et dont le nom comprend l'indice de génération (qui permet de localiser le plan dans la base de donnée) et la valeur du paramètre de classe qui nous intéresse (de valeur 1,2 ou 3).

3.2.2 La version «Évoluée» de la base de données (version E)

Après les premiers essais d'entraînement (qui seront détaillés plus loin), on a rapidement constaté des failles dans la robustesse du système dues à la trop grande homogénéité des données d'entraînement. Par exemple, le réseau de neurones était incapable de reconnaître un plan sur lequel il avait été entraîné avec un taux d'erreur de 0% si celui-ci subissait une simple rotation d'un angle non compris dans la liste des angles possibles. Entre la version basique et la version évoluée, ce sont donc quelques modifications simples qui ont été opérées, permettant d'augmenter la diversité des plans à soumettre au réseau. On a notamment rajouté un degré de symétrie aléatoire, ainsi qu'un intervalle continu d'angles

Extrait de version «Évoluée» de la base de donnée: on peut le remarquer par les symétries, les degrés de rotation ou encore les noms des pièces correspondant au nombre réellement présent. On peut également visualiser la logique d'étiquetage via les noms des fichiers d'export sous chacun des plans.

possibles. Enfin, afin de pousser le réseau à baser sa lecture sur les paramètres textuels, on a fait en sorte que les noms des chambres attribués correspondent bien au nombre réellement présent (c'est-à-dire que le nom «Ch.3» ne soit pas utilisé pour nommer une chambre si il n'y en a que deux par exemple).



plan501_3.png

Exemple d'un élément ajouté manuellement à la base de donnée «Mixte»: il s'agit d'un plan trouvé sur internet[6] et sélectionné pour sa simplicité et sa proximité avec les plans générés. On notera que la salle de bain est ici considérée comme une chambre pour simplifier.

3.2.3 La version «Mixte» de la base de données (version M)

Après d'autres tests d'entraînements sur la version «Évoluée» de la base de données, on a pu constater d'autres limites (qui seront détaillées plus loin), mais on a surtout pu mesurer la distance à parcourir dans le perfectionnement des plans à générer si on voulait avoir une chance de pouvoir lire des plans réels. Finalement, le problème ainsi posé reviendrait à décrire très précisément par une grammaire de forme très aboutie, une grande diversité de plans, ce qui remettrait en cause tout l'intérêt supposé de l'utilisation de l'apprentissage profond, décrit dans les parties précédentes. Cette dernière tentative est donc le début de ce qui mériterait un réel travail de recherche: combiner un apprentissage sur des données générées et sur des données réelles. En effet, le temps imparti pour ce travail de mémoire ne permettait pas le long travail de sélection et d'étiquetage de plans réels en quantités suffisantes. On verra par la suite que l'objectif ne serait d'ailleurs pas plus atteint de cette manière, qui ne fait décaler le problème, et qu'il existe des méthodes alternatives pour traiter le problème des données non étiquetées (reposant sur une combinaison entre l'apprentissage supervisé et l'apprentissage non supervisé). En attendant on a donc «bricolé» cette troisième base de données en ajoutant des éléments manuellement, dans une proportion peu signifiante malheureusement.

Finalement, une fois la base de donnée à utiliser constituée, il reste à formater les données pour l'entraînement: en d'autres termes, il faut les convertir dans un format qui soit lisible par le réseau de neurones. Cet aspect sera détaillé dans la partie suivante, puisqu'elle est conditionnée par la forme même du réseau utilisé.

3.3 Créer et entraîner un modèle

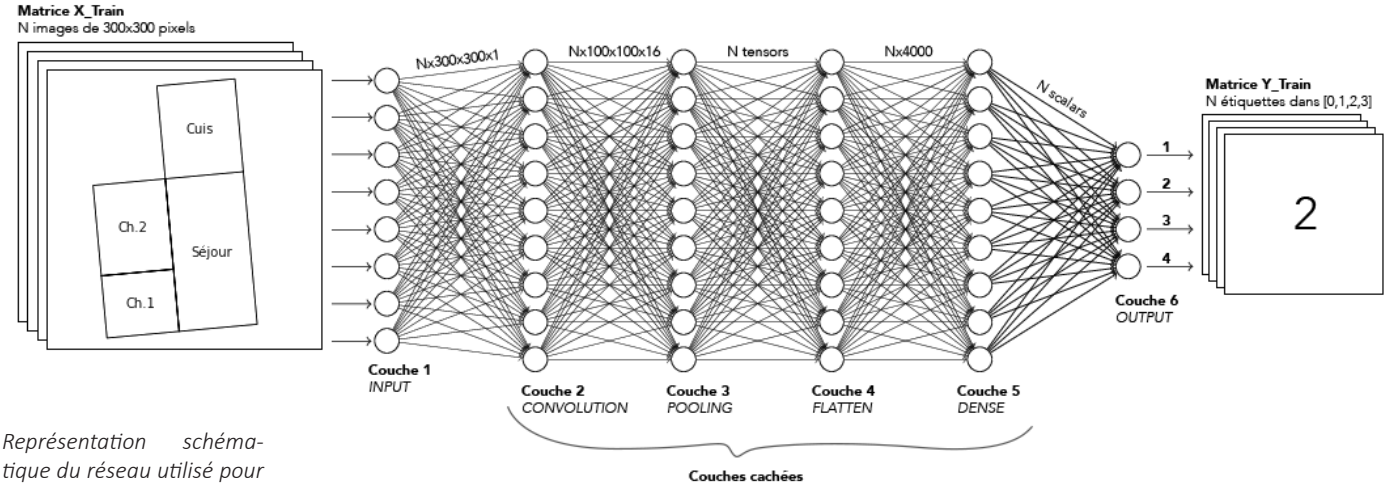
Une fois qu'on dispose d'une base de données exploitable, il faut donc y prélever un échantillon de données d'entraînement ainsi qu'un échantillon de test. En effet, le réseau de neurone utilisera les données d'entraînement pour modifier ses poids (processus dit de *rétro-propagation*), et les données de test serviront à offrir une visualisation de la performance du réseau au cours de l'entraînement, ce qui est surtout utile pour connaître le nombre d'époques nécessaires pour avoir un entraînement satisfaisant. On comprend que seul l'échantillon d'entraînement a un impact sur l'apprentissage du réseau.

3.3.1 Formater les données d'entrée

Comme énoncé précédemment, les données brutes de la base de données doivent tout d'abord être importées puis formatées pour pouvoir être lues par le réseau. Plus spécifiquement, c'est la première couche du réseau qui détermine le format nécessaire (ou inversement, on peut être amené à choisir le type de la première couche adapté au format des données à traiter). Nos plans étant générés, ils sont déjà dans un format homogène (format, dimension et profil colorimétrique identiques), ce qui nous épargne une étape d'harmonisation qui serait nécessaire pour des plans réels. Toutefois il est nécessaire de les convertir en tableaux de valeurs numériques (ou matrice), pour qu'ils puissent être exploités par

6. *« 6º Prêmio Pré-Fabricados para Estudantes – 1° Lugar – Parque Guaianazes / Jhonny Rezende », ArchDaily Brasil, 02-janv-2012. Disponible sur: http://www.archdaily.com.br/br/01-18871/6o-premio-pre-fabricados-para-estudantes-1-graus-lugar-parque-guaianazes-jhonny-rezende.*

le réseau qui fonctionne par opérations matricielles. Enfin, il est également nécessaire de mettre sous forme matricielle les étiquettes (ou labels) décrivant les plans. On produit donc deux matrices d'entraînement X_train et Y_train comprenant respectivement l'ensemble des plans et les valeurs de classe correspondantes (le nombre de pièces, allant de 0 à 4), dans un ordre identique qui permettra au réseau de les associer. On procède de même pour les matrices de test X_test et Y_test. Parmi les paramètres que l'on fait varier pour l'expérience, celui de la taille de l'échantillon d'entraînement (on entraîne le réseau avec 300 puis avec 3000 plans) nous permet d'évaluer la bonne proportion à adopter entre la complexité des objets à classer, le nombre de classes possible et la taille de l'échantillon à soumettre au réseau pour obtenir un degré de reconnaissance satisfaisant.



Représentation schématique du réseau utilisé pour l'expérience: en «entrée», on lui donne en réalité une matrice de valeurs d'entrées associée à une matrice de valeurs de sorties, grâce auxquelles il pourra réaliser son entraînement (on dit que le modèle s'ajuste aux données d'entraînement). On voit aussi les différentes couches qui composent ce réseau (représentation simplifiée), ainsi que le format des entrées et sorties de chacune d'elle.

3.3.2 Choisir la structure du réseau

Pour choisir la structure du réseau adaptée à nos besoins, une façon simple était de partir d'un exemple de base disponible sur le site web de TensorFlow[7], puis de l'adapter en modifiant simplement la nature et le nombre de couches. La seule contrainte est de s'assurer que chaque couche est compatible avec le format de sortie de la couche précédente, et que la dernière sortie est de dimension scalaire (numéro de la classe). On ajoutera par exemple une couche de convolution, dont l'utilisation dans de nombreux exemples (par exemple dans le réseau utilisé par *Pix2Pix*[8]) semble indiquer l'efficacité. Il est d'ailleurs reconnu que les réseaux de neurones dits convolutifs (ayant au moins un couche de convolution), sont très efficaces pour les tâches de classification d'image mais aussi pour la reconnaissance du langage naturel[9]. Mais pour avoir un aperçu général du principe de fonctionnement de notre réseau (pourquoi il fonctionne autrement dit), expliquons le principe et l'utilité de chacune des couches qui le composent:

- La couche d'entrée
  Cette couche est indispensable puisqu'elle a pour fonction de distribuer les entrées sur la couche suivante. Toutefois elle ne subit pas d'ajustement, et ne sert à rien d'autre qu'à permettre au système de lire la matrice des données. Elle ne fait pas partie des couches dites cachées du modèle.

7. *https://www.tensorflow.org/tutorials/*
8. *https://phillipi.github.io/pix2pix/*
9. *A. Géron, Deep Learning avec TensorFlow - Mise en œuvre et cas concrets. Dunod, 2017, p.179*

```
In [62]: import tensorflow as tf
         import tensorflow.keras as k


         model = k.models.Sequential([

           k.layers.Conv2D(16, (3, 3), strides=3, input_shape=(300,300,1),activation='relu')
         ,
           k.layers.MaxPooling2D(pool_size=(2, 2)),
           k.layers.Flatten(),
           k.layers.Dense(4,activation=tf.nn.softmax)
         ])

         model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',
         metrics=['accuracy'])




         model.fit(X_train, y_train, epochs=10,batch_size=8) model.evaluate(X_test, y_test)

         Epoch 1/10
         2000/2000 [==============================] - 10s 5ms/step - loss: 1.1092 - acc: 0.5460
         Epoch 2/10
         2000/2000 [==============================] - 10s 5ms/step - loss: 0.5664 - acc: 0.7955
         Epoch 3/10
         2000/2000 [==============================] - 6s 3ms/step - loss: 0.3797 - acc: 0 .8665
         Epoch 4/10
         2000/2000 [==============================] - 4s 2ms/step - loss: 0.2646 - acc: 0 .9280
         Epoch 5/10
         2000/2000 [==============================] - 4s 2ms/step - loss: 0.1912 - acc: 0 .9510
         Epoch 6/10
         2000/2000 [==============================] - 4s 2ms/step - loss: 0.1300 - acc: 0 .9750
         Epoch 7/10
         2000/2000 [==============================] - 4s 2ms/step - loss: 0.0980 - acc: 0 .9835
         Epoch 8/10
         2000/2000 [==============================] - 4s 2ms/step - loss: 0.0680 - acc: 0 .9915
         Epoch 9/10
         2000/2000 [==============================] - 4s 2ms/step - loss: 0.0570 - acc: 0 .9930
         Epoch 10/10
         2000/2000 [==============================] - 4s 2ms/step - loss: 0.0422 - acc: 0 .9945

         200/200 [==============================] - 1s 3ms/step

Out[62]: [0.29691603004932404, 0.915]
```

Extrait du code qui suffit à définir, entraîner et évaluer le réseau de neurone utilisé dans notre expérience: quelques lignes suffisent pour définir l'ensemble des couches de neurones (surlignées en jaune), et les nombreux paramètres tels que le type de fonction de coût, la fonction d'activation, la méthode d'optimisation ou encore le nombre d'époques d'entraînement. En sortie on peut lire la valeur du coût (loss) et de la précision (accuracy) à chaque étape de l'entraînement (epochs). Ici, on a entraîné le réseau sur les données «Mixtes».

- La couche de convolution
  C'est le bloc de construction le plus important d'un RNC (Réseau de Neurones Convolutif). Dans la première couche de convolution, les neurones ne sont pas connectés à chaque pixel de l'image d'entrée mais uniquement aux pixels dans leur champs récepteurs (c'est à dire une matrice de dimension réduite appliquant un filtre sur toutes les sous-parties de même dimension de l'image, par balayage). Cette architecture, inventée en imitant le fonctionnement du cortex visuel humain[10], permet au réseau de se focaliser sur des caractéristiques de bas niveau dans les premières couches cachées, puis de les assembler en caractéristiques de plus haut niveau dans les couches supérieures. Ici, les motifs à reconnaître étant assez simples, un seul niveau de convolution suffit à obtenir des résultats satisfaisants. Si toutefois on souhaiter améliorer notre sys-

10. *Ibid, p.181*

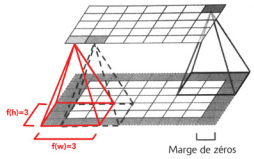


Schéma de principe[10] d'une opération de convolution: ici la dimension de balayage est 3x3 pour une image de 7x7.

tème (cf. Considérations de la partie 4), on gagnerait à multiplier ce type de couches dans la mesure du niveau de diversité de «motifs» à reconnaître. La dimension de balayage est également un paramètre important à adapter suivant le type de motifs à repérer par chaque couche.

- La couche de *pooling*
  Cette couche a pour objectif de réduire la charge de calcul, l'utilisation de la mémoire et le nombre de paramètres (limitant ainsi le risque de sur ajustement). Son fonctionnement est très similaire aux couches de convolution, mais les neurones de *pooling* ne possèdent aucun poids. Elle se contente donc d'agréger les entrées en utilisant une fonction d'agrégation, comme la valeur maximale ou la moyenne. Les autres entrées sont ignorées. Cette couche est très destructrice, à utiliser avec parcimonie dans un réseau plus complexe.

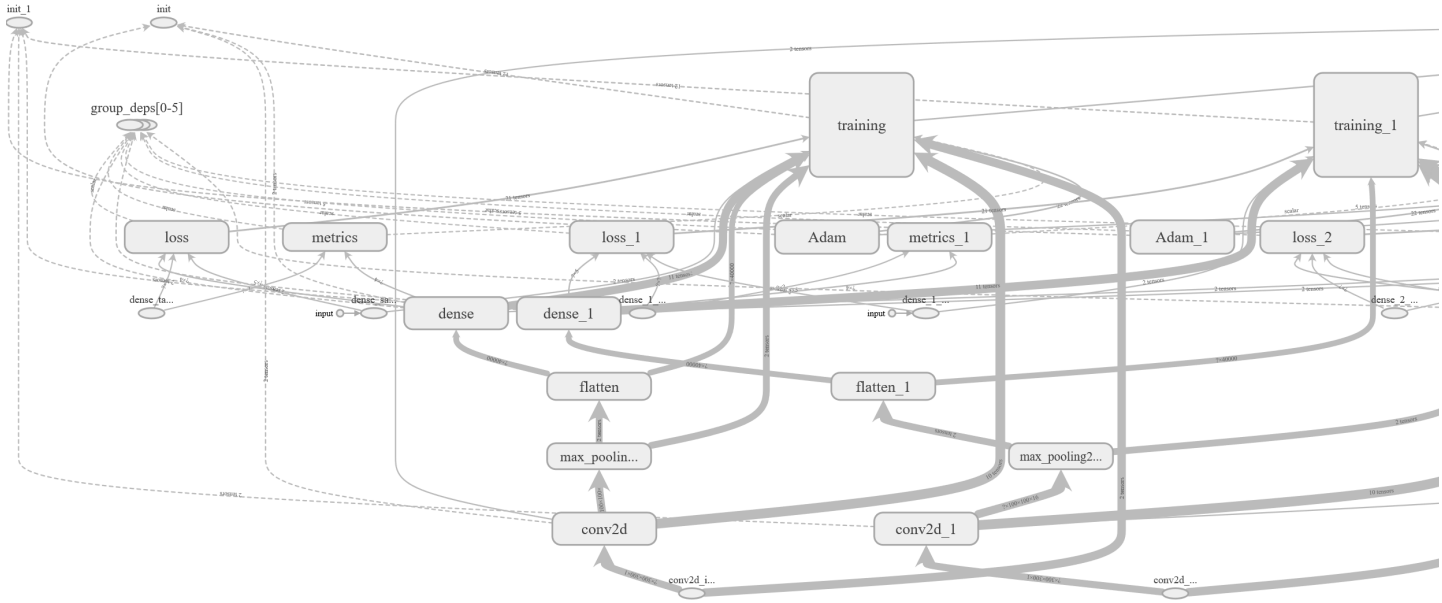
- La couche d'aplatissement (ou *flatten*)
  Cette couche a uniquement pour fonction de convertir la sortie de la couche précédente dans un format lisible par la couche d'activation suivante.

- La couche *dense*
  Cette couche est indispensable pour réaliser la tâche de classification. En effet c'est elle qui permet d'appliquer la fonction d'activation, dont le rôle et de déterminer la classe de sortie. Ici, on utilise une fonction d'activation *«softmax»,* ce qui est la solution standard si on ne veut pas spécialement s'y intéresser (fonctionne dans beaucoup de cas).

- La couche de sortie
  De même que la couche d'entrée, celle-ci sert à distribuer les valeurs de sortie. Elle permet notamment de lancer la rétro-propagation dans l'autre sens, après une évaluation de la réponse donnée à chaque tour par le réseau, par comparaison de celle-ci avec le résultat connu (étiquette donnée par la matrice d'entraînement).

Ainsi on dispose d'un réseau de neurones (ou modèle) assez simple, qui donne des résultats satisfaisants pour la tâche qui lui est demandée. On verra dans la partie 3.5 pourquoi un paramétrage plus fin du réseau n'a pas été nécessaire, dans la mesure où le facteur

Visualisation de la structure de notre réseau de neurone (appelé «graphe»), grâce à TensorBoard, un outil inclus dans TensorFlow qui offre une interface web de visualisation pour visualiser aussi bien la composition du réseau, que les courbes d'apprentissage (voir ci-après). On peut constater la complexité des connexions, pour notre réseau qui est pourtant très basique par rapport à l'état de l'art des CNN (moins de 10 couches cachées). A noter, on ne visualise ici le modèle qu'à un niveau très peu détaillé (absolument pas à l'échelle d'un «neurone» unitaire).



limitant est surtout la question des données d'entraînement et du type de système mis en place (système simple de classification supervisé) pour arriver à nos objectifs les plus ambitieux (lire des plans réels, sans avoir à en étiqueter un grande quantité manuellement).

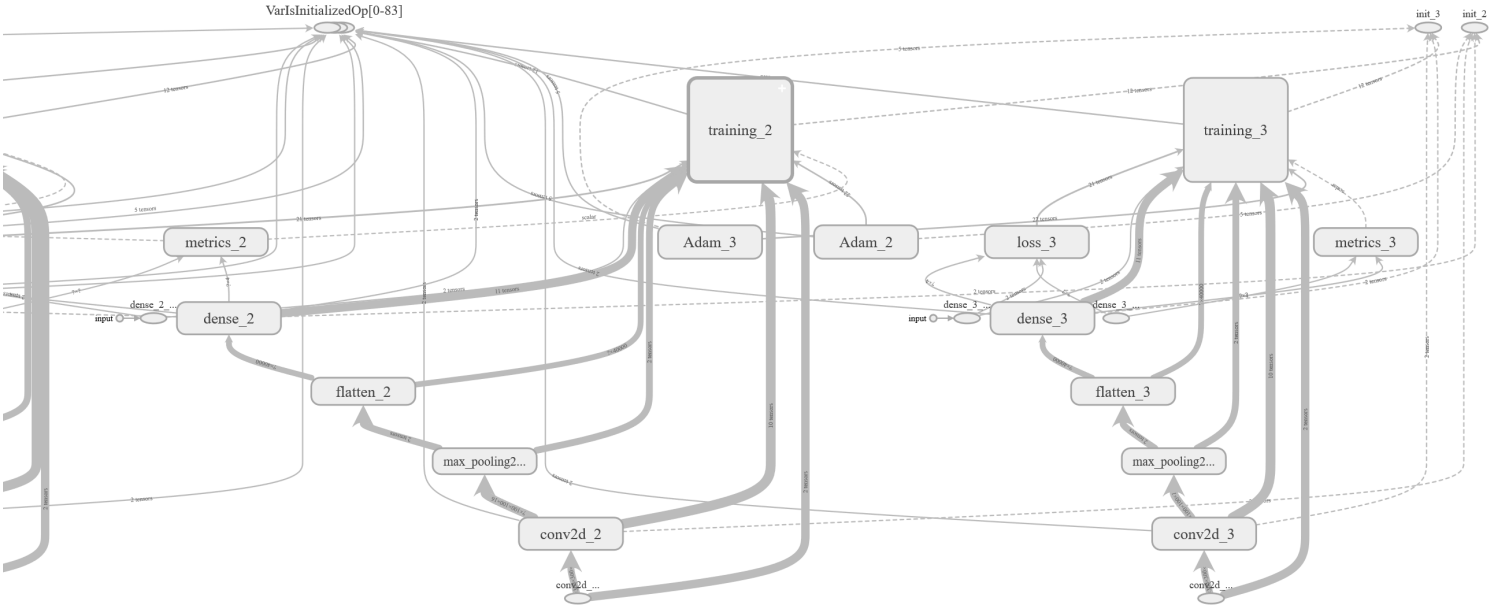
Remarque: la représentation du graphe de notre modèle, visible ci-dessous, ressemble vaguement à un «canvas Grasshopper» d'une certaine complexité. Or le très faible nombre de lignes de code qui en est à l'origine (cf. Illustration) nous évoque l'intérêt que pourraient avoir les réseaux de neurones pour les concepteurs du point de vue de la compacité de la programmation. On peut s'intéresser en la matière à la thèse de Daniel Davis[11] qui étudie le manque de flexibilité de la «programmation visuelle» pour les concepteurs.

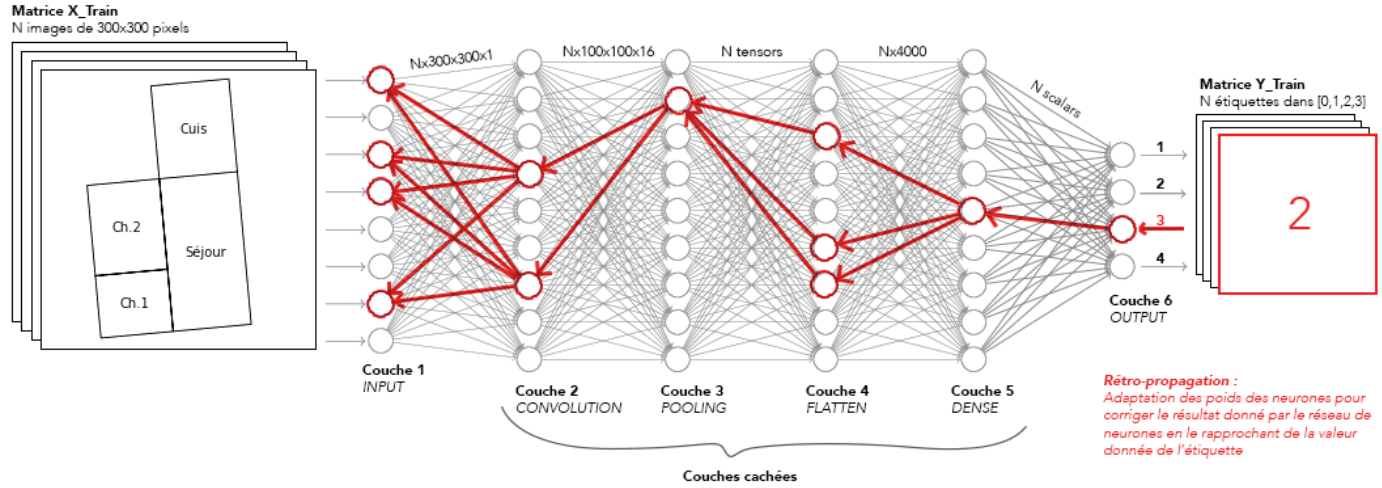
3.3.3 Visualiser l'entraînement

Afin d'apprendre à réaliser la tâche de classification des plans, le modèle réalise un certain nombre «d'époques» d'entraînement. A chaque époque, il s'ajuste pour minimiser son erreur sur un certain nombre de plans, grâce au mécanisme dit de rétro-propagation, dont voici une courte définition[12]:

« L'algorithme de rétro-propagation :
Pendant de nombreuses années, les chercheurs se sont efforcés de trouver une manière d'entraîner les RNA (réseaux de neurones artificiels), sans succès. En 1986, D.Rumlhart et al. ont publié un article révolutionnaire dans lequel ils introduisent un algorithme d'entraînement à rétro-propagation (Learning International Representation by Error Propagation, D.Rumelhart, G.Hinton et R.Williams, 1986). Pour l'exprimer de façon concise : pour chaque instance d'entraînement, l'algorithme de rétro-propagation commence par effectuer une prédiction (passe vers l'avant), mesure l'erreur, traverse chaque couche en arrière pour mesurer la contribution à l'erreur de chaque connexion (passe vers l'arrière) et termine en ajustant légèrement les poids des connexions de manière à réduire l'erreur (étape de descente du gradient).»

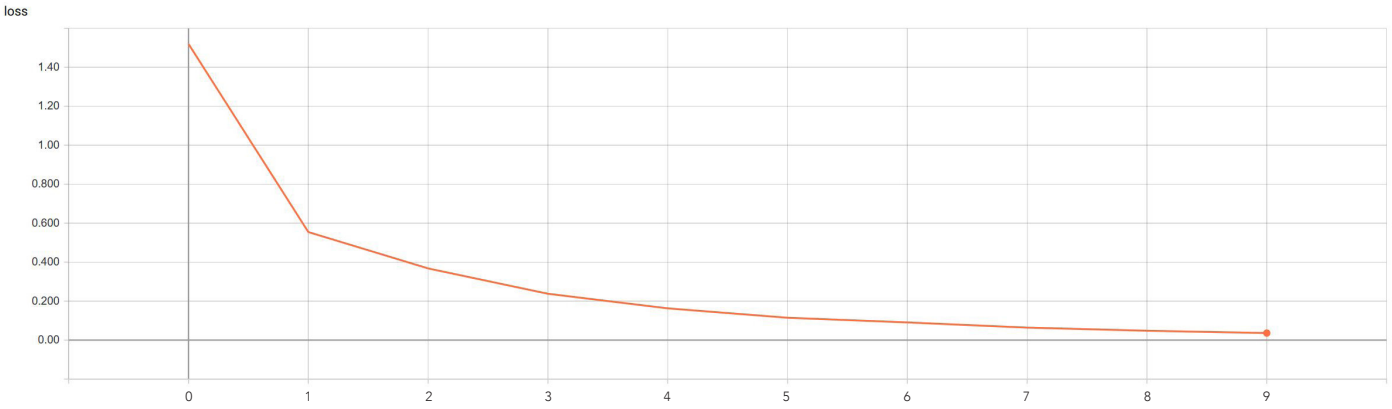
11. *Davis, Daniel. 2013. "Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture." PhD dissertation, RMIT University.*
12. *A. Géron, Deep Learning avec TensorFlow - Mise en œuvre et cas concrets. Dunod, 2017, p.81*



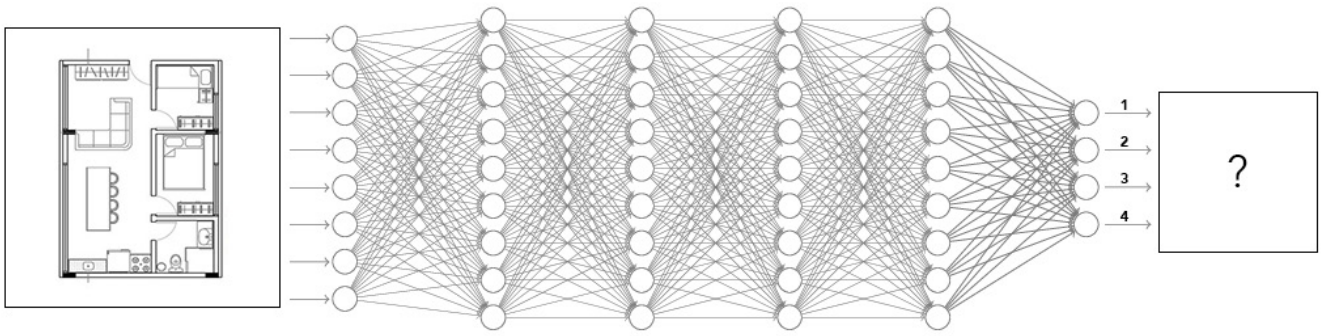


Représentation schématique du réseau utilisé pour l'expérience, pendant la phase d'apprentissage.

Si il n'y a rien à faire pendant l'apprentissage (d'où l'appellation «apprentissage automatique»), il est tout de même indispensable d'avoir une certaine visibilité sur la façon dont le système apprend. En effet, dans la phase de conception du modèle (ou du réseau de neurone), on doit choisir un certain nombre de paramètres afin qu'il puisse apprendre de manière satisfaisante. Pour avoir des indications sur les défauts à corriger, on peut notamment observer la *courbe d'apprentissage*, qui donne l'évolution de la mesure de l'erreur au cours du temps. Cette courbe nous permet par exemple de savoir si l'apprentissage converge ou diverge, mais encore au bout de combien de temps l'erreur est suffisamment faible (pour utiliser le temps d'entraînement optimal). Dans notre cas, la *courbe d'apprentissage* est assez basique et converge plus ou moins rapidement selon la quantité de donnée, ou selon la base de donnée utilisée. Nous n'avons pas observé de grande différence suivant les modalités d'expérience. D'autres indications nous renseignent aussi sur l'optimisation de l'apprentissage, comme la durée d'entraînement de chaque *époque* (cf. Extrait de code précédemment). Enfin, il reste important de vérifier qu'on ne se situe pas dans une situation trompeuse, comme un *sur-ajustement* ou une convergence vers un mauvais optimum.



Courbe d'apprentissage d'un des entraînements réalisés, visualisée via l'outil TensorBoard.



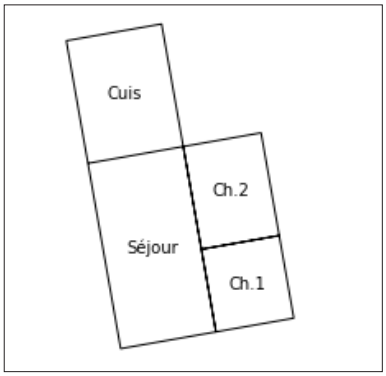
Représentation schématique du réseau utilisé pour l'expérience, pendant la phase d'utilisation.

3.4 Utiliser le modèle

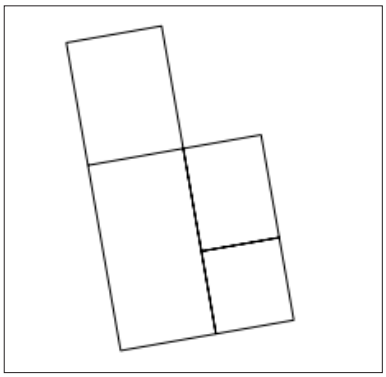
Une fois entraîné, le modèle a surtout été conçu pour être utilisé. Si nous n'avons pas pu faire une analyse très poussée des biais d'apprentissage du réseau dans le cadre de ce mémoire, nous avons utilisé la phase d'implémentation du réseau pour tester sa réaction à différents types de plans plus ou moins éloignés de ce pour quoi il a été entraîné. Ainsi les caractéristiques des plans testés, comparées à la capacité du réseau à les identifier nous a permis de déduire certains paramètres sensibles et d'autres moins sensibles au changement. Les tests étant menés dans l'optique de se rapprocher de la capacité à lire (ou classer) des plans réels, ils consistent essentiellement à des ajouts de détails sur des plans ayant servi à l'entraînement (qui sont donc reconnus avec une incertitude minime par le réseau). Le résultat donné par le modèle, se présente sous la forme des probabilités d'attribution de chacune de 4 classes, dont la plus haute est retenue comme le classement du plan analysé. Pour chaque test, on observera donc non seulement la réponse donnée par le réseau, mais surtout les variations dans les probabilités des différentes classes (voir partir 3.5). Comme on le verra dans la synthèse des résultats, on utilisera les résultats de ces différents tests pour faire évoluer l'expérience. On utilisera notamment les plans modifiés pour tenter d'influencer l'apprentissage, en les intégrant dans une base de donnée mixte (évoquée précédemment) qui servira à son tour à entraîner le réseau.

3.4.1 Utiliser le modèle sur des plans modifiés

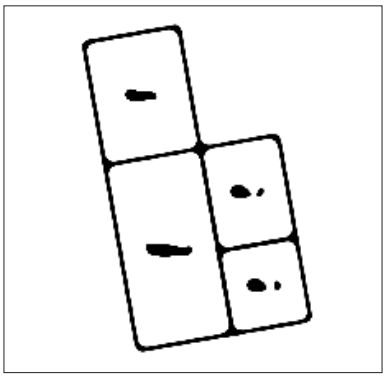
Afin d'introduire une synthèse des résultats des différentes expériences menées , commençons donc par détailler les types de transformations testées. Ici, il s'agit donc de modifications réalisées «manuellement» (avec un logiciel de dessin) sur des plans issus de notre base de donnée générée Cela permet notamment de conserver le même format, ce qui facilite l'opération de formatage afin de les soumettre à l'évaluation du réseau (même conversion matricielle). On distinguera une première «vague» de modifications, dont le test nous a conduit à réaliser la base de donnée «Évoluée», puis quelques modifications supplémentaires destinées à tester plus spécifiquement cette deuxième base de donnée. Si ces nuances seront détaillée lors de la synthèse des résultats, on se contentera ici d'illustrer par catégories l'ensemble des modifications testées dans l'expérience. On a représenté ici côte à côte ces modifications réalisées sur un même plan de départ (voir page suivante), qui a servi d'échantillon témoins pour chacune de nos expériences.



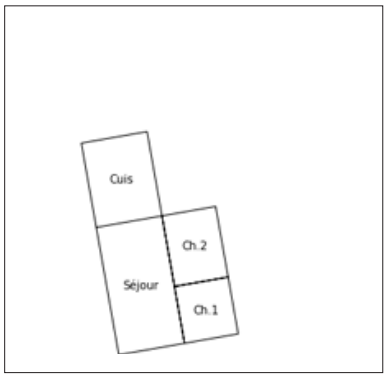
Plan d'origine
Plan sur lequel on applique les modifications décrites ci-contre. Ce plan issu de la base de donnée «Basique», pourrait aussi bien avoir été généré par la deuxième version «Évoluée» de l'algorithme génératif (mêmes caractéristiques).



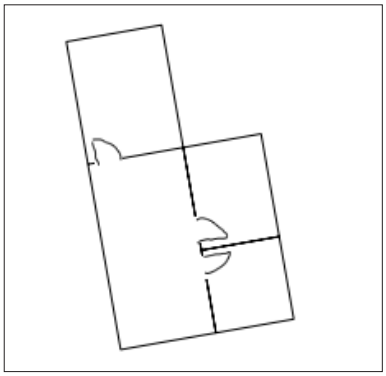
Plan modifié 1
Plan sur lequel on a appliqué la suppression des textes (réduction du nombre d'indices)



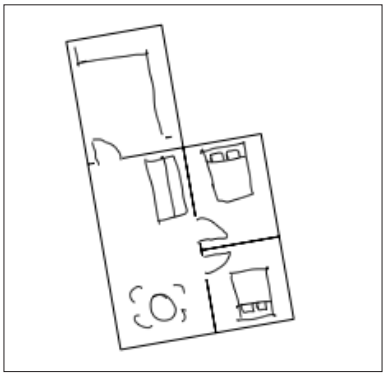
Plan modifié 2
Plan sur lequel on a appliqué le changement de style de trait, en appliquant un filtre «effet photocopie» (modification de l'apparence des indices).



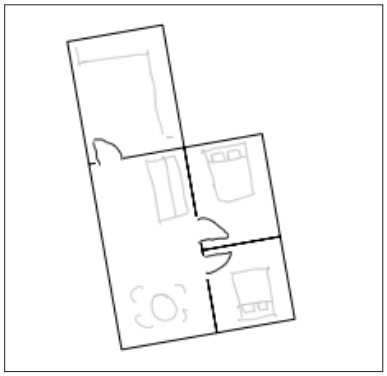
Plan modifié 3
Plan sur lequel on a appliqué le changement d'échelle, ou de cadrage (modification de l'apparence des indices).



Plan modifié 4
Plan sur lequel on a appliqué l'ajout de portes (complexification du plan).



Plan modifié 5
Plan sur lequel on a appliqué l'ajout de mobilier, d'épaisseur de trait identique aux murs (complexification du plan).



Plan modifié 6
Plan sur lequel on a appliqué l'ajout de mobilier, d'épaisseur de trait différente de celle des murs (complexification du plan).

1. Supprimer le texte (réduire le nombre d'indices)

   Les textes qui indiquent le nom des pièces ont été intégré dès la base de donnée «Basique», avec l'idée d'identifier si le réseau préfèrerait s'appuyer sur ces petits éléments plutôt que sur les «motifs» constitués par les murs pour identifier le nombre de pièces. Un premier test intéressant consiste donc à supprimer ces textes, afin de répondre à cette question. Cette opération revient donc à réduire le nombre d'indices (ou de propriétés) sur lesquels le modèle peut se baser pour évaluer le plan.

2. Changer le type de traits (modifier l'apparence des indices)

   Un autre paramètre qui varie d'un plan à un autre malgré les conventions, est le type de traits. Ces variations peuvent être dues à des résolutions différentes, mais aussi à des modes (ou styles) de représentation différents (du fait d'outils de dessins différents par exemple). Il est donc important de tester l'effet la modification des styles de traits. Il existe bien sûr de multiples manières d'obtenir un style de ligne différent, ici on a choisi d'appliquer un filtre «effet photocopie».

3. Changer l'échelle (modifier l'apparence des indices)

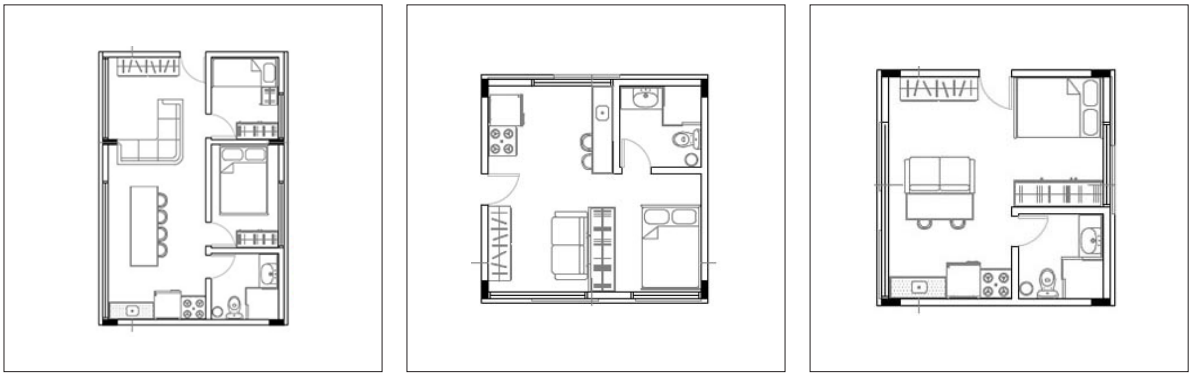
   Dans l'optique de traiter des plans réels issus d'internet par exemple, on peut envisager que le cadrage des plans récoltés varie beaucoup. Dans l'idée d'automatiser la récolte d'un grand nombre de plans, on doit pouvoir maîtriser ce paramètre: soit en automatisant un cadrage optimal des plans récoltés, soit en entraînant le réseau de sorte qu'il soit insensible à un changement d'échelle. Comme on le verra dans les résultats, cette deuxième option n'est pas si évidente. Elle pose aussi la question encore plus complexe de l'échelle des plans, qu'il serait intéressant d'apprendre à reconnaître de manière automatique...

4. Ajouter des portes (complexifier le plan)

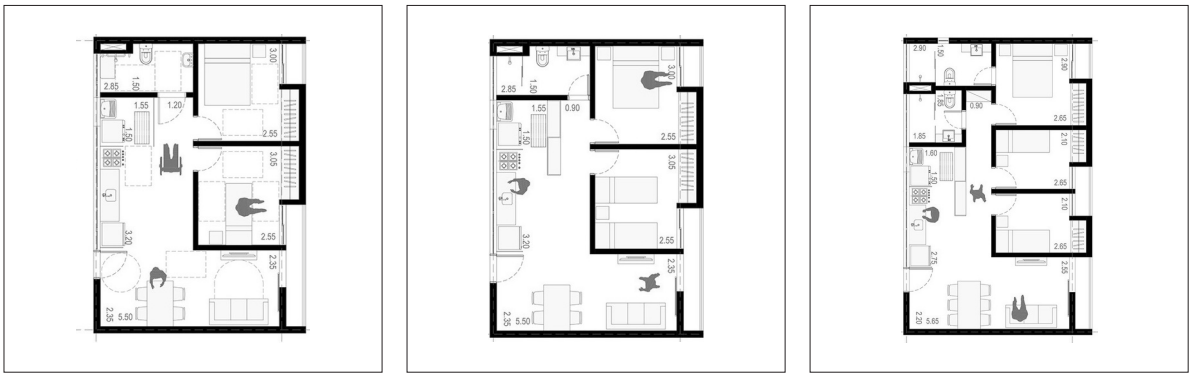
   Dans la perspective de pouvoir évaluer des plans réels, on va ensuite procéder à des transformations consistant à complexifier (ou enrichir) le plan, afin de le rendre plus réaliste. On testera ainsi la «robustesse» du réseau à diversité des modes de représentation, mais surtout sa capacité à extraire des «motifs» importants, sur lesquels il aura été spécifiquement entraîné. Cette première complexification qui consiste à ajouter des portes n'est pas anodine, puisqu'elle implique d'interrompre les lignes des murs tout en ajoutant des «motifs» perturbateurs (de forme arrondie).

5. Ajouter des meubles (complexifier le plan)

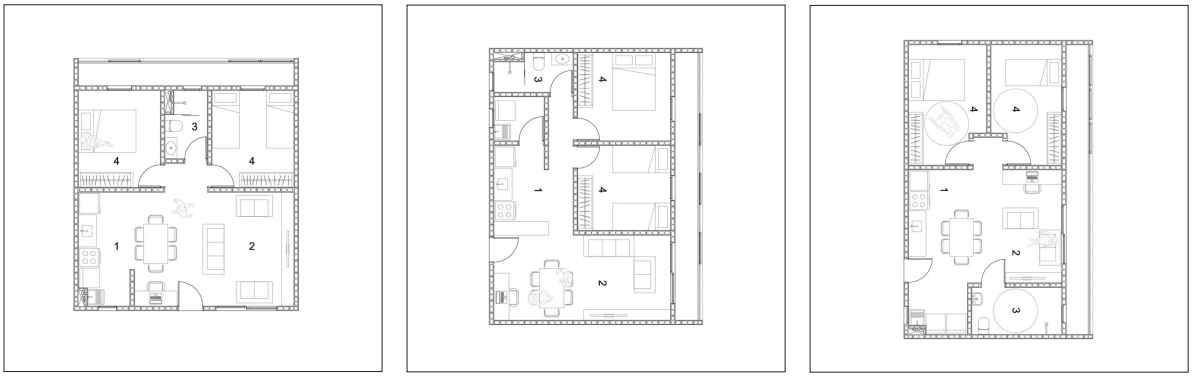
   Dans la même logique que pour l'ajout des portes, on ajoute ensuite un degré de complexité supplémentaire en dessinant des éléments de mobilier (à une échelle approximative, mais visuellement réaliste). On distinguera ici deux versions de la transformation: l'une en conservant le même type de trait (épaisseur et couleur identique), et l'autre avec des traits distinctifs (couleur atténuée). En effet, dans l'hypothèse ou le réseau s'appuie sur les motifs constitués par les lignes de mur, on peut craindre que les éléments de mobilier d'une échelle comparable (les lits par exemple), soient interprétés comme des murs. D'ailleurs, toujours dans la perspective de la lecture de plans réels, on notera que les conventions de représentation, quel qu'elles soient, impliquent toujours une distinction assez nette entre les murs et le mobilier.



Série de plans réels N°1
Source: « 6º Prêmio Pré-Fabricados para Estudantes – 1° Lugar – Parque Guaianazes / Jhonny Rezende », ArchDaily Brasil, 02-janv-2012. Disponible sur: http://www.archdaily.com.br/br/01-18871/6o-premio-pre-fabricados-para-estudantes-1-graus-lugar-parque-guaianazes-jhonny-rezende.



Série de plans réels N°2
Source: « Menção honrosa no concurso CODHAB Sol Nascente – trecho 2, por Metamoorfose Studio, Eduardo Martorelli e Bianca de Cillo », ArchDaily Brasil, 24-mars-2017. Disponible sur: http://www.archdaily.com.br/br/805831/mencao-honrosa-no-concurso-codhab-sol-nascente-nil-trecho-2-por-metamoorfose-studio-eduardo-martorelli-e-bianca-de-cillo.



Série de plans réels N°3
Source: « Primeiro Lugar no concurso da Operação Urbana Consorciada Água Branca / Estúdio 41 », ArchDaily Brasil, 22-juin-2015. Disponible sur: http://www.archdaily.com.br/br/768842/primeiro-lugar-no-concurso-da-operacao-urbana-consorciada-agua-branca-estudio-41.

3.4.2 Utiliser le modèle sur des plans réels

Bien qu'il soit très peu probable que le réseau puisse miraculeusement déchiffrer des plans réels, après n'avoir appris que sur nos simili de plans générés (ce serait bien trop beau), on a tout de même tenté de lui faire lire certains plans réels, autant pour combler une certaine frustration due aux limites de notre expérience, que pour en déduire des pistes d'amélioration possibles. Afin de ne pas non plus jeter des «pavés dans la marre», on a soigneusement choisi les plans réels à tester, afin qu'ils remplissent au moins quelques critères propres aux conditions de notre expérience. Les plans choisis sont donc des plans rectangulaires à murs orthogonaux, dont le nombre de pièces ne dépasse pas 5 (dans la mesure du possible). Concernant le nombre de pièces, on notera d'ailleurs que les salles de bain sont souvent des pièces à part entière, alors que les cuisines sont rarement des pièces fermées, ce qui nous éclaire déjà sur une des limites de l'exercice (dont l'ambiguité de la définition de ce qu'est une pièce). Pour simplifier l'expérience, on considérera donc comme «pièce» tout espace rectangulaire délimité par quatre murs, et le nombre de chambre sera déduit du nombre total de pièces, auquel sera soustrait le nombre de 2 (équivalent à l'ensemble séjour+cuisine). Une fois cette convention fixée, on peut évaluer la réponse donnée par le réseau comme sa capacité à reconnaître les «motifs» qui correspondent à des murs. On notera aussi que les plans sélectionnés sont regroupés par familles, au nombre de trois. En effet, nos recherches nous on conduit à trouver des séries de plans similaires et compatibles avec nos critères, ce qui permet de tester des variations d'une série à l'autre, tout en multipliant facilement le nombre d'éléments pouvant être ajoutés dans notre dernière base de donnée «Mixte». On utilisera également une dernière astuce consistant à modifier manuellement ces plans pour augmenter encore notre échantillon de plan réels (on appliquera par exemple des rotations, des symétries, mais encore des découpages ou collage de pièces pour modifier leur nombre). Cette astuce permettra aussi de supprimer les pièces en excès si nécessaire (se référer à l'échantillon de plans réels «augmenté» illustré en annexe 3).

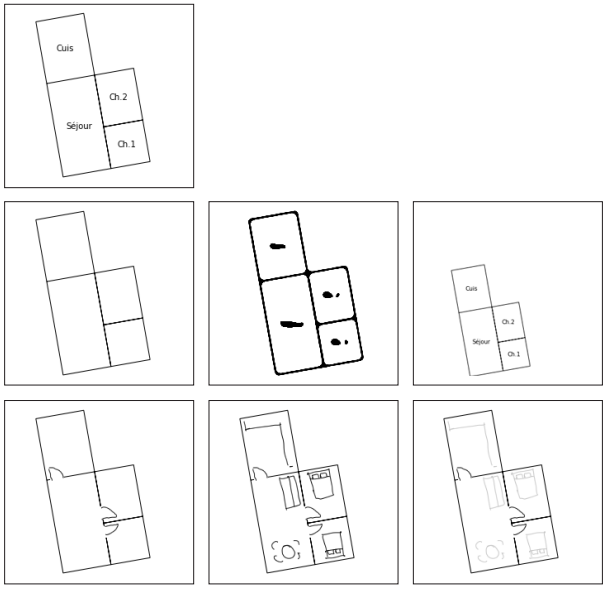
Remarque: Il est important de préciser que les plans utilisés pour «augmenter» la base de donnée «Mixte» ne seront pas ceux utilisés pour tester le réseau (on parle ici des tests «manuels» en mode utilisation, et non des tests qui permettent le suivi du taux d'erreur durant l'apprentissage).

3.5 Synthèse des résultats

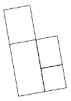
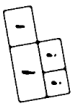
Pour finir cette partie, on analysera une série de tests réalisés afin de «visualiser» les compétences acquises par le modèle, selon qu'il ait été entraîné sur la base de donnée «Basique», la base de donnée «Évoluée» ou la base de donnée «Mixte». Bien entendu, ces tests ne constituent pas une analyse complète ni exhaustive des capacités et des limites du réseau, puisqu'elles sont menées sur un échantillon assez peu représentatif (d'un effectif de 16 éléments). On notera aussi un aspect important qui n'est pas visible dans cet «extrait de résultat»: les résultats donnés par le modèle changent dès qu'il subit un nouvel entraînement, et ce même si c'est sur le même échantillon. En effet, l'entraînement comprend une phase d'initialisation aléatoire qui explique en partie ces différences. Il ne faut donc pas prendre les résultats présentés das les tableaux suivants pour une évaluation ferme et définitive. Enfin, on comprendra que la mise «au propre» de ces résultats étant un travail conséquent, il n'a pas été possible de récolter des données aussi précises sur l'ensemble des expériences réalisées durant le processus de recherche. Certains résultats ayant contribué à des évolutions importantes de l'expérience, tels que ceux ayant conduit à la constitution d'une nouvelle base de donnée plus évoluée n'ont en effet pas été relatés ici. Voici donc une présentation partielle des résultats, qui malgré ses limites, a l'avantage d'être structurée afin de pouvoir comparer des expériences menées dans des conditions identiques.

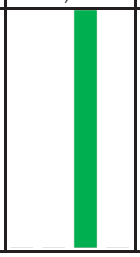
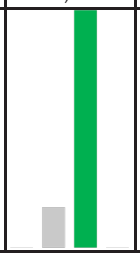
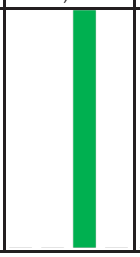
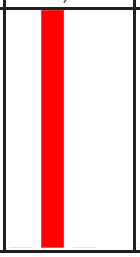
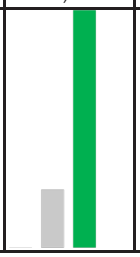
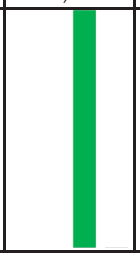
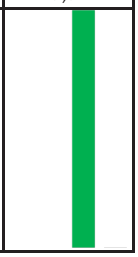
3.5.1 Série de tests sur plans modifiés

Les plans testés ici n'ont pas été ajouté dans les bases de données, ils ont simplement été utilisés comme tests de robustesse à différents types de transformations. A propos du plan original utilisé pour subir les modifications, il a été choisi pour être le plus quelconque possible, et pouvant être trouvé dans chacune des bases de données. Pourtant il semble qu'il n'ait pas été correctement identifié par le deuxième réseau, entraîné avec la base de donnée E. L'une des hypothèses qu'on peut avancer, (et qui expliquerait aussi plus généralement les mauvaises performance de ce réseau sur l'ensemble des tests) est que la grande diversité des plans sur lesquels il s'est entraîné nécessitait un effectif plus important de données ou encore un nombre d'époques d'entraînement plus grand que 10. En effet, un indice qui abonde dans ce sens est le «taux de réussite à l'entraînement» donné par le réseau (et son processus de suivi interne de l'entraînement, par test sur un échantillon dédié): au bout de 10 époques d'entraînement, ce réseau présente un «taux de réussite à l'entraînement» de 90%, ce qui est relativement faible en comparaison aux 98% du réseau entraîné sur la base de donnée B. Mais à ces considérations s'ajoutent deux observations: d'une part le réseau entraîné sur la base M ayant de meilleurs résultats, affiche pourtant un «taux de réussite à l'entraînement» de 89%, et d'autre part un «taux de réussite à l'entraînement» très élevé peut être le signe d'un sur-ajustement et donc paradoxalement d'une mauvaise performance (bien que l'usage d'un échantillon de test d'une taille suffisante permet normalement de s'en prémunir). Pour approfondir ces considérations, on pourra se référer à l'annexe 7.3 qui rassemble les courbes d'apprentissage, ainsi que le valeurs du suivi de l'entraînement de chaque expérience. On peut donc observer beaucoup de choses à partir de ces quelques résultats, bien que ça ne reste à cette échelle que des hypothèses. Il est donc intéressant d'établir cette sorte de «test témoin» pour s'en servir dans la conception d'un système d'apprentissage, et cette méthode pourrait être approfondie.

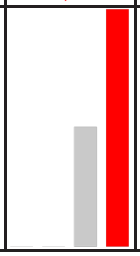
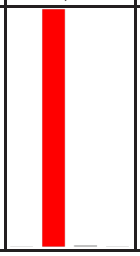
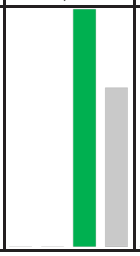
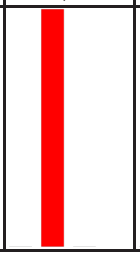
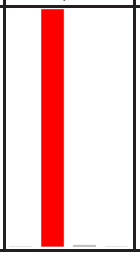
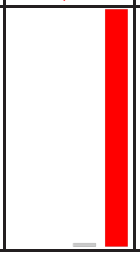
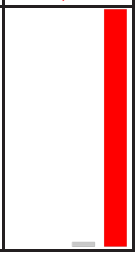


Série test de plans modifiés
Pour se faire une idée des capacités du réseau entraîné, et tenter de comprendre un peu sa «manière d'apprendre», on l'utilise pour prédire (ou évaluer) le nombre de pièce des plans de cet échantillon témoin, issu de la modification de plans générés par notre algorithme.

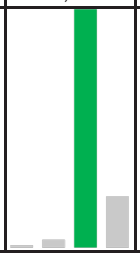
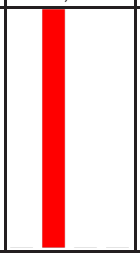
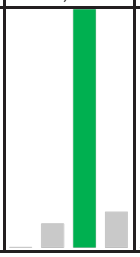
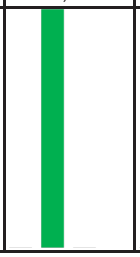
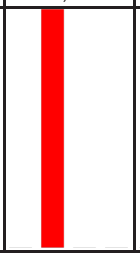
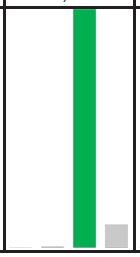
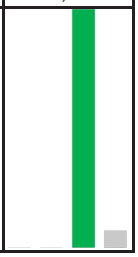
	Plans modifiés manuellement						
	Original	Modifié 1	Modifié 2	Modifié 3	Modifié 4	Modifié 5	Modifié 6
Image							
Réponse attendue	2	2	2	2	2	2	2

Réponses du modèle entraîné avec la base de donnée BASIQUE							
0	1,38E-05	4,92E-04	3,76E-05	1,55E-07	5,40E-05	4,54E-10	5,22E-10
1	2,44E-05	1,45E-01	4,96E-04	1,00E+00	1,96E-01	2,48E-09	2,90E-09
2	1,00E+00	8,54E-01	9,99E-01	7,70E-08	8,04E-01	9,99E-01	9,99E-01
3	1,64E-04	4,00E-08	1,09E-04	8,36E-16	2,98E-09	8,81E-04	7,74E-04
Histogrammes (échelle relative)							

Taux de réussite= 86%

Réponses du modèle entraîné avec la base de donnée EVOLUEE							
0	1,46E-04	9,59E-05	2,09E-05	7,70E-07	7,39E-06	4,97E-15	8,89E-15
1	4,93E-04	9,94E-01	6,08E-04	1,00E+00	9,94E-01	1,19E-10	1,18E-10
2	3,35E-01	5,72E-03	5,99E-01	1,36E-07	6,00E-03	1,41E-02	1,92E-02
3	6,64E-01	4,49E-05	4,00E-01	5,51E-12	1,16E-05	9,86E-01	9,81E-01
Histogrammes (échelle relative)							

Taux de réussite= 14%

Réponses du modèle entraîné avec la base de donnée MIXTE							
0	1,14E-02	9,35E-05	1,18E-03	1,58E-05	1,78E-06	2,14E-07	2,07E-07
1	2,90E-02	9,99E-01	7,99E-02	1,00E+00	1,00E+00	2,54E-03	1,22E-03
2	7,88E-01	4,37E-04	8,00E-01	4,14E-08	4,67E-04	9,10E-01	9,33E-01
3	1,72E-01	3,35E-07	1,19E-01	9,86E-14	3,69E-08	8,74E-02	6,62E-02
Histogrammes (échelle relative)							

Taux de réussite= 71%

Tableau de résultats des tests sur plans modifiés

Pour chaque plan, la réponse du modèle se décompose en quatre valeurs, chaque valeur étant la probabilité d'appartenir à une classe. La prédiction du modèle, c'est à dire la probabilité la plus haute est soit vraie (couleur verte), soit fausse (couleur rouge) par rapport à la valeur attendue.

3.5.2 Série de tests sur plans réels

Dans cette phase de l'expérience on réalise cette fois des test sur des plans réels. On notera que l'expérience est réalisée «dans la foulée» de la phase précédente, afin de comparer non pas six, mais bien trois réseaux de neurones ne différant que par la base de donnée sur laquelle ils sont entraînés (c'est à dire qu'un réseau une fois entraîné est soumis à la suite aux deux séries de test sur plans modifiés, et sur plans réels). Cela permettra entre autre de tirer des conclusions sur les «scores totaux» de chacun des réseaux, en plus de les comparer sur chacun des tests séparément. Lorsqu'on observe donc plus particulièrement les résultats de cette phase de test, la première chose qu'on peut remarquer est la progression dans la capacité à classer les plans réels que permet la base de données M. Cela semble confirmer nos hypothèses, et surtout il est intéressant de noter qu'une proportion de 36 plans réels sur 2000 plans d'entraînement au total (soit moins de 2%), permet d'influencer notablement la capacité du réseau. Ce résultat est intéressant, bien qu'il doive être nuancé, en tenant compte du fait qu'il a fallu non seulement étiqueter, mais aussi modifier «manuellement» ces plans un à un pour augmenter artificiellement notre échantillon. Ce processus est non seulement laborieux, mais il présente aussi quelques biais, notamment en ce qu'il limite la diversité des styles de graphisme auquel le réseau est soumis. Un autre biais tout aussi problématique est l'ambiguïté qu'on peut avoir pour étiqueter ces plans: en effet, notre définition du nombre de pièces basée sur la logique de notre générateur, se retrouve vite confrontée à la grande diversité des cloisonnements et des types d'organisation possibles des pièces. De ce point de vue, si le but est d'apprendre au système à «lire» des murs, il faudra un travail beaucoup plus subtil et un générateur bien plus diversifié. On comprend donc par cet exemple toute la difficulté qu'il y a à vouloir identifier ne serait-ce qu'un paramètre aussi simple qu'un nombre de pièces. On peut donc s'amuser à analyser dans le détail les nuances de ces résultats, mais il est surtout intéressant de prendre du recul et d'identifier les perspectives d'améliorations possible pour ne pas tomber à nouveau dans l'impasse du «problème de la formulation explicite» évoqué en introduction.

Série test de plans réels
Pour chaque famille de plans réels présentée en partie 3, on a sélectionné trois configurations issues de modifications «manuelles» (chacune correspondant aux nombres de pièces 1, 2 et 3), afin de les utiliser comme échantillon témoin. Cet échantillon complétera celui constitué par les plans générés modifiés (page précédente), pour l'évaluation des performance du réseau selon les différents entraînements. On notera que ces plans particuliers n'ont pas été ajoutés à la base de donnée «Mixte», afin d'éviter de biaiser les résultats prédictifs.

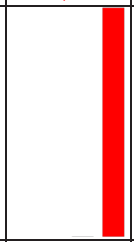
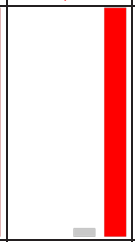
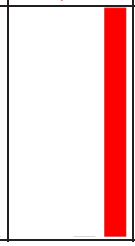
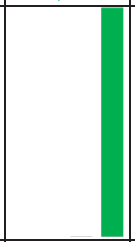
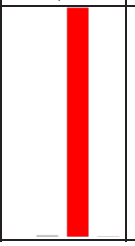
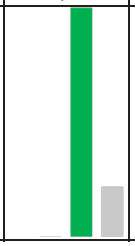
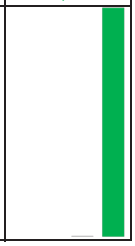
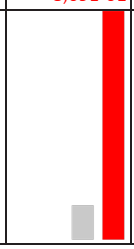
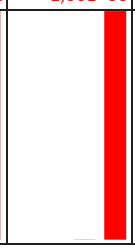
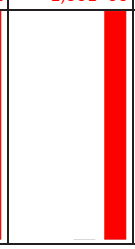
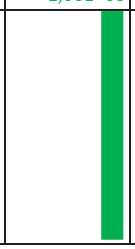
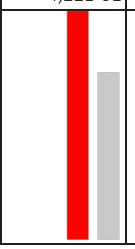
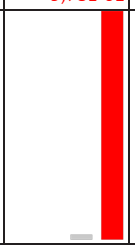
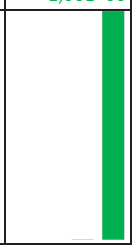
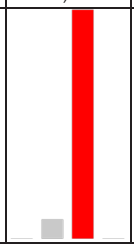
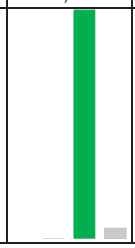
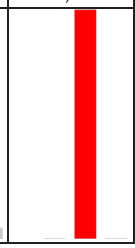
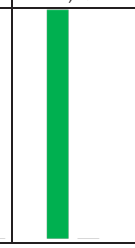
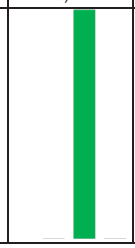
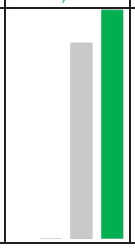
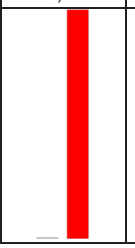
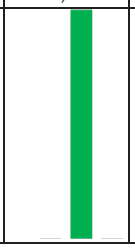
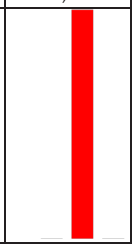
	Plans réels								
	Famille 1			Famille 2			Famille 3		
Image									
Réponse attendue	1	2	3	1	2	3	1	2	3
Réponses du modèle entraîné avec la base de donnée BASIQUE									
0	1,12E-15	9,76E-18	4,96E-19	1,26E-08	9,88E-22	5,58E-24	1,99E-10	1,35E-13	9,05E-19
1	1,43E-20	1,09E-18	2,42E-16	8,96E-06	2,02E-17	6,65E-22	5,21E-03	9,19E-05	4,41E-12
2	5,89E-04	3,46E-02	1,05E-01	9,98E-01	1,87E-04	6,82E-07	9,95E-01	8,20E-01	2,50E-03
3	9,99E-01	9,65E-01	8,95E-01	1,59E-03	1,00E+00	1,00E+00	1,49E-05	1,80E-01	9,98E-01
Histogrammes (échelle relative)									
Taux de réussite= 44%									
Réponses du modèle entraîné avec la base de donnée EVOLUEE									
0	8,85E-12	1,09E-24	8,62E-27	7,87E-17	1,19E-30	3,25E-32	1,91E-15	7,21E-24	9,25E-27
1	6,89E-13	1,18E-23	3,75E-27	4,30E-12	1,83E-24	1,12E-32	9,02E-10	1,57E-18	4,52E-26
2	1,31E-01	1,22E-06	6,07E-07	2,50E-02	6,82E-07	4,22E-10	5,78E-01	2,21E-02	1,84E-04
3	8,69E-01	1,00E+00	1,00E+00	9,75E-01	1,00E+00	1,00E+00	4,22E-01	9,78E-01	1,00E+00
Histogrammes (échelle relative)									
Taux de réussite= 33%									
Réponses du modèle entraîné avec la base de donnée MIXTE									
0	2,43E-04	3,11E-11	2,11E-15	9,47E-10	6,93E-18	1,17E-14	2,23E-12	9,27E-18	1,63E-14
1	7,75E-02	8,87E-04	9,46E-07	1,00E+00	5,46E-06	7,15E-07	6,15E-03	3,25E-06	1,44E-04
2	9,21E-01	9,55E-01	1,00E+00	1,85E-05	9,99E-01	4,61E-01	9,94E-01	1,00E+00	9,98E-01
3	9,93E-04	4,42E-02	2,35E-04	3,37E-10	7,29E-04	5,39E-01	7,99E-08	3,07E-06	1,59E-03
Histogrammes (échelle relative)									
Taux de réussite= 56%									

Tableau de résultats des tests sur plans réels
Pour chaque plan, la réponse du modèle se décompose en quatre valeurs, chaque valeur étant la probabilité d'appartenir à une classe. La prédiction du modèle, c'est à dire la probabilité la plus haute est soit vraie (couleur verte), soit fausse (couleur rouge) par rapport à la valeur attendue.

PERSPECTIVES

Vers un apprentissage et une utilisation sur des plans réels

4| PERSPECTIVES

4.1 Pistes d'amélioration possibles

Comme on a pu le remarquer à travers la synthèse de notre expérimentation, nous sommes encore loin d'atteindre l'objectif qui est de déchiffrer des "caractéristiques architecturales" sur des plans réels quelconques. Pourtant il existe de nombreuses techniques qui permettraient d'améliorer notre système, et en particulier toutes celles qui permettent de réaliser un pré-entraînement non supervisé du réseau. En effet, la limite principale de notre expérience réside dans la questions des données d'entraînement dont on dispose. Si le fait de générer certaines données afin de "diriger" l'apprentissage pour qu'il reconnaisse les critères souhaités reste intéressante, on ne peut absolument pas envisager d'aller dans un détail tel qu'on obtienne des plans quasiment réels (cela reviendrait à encoder de manière extrêmement précise un ensemble de plans qu'on souhaite le plus diversifié possible). Il n'est pas non plus envisageable d'étiqueter manuellement des centaines de plans réels pour "augmenter artificiellement" notre base de donnée (notre but étant d'avoir un système généralisable). Une alternative serait donc de pré-entraîner le réseau sur une grande quantité de plans réels de manière non supervisée (car on ne disposera jamais de plans étiquetés avec exactement le paramètre recherché). Cette solution présente un avantage majeur: une fois pré-entraîné sur une grande quantité de plans, un petit échantillon étiqueté suffit à orienter l'apprentissage du réseau qui saura déjà reconnaître des motifs communs dans les plans (tels que des murs ou des textes)[1]. Cette méthode de pré-entraînement est rendue possible grâce au principe de transfert d'apprentissage: ce principe stipule que les couches de bas niveau apprenant spontanément à détecter des caractéristiques de bas niveau peuvent être conservées et verrouillées, afin de simplement ré-entraîner les couches les plus élevées du réseau. Reste bien-sûr à bien dimensionner tous les éléments, et notamment le nombre de couches à conserver entre les deux entraînement. Une autre astuce intéressante est le pré-entraînement à partir d'une tâche secondaire, pour laquelle on peut aisément obtenir ou générer des données d'entraînement étiquetées. On pourra par exemple étiqueter comme "bons" tous les plans réels dont on dispose, puis générer des nouvelles instances d'entraînement par altération des bonnes en étiquetant celles-ci comme "mauvaises", puis pré-entraîner le modèle à classer cet échantillon. Enfin, un aspect qu'il serait intéressant d'explorer, serait la visualisation des caractéristiques apprises, afin d'évaluer leur degré d'intelligibilité, et donc leur usage potentiel comme outil d'analyse ou de conception. Cela est possible[2], et il existe des méthodes pour visualiser pour chaque neurone, les instances qui l'activent le plus. On a d'ailleurs pu piéger des réseaux de neurones[3], en utilisant ces "marqueurs", afin de les faire identifier des éléments qui n'étaient pas réels.

4.2 Applications pour la conception

Si l'objectif de départ de cette recherche vise à instrumenter en particulier le processus de conception, il reste encore du chemin à parcourir avant de disposer d'un outil opérationnel, avec une application utile à la pratique de l'architecte dans des conditions réelles. Si on peut désormais espérer avoir un modèle qui apprenne à partir de références, ce qui permettrait par exemple de dépasser les grammaires de formes "manuelles" à partir d'un corpus, cela pose tout de même la question de la transposition entre des projets existants et les paramètres toujours uniques du problème posé. Une autre application inté-

1. *A. Géron, Deep Learning avec TensorFlow - Mise en œuvre et cas concrets. Dunod, 2017, p.251*
2. *Ibid, p.249*
3. *A. Nguyen, J. Yosinski, et J. Clune, « Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images », arXiv:1412.1897 [cs], déc. 2014.*

ressante serait alors l'apprentissage pendant le processus de conception. Comme vu dans certains papiers de recherche de l'état de l'art[4], ce genre de résultat requiert la combinaison de notre système, que l'on peut qualifier de "système d'évaluation", avec un algorithme de type génératif (par exemple, un algorithme génétique). Mais dans l'idée d'utiliser ce genre de système dans les agences d'architecture, il faudrait alors explorer les conditions nécessaires à leur déploiement à l'échelle des agences d'architecture. La possibilité de greffer à un réseau entraîné de manière «généraliste», un petit réseau entraîné de manière spécifique (sur un ensemble de données plus restreint), pourrait alors permettre de dépasser les limites techniques, et notamment offrir des outils personnalisables à petite échelle, correspondant au besoin spécifique du concepteur. Restent les questions autour de la propriété intellectuelle, qui à l'heure de l'open source et de la conception collaborative gagnerait peut-être à être réinterrogé.

4.3 Applications pour la recherche

Si ça ne constituait pas un objectif de notre recherche, les applications possibles pour la recherche en architecture sont pourtant très nombreuses. En effet, on pourrait imaginer utiliser des systèmes d'apprentissage dédiés au classement des archives (comme on le fait déjà par exemple pour les peintures[5]), ou encore à l'analyse des types par analyse comparative de corpus de plans. Mais ces recherches peuvent aussi être utiles concernant la recherche en informatique elle même: en effet, dans le domaine de l'IA, la recherche se nourrit de toutes les exigences des domaines intéressés par le fait de s'approprier cette technologie. Parmi les nombreux domaines touchés par ce développement, l'architecture (ainsi que d'autres domaines créatifs) fait figure d'exception en ce qu'elle relève d'une grande complexité, qui met au défi les techniques les plus avancées. De plus, les supports de représentation graphique étudiés ont la caractéristique d'être à haut degré de sémantique (c'est à dire, contenant beaucoup de conventions signifiantes), ce qui en fait des cas d'étude intermédiaires entre le traitement du langage écrit et le traitement des photos (notons qu'on utilise aussi des plans et des schémas dans d'autres domaines comme l'électronique, qui pourraient bénéficier de telles recherches).

4. *C. Sjoberg, C. Beorkrem, et J. Ellinger, « Emergent Syntax: Machine Learning for the Curation of Design Solution Space », in ACADIA 2017: DISCIPLINES & DISRUPTION [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 552- 561, 2017.*
5. *B. L. A. Seguin, « Making large art historical photo archives searchable », Laboratoire d'humanités digitales, EPFL, 2018*
6. D. Boudet, Nouveaux logements à Zurich: la renaissance des coopératives d'habitat. 2017.

Plan illustrant la couverture du mémoire
Ce plan d'étage de l'opération suisse Hunziker Areal[6] illustre les limites rencontrées lors de cette recherche, comme un rappel que la subtilité du langage architectural ne se laisse pas dompter facilement. En effet, ce plan possède des caractéristiques d'adaptabilités très riches, qu'il serait extrêmement difficile de pouvoir repérer par un système de lecture automatique (on peut par exemple aussi bien aménager l'étage en deux grandes collocations, que le cloisonner pour avoir quatre logements indépendants). Et pourtant cette subtilité est à l'origine même de toute la conception de ce bâtiment, ce qui en fait un caractère fondamental à interpréter.

CONCLUSION

5| CONCLUSION

Dans ce mémoire, on a donc pris le parti de l'expérimentation, afin de pouvoir aborder ce sujet par la pratique, sans quoi il aurait été trop abstrait à traiter. Pour finir sur une note qualitative, ce travail fût très enthousiasmant, bien que les résultats ne soient pas vraiment au rendez-vous, car il balaye beaucoup de questions autour de la nature de la conception architecturale, ainsi que de l'interface homme-machine dans l'activité de conception. Malheureusement, les aspects plus théoriques, notamment sur les typologies, les diagrammes ou encore les grammaires de forme n'ont pu être abordés que brièvement, alors qu'ils constituent avec les objectifs expérimentaux un tout indissociable. De même, on aura pas pris le temps de développer les questions philosophiques et éthiques que soulève ce sujet, car ce n'était pas l'enjeu du présent travail. Toutefois, fort de cette expérience pratique, on peut aujourd'hui relativiser à propos des nombreux fantasmes sur la toute puissance de l'IA. En effet, les tâches créatives restent encore assez difficile à automatiser, et ce pour de multiples raisons qui n'ont pas toutes à voir avec leur "complexité" inhérente. En effet, il semble que la limite principale en terme d'apprentissage machine soit la question des données, tant du point de vue de leur quantité que de leur degré d'étiquetage (sans parler du problème de l'accès et de la propriété intellectuelle).

BIBLIOGRAPHIE

6| BIBLIOGRAPHIE

6.1 Travaux de recherche

[1]
S. Ahmed, M. Weber, M. Liwicki, C. Langenhan, A. Dengel, et F. Petzold, « Automatic analysis and sketch-based retrieval of architectural floor plans », Pattern Recognition Letters, vol. 35, p. 91-100, janv. 2014.

[2]
J. Algeciras-Rodríguez, « TRAINED ARCHITECTONICS », p. 8.

[3]
A. Andia, « Automated Architecture: Why CAD, Parametrics and Fabrication are Really Old News », in Proceedings of the XVII Conference of the Iberoamerican Society of Digital Graphics- SIGraDi: Knowledge-based Design, Valparaiso, Chile, 2013, p. 83-86.

[4]
N. Audebert et al., « Deep learning for urban remote sensing », in 2017 Joint Urban Remote Sensing Event (JURSE), Dubai, United Arab Emirates, 2017, p. 1-4.

[5]
M. Barros, J. P. Duarte, et B. M. Chaparro, « A Grammar-Based Model for the Mass Customisation of Chairs: Modelling the Optimisation Part », Nexus Netw J Nexus Network Journal : Architecture and Mathematics, vol. 17, n° 3, p. 875-898, 2015.

[6]
G. Cleuziou, « Une méthode de classification non-supervisée pour l'apprentissage de règles et la recherche d'information », p. 209.

[7]
L.-P. de las Heras, S. Ahmed, M. Liwicki, E. Valveny, et G. Sánchez, « Statistical segmentation and structural recognition for floor plan interpretation », IJDAR, vol. 17, n° 3, p. 221-237, sept. 2014.

[8]
S. Dodge, J. Xu, et B. Stenger, « Parsing floor plan images », in 2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA), 2017, p. 358-361.

[9]
J. P. Duarte, « Towards the Mass Customization of Housing: The Grammar of Siza's Houses at Malagueira », Environment and Planning B: Planning and Design, vol. 32, n° 3, p. 347-380, juin 2005.

[10]
R. Gadde, V. Jampani, R. Marlet, et P. V. Gehler, « Efficient 2D and 3D Facade Segmentation using Auto-Context », arXiv:1606.06437 [cs], juin 2016.

[11]
R. Gadde, R. Marlet, et N. Paragios, « Learning Grammars for Architecture-Specific Facade Parsing », International Journal of Computer Vision, vol. 117, n° 3, p. 290-316, mai 2016.

[12]
D. Ha et D. Eck, « A Neural Representation of Sketch Drawings », arXiv:1704.03477 [cs, stat], avr. 2017.

[13]
P. Isola, J.-Y. Zhu, T. Zhou, et A. A. Efros, « Image-to-Image Translation with Conditional Adversarial Networks », arXiv:1611.07004 [cs], nov. 2016.

[14]
S. Kacher, « PROPOSITION D'UNE METHODE DE REFERENCEMENT D'IMAGES POURASSISTER LA CONCEPTION ARCHITECTURALE:Application à la recherche d'ouvrages. », p. 238.

[15]
M. Koziński, G. Obozinski, et R. Marlet, « Beyond Procedural Facade Parsing: Bidirectional Alignment via Linear Programming », in 2014 Asian Conference on Computer Vision (ACCV), Singapore, Singapore, 2014, vol. 9006, p. 79-94.

[16]
M. Lin, Q. Chen, et S. Yan, « Network In Network », arXiv:1312.4400 [cs], déc. 2013.

[17]
A. Martinović, M. Mathias, J. Weissenberg, et L. Van Gool, « A Three-Layered Approach to Facade Parsing », in Computer Vision – ECCV 2012, vol. 7578, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, et C. Schmid, Éd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, p. 416-429.

[18]
M. Mathias, A. Martinovic, J. Weissenberg, et L. V. Gool, « Procedural 3D Building Reconstruction Using Shape Grammars and Detectors », in 2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, Hangzhou, TBD, China, 2011, p. 304-311.

[19]
A. Nguyen, J. Yosinski, et J. Clune, « Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images », arXiv:1412.1897 [cs], déc. 2014.

[20]
S. A. Oliveira, « Machine vision algorithms on cadaster maps », p. 27.

[21]
S. A. Oliveira, F. Kaplan, et I. di Lenardo, « Abstract DH2017 Machine Vision algorithms on cadaster plans », p. 6.

[22]
S. Or, K.-H. Wong, Y. Yu, et M. M. Chang, « Highly Automatic Approach to Architectural Floorplan Image Understanding & Model Generation », p. 9.

[23]
W. Peng, F. Zhang, et T. Nagakura, « Machines' Perception of Space: Employing 3D Isovist Methods and a Convolutional Neural Network in Architectural Space Classification », in ACADIA 2017: DISCIPLINES & DISRUPTION [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 474- 481, 2017.

[24]
P. S. Rajpura, H. Bojinov, et R. S. Hegde, « Object Detection Using Deep CNNs Trained on Synthetic Images », arXiv:1706.06782 [cs], juin 2017.

[25]
B. L. A. Seguin, « Making large art historical photo archives searchable », p. 169.

[26]
J. Shin, R. Triebel, et R. Siegwart, « Unsupervised discovery of repetitive objects », in 2010 IEEE International Conference on Robotics and Automation, 2010, p. 5041-5046.

[27]
J. Silvestre et Y. Ikeda, « ARTIFICIAL IMAGINATION OF ARCHITECTURE WITH DEEP CONVOLUTIONAL NEURAL NETWORK », p. 10.

[28]
C. Sjoberg, C. Beorkrem, et J. Ellinger, « Emergent Syntax: Machine Learning for the Curation of Design Solution Space », in ACADIA 2017: DISCIPLINES & DISRUPTION [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 552- 561, 2017.

[29]
G. Stiny, « Introduction to shape and shape grammars », Environment and Planning B: Planning and Design, vol. 7, n° 3, p. 343-351, 1980.

[30]
C. Vargas et D. Alejandro, « Wall extraction and room detection for multi-unit architectural floor plans », Thesis, 2018.

[31]
« Proposition d'un modèle et d'un outil dédiés à la conception morphologique architecturale en phase esquisse », ResearchGate. Disponible sur: https://www.researchgate.net/publication/324474290_Proposition_d%27un_modele_et_d%27un_outil_dedies_a_la_conception_morphologique_architecturale_en_phase_esquisse.

[32]
« Image-to-Image Translation with Conditional Adversarial Networks ». [En ligne]. Disponible sur: https://phillipi.github.io/pix2pix/.

[33]
« PhD Thesis – Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture – Daniel Davis ». Disponible sur: http://www.danieldavis.com/thesis/.

6.2 Ouvrages

[34]
A. Géron, Deep Learning avec TensorFlow- Mise en oeuvre et cas concrets. Dunod, 2017.

[35]
D. Boudet, Nouveaux logements à Zurich: la renaissance des coopératives d'habitat. 2017.

[36]
P. Boudon, Sur l'espace architectural: essai d'épistémologie de l'architecture. Marseille: Parenthèses, 2003.

[37]
M. Carpo, The second digital turn design beyond intelligence. Cambridge, MA: The MIT Press, 2017.

[38]
J.-P. Chupin, Analogie et théorie en architecture: de la vie, de la ville et de la conception, même. Gollion: Infolio, 2013.

[39]
A. Farel, Architecture et complexité: le troisième labyrinthe. Marseille: Editions Parenthèses, 2008.

[40]
J. Lucan, Précisions sur un état présent de l'architecture. Lausanne: Presses Polytechniques et Universitaires Romandes, 2015.

[41]
B. Stiegler, Automatic society. Vol. 1: the future of work. Cambridge: Polity Press, 2016.

[42]
G. Stiny, « Introduction to shape and shape grammars », Environment and Planning B: Planning and Design, vol. 7, n° 3, p. 343-351, 1980.

[43]
K. Terzidis, Permutation Design: Buildings, Texts, and Contexts. London ; New York: Routledge, 2014.

6.3 Guides et tutoriels de mise en pratique

https://github.com

https://keras.io/getting-started/sequential-model-guide/

https://www.tensorflow.org/

https://www.python.org/

ANNEXES

7| ANNEXE

7.1 Code du générateur de plans

In [ ]:
```
# Imports des bibliothèques et des raccourcis utiles
import random
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.path import Path
from matplotlib.patches import PathPatch
import matplotlib.text as mtext
import os
import math

# Chemin d'accès vers les plans
dossier="C:\Users\ProjetMémoire\Data\Plans\SérieEvoluée"
```

In [ ]:
```
# Définition des fonctions de transformation:

def rotate(P,a):
      (x,y)=P
      A= math.radians(a)
      X = math.cos(A)*x - math.sin(A)*y
      Y = math.sin(A)*x + math.cos(A)*y
      return (X,Y)

def symetrie(P,axeV,axeH):
      (x,y)=P
      if axeV>0:
            P=(axeV-(x-axeV),y)
      if axeH>0:
            P=(x,axeH-(y-axeH))
      return P
```

In [ ]:
```
# Génération des plans

nombre_de_plans = 5000

i=0

for i in range (nombre_de_plans):

      # Attribution des indices des pièces

      [c1]=random.sample([0,1], k=1)
      [c2]=random.sample([0,1], k=1)
      [c3]=random.sample([0,1], k=1)
      C=[c1,c2,c3]
      [ch1,ch2,ch3]=[str(),str(),str()]
      Ch=[ch1,ch2,ch3]
      T=['Ch.1','Ch.2','Ch.3']
      N=random.sample([0,1,2], k=3)
      j=0
      for k in range(3):
            if C[N[k]]>0:
                  Ch[N[k]]=T[j] j=j+1
            else:
                  Ch[N[k]]=str()

      # Attribution des variables aléatoires
      [angle]=random.sample([0,0,0,0,0,0,0,0,random.uniform(-30,30)], k=1)
      [w0,h0]=[random.uniform(8,15),random.uniform(10,20)]

      # Composition du nom de fichier
      filename="plan"+str(i+1)+"_"+str(c1+c2+c3)+".png"

      # Liste des distances
      d12=random.uniform(0,w0/7)+w0/2
      d23=w0-d12
      d34=random.uniform(0,w0/7)+h0/4
      d45=d23
      d56=random.uniform(0,h0/10)+h0/3
      d67=d23
      d78=h0-d34-d56
      d89=d23
      d9A=d12
      dAB=d78
```

```
# Axes de symétrie
[axeV]=random.sample([0,(d12+d23)/2], k=1)
[axeH]=random.sample([0,h0/2], k=1)

# Liste des points
P1=(0,0)
P2=(d12,0)
P3=(d12+d23,0)
P4=(d12+d23,d34)
P5=(d12,d34)
P6=(d12,d34+d56)
P7=(d12+d23,d34+d56)
P8=(d12+d23,d34+d56+d78)
P9=(d12,d34+d56+d78)
P10=(0,d34+d56+d78)
P11=(0,d34+d56)
P12=(d12,d34+d56)

# Coordonnées des points
(x1,y1)=P1
(x2,y2)=P2
(x3,y3)=P3
(x4,y4)=P4
(x5,y5)=P5
(x6,y6)=P6
(x7,y7)=P7
(x8,y8)=P8
(x9,y9)=P9
(x10,y10)=P10
(x11,y11)=P11
(x12,y12)=P12

# Positions des textes au centre des pièces

PA=(0.5*(x1+x2), 0.5*(y1+y11))
PA=rotate(PA,angle)
PA=symetrie(PA,axeV,axeH)
(xA,yA)=PA

PB=(0.5*(x1+x2), 0.5*(y11+y10))
PB=rotate(PB,angle)
PB=symetrie(PB,axeV,axeH)
(xB,yB)=PB

PC1=(0.5*(x2+x3), 0.5*(y3+y4))
PC1=rotate(PC1,angle)
PC1=symetrie(PC1,axeV,axeH)
(xC1,yC1)=PC1

PC2=(0.5*(x5+x4), 0.5*(y5+y6))
PC2=rotate(PC2,angle)
PC2=symetrie(PC2,axeV,axeH)
(xC2,yC2)=PC2

PC3=(0.5*(x6+x7), 0.5*(y7+y8))
PC3=rotate(PC3,angle)
PC3=symetrie(PC3,axeV,axeH)
(xC3,yC3)=PC3

# Rotation des points
P1=rotate(P1,angle)
P2=rotate(P2,angle)
P3=rotate(P3,angle)
P4=rotate(P4,angle)
P5=rotate(P5,angle)
P6=rotate(P6,angle)
P7=rotate(P7,angle)
P8=rotate(P8,angle)
P9=rotate(P9,angle)
P10=rotate(P10,angle)
P11=rotate(P11,angle)
P12=rotate(P12,angle)

# Symétrie des points
P1=symetrie(P1,axeV,axeH)
P2=symetrie(P2,axeV,axeH)
P3=symetrie(P3,axeV,axeH)
P4=symetrie(P4,axeV,axeH)
P5=symetrie(P5,axeV,axeH)
P6=symetrie(P6,axeV,axeH)
P7=symetrie(P7,axeV,axeH)
P8=symetrie(P8,axeV,axeH)
P9=symetrie(P9,axeV,axeH)
P10=symetrie(P10,axeV,axeH)
P11=symetrie(P11,axeV,axeH)
P12=symetrie(P12,axeV,axeH)

# Coordonnées des points
(x1,y1)=P1
(x2,y2)=P2
(x3,y3)=P3
(x4,y4)=P4
(x5,y5)=P5
(x6,y6)=P6
(x7,y7)=P7
(x8,y8)=P8
(x9,y9)=P9
(x10,y10)=P10
(x11,y11)=P11
(x12,y12)=P12
```

```
# Tracé du plan pièce par pièce

vertices = []
codes = []

# Sejour-cuisine
codes = [Path.MOVETO] + [Path.LINETO]*3 + [Path.CLOSEPOLY]
vertices = [P1,P2,P9,P10,P1]
codes += [Path.MOVETO] + [Path.LINETO]*1
vertices += [P11,P12]

# Chambre 1
if c1>0:
    codes += [Path.MOVETO] + [Path.LINETO]*3 + [Path.CLOSEPOLY]
    vertices += [P2,P3,P4,P5,P2]

# Chambre 2
if c2>0:
    codes += [Path.MOVETO] + [Path.LINETO]*3 + [Path.CLOSEPOLY]
    vertices += [P4,P5,P6,P7,P4]

# Chambre 3
if c3>0:
    codes += [Path.MOVETO] + [Path.LINETO]*3 + [Path.CLOSEPOLY]
    vertices += [P6,P7,P8,P9,P6]

vertices = np.array(vertices, float)
path = Path(vertices, codes)
pathpatch = PathPatch(path, facecolor='None', edgecolor='black')
fig = plt.figure(1, figsize=(300, 300))
fig, ax = plt.subplots()
ax.add_patch(pathpatch)

# Texte séjour
T1=ax.text(xA, yA, 'Séjour', horizontalalignment='center', verticalalignment='center',fontsize=10, color='black')
# Texte cuisine
T2=ax.text(xB, yB, 'Cuis', horizontalalignment='center', verticalalignment='center',fontsize=10, color='black')
# Texte ch1
if c1>0:
    T3=ax.text(xC1, yC1, Ch[0], horizontalalignment='center', verticalalignment='center',fontsize=10, color='black')
# Texte ch2
if c2>0:
    T4=ax.text(xC2, yC2, Ch[1], horizontalalignment='center', verticalalignment='center',fontsize=10, color='black')
# Texte ch3
if c3>0:
    T5=ax.text(xC3, yC3, Ch[2], horizontalalignment='center', verticalalignment='center',fontsize=10, color='black')


    ax.set_axis_off()
    ax.autoscale_view(tight=True, scalex=True, scaley=True)
    ax.autoscale()
    plt.axis('equal')

    # Export de l'image dans le dossier nommé selon ses paramètres
    fig.savefig(os.path.join(dossier,filename),bbox_inches='tight',format='png')

print('ok')
```

7.2 Code de formatage des données, d'entraînement et d'utilisation du réseau

In [ ]:
```
# Importation des bibliothèques et raccourcis utiles
import os
import matplotlib.pyplot as plt
import numpy as np
import skimage
from skimage import io
from skimage.transform import resize
import tensorflow as tf
import tensorflow.keras as k



# Chemin d'accès aux données
path_Basique="C:\Users\ProjetMémoire\Data\Plans\SérieBasique"
path_Evolué="C:\Users\ProjetMémoire\Data\Plans\SérieEvoluée"
path_Mixte="C:\Users\ProjetMémoire\Data\Plans\SérieMixte"
```

In [2]:
```
path_train=path_Evolué

# Fonction pour extraire les noms dans l'ordre alphanumérique :

import re

def sorted_aphanumeric(data):
    convert = lambda text: int(text) if text.isdigit() else text.lower() alphanum_key = lambda key:
    [ convert(c) for c in re.split('([0-9]+)', key) ]
    return sorted(data, key=alphanum_key)

# Liste des noms de fichier dans l'ordre alphanumérique
ListePlans0=sorted_aphanumeric(os.listdir(path_train))
ListePlans = []
for names in ListePlans0:
    if names.endswith(".png"):
        ListePlans.append(names)
nbfiles=len(ListePlans)

# Fonction pour importer le plan(k)
def plan(k):
    img = io.imread(os.path.join(path_train,ListePlans[k]),as_gray=True,mode='wa
rp')
    arr = np.array(resize(img, (300, 300)),dtype='uint8')
    return arr


# Fonction pour lire le tag(k)
def tag(k):
    nom=ListePlans[k] #nom du fichier du plan k
    sep=nom.find('_',4) #position du séparateur dans la chaîne
    [ccc]=nom[sep+1] #extraire une sous-chaîne
    return ccc#somme des elements convertis en entiers

print('ok')
```

```
ok
```

In [3]:
```
# Boucle pour créer la table des datasets d'entraînement


n_train=2000
n_test=200

X_train=np.zeros((n_train,300,300,1))
X_train0=np.stack(([plan(k) for k in range(n_train)]))
X_train[:,:,:,0]=X_train0

y_train=np.array([tag(k) for k in range(n_train)])

X_test=np.zeros((n_test,300,300,1))
X_test0=np.stack(([plan(k) for k in range(n_train,n_train+n_test)])) X_test[:,:,:,0]=X_test0

y_test=np.array([tag(k) for k in range(n_train,n_train+n_test)])

print ('ok')
```

```
ok
```

In [4]:
```
# Import des bibliothèques spécifiques pour entraîner le réseau
import tensorflow as tf
import tensorflow.keras as k

# Marqueur à envoyer à TensorBoard pour le suivi de l'entraînement

root_logdir="C:\Users\ProjetMémoire\Data\Exports\Tensorboard.\\tflogs_E_2000"
tbCallBack = tf.keras.callbacks.TensorBoard(log_dir=root_logdir, histogram_freq= 0, batch_size=32,
write_graph=True,write_grads=False, write_images=True)

# Définition et entraînement du modèle

model = k.models.Sequential([
   k.layers.Conv2D(16, (3, 3), strides=3, input_shape=(300,300,1),activation='rel u'),
   k.layers.MaxPooling2D(pool_size=(2, 2)),
   k.layers.Flatten(),
   k.layers.Dense(4,activation=tf.nn.softmax)
])




model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])


model.fit(X_train, y_train, epochs=10,batch_size=8,callbacks=[tbCallBack])
model.evaluate(X_test, y_test)
```

```
Epoch 1/10
2000/2000 [==============================] - 11s 5ms/step - loss: 1.0064 - acc : 0.5560
Epoch 2/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.4872 - acc: 0.8250
Epoch 3/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.3221 - acc: 0.8975
Epoch 4/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.2275 - acc: 0.9305
Epoch 5/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.1653 - acc: 0.9505
Epoch 6/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.1099 - acc: 0.9760
Epoch 7/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.0796 - acc: 0.9880
Epoch 8/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.0601 - acc: 0.9920
Epoch 9/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.0433 - acc: 0.9970
Epoch 10/10
2000/2000 [==============================] - 9s 5ms/step - loss: 0.0295 - acc: 0.9970 200/200
[==============================] - 1s 4ms/step
```

Out[4]:
```
[0.3476581168174744, 0.9]
```

In [9]:
```
# Utiliser le réseau entraîné :


path_transformations="C:\Users\ProjetMémoire\Data\Plans\Modifiés\Série_transformations"
path_réels="C:\\Users\ProjetMémoire\Data\Plans\Modifiés\Série_réels"
```

In [10]:
```
# Redéfinir les fonctions d'import les plans (fichier spécifique)

path_pred=path_réels

ListePlans1=sorted_aphanumeric(os.listdir(path_pred))

ListePlans_pred = []
for names in ListePlans1:
     if names.endswith(".png"):
                    ListePlans_pred.append(names)

nbfiles_pred=len(ListePlans_pred)



def plan_pred(k):
     img = io.imread(os.path.join(path_pred,ListePlans_pred[k]),as_gray=True,mode ='warp')
     arr = np.array(resize(img, (300, 300)),dtype='uint8')
     return arr



print('ok')
```

```
ok
```

In [11]:
```
# Formater les plans à évaluer


n=9
dep=0
arr=dep+n

X_pred0=np.zeros((n,300,300))
X_pred=np.zeros((n,300,300,1))
X_pred0=np.stack(([plan_pred(k) for k in range(dep,arr)]))
X_pred[:,:,:,0]=X_pred0

print('ok')
```

```
ok
```

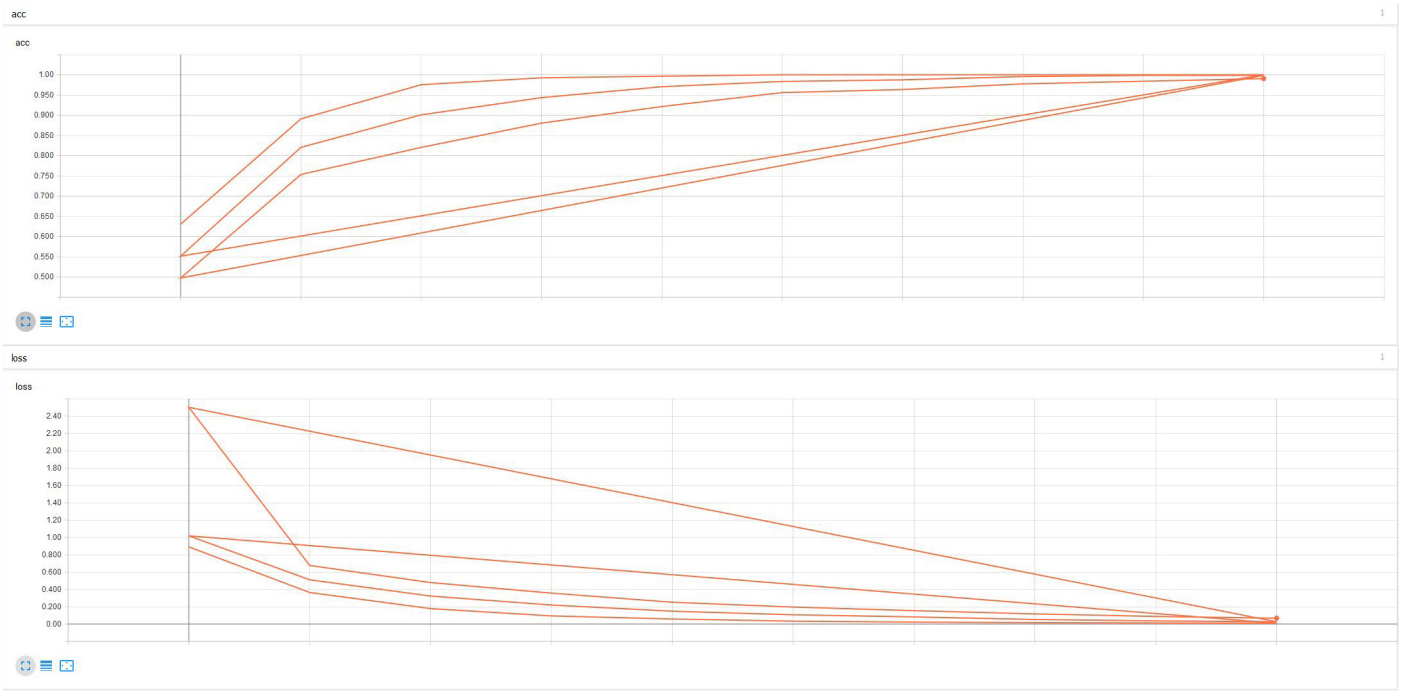
In [12]:
```
# Prédictions:


classes = model.predict_classes(X_pred, batch_size=32) proba =
model.predict_proba(X_pred, batch_size=32)
print(classes)
print(proba)
```

```
[3 3 3 3 3 3 2 3 3]

[
[8.8510258e-12  6.8902252e-13  1.3057047e-01   8.6942953e-01]
[1.0912837e-24   1.1756238e-23   1.2221698e-06   9.9999881e-01]
[8.6232805e-27   3.7461236e-27   6.0699949e-07   9.9999940e-01]
[7.8679249e-17   4.2974604e-12   2.4954138e-02   9.7504586e-01]
[1.1916853e-30   1.8311998e-24   6.8162279e-07   9.9999928e-01]
[3.2501784e-32   1.1239174e-32   4.2158974e-10   1.0000000e+00]
[1.9060183e-15   9.0173580e-10   5.7813150e-01   4.2186847e-01]
[7.2068439e-24   1.5667010e-18   2.2114610e-02   9.7788543e-01]
[9.2460338e-27   4.5249524e-26   1.8356898e-04   9.9981648e-01]
]
```

7.3 Suivi de l'apprentissage des 3 configurations étudiées



Courbes d'apprentissage

Superposition des trois entraînements avec chacune des bases de données B, E et M (seules les lignes brisées sont à prendre en compte). Ces courbes représentent les valeurs de la précision ou taux de réussite (courbe du haut) et du coût ou taux d'échec (courbe du bas), mesurés sur l'échantillon de test fourni au réseau (proportion utilisée: 1 plan de test pour 10 plans d'entraînement, comparativement à ce qu'il se fait dans l'état de l'art).

```
Epoch 1/10
2000/2000 [==============================] - 13s 7ms/step - loss: 1.1132 - acc
: 0.5770
Epoch 2/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.3510 - acc:
0.9005A: 0s - loss: 0.3633 - ac - ETA: 0s - loss: 0.3568 - acc:
Epoch 3/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.1898 - acc:
0.9590
Epoch 4/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.1203 - acc:
0.9770
Epoch 5/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.0701 - acc:
0.9955
Epoch 6/10
2000/2000 [==============================] - 9s 4ms/step - loss: 0.0421 - acc:
0.9985
Epoch 7/10
2000/2000 [==============================] - 9s 4ms/step - loss: 0.0297 - acc:
0.9990
Epoch 8/10
2000/2000 [==============================] - 9s 4ms/step - loss: 0.0199 - acc:
1.0000
Epoch 9/10
2000/2000 [==============================] - 9s 4ms/step - loss: 0.0149 - acc:
1.0000
Epoch 10/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.0104 - acc:
1.0000
200/200 [==============================] - 1s 4ms/step

[0.06463677272200584, 0.98]
```

Scores d'apprentissage à chaque époque d'entraînement sur la base de donnée B.

```
Epoch 1/10
2000/2000 [==============================] - 11s 5ms/step - loss: 1.0064 - acc
: 0.5560
Epoch 2/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.4872 - acc:
0.8250
Epoch 3/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.3221 - acc:
0.8975
Epoch 4/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.2275 - acc:
0.9305
Epoch 5/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.1653 - acc:
0.9505A: 1s - loss:
Epoch 6/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.1099 - acc:
0.9760A: 1s - loss: 0.1095 - acc:  - ETA: 0s - loss: 0.1118 -
Epoch 7/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.0796 - acc:
0.9880
Epoch 8/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.0601 - acc:
0.9920A: 0s - loss: 0.0603
Epoch 9/10
2000/2000 [==============================] - 8s 4ms/step - loss: 0.0433 - acc:
0.9970
Epoch 10/10
2000/2000 [==============================] - 9s 5ms/step - loss: 0.0295 - acc:
0.9970
200/200 [==============================] - 1s 4ms/step

[0.3476581168174744, 0.9]
```

Scores d'apprentissage à chaque époque d'entraînement sur la base de donnée E.

```
Epoch 1/10
2000/2000 [==============================] - 6s 3ms/step - loss: 1.2290 - acc:
0.5060
Epoch 2/10
2000/2000 [==============================] - 4s 2ms/step - loss: 0.5972 - acc:
0.7760A: 0s - loss: 0.6027 -
Epoch 3/10
2000/2000 [==============================] - 4s 2ms/step - loss: 0.3953 - acc:
0.8665
Epoch 4/10
2000/2000 [==============================] - 4s 2ms/step - loss: 0.2623 - acc:
0.9240
Epoch 5/10
2000/2000 [==============================] - 4s 2ms/step - loss: 0.1813 - acc:
0.9575
Epoch 6/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.1442 - acc:
0.9655A: 0s - loss: 0.1393
Epoch 7/10
2000/2000 [==============================] - 4s 2ms/step - loss: 0.1010 - acc:
0.9835
Epoch 8/10
2000/2000 [==============================] - 4s 2ms/step - loss: 0.0736 - acc:
0.9875A: 1s
Epoch 9/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.0532 - acc:
0.9945A: 0s - loss: 0.055
Epoch 10/10
2000/2000 [==============================] - 4s 2ms/step - loss: 0.0344 - acc:
0.9980
200/200 [==============================] - 0s 2ms/step

[0.31143479168415067, 0.89]
```

Scores d'apprentissage à chaque époque d'entraînement sur la base de donnée M.

7.4 Extrait de la base de donnée «Mixte»: échantillon «augmenté» de plans réels