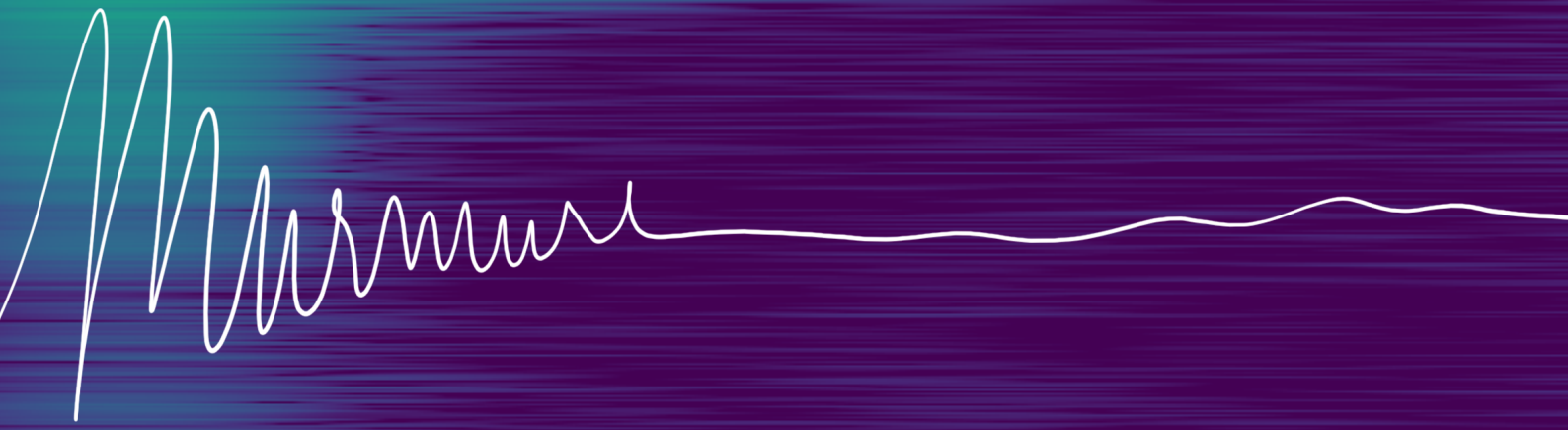


RECONNAISSANCE ACOUSTIQUE DES MATÉRIAUX DE CONSTRUCTION : UNE MÉTHODE NON-DESTRUCTIVE PAR DES RÉSEAUX DE NEURONES

Comment les réseaux LSTM, Conv1D et Conv2D peuvent-ils identifier les matériaux de construction à partir de leurs signatures acoustiques d'impact ?

MÉMOIRE **2024**

DILARA KAYSAL

Séminaire **Savoirs des Activités de Projet Instrumentées**
Sous la direction de **Anne Tuscher** et **Joaquim Silvestre**

cnsm paris la villette

Je tiens à exprimer toute ma gratitude aux professeurs qui m'ont encadrée, guidée et conseillée. Merci à Anne Tüscher et Joaquim Silvestre qui m'ont accompagnée tout au long de ce travail. Je remercie également François Guena pour avoir nourri mes réflexions ainsi que Léa Sattler pour avoir inspiré cette recherche.

Je remercie enfin toutes les personnes qui m'ont apporté leur inconditionnel soutien. À mes parents, Lütfiye et Himmet, et mes proches pour leur enthousiasme et leur encouragement constant.

LA RECONNAISSANCE DES MATÉRIAUX DE CONSTRUCTION PAR UN RÉSEAU DE NEURONES PAR LE BIAIS DES SONS D'IMPACT

Comment les réseaux LSTM, Conv1D et Conv2D peuvent-ils identifier les matériaux de construction à partir de leurs signatures acoustiques d'impact ?

DILARA KAYSAL

Séminaire **Savoirs des Activités de Projet Instrumentées**
Sous la direction de **Anne Tuscher** et **Joaquim Silvestre**

École Nationale Supérieure d'Architecture de Paris-La Villette

RÉSUMÉ

Dans un monde où l'on pense écologie, la sauvegarde du bâti est considérée comme un levier environnemental et patrimonial. La connaissance du bâti par la matière se joint à la préservation de l'architecture. Dans ce contexte, je propose de marier acoustique et intelligence artificielle pour réaliser une reconnaissance matérielle non-destructive.

De nombreux outils permettent d'identifier les matériaux de construction non-visibles dans le cadre de sondages sur les édifices existants. Ces techniques d'investigation, déployées lors de campagnes de sondages, sont pour la plupart sophistiquées mais onéreuses, comme le radar géophysique ou le scléromètre, sinon abordables, mais sans fiabilité. Il existe également des méthodes destructives, tel le burinage ou le carottage, qui s'avèrent non coûteuses et invasives pour le bâti.

En parallèle, l'intelligence artificielle a vu une progression fulgurante depuis sept décennies, particulièrement dans le traitement du son. Des travaux récents ont démontré que des modèles d'apprentissage profond permettent de classifier les matériaux d'objets par leurs sons d'impact. Le modèle d'apprentissage pourrait alors s'affirmer comme un instrument pour la reconnaissance des matériaux de construction.

Cette recherche vise à évaluer la performance de trois architectures de réseaux de neurones, des algorithmes d'apprentissage profond, pour la classification des matériaux du bâti grâce au son qu'émet un mur lors d'un impact. L'objectif est de proposer un instrument gratuit, facilement transportable, fiable et non-destructif pour identifier des matériaux de construction.

SOMMAIRE

AVANT-PROPOS

C'est en réalisant un stage en agence d'architecture spécialisée dans la réhabilitation que j'ai été sensibilisée aux méthodes de réhabilitation, de rénovation et de restauration. La sauvegarde du bâti est un des domaines auxquels je suis le plus attachée. En constatant les difficultés auxquelles tous les acteurs de la construction sont confrontés lors de la réhabilitation d'un bâtiment, j'ai souhaité proposer une recherche qui pourrait éventuellement leur apporter une solution numérique. J'avais déjà une idée en tête : le diagnostic. Cette phase est le socle du projet. L'analyse de site donne les ingrédients pour réaliser un projet pérenne. Elle permet non seulement de mieux comprendre le contexte, mais elle permet surtout à l'architecte de manifester les intentions du projet, adoptant ainsi un regard prospectif. Elle fait donc selon moi partie intégrante de la phase de conception. C'est pourquoi je m'étais d'emblée arrêtée sur un questionnement concernant l'élaboration des diagnostics techniques globaux (DTG).

L'intérêt de cette analyse est d'élucider les problèmes architecturaux et techniques au sein d'une parcelle et de recommander certains travaux pour y remédier. De mon expérience, la réalisation des documents est loin d'être une sinécure. C'est pour cette raison que je me suis naturellement orientée vers ce thème. Les réglementations ont beaucoup évolué. En France, les DTG sont obligatoires pour les immeubles insalubres à la demande de l'administration et pour les immeubles nouvellement mis en copropriété. La coercition n'empêche cependant pas le manque de clarté et de précision sur la mise en forme et le contenu du DTG, perceptible par quelques disparités des diagnostics réalisés par les différentes sociétés. Il n'y a pas seulement un enjeu technique, mais cela devient également une opportunité financière pour certains prestataires. Le manque de données exploitables, trop vagues et hétérogènes, m'a conduite à continuer ma recherche de sujet.

Tout en maintenant le thème du diagnostic, de fil en aiguille, avec l'aide des encadrants, le sujet « la classification des matériaux de construction » s'est révélé comme une évidence. En tentant de sortir de ce que l'on a l'habitude de faire à l'école, je me suis tournée vers un instrument que les étudiants utilisent de plus en plus : l'apprentissage artificiel profond. La classification des sons est un travail de longue haleine mené par de nombreux scientifiques dans des cadres et contextes différents. Des études ont été conduites pour comprendre les performances des hommes puis celles de la machine. Très vite, le potentiel de l'ordinateur s'est affirmé. Les logiciels de reconnaissance vocale et de classification des sons ont gagné en popularité. J'ai découvert des recherches se focalisant plus précisément sur les matériaux et j'ai tout de suite compris que ma recherche pourrait intéresser ceux qui ont le goût pour la préservation du bâti.

Parmi les outils les plus fréquemment utilisés pour la détection de matériaux, on retrouve le Ferroscan et le Géoradar béton. Les deux dispositifs émettent des ondes magnétiques afin de détecter les armatures et objets, les épaisseurs d'éléments bétonnés et les vides. D'autres outils un peu moins connus permettent de sonder d'autres types de matériaux. Compte tenu du prix élevé de ces engins sophistiqués, ce sont souvent les entreprises spécialisées qui les détiennent et notamment les bureaux d'études. Les architectes formulent plutôt des hypothèses basées sur la date de construction et des caractéristiques du bâti. Il manque un instrument qui soit accessible aux architectes, mais également à tous ceux qui disposent de moyens pour détecter les matériaux de construction. C'est ainsi que j'ai décidé d'allier l'intelligence artificielle, plus particulièrement l'apprentissage profond, à la reconnaissance des matériaux. L'outil qui fait l'objet de ma recherche pourrait servir dans le cadre des diagnostics comme un dispositif de sondage non destructif. Elle pourrait donc permettre de préserver le bâti dans son état initial sans rien détruire. Le patrimoine serait conservé.

Faire cette recherche m'a poussée à entraîner et tester plusieurs algorithmes capables de créer un modèle de prédiction à partir de données classées. La première étape était de construire un jeu de données à partir de sons de percussion d'un marteau sur des murs enregistrés par mes soins. J'ai collecté une multitude de sons aux tonalités différentes propres à chaque époque et chaque lieu. Ces bandes sonores méritent-elles une place dans une sonothèque ?

L'investissement intellectuel et émotionnel consacré au développement du modèle d'apprentissage en fait bien plus qu'un simple outil de recherche. J'ai la sensation d'avoir vu grandir le projet à travers les succès et les difficultés. Peut-être qu'une autre personne pourra poursuivre ma recherche, la faire grandir ou, au contraire, nous éclairer plus amplement sur les limites de l'identification des matériaux de construction par les réseaux de neurones.

INTRODUCTION

C'est par notre perception du monde à travers nos sens que nous expérimentons l'architecture. Tout ce que l'on perçoit vient de la matière. La construction est faite à partir de matériaux et de méthodes légués de génération en génération et parfois oubliés. L'architecture vernaculaire, faite de murs de pierre, de paille, de bois, de terre crue, s'amenuise suite à la révolution industrielle, en faisant table rase du passé. On construit alors en mâchefer, béton, verre, acier. Et puis, alors que la construction était encouragée par l'État il y a 70 ans, aujourd'hui, on cherche à transformer les bâtiments existants par la réhabilitation. Bien que la volonté de conservation se soit développée au XVIIIème siècle, la réhabilitation prédomine dans les pays européens depuis peu. Les acteurs du domaine du bâtiment, tels des archéologues, doivent découvrir les strates de construction, vestiges du passé, laissées par les ouvriers. Ce qui m'intéresse dans cette recherche, c'est justement cela : l'archéologie par le son pour découvrir les matériaux qui composent un bâtiment. Comprendre le site du projet, c'est se donner les moyens de prendre parti et de dessiner la trajectoire du bâtiment par le projet en toute conscience.

En parallèle, les outils algorithmiques ont émergé dans la seconde moitié du XXème siècle, portés par le développement des ordinateurs et la croissance exponentielle de la puissance des processeurs, autrement dit le cerveau de nos appareils informatiques. Les modèles d'intelligence artificielle s'immiscent ainsi dans nos vies. Ce sont d'abord les systèmes experts, des programmes spécialisés dans un domaine, raisonnant à partir de règles explicites et organisées, qui attirent l'attention des chercheurs. Ce sont ensuite les algorithmes d'apprentissage, plus particulièrement les **réseaux de neurones**, qui attisent la curiosité des scientifiques. Les réseaux de neurones tirent leur nom d'une partie du fonctionnement du cerveau humain, la transmission des messages neuronaux. Ceux auxquels je me suis intéressée dans le cadre de ma recherche sont liés à la classification, plus particulièrement la classification d'éléments sonores.

Les réseaux de neurones ont fait leurs preuves à maintes reprises lors de recherches, en s'affirmant comme étant plus performants que les humains pour certaines tâches, comme celles liées à la classification des sons d'impact similaires. C'est ainsi que la mémoire à long court terme (en anglais **Long Short Term Memory, LSTM)** et les **réseaux de neurones convolutifs,** le Conv1d et le Conv2d, font leur entrée. Le LSTM a comme particularité de comprendre l'évolution de la donnée, la piste sonore, au fil du temps et de ne retenir que les éléments importants pour la prédiction. Le Conv1d et le Conv2d traitent un enregistrement comme une image. Ils vont examiner l'intégralité d'un enregistrement par l'application de filtres.

Dans le domaine de l'architecture, ce n'est pas tant la classification des matériaux qui me passionne, mais l'identification des matériaux pour la connaissance du bâti. Cette connaissance, elle est intéressante pour contourner certains éléments muraux comme les rails métalliques. Elle peut également nous servir à trouver des objets dissimulés à l'intérieur des murs, comme d'anciens éléments de plomberie. C'est aussi un moyen pour justifier l'utilisation de matériaux spécifiques pour notre projet.

Les diagnostiqueurs emploient divers dispositifs pour décoder la composition matérielle d'un édifice. Certains sont onéreux, certains peu fiables, d'autres destructifs. Pourtant, un geste instinctif, celui de toquer sur les murs, est très peu mis sur le devant de la scène. Cette habitude à laquelle nous adhérons tous est un potentiel moyen de reconnaissance des matériaux par des algorithmes d'apprentissage profond. Et si les enregistrements sonores de coups donnés à un mur étaient une méthode de reconnaissance de matériaux par des réseaux de neurones, tels que le LSTM, le Conv1d et le Conv2d ?

Je me suis demandée s'il était possible de créer un outil fiable, économique et non invasif pour la classification des matériaux de construction en me basant sur le son émis lorsque l'on toque sur des murs. Cette recherche porte donc sur l'évaluation de la fiabilité des réseaux de neurones pour la classification de matériaux grâce aux sons d'impact. La comparaison des performances entre un LSTM, un conv1D et un conv2D a également été réalisée.

Avant d'expliquer plus en détail l'objet de cette recherche, je ferai l'**état de l'art des techniques de classification des matériaux**. Dans cette partie seront exposées les différentes classifications réalisées par l'intelligence artificielle, mais également par l'homme, dans divers domaines. Une vision holistique donnera matière à l'expérience et suggérera des pistes de méthodologies.

Je **définirai ensuite les termes** qui permettent de saisir les informations importantes. Une découverte des éléments caractéristiques du son puis de la classification audio par les réseaux de neurones permettra de saisir toutes les informations importantes énoncées dans la partie experimentale.

Dans une troisième et quatrième partie, la problématique sera rappelée et la méthodologie de recherche sera plus amplement décrite.

Puis, je relaterai **les étapes** pour mettre en place l'**expérience** consistant à évaluer les performances des trois architectures entraînées. Les étapes préliminaires, comme la réalisation d'un caisson isolé, en passant par la préparation des données, l'entraînement des algorithmes d'apprentissage profond jusqu'à la phase de test, seront montrées. Mes difficultés et mon point de vue personnel guideront cette section.

Je conclurai par le **bilan de l'expérience** en soulignant ses limites et en proposant des pistes d'amélioration.

Un **glossaire** permet de rappeler la signification des mots spécifiques à la fin de cet ouvrage.

PARTIE 1

ÉTAT DE L'ART

DES MÉTHODES MULTIPLES POUR RÉPONDRE À DES BESOINS DIFFÉRENTS

Il faut en premier lieu estimer la relation entre les propriétés des matériaux et leur son d'impact pour étayer le propos. S'il y a bien une corrélation entre ces deux éléments, alors une reconnaissance des matériaux à travers les modalités auditives peut être envisagée. Je me suis posée un certain nombre de questions par la suite : est-ce que d'autres facteurs que ceux liés à la matérialité, comme l'outil avec lequel on toque ou encore la géométrie locale de l'objet étudié, peuvent avoir une influence sur la reconnaissance des matériaux ? Est-ce qu'une évaluation des performances humaines et computationnelles a été aboutie et quels en sont les résultats ? Une étude sur la reconnaissance des matériaux de construction par le biais de la classification des sons d'impact de murs a-t-elle déjà été menée ?

1 La reconnaissance matérielle par l'oreille humaine

Comprendre la composition d'un édifice, par ses matériaux, enrichit notre connaissance de l'état de l'ouvrage. L'édifice livre des informations sur sa mise en œuvre, son époque, les modes d'habiter des résidents. Toutes ses histoires peuvent aussi devenir des arguments de projet. C'est pourquoi, tout projet commence par une phase d'état des lieux et de diagnostic. L'identification des matériaux est pratiquée en amont du dessin détaillé du projet. D'une part, l'état des lieux donne un indice sur la vie du site existant, peu importe sa condition. D'autre part, le diagnostic, à l'inverse, signale les pathologies, les désordres. Le diagnostiqueur est comparable à un docteur qui cherche grâce à divers outils, les signes d'une maladie. La mise en évidence de problèmes passe par l'emploi de techniques de reconnaissance des matériaux.

La manière la plus intuitive de comprendre comment fonctionne un mur, par exemple, est de regarder et de taper sur ce dernier. Observer donne d'emblée une idée de la matérialité d'un mur. La dureté de la surface, ses aspérités, sa brillance. Ils nous permettent de faire des hypothèses sur la matière. Les différentes strates et leur épaisseur pourraient apparaître clairement.

La percussion donne aux **caractéristiques physiques propres à chaque matériau** à se manifester. Chaque matériau aurait des propriétés invariantes, quelle que soit sa forme et sa taille.. Richard P. Wildes et Whitman A. Richards montrent en 1988 que la manière dont se propagent les ondes change en fonction du matériau[1]. Ces propriétés sont mis en exergue par l'angle de frottement interne. Ce terme est employé pour les matériaux granulaires, comme pour la terre ou le sable. Il s'agit de l'angle maximal que peut former un matériau granulaire avec l'horizontale avant de glisser sur lui-même. Il caractérise donc la résistance au cisaillement du matériau par rapport à une contrainte normale. Cela aurait notamment une incidence sur le temps d'affaiblissement du son. C'est ainsi qu'une classification des matériaux à partir du son généré lorsque celui-ci est frappé est possible.

1 R.P. Wildes. W.A. Richards. Recovering material properties from sound. Natural Computation, 1988. Consulté le 2 janvier 2024

Nous pouvons toquer pour comprendre si un mur est porteur ou non. Pour entendre ses sonorités et dire de quoi le mur est fait. Pour comprendre si d'autres objets sont dissimulés. Un rail métallique. Un tuyau. Un quelconque objet. Toutes les nuances dans le son proféré dénotent des caractéristiques qui pourraient servir durant les phases de conception et de travaux.

En 1996, Eric Krotkov, Roberta Klatsky et Nina Zumel s'appuient sur les formules de R.P. Wildes et W.A. Richards[2] pour analyser et synthétiser les sons d'impact. Ils soulignent que ce travail pourrait servir aux sondages non destructifs mais également au domaine de la réalité virtuelle, dans lequel on génère du son plutôt que d'analyser, pour concourir au sentiment de présence et d'en améliorer l'immersion : ce sentiment qui nous donne l'impression de nous trouver physiquement dans un lieu pourtant en réalité virtuelle. Ils confirment que **la durée des sons d'impact est liée au matériau frappé**. Par exemple, le son d'impact sur du métal décline moins vite que celui du bois. Enfin, les **dimensions** des objets jouent un rôle important, particulèrement sur les **fréquences** sonores émises. Plus la taille de l'objet est petite, plus le son a une fréquence haute, caractéristique d'un son aigu. Wert, Lutfi, Gaver et leurs associés nuancent cependant les résultats de Krotkov à propos de l'angle de frottement interne, qui varie tout de même légèrement selon la forme de l'objet. Lutfi et ses associés, Klatzky et al. ainsi que Rocchesso et al.[3] insistent sur l'i**mportance des fréquences pour la reconnaissance des matériaux**, mettant le temps de déclin du son au second plan. Ce sont donc les fréquences qui endossent un rôle majeur.

Les instruments de musique sont des bons exemples pour montrer la relation entre les dimensions des objets et le son qu'ils émettent. Le xylophone est composé de plusieurs lames de bois aux longueurs variables. Les touches sont alignées de la plus grande à la plus petite, du plus grave au plus aigu. Le diapason, un instrument métallique, produit une note presque pure, d'une fréquence constante. Le diapason le plus connu donne la fréquence de 440 Hz, jouant la note du la. Mais il en existe d'autres de taille, de forme et de masse différentes. Pour la même matière, un diapason

2 Krotkov, Eric. Klatzky, Roberta. Zumel, Nina. «Analysis and synthesis of the sounds of impact based on Shape-Invariant Properties of Materials.» Université de Carnegie Mellon. Pittsburgh, Pennsylvanie. 1996.

3 Rocchesso, Davide. Fontana, Federico. The Sounding Object. Italie, Firenze : Mondo estremo, 2003. 399 p.

de 1024 Hz, plus petit que celui de 440 Hz, produit un son plus aigu. Les objets d'un même matériau peuvent produire des fréquences différentes. Il est important de **décomposer le signal sonore selon toutes ses fréquences au fil du temps pour pouvoir faire apparaître les spécificités des matériaux.**

Malgré cela, nous sommes plus ou moins en mesure d'entendre les spécificités de chaque son et d'en identifier leur nature. L'acuité auditive, donc notre seuil de perceptibilité et notre sensibilité auditive, varie d'une personne à une autre. D'autres éléments liés au contexte lors de l'expérience auditive pourraient affaiblir la reconnaissance.

En 2003, Bruno L. Giordano[4] se base sur le modèle de Krotkov et ses associés pour diriger une expérience mesurant la reconnaissance par la perception humaine. Elle consistait à faire écouter à travers un casque une série d'enregistrements de sons d'impact de plaques à plusieurs personnes. Le but était de reconnaître le matériau émettant le son. Les plaques variaient en taille, en dimensions et en matériaux. Il a été conclu que les caractéristiques telles que les dimensions des objets entrent en ligne de compte dans la justesse de la réponse donnée. Enfin, un paramètre supplémentaire a été modifié. **Le changement de matériau de l'objet qui inflige l'impact semble porter peu d'importance au résultat de la reconnaissance par les humains.**

Pour ce qui est de l'observation visuelle, elle n'est souvent pas suffisante pour distinguer les matériaux composant un bâtiment. La surface visible n'est parfois qu'apparence. Pour le bâti ancien, la question se pose peu, car la plupart du temps, il n'existait qu'une, deux, voire trois couches de matériaux différents. Aujourd'hui, l'enveloppe des édifices se complexifie. On peut compter jusqu'à cinq couches différentes. L'enduit extérieur, le mur, l'isolant, les membranes étanches, les rails, le placo, l'enduit intérieur... La surface n'est qu'un revêtement, cachant plusieurs autres strates. L'expérience forme davantage le sachant, qu'il s'agisse du diagnostiqueur, de l'architecte, de l'ouvrier. Il acquiert des connaissances par ses études antérieures et sa réflexion. Mais une personne sans expérience pourrait manquer de compétence pour interpréter les données avec discernement.

---

4 L. Giordano, Bruno. «Material categorization and hardness scaling in real and synthetic impact sounds», The sounding object. Italie, Firenze : Mondo estremo, 2003. pp.73-93

2 La reconnaissance des matériaux par les dispositifs numériques par le biais de signaux perceptibles

Plusieurs chercheurs ont vérifié la pertinence de l'emploi des outils computationnels pour la classification. Sofia Cavaco et José Rodeia ont proposé en 2010 une classification de sons similaires produits à partir d'objets de géométrie semblable mais de matériaux différents. Ils ont relevé que l'ordinateur était plus performant que l'humain dans le cadre de leur recherche[5]. Il semblerait par ailleurs que la machine peut apprendre beaucoup plus rapidement et commettre moins d'erreurs.

Les techniques de classification des matériaux par des algorithmes d'intelligence artificielle sont surtout appliquées aux domaines de l'ingénierie civile et à la robotique notamment pour le tri des matériaux recyclables. Mais des études sont également menées à la fois pour l'analyse des éléments réels pour simuler des sons synthétiques plus fidèles à la réalité. D'autres exploitent les données synthètiques pour expliquer des faits physiques réels. La réalité virtuelle s'empare donc également de techniques de reconnaissance. En s'éloignant du domaine architectural, le computer vision prédomine dans la classification des matériaux. Lorsque la classification audio est exploitée, cette dernière se conjugue à des images ou des éléments tactiles. Tous ces outils ont un potentiel qui n'est pas exploité en architecture. Une technique pourrait s'appliquer dans le domaine de la construction.

a La reconnaissance tactile

En 2021, Di Guo[6] et ses associés publient l'article « Reconnaissance tactile des matériaux guidée par l'affordance visuelle pour le recyclage des déchets ». De leur désir d'adapter un système de tri robotisé à un cas d'application réel est née une expérimentation reposant sur la prise d'objets robotique guidée par la reconnaissance visuelle et la classification tactile des matériaux.

---

5 Cavaco, Sofia. Rodeia, « Classification of Similar Impact Sounds », in Image and Signal Processing, éd. par Abderrahim Elmoataz et al., vol. 6134, Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. pp. 307-14, https://doi.org/10.1007/978-3-642-13681-8_36.

6 Guo Di, Liu Huaping, Fang Bin, Sun Fuchun et Yang Wuqiang, « Visual Affordance Guided Tactile Material Recognition for Waste Recycling », IEEE Transactions on Automation Science and Engineering, 19-4, octobre 2022, p. 2656-2664.

Ils s'intéressent à une machine à vecteurs de support (SVM). Presque tout processus d'intelligence artificielle peut être visualisé. On part du principe que toute donnée peut être située dans une, deux, trois ou des dimensions infinies. Le but du SVM est de rechercher un plan, sur plusieurs dimensions, donc un hyperplan optimal pour séparer les données en classes distinctes. Pour cela, il fait appel à l'astuce du noyau (kernel trick). C'est en réalité une transformation mathématique qui projette des données dans un espace de dimension supérieure pour qu'une séparation linéaire soit possible. Les vecteurs de support, les points les plus proches de l'hyperplan, ont une incidence sur le placement de l'hyperplan. Ils ont donné leur nom à l'algorithme de classification.

Pour en revenir à la recherche de Di Guo, le SVM est entraîné pour la classification grâce à des données tactiles recueillies par des capteurs situés sur les pinces du robot. Les capteurs sont composés d'électrodes acquérant des informations liées à la contre-pression des objets en fonction du temps. La répartition de la pression sur les électrodes est relevée. Sont différenciées quatre classes : le papier, le métal, le plastique et le verre. La précision de la reconnaissance tactile s'élève à 78.3%.

Les données reflètent la dureté des matériaux, marquant des propriétés intrinsèques comme la résistance du matériau. Ce système est une piste de classification. Ce type de données façonne d'autres outils dans le domaine de la construction que nous verrons plus tard. Nous pouvons nous demander si cette technique est appropriée à la classification d'objets de rigidité plus importante et de surfaces planes comme les murs.

b La reconnaissance d'image

Nous faisons certes abstraction des détails cachés quand nous regardons notre environnement. Cependant, il faut prendre en considération la vision puisqu'elle a toujours été prédominante dans le domaine de la reconnaissance par l'ordinateur. Elle a été utilisée pour différencier des matériaux par leur texture et leur couleur. Parmi les algorithmes de classification, on retrouve les réseaux de neurones convolutifs (CNN).

Dans le cadre d'une recherche concernant des matériaux de construction sur des sites de chantier[7], la comparaison des teintes et des saturations

7 Son Hyojoo, Kim Changmin, Hwang Nahyae, Kim Changwan et Kang Youngcheol, « Classification of major construction materials in construction environments using ensemble

des couleurs au sein des images a été probante. H. Son et al. font une reconnaissance distinctement du bois, puis du béton et enfin de l'acier en unissant les forces de modèles d'apprentissage aux différentes architectures. La précision des prédictions varie entre 88.73 % et 95.23 %.

Ils ont proposé d'appliquer cette technique pour générer des modèles 3D conformes à l'exécution. Le suivi de chantier pourrait ainsi être facilité.

Cependant, plusieurs facteurs, comme la lumière, influencent les prédictions. Les éléments constructifs sont parfois complexes, de par leur texture et leur forme, ou masqués et peuvent présenter des désordres. Le plus grand défi est l'identification des matériaux dissimulés sous les revêtements. L'image peut être une donnée qui manque de fiabilité.

c La reconnaissance par l'image complétée par le son audible

Et si l'on mariait plusieurs modalités sensorielles pour la reconnaissance avec des modèles d'apprentissage profond ? Cela résoudrait quelques problèmes cités précédemment, comme l'ont déclaré Fengmin Shi et ses collaborateurs dans « GLAVnet : indices audio-visuels globaux et locaux pour la classification détaillée des matériaux »[8]. Ils estiment que les caractéristiques du son d'un objet frappé varient en fonction de la localisation de la percussion. Par exemple, le son du coup sur la hanse d'une tasse n'est pas le même que celui sur le rebord. Le son et la géométrie locale de l'objet devraient être mis en relation pour pouvoir faire une classification. Leurs datasets sont faits d'objets de 17 classes de matériaux représentés sous forme de voxels et de sons générés par synthèse sonore ou par impact d'objets réels. Le modèle utilisé est une variante du CNN. Alors que le modèle parvient à de bons résultats (entre 85% et 99%) pour la catégorisation des sons synthétiques, celle des objets réels est moindre (53,4%).

Allier deux modalités différentes permet d'obtenir plus d'informations. Articuler l'image avec le son pourrait être utile. Cependant, si nous supposons

classifiers », Advanced Engineering Informatics, 28-1, 1 janvier 2014, p. 1-10. https://doi.org/10.1016/j.aei.2013.10.001.

8 Shi Fengmin, Guo Jie, Zhang Haonan, Yang Shan, Wang Xiying et Guo Yanwen. « GLAVNet: Global-Local Audio-Visual Cues for Fine-Grained Material Recognition », in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA: IEEE, 2021), 14428-37, https://doi.org/10.1109/CVPR46437.2021.01420.

que les matériaux ont des propriétés invariantes quelle que soit la forme alors il ne serait pas nécessaire de rajouter des données supplémentaires. En revanche, une variante de cette technique est exploitable pour localiser les rails métalliques derrière les plaques de plâtre.

d La reconnaissance matérielle par le son audible

Dimiccoli, Patni, Haffmann et Moreno-Noguer[9] ont prouvé qu'il était possible de se fier aux sons d'impact pour une reconnaissance des matériaux par l'ordinateur.

Ils ont pour cela créé un dataset, le « YCB-impact sound dataset » disponible en ligne contenant 3000 enregistrements de 75 objets. On trouvera les matériaux suivants : l'acier, l'aluminium, le plastique dur, le plastique souple, d'autres types de plastique, la céramique, le bois, le papier/carton, la mousse, le verre et le caoutchouc. Trois modes d'exploration pour générer des sons d'impact différents ont été mis en œuvre. Le premier consiste à utiliser des gestes manuels, tels que taper, gratter, laisser tomber les objets à l'aide d'une pince métallique. Le deuxième et le troisième consistent à se servir d'un robot tapant respectivement verticalement et horizontalement, en variant la force et la localisation de la frappe. Le modèle Resnet34 analyse enfin les données du dataset.

Pour résumer les résultats, l'intelligence artificielle distingue les différents matériaux des objets du quotidien entre eux. Cependant, les matériaux des objets sont inégalement répartis. Les objets en plastique et en acier sont plus nombreux. C'est pourquoi les données ont été augmentées informatiquement. Entraîné sur tout le dataset, le modèle a prédit le bon matériau avec une précision de 60% les sons produits par un geste manuel et de 85% lorsque le robot tape verticalement. **Le modèle est donc plus à même de prédire correctement des sons provenant d'un geste répété gardant la même force et la même direction.**

Le travail de Dimiccoli exhibe un modèle qui pourrait être transplanté à la discipline de la construction. Voyons maintenant comment sont effectués les sondages non destructifs sur les bâtiments. Est-ce que les techniques actuelles se nourrissent des savoirs en acoustique ? Si oui, comment ?

9 Dimiccoli Mariella, Patni Shubhan, Hoffmann Matej et Moreno-Noguer Francesc. « Recognizing object surface material from impact sounds for robot manipulation ». In 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Kyoto, Japan: IEEE, 2022 .9280-87. doi:10.1109/IROS47612.2022.9981578.

3 Dans le secteur de la construction, des dispositifs numériques captant l'imperceptible pour l'humain

Il faut noter que rares sont les travaux récents s'orientant vers la classification audio. En architecture, les sondages non destructifs font usage de données qui sont de l'ordre de l'invisible ou de l'inaudible. Les détections par ondes électromagnétiques ou ondes mécaniques sont les plus répandues. Il existe également des instruments qui estiment la résistance à la pression des matériaux, comme le scléromètre.

a La reconnaissance par des ondes électromagnétiques non visibles

Des outils comme le géoradar et le ferroscan dévoilent les matériaux invisibles en sillonnant l'univers complexe des rayonnements électromagnétiques. Les rayons X (30 PHz à 60 EHz), les ondes infrarouges (300 GHz à 384 THz), les micro-ondes, les hautes fréquences et les ondes radios (1 GHz à 300 GHz) sont ainsi employés pour déduire les propriétés des matériaux. Chaque matériau a un comportement spécifique face aux ondes. Ainsi, l'outil émet une onde à l'aide d'une antenne qui est ensuite réfléchie par l'objet et transmise à un récepteur.

Le géoradar ou GPR, par exemple, se sert des hautes fréquences (entre 500 MHz et 2.6 GHz)[10] pour produire des images radar des structures des bétons ou encore des sous-sols. Il mesure la trace, c'est-à-dire l'amplitude du signal électromagnétique au cours du temps. Le changement de matériaux, les zones humides, la corrosion des armatures peuvent ainsi être détectés grâce au coefficient de réflexion des ondes sur l'échantillon étudié. Le géoradar demande malgré cela une expertise pour traiter la donnée après la prise d'images.

Le ferroscan détecte les forces des champs magnétiques, par la suite converties en signaux électriques mesurables[11]. Il repère les positions, les

10 Van der Wielen, Audrey. Nguyen, Frédéric. Courard, Luc. «Détermination des propriétés de couches minces dans le béton à l'aide d'un géoradar commercial à hautes fréquences: approche pic-à-pic et analyse fréquentielle du coefficient de réflexion.». Annales Du bâtiment Et Des Travaux Publics, 4(01). Université de Liège: 2014. https://orbi.uliege.be/bitstream/2268/165565/1/ArticleAvdWielen_GPR.pdf.

11 James, Amala et al., « Rebar corrosion detection, protection, and rehabilitation of

alignements de barres métalliques ou calcule la profondeur de l'enrobage. La corrosion pourrait être reconnue et les fuites d'eau trouvées lors de l'inspection des réseaux de plomberie. Bien entendu, le ferroscan a ses avantages mais aussi ses inconvénients. Il est difficile de l'utiliser sur des ouvrages à reliefs importants.

Les deux dispositifs décrits sont précis. Pourtant, ils se limitent au sondage d'ouvrages en béton. Dans l'ensemble, ces outils sophistiqués sont généralement très onéreux (plus de 3000 euros). Les autres dispositifs à prix plus abordables sont beaucoup moins fiables.

b La reconnaissance par des ondes mécaniques non audibles

L'acoustique semble en effet contribuer à l'enquête sur les matériaux. Ce qui est inaudible, au détriment de ce qui est audible, est étudié. Les vibrations mécaniques des objets, imperceptibles, dépendent de la forme des objets mais également de leur matérialité. Des capteurs avec contact ou des vibromètres (capteurs sans contact) mesurent ces vibrations pour déterminer l'élasticité des matériaux afin d'en tirer des conclusions sur les zones dégradées des matériaux de construction. Les fréquences hautes sont émises par le dispositif, puis interagissent avec l'objet qui renvoie et absorbe une partie des ondes.

Le spectromètre, plutôt que d'être équipé d'un émetteur de faisceau lumineux, est armé d'un émetteur d'onde acoustique de domaine incluant les ultrasons et les infrasons. C'est donc la signature spectrale de résonance acoustique qui détermine les propriétés élastiques du matériau. Le vibromètre[12], autrement appelé vibromètre laser Doppler (LDV), calcule la vélocité d'une surface en mesurant le décalage Doppler d'un rayon laser réfléchi. Enfin, le scléromètre, par le contact avec l'échantillon, évalue la résistance du béton à partir du rebond d'une masse sur la surface du matériau. Ces instruments, tout comme ceux dépendant des ondes électromagnétiques, sont onéreux. Ils impliquent un investissement financier.

---

reinforced concrete structures in coastal environments: A review », Construction and Building Materials 224, 10 novembre 2019. 1026-39. https://doi.org/10.1016/j.conbuildmat.2019.07.250.

12 Costas P. Providakis, Maria G. Mousteraki, et Georgia C. Providaki, « Operational Modal Analysis of Historical Buildings and Finite Element Model Updating Using α Laser Scanning Vibrometer », Infrastructures 8, no 2. Février 2023. https://doi.org/10.3390/infrastructures8020037.

Pour conclure, des chercheurs (Wildes et Richards, 1988) ont prouvé que certaines propriétés des matériaux sont invariantes quelle que soit leur forme. Les fréquences sonores, bien qu'elles se transforment en fonction des dimensions et des formes des objets, peuvent caractériser les matériaux (Rocchesso et al, 2003). Le matériau de l'instrument avec lequel on percute l'échantillon a peu d'importance tant que le jeu de données est créé par un unique matériau de percussion. De plus, les réseaux de neurones aboutissent à une meilleure performance que l'humain pour la classification de sons similaires (Cavaco et al, 2010). Parmi l'image, la texture et le son, le son semble porter un potentiel très peu exploité pour la classification des matériaux. Dans le cadre d'un sondage non destructif, le son audible est exclu. Je n'ai pas trouvé de recherche sur la classification des matériaux de construction dans un environnement construit par le biais du son d'impact. Toutes ces études laissent pourtant penser qu'un réseau de neurones peut classifier avec justesse les sons d'impact. Je peux à mon tour construire mon propre jeu de données en enregistrant des sons depuis un téléphone portable, puis entraîner et tester une variante d'un algorithme d'apprentissage profond préalablement codé.

PARTIE 2

PRENDRE SES MARQUES

DU SIGNAL ANALOGIQUE AU SIGNAL DIGITAL POUR UNE CLASSIFICATION PAR DES RÉSEAUX DE NEURONES

Avant de faire des tests de reconnaissance audio par un modèle d'apprentissage, il est important d'exposer les termes à connaître pour une bonne compréhension de l'expérience. Dans cette partie seront présentées plus en détail les notions relatives au son pour entrevoir comment s'effectue la classification audio par les réseaux de neurones.

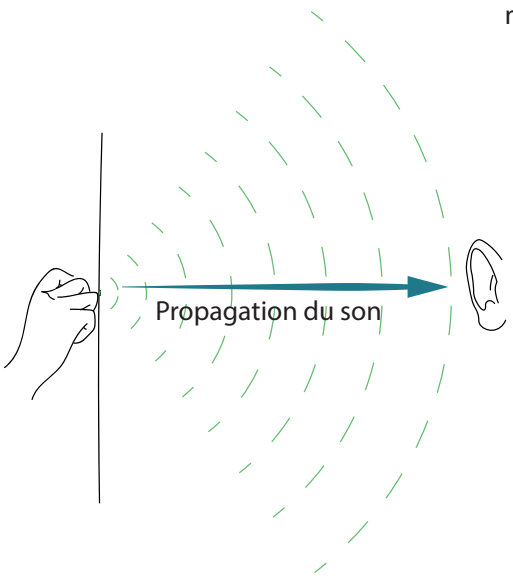


Figure 1 : La propagation du son

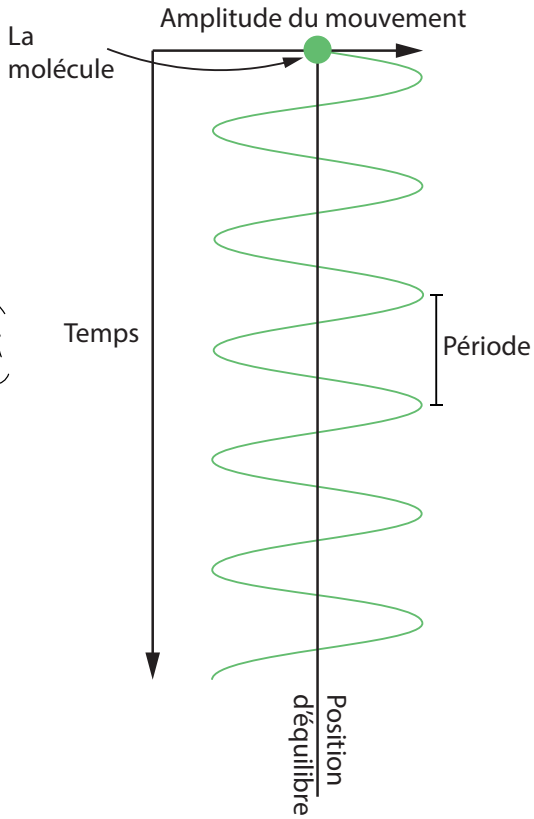


Figure 2 : L'oscillation à l'échelle de la molécule

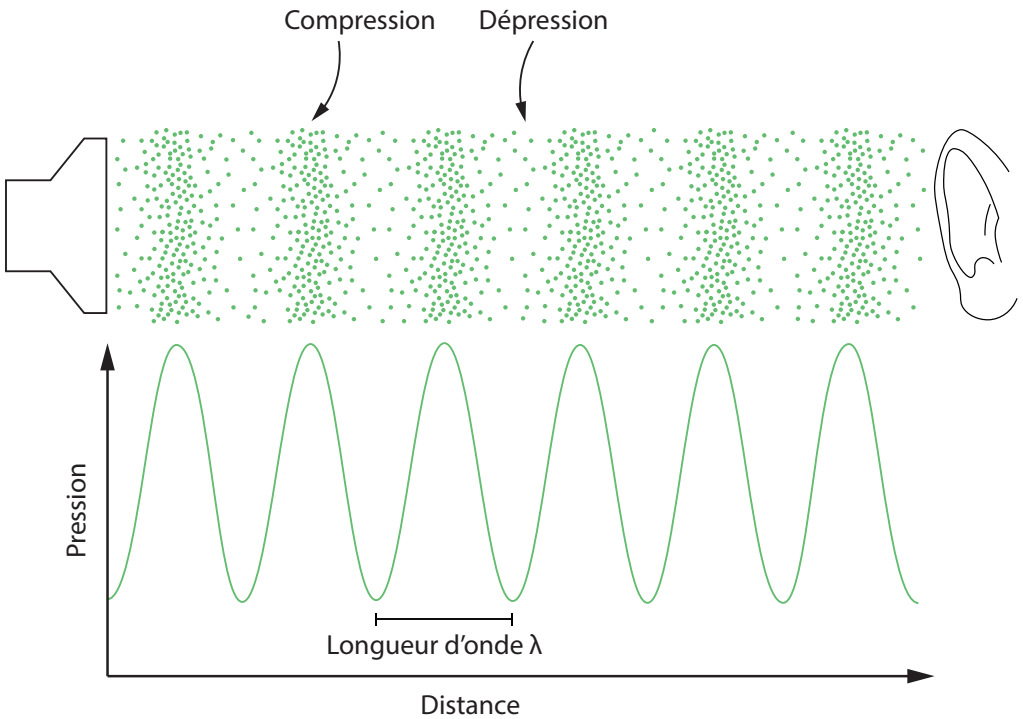


Figure 3 : Succession de zone de compression et de dilatation

1 Le son, un signal riche en information

Le son se manifeste comme une **onde mécanique** se propageant dans un milieu matériel sous forme de vibrations continues. Sa transcription dans l'environnement informatique se traduit par une **conversion du signal analogique**, tel qu'il existe dans la nature, en **signal numérique** exploitable par les systèmes de traitement. Pour comprendre le signal, le son peut être dessiné. Les diverses **représentations graphiques** du son, qu'elles soient temporelles ou fréquentielles, permettent non seulement une analyse approfondie pour en extraire les caractéristiques spécifiques, mais offrent également la possibilité de manipuler le signal à d'autres fins.

a Une vibration continue

Le son est généré par la vibration d'un ou de plusieurs objets (figure 1). La vibration se propage ensuite dans un milieu matériel (air, eau, bois, etc.). Elle transporte de l'énergie d'un point à un autre, sans déplacement effectif de la matière.

À un niveau moléculaire, la molécule oscille suivant la direction de la propagation de l'onde sonore (figure 2). En prenant en compte le fait que le son se propage, si nous dézoomons légèrement, les molécules se compressent entre elles à certaines régions et se décompressent à d'autres régions (figure 3).

Ainsi, le son est défini par une onde sonore mécanique qui se propage grâce à une perturbation locale et temporaire dans un milieu matériel.

L'onde sonore est caractérisée par cinq éléments. La position d'équilibre représente la position de la molécule lorsqu'elle n'est pas perturbée. L'**amplitude** est la position de la molécule, à un instant donné. Elle se traduit par le niveau de pression acoustique à un instant t, mesuré en décibels (dB). Plus l'amplitude est forte et plus le son est fort. La **période T** est l'intervalle de temps pour que le signal sonore se reproduise à l'identique (figure 4). Elle s'exprime en seconde. La **fréquence f** (figure 4) constitue le nombre de répétitions de l'oscillation par seconde. Elle s'exprime en hertz. La relation entre la fréquence et la période est définie par la fonction : $f = 1/T$.

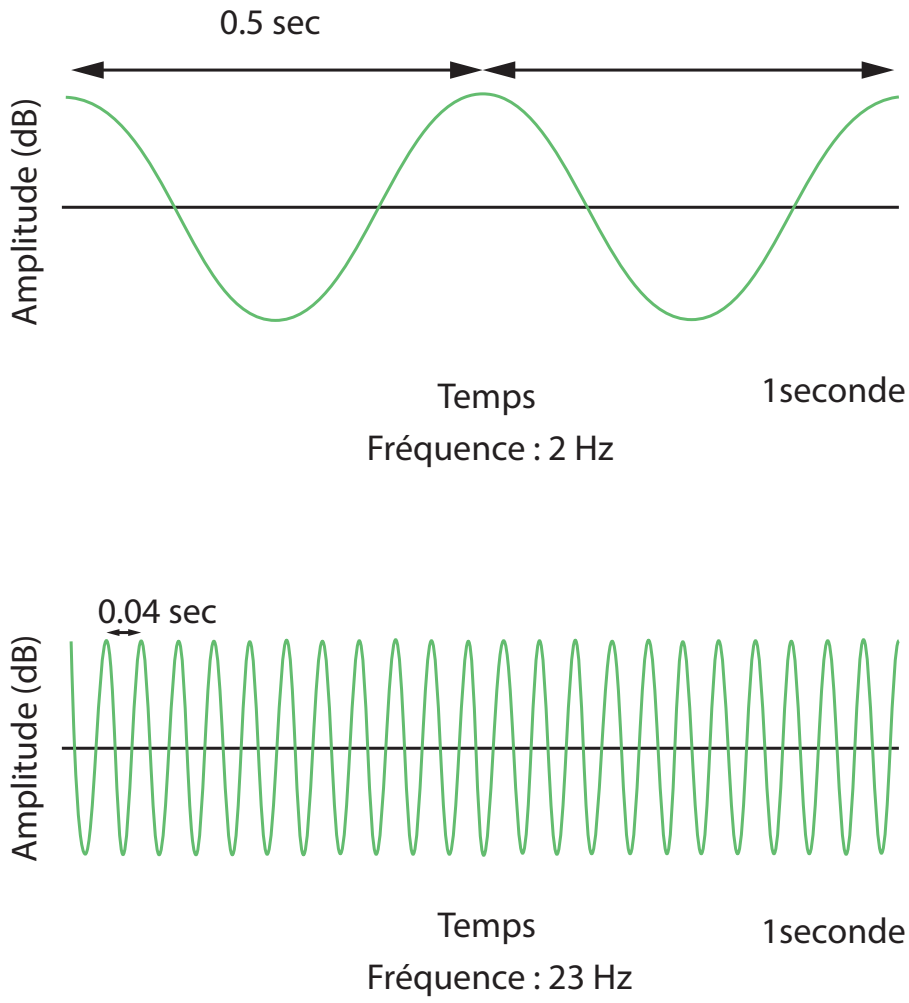


Figure 4 : Les caractéristiques du son

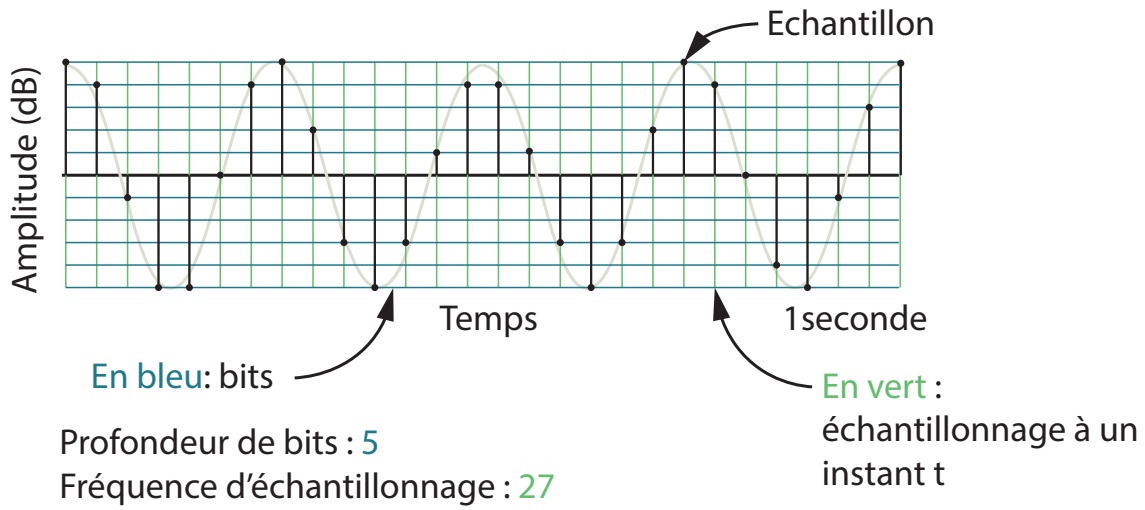


Figure 5 : La digitalisation

Enfin, la longueur d'onde λ est la distance entre les deux régions de molécules compressées entre elles. En d'autres termes, c'est la distance parcourue par l'onde pendant la durée T. Elle s'exprime en mètre. La relation entre la période T, la longueur d'onde λ et la célérité c : $c = \lambda / T$.

Le son émis par un matériau est intimement relié à ses propriétés. La **fréquence f est la principale caractéristique analysée lors d'une classification de matériaux par le son.** De plus, l'amplitude enrichit l'information et vient donc s'ajouter à la matrice pour la classification.

Le son que l'on perçoit est un **signal continu** que l'on peut qualifier de **signal analogique**. Le disque vinyle et la cassette audio sont gravés. Ils donnent un relief au son de la manière la plus fidèle possible. Les cavités et les protubérances correspondent à l'amplitude du signal. Les casques, les microphones et les haut-parleurs transmettent également un signal analogique, cette fois-ci sous forme de variations proportionnelles de tension électrique. Le problème du signal analogique est la présence de petits bruits parasites altérant l'enregistrement. Mais le second problème, et qui est de loin le plus handicapant dans mon cas, c'est qu'il ne peut pas être traité par l'ordinateur. Pour effectuer mon expérience, j'ai eu besoin d'un langage codé pour que l'algorithme d'intelligence artificielle comprenne le message. Numériser est la première étape du processus.

b Un signal sonore numérique

Pour obtenir un **signal sonore numérique,** la clé est l'**échantillonnage**. Cela consiste à prélever le son à un intervalle de temps régulier. Plus l'intervalle est court, plus il y aura d'échantillons[13]. Chaque échantillon est mesuré sous forme de valeurs. Le signal continu devient discontinu, donc numérique (figure 5)[14].

Sommairement, le son est converti en signal numérique le plus tôt possible. Le microphone capture le son sous forme de signal électrique analogique, le filtre, l'échantillonne, le quantifie puis le transmet sous forme de flux en basse et haute tension. L'ordinateur retranscrit donc une suite de nombres, un signal discret, sous forme binaire. Cela permettra ensuite de traiter le son. D'en optimiser la fidélité. Ou au contraire, de la tordre, de

13 La discrétisation du signal continu est obtenue grâce à une fonction, le peigne de Dirac définissant une série d'impulsions régulièrement espacées.

14 « Son analogique / Son numérique | AFSI », AssoConnect, consulté le 2 avril 2024, https://www.afsi.eu/articles/12926-son-analogique-son-numerique.

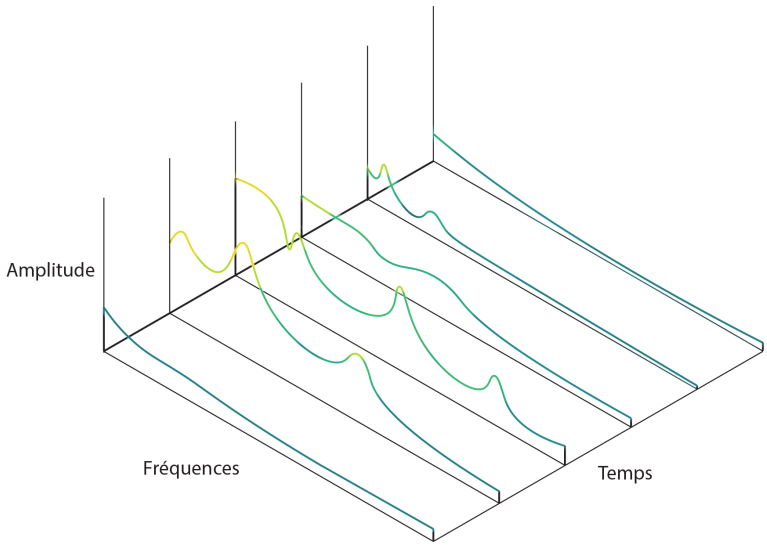


Figure 6 : Passage de l'oscillogramme au spectrogramme, vu en trois dimensions

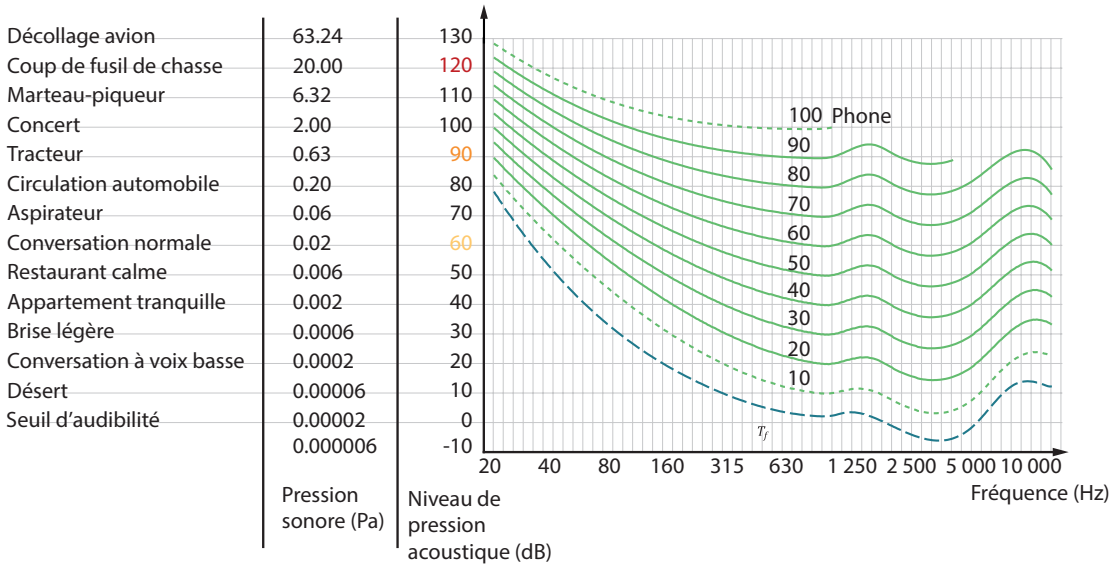


Figure 7 : Echelle des décibel et courbe isophonique, illustrant la différence de niveau de pression acoustique perçue par l'humain, selon la norme ISO 226:2023

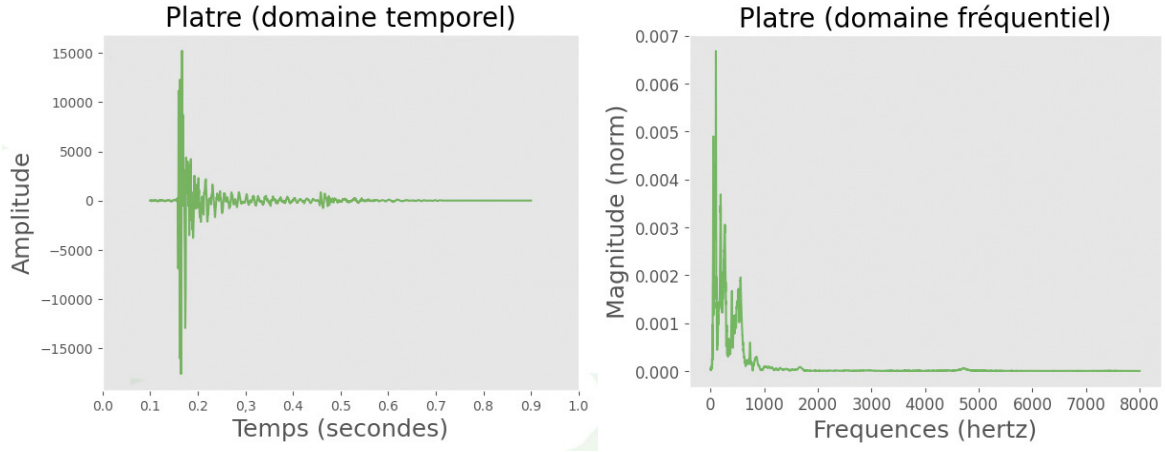


Figure 8 : Forme d'onde et spectre de fréquences

la transformer afin de créer de toutes nouvelles données à partir du son capturé au format numérique.

Pour aller plus en détail, il y a deux éléments importants dans le signal sonore numérique : la fréquence d'échantillonnage et la profondeur de bits (la quantification).

Mesurée en Hertz, la **fréquence d'échantillonnage** est le nombre d'échantillons prélevés en une seconde. Les petites variations peuvent échapper au signal résultant, celles qui correspondent aux fréquences les plus élevées. On définit alors une fréquence d'échantillonnage suffisante pour reproduire le son correctement. D'après le théorème de Nyquist-Shannon, la fréquence d'échantillonnage doit correspondre au double de la fréquence à capturer. Sinon, le signal n'est pas fidèlement reconstitué. Nous, êtres humains, percevons jusqu'à 20 kHz. C'est pour cette raison que la fréquence d'échantillonnage standard des CD est à 44,1 kHz.

La **profondeur de bits** représente le nombre de bits utilisés pour coder chaque échantillon. Elle indique donc le degré de précision de la valeur d'amplitude échantillonnée. On aura alors 2n valeurs possibles d'intensités sonores, communément appelées étapes. Les profondeurs de bits les plus courantes sont de 16 bits, soit de 216 = 65 536 étapes, et de 24 bits, soit 224 = 16 777 216 étapes.

Il faut savoir que les valeurs enregistrées représentent les amplitudes, en décibels. Ces unités logarithmiques reflètent bien la manière dont l'oreille humaine perçoit le son. Notre oreille n'interprète en effet pas les sons de manière linéaire. Nous sommes plus sensibles aux petites variations des sons faibles qu'aux petites variations de sons forts. Notre sensibilité est décroissante selon une logique logarithmique (figure 7).

La composition d'un signal a besoin d'être représentée pour en faciliter la compréhension et la transmission. La représentation graphique peut prendre la forme d'oscillogrammes, comptant la forme d'onde et le spectre de fréquences, et de spectrogrammes, tels que le spectrogramme log et le spectrogramme mel.

L'**oscillogramme** consiste à représenter l'onde sous forme d'une courbe qui oscille verticalement le long d'un axe x.

La **forme d'onde** (figure 8 à gauche) est une représentation du domaine temporel d'un son. Sur les ordonnées, les amplitudes sont exposées, tandis

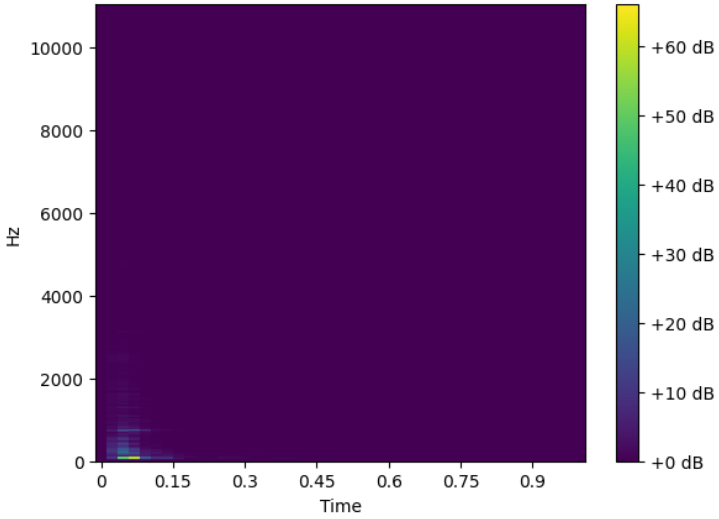


Figure 9 : Spectrogramme d'un son d'impact sur un mur en parpaing

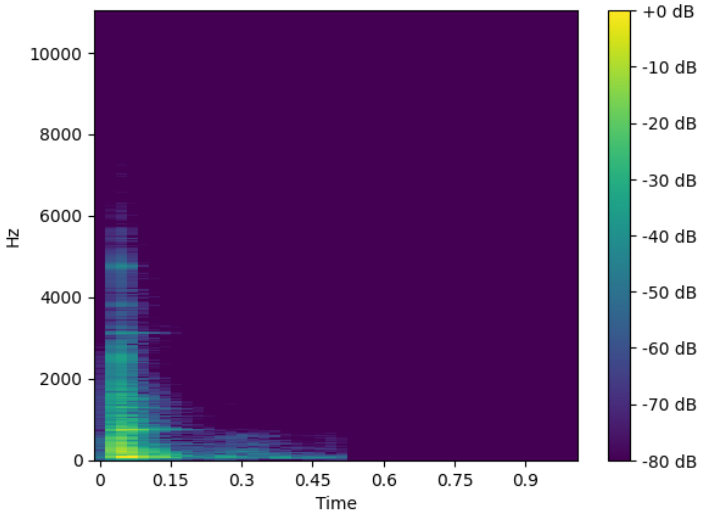


Figure 10 : Spectrogramme de décibels d'un son d'impact sur un mur en parpaing

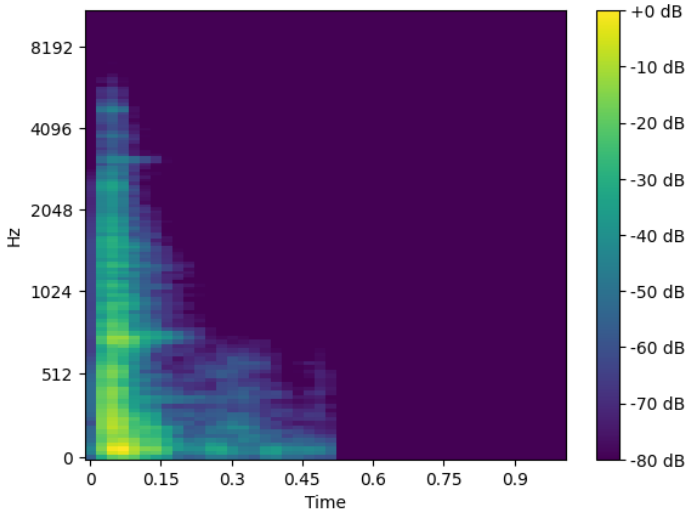


Figure 11 : Spectrogramme mel de décibels d'un son d'impact sur un mur en parpaing

que sur les abscisses figure le temps.

Le **spectre de fréquences** (figure 8 à droite) est une représentation du domaine fréquentiel sur un court segment du signal sonore. Il est possible de tracer le spectre de l'ensemble du son, mais il est plus intéressant de ne s'intéresser qu'à une petite partie. Cela permet de détailler les fréquences individuelles qui composent le signal (abscisses) selon leur amplitude ou leur magnitude (ordonnées) grâce à un procédé mathématique que l'on appelle la transformée de Fourier discrète (FTD). Cette donnée est particulièrement importante, car les matériaux de construction se distinguent par leurs propriétés de résonance ou d'absorption, associées à des fréquences spécifiques.

Le **spectrogramme**, quant à lui, pourrait être comparé à une photographie du spectre détaillé d'un signal sonore dans le temps.

Le spectrogramme (figure 9) est construit à partir de courts segments du signal de quelques millisecondes. On applique la FTD sur une série de fenêtres de temps très courtes. On parle alors de transformée de Fourier à court terme (TFCT). Les spectres résultants sont empilés le long d'un axe temporel. En somme, c'est l'assemblement de plusieurs FTD côte à côte, ce qui permet d'apporter une troisième dimension : le temps (figure 6). Trois mesures sont mises en évidence : l'amplitude ou la puissance de la fréquence traduite par la couleur, la fréquence sur l'axe des y et le temps sur l'axe des x. Le spectrogramme représente l'ensemble des fréquences composant le signal sonore et leur amplitude au fil du temps. Cela permet de rassembler à la fois les données montrées par la forme d'onde (amplitude et temps) et le spectre de fréquences (décomposition du signal en toutes ses fréquences). Finalement, si on comparait la représentation du son à la représentation architecturale, le spectrogramme serait le plan topographique et le spectre de fréquence, une coupe.

Différents processus peuvent s'ensuivre.

Le premier processus consiste à appliquer une formule logarithmique pour mettre en évidence les variations de sons de faibles niveaux sonores. Il en résulte un **spectrogramme de décibels** ou logarithmique (figure 10).

Le second processus permet d'obtenir le **spectrogramme mel**. Il est tiré d'une observation. Comme dit précédemment, le système humain est plus sensible aux faibles variations des sons de faibles niveaux sonores qu'aux forts niveaux. Il est également plus sensible aux petits changements dans

les fréquences basses que dans les fréquences élevées. Notre sensibilité est décroissante dans une logique logarithmique (figure 7).

Le **spectrogramme mel** (figure 11) est donc une variante du spectrogramme. Au moment d'effectuer la TFCT pour passer des spectres de fréquences au spectrogramme, un ensemble de filtres est mis en œuvre, nommé mel filterbank. Ces filtres permettent de construire les différentes bandes du spectrogramme, autrement dit des gammes de fréquences. On passe alors d'un spectrogramme à un axe y à une échelle linéaire à une échelle logarithmique, donc mel. Il faut noter que le signal étant filtré, certaines données sont perdues. En outre, ce type de représentation est principalement utilisé pour la reconnaissance vocale ou encore pour la classification des genres musicaux, car il se rapproche le plus de la perception humaine.

> À partir de ces informations, j'en ai conclu que les objets de même matériau doivent avoir, eux aussi, leur signature spectrale. Je ne savais que choisir au début de ma recherche. Mais au fil de mes recherches, j'ai remarqué que le spectrogramme log-mel était valorisé pour la classification. J'ai donc choisi de nourrir mon réseau de neurones de spectrogrammes log-mel. Ce choix peut être remis en question, car il n'est pas nécessaire de reproduire la perception humaine. S'en émanciper pourrait permettre de percevoir des caractéristiques non audibles, encore faut-il que le dispositif d'enregistrement puisse capturer toute la plage de fréquences importantes pour la reconnaissance.
>
> Suite à l'acquisition des connaissances fondamentales sur le signal sonore numérique, l'étape suivante consiste en l'identification d'un modèle d'intelligence artificielle adapté à la classification audio.

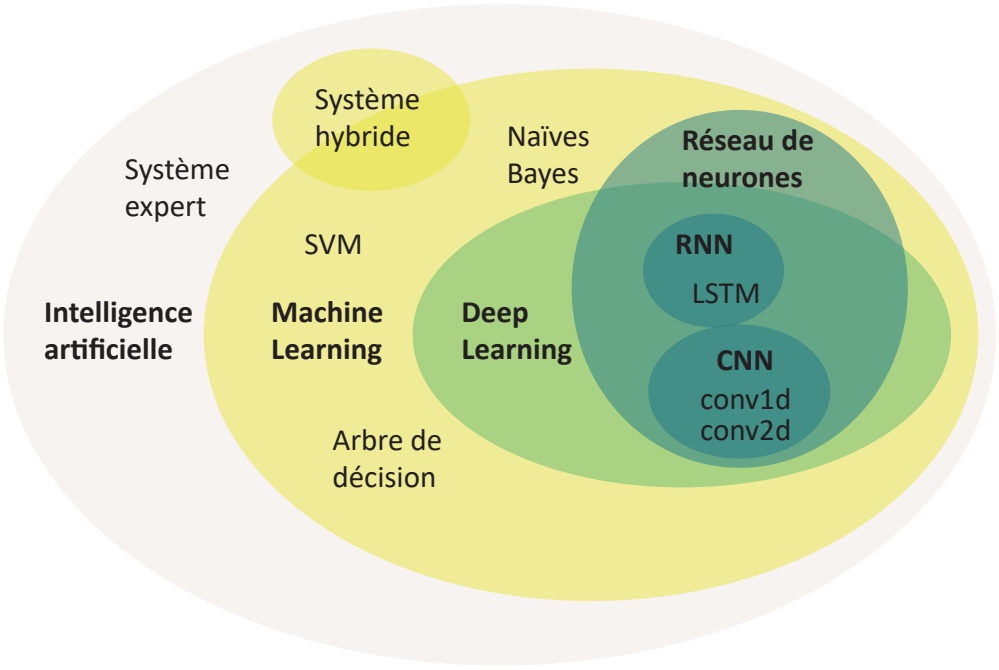


Figure 12 : Ensemble et sous-ensemble de l'intelligence artificielle

2 La classification audio par les réseaux de neurones

Il convient maintenant d'entrer plus en profondeur dans l'univers du numérique en illustrant les rouages de l'intelligence artificielle et de la classification par l'ordinateur. Cette sous-partie permettra de donner les clés de compréhension des algorithmes, en commençant par la définition des **intelligences artificielles**, puis de l'**Apprentissage Machine** (Machine Learning) et de l'**apprentissage profond** (Deep Learning) pour finir sur les **réseaux de neurones**, particulièrement du réseau récurrent à mémoire court et long terme (**LSTM**) et des réseaux neuronaux convolutifs, **Conv1D** et **Conv2D**. Des mécanismes comme l'apprentissage supervisé et la descente de gradient, la data augmentation seront développés.

a L'intelligence artificielle

L'intelligence artificielle (IA ou AI en anglais) s'introduit dans notre vie quotidienne, sur nos téléphones et nos ordinateurs. Ce terme, au départ donné à des outils qui ont pour but d'imiter l'intelligence humaine, est omniprésent. Il est employé sans cesse pour désigner des modèles algorithmiques dont les structures sont distinctes. Il englobe en effet l'Apprentissage Machine, qui renferme lui-même l'apprentissage profond (figure 12). On pourrait alors parler d'intelligences artificielles au pluriel.

Ce que le grand public entend par intelligence artificielle, ce sont des modèles algorithmiques qui nous questionnent sur la singularité humaine, car ils semblent la mettre en concurrence par l'utilisation du langage ou de l'image. L'IA est le robot qui nous affronte lors d'une partie de dames, d'échecs ou encore de go. Mais ce sont, précisément, des techniques de résolution d'un certain nombre de problèmes par des calculs. La classification des courriers électroniques indésirables. Les publicités personnalisées dans nos navigateurs. Les systèmes de recommandations de vidéos ou de produits. Le système de recherche de Google par la pertinence. Ou encore les prévisions météorologiques. En somme, l'intelligence artificielle nous entoure, et il peut parfois être difficile de faire la distinction entre les algorithmes qui font partie de la sphère de l'IA et ceux qui ne la font pas.

La volonté de reproduire l'intelligence humaine est par ailleurs remise

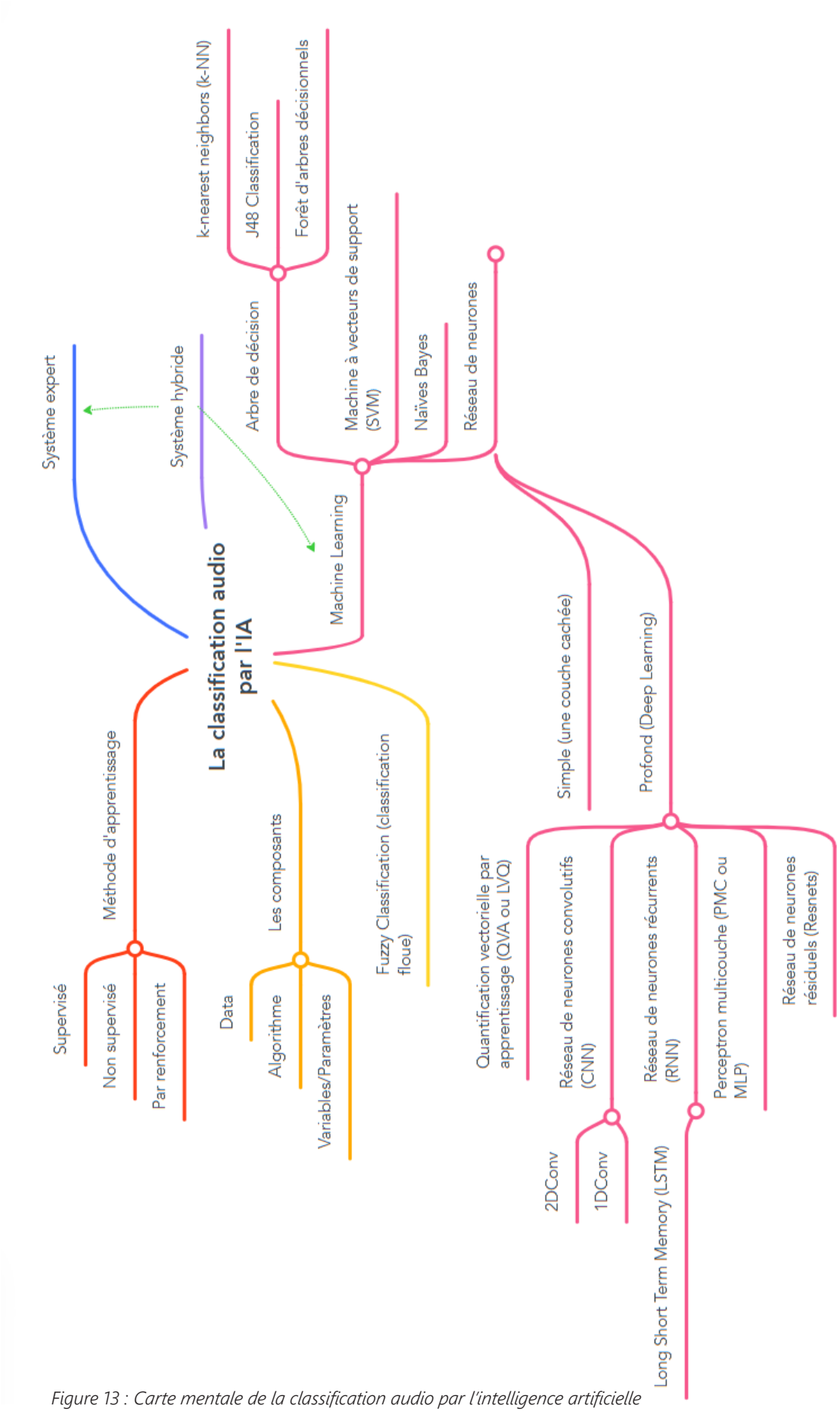


Figure 13 : Carte mentale de la classification audio par l'intelligence artificielle

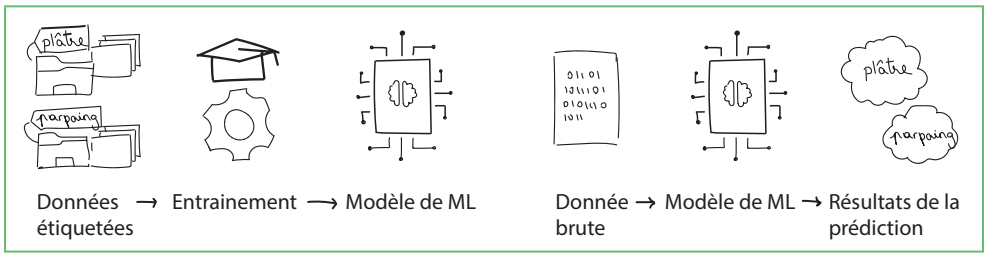
en question (Marsault et Nguyen, Negrotti)[15]. La remise en cause de la singularité humaine est écartée dans le domaine de la créativité. La machine est capable de réaliser des éléments selon une logique différente de celle des humains. L'expertise humaine n'est alors pas remplaçable et l'IA augmente nos capacités. « AI » devient alors l'abréviation de « Augmented Intelligence » ou de « Alternative Intelligence », montrant la complémentarité entre l'humain et la machine.

De manière générale, les algorithmes d'intelligence artificielle transforment une donnée une ou plusieurs fois pour aboutir à une nouvelle donnée. Celle-ci est la réponse attendue par l'utilisateur. Il existe une multitude de structures d'intelligence artificielle pour la classification audio, couramment utilisée pour la classification de la musique par genre ou pour la reconnaissance vocale.

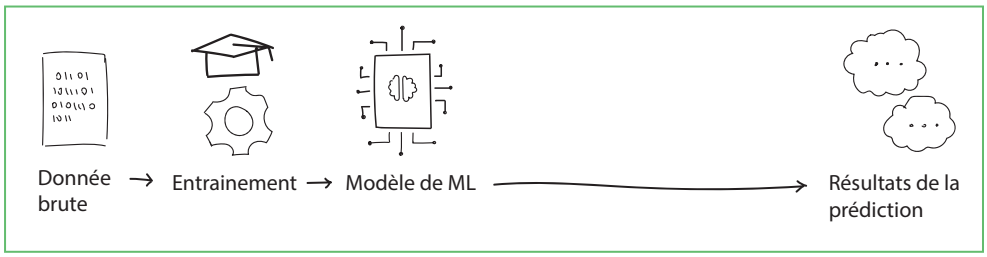
Un panorama des modèles d'intelligence artificielle les plus utilisés dans le cadre de recherches scientifiques pour la reconnaissance audio a été dressé (figure 13). La carte mentale n'est cependant pas une liste exhaustive des algorithmes existants pour la classification audio tant leur nombre est important.

Pour synthétiser, le système expert prédit en suivant des règles strictes définies en amont par l'utilisateur. Le système hybride regroupe plusieurs algorithmes d'intelligence artificielle, de l'ordre du système expert et/ou de l'apprentissage machine, que je développerai dans quelques lignes. Enfin, la classification floue est une technique qui consiste à attribuer des degrés d'appartenance aux classes données. Au lieu de fournir une réponse unique, elle génère plusieurs réponses, chacune associée à un pourcentage d'appartenance. La classification floue peut être implémentée au système expert et à l'apprentissage machine. Le système expert, le système hybride et la classification floue n'étant pas les objets de cette recherche, ils ne seront pas développés davantage. En revanche, l'apprentissage machine forme le cœur du sujet.

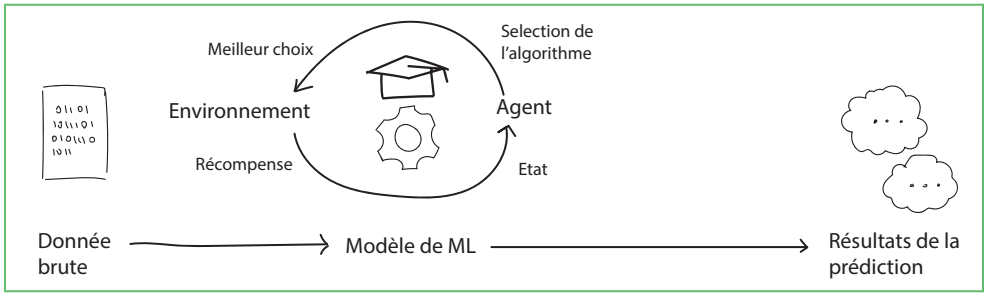
15 Xavier Marsault et Hong Minh-Chau Nguyen, « Les GANs : stimulateurs de créativité en phase d'idéation », in SHS Web of Conferences, éd. par K. Jacquot et H. Lequay, vol. 147, SHS Web of Conferences (Lyon, France, 2022), 06003, https://doi.org/10.1051/shsconf/202214706003.



Apprentissage supervisé



Apprentissage non supervisé



Apprentissage par renforcement

Figure 14 : Les techniques d'apprentissage

b Le Machine Learning

Enfin, le Machine Learning (ML), une branche de l'IA, désigne le processus et les **algorithmes** capables d'**apprendre des données** qu'on leur soumet. Sans règles strictes, le Machine Learning transforme la donnée et en identifie des patterns. Il est important de garder en tête que le langage informatique est numérique, toute l'information, même textuelle, est chiffrée, quantifiée et transformée par des calculs avant qu'elle ne nous parvienne à travers un écran. Pour résoudre des problèmes, les algorithmes construisent des modèles à partir de données en jouant sur des **paramètres variables**. Des prédictions auront enfin lieu à partir de nouvelles données.

Nous pouvons séparer les méthodes de ML selon leur méthode d'apprentissage (figure 14) : l'apprentissage supervisé, l'apprentissage non supervisé, l'apprentissage par renforcement et l'apprentissage semi-supervisé.

L'**apprentissage supervisé** consiste à former un modèle à l'aide d'un ensemble de données étiquetées. Ainsi, les phases d'entraînement et de test de l'algorithme se divisent en plusieurs boucles (epochs ou époques). À la fin de chaque boucle, un algorithme compare les résultats aux étiquettes afin d'optimiser les résultats en ajustant ses paramètres pour minimiser les erreurs. Cette technique est la plus utilisée pour la **classification**.

Dans l'apprentissage non supervisé, les éléments caractéristiques des données sont déterminés sur un jeu de données non étiqueté. Il peut par exemple être utilisé pour le clustering, un algorithme de partitionnement de données, ou pour la détection d'anomalie. L'apprentissage semi-supervisé est utilisé dans le cas où le jeu de données est partiellement étiqueté.

L'apprentissage par renforcement consiste à récompenser les comportements souhaités et à pénaliser ceux à éviter par la mise en place d'un score, par exemple. Le but est donc de maximiser les récompenses sur le long terme grâce à une stratégie, que l'on nomme « politique ».

Étant donné que la recherche porte sur la classification, il serait intéressant de comprendre comment un algorithme d'apprentissage supervisé apprend. Au centre de ces techniques d'apprentissage se déroulent des calculs pour la quête du meilleur modèle de Machine Learning. Le premier essai démarre par la sélection de poids aléatoires. Cette approche empirique est basée sur un ajustement itératif des paramètres pour se rapprocher de

la solution optimale. Il faut d'abord analyser les résultats des prédictions et les comparer avec les valeurs réelles. L'écart entre les prédictions du modèle et les résultats est mesuré, appelé **fonction de perte**[16]. Plus l'écart est faible, plus le modèle est performant. On peut visualiser cette fonction comme un paysage embrumé, où l'objectif est de descendre au point le plus bas de la vallée (minimum global). L'analyse découle de la recherche pour l'optimisation. Le **gradient** caractérise la pente locale de la fonction de coût par rapport aux paramètres du modèle. Il donne alors, telle une boussole, la direction à suivre : la descente la plus raide. Puis, la stratégie pour descendre est adoptée. La descente de gradient, un des **algorithmes d'optimisation** les plus courants, détermine la taille du pas pour minimiser la fonction de perte, qui est une fonction convexe. Il permet d'ajuster les paramètres en fonction du gradient tout en contrôlant la taille de l'ajustement. Les pas représentent le **taux d'apprentissage**. Au fil des entraînements, le modèle s'améliore. Progressivement.

Or, des difficultés peuvent survenir, comme l'explosion ou la disparition du gradient.

L'explosion du gradient se produit lorsque les valeurs des gradients deviennent excessivement grandes. Les pas deviennent alors trop importants. On oscille alors autour de l'objectif sans s'en rapprocher. Cela peut entraîner des ajustements incontrôlés des paramètres. L'entraînement devient instable ou empêche le modèle de converger.

La disparition du gradient intervient lorsque les gradients deviennent si petits qu'ils sont presque infimes. Cela arrive particulièrement dans les premières couches du réseau. Les mises à jour des paramètres sont presque nulles. L'apprentissage prendrait un temps infini pour atteindre l'objectif.

Ces deux problèmes peuvent survenir plus ou moins fréquemment selon les modèles de ML. Ils empêchent le réseau d'apprendre. Par conséquent, d'autres techniques et d'autres algorithmes d'optimisation peuvent y remédier.

16 La fonction de perte est également appelée la fonction de coût ou la fonction d'erreur.

Comprendre les modes d'apprentissage du Machine Learning donne à voir les types d'algorithmes les plus adaptés à la classification. L'apprentissage supervisé semble ainsi être la technique la plus à même de répondre aux exigences de l'expérience. La branche du Machine Learning se ramifie, incluant parmi ses rameaux le Deep Learning, qui regroupe des algorithmes spécialisés dans le traitement de données complexes comme le son et l'image.

c Le Deep Learning

Généralement, on parle de Machine Learning lorsque la donnée n'est transformée qu'une fois avant de formuler la prédiction. Lorsque la donnée subit plusieurs transformations successives avant de générer la donnée de sortie, donc la prédiction, il est question de **Deep Learning** (DL). Le DL représente donc une avancée par rapport aux algorithmes traditionnels en raison de la capacité à traiter des données complexes.

Les algorithmes sont organisés en plusieurs couches d'opérations (figure 15), permettant une transformation progressive de la donnée. La première couche, celle qui reçoit nos données, se nomme la **couche d'entrée**, et la dernière couche est la **couche de sortie**. Les couches intermédiaires sont connues sous le nom de **couches cachées**. Cette hiérarchie de transformation permet au modèle d'extraire des caractéristiques de plus en plus abstraites à chaque étape, ce qui rend cette approche particulièrement adaptée aux données à haute dimensionnalité. Ils reposent sur des architectures de **réseaux de neurones.** Le processus de transformation dans un réseau de neurones s'inspire, dans une certaine mesure, du fonctionnement des neurones biologiques.

Dans un cerveau humain, les dendrites reçoivent les signaux provenant d'autres neurones, qui sont ensuite transmis à travers l'axone vers d'autres cellules nerveuses via des synapses. De manière similaire, dans un réseau de neurones artificiel, les données sont reçues par les neurones d'une couche. Tout commence lorsqu'un neurone reçoit des signaux d'entrée. Chaque neurone détermine s'il doit ou non s'activer en fonction de son **biais** et des **poids** associés aux connexions. Si ce dernier s'active, il transforme l'information et envoie le résultat aux neurones de la couche suivante. Ce processus se répète sur les différentes couches du réseau, chaque couche ajoutant une dimension de transformation.

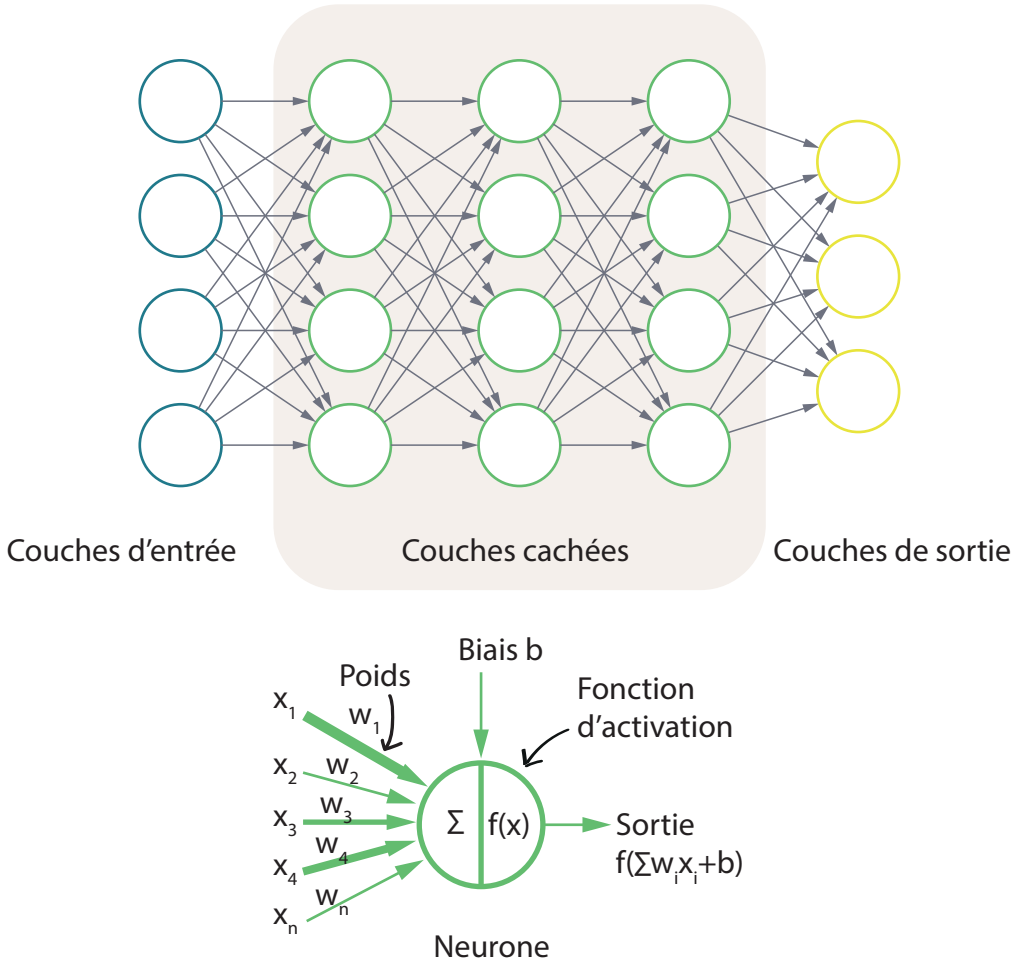


Figure 15 : Architecture de Deep Learning en réseau de neurone

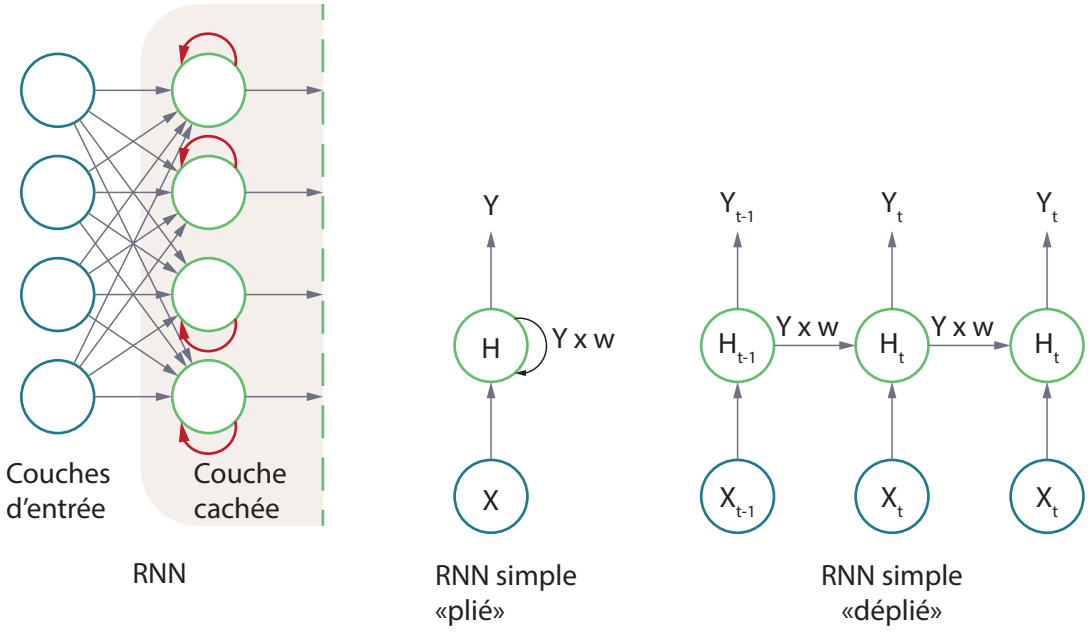


Figure 16 : Schéma d'un réseau de neurone récurrent (RNN)

Chaque neurone dans un réseau de neurones artificiel peut être vu comme une unité de traitement. À l'instar des neurones biologiques, ils génèrent une réponse lorsqu'ils sont stimulés au-delà d'un seuil électrique. Ce **seuil** est défini par la **fonction d'activation.** Ainsi, l'activation progressive des neurones à travers les couches successives dans un réseau de neurones artificiel permet de transformer et d'abstraire les informations de manière de plus en plus complexe, jusqu'à aboutir à une décision ou une prédiction[17].

Sans aborder davantage la partie technique du Deep Learning, nous pouvons à présent nous intéresser à trois architectures de réseaux de neurones, supports de l'expérience. Les **RNN**, particulièrement le **LSTM**, et les **CNN**, notamment le **Conv1D** et le **Conv2D**, sont deux des techniques les plus couramment utilisées pour la classification audio.

d Long-Short Time Memory (LSTM)

Les réseaux de neurones récurrents, ou recurrent neural network en anglais (RNN), ont l'avantage de lire les données séquentielles et des séries temporelles. Leur principal atout réside dans leur capacité à comprendre les relations internes d'une séquence d'entrée. Cela leur permet de modéliser les séquences efficacement et d'en comprendre les dynamiques sous-jacentes.

Les séquences temporelles sont des suites de données numériques. Elles sont traitées segment par segment dans un ordre précis, habituellement dans l'ordre chronologique.

L'idée fondamentale des RNN est que le réseau est constitué de neurones récurrents. Ainsi, une couche est composée d'un neurone qui traite la donnée un temps après l'autre, comme nous le ferions en lisant un livre. Il découvre le premier segment Xn. Il fait ses premiers calculs et produit un résultat Yn. Puis il garde en mémoire la réponse Yn. L'état de cette mémoire à un instant t est appelé le « hidden state », l'état caché[18] Hn. Il découvre le deuxième segment, Xn+1. Il effectue les mêmes opérations, mais cette fois en ajoutant à l'équation ce qu'il a gardé en mémoire. Les données d'entrée sont donc Xn+1 et Hn. Il génère une nouvelle réponse Yn+1. Il garde en mémoire la réponse, l'état caché est réactualisé, devenant

17 Shanaka Kristombu Baduge et al., « Artificial intelligence and smart vision for building and construction 4.0: Machine and deep learning methods and applications », Automation in Construction 141, 1 septembre 2022. https://doi.org/10.1016/j.autcon.2022.104440.

18 L'état caché exprime donc la mémoire à court terme du réseau de neurones.

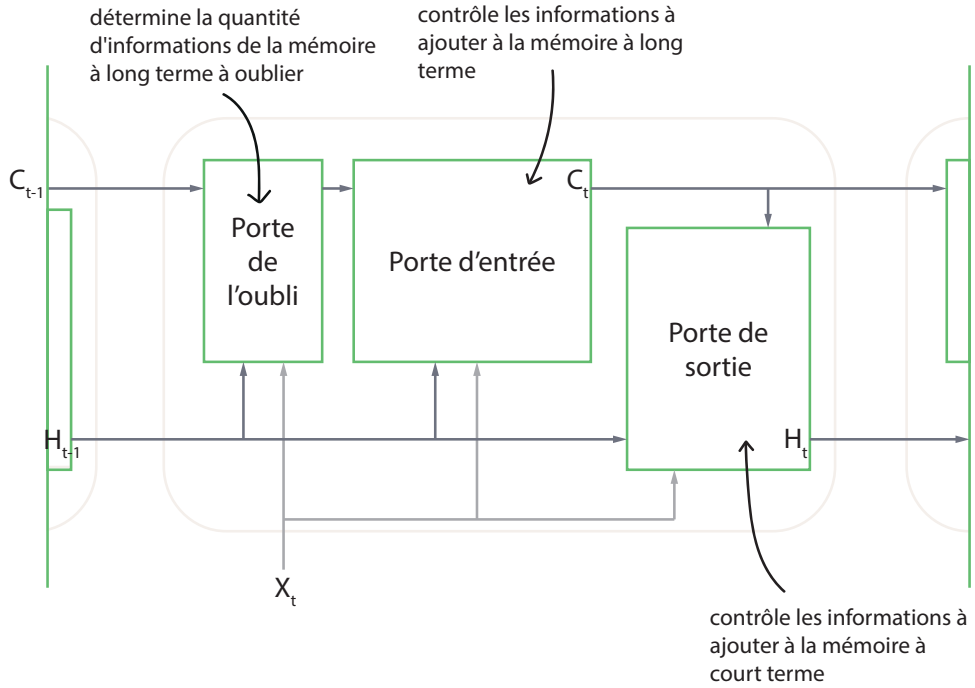


Figure 17 : Une unité LSTM

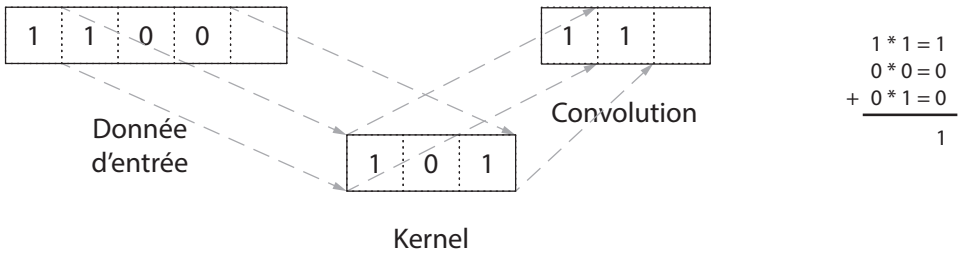


Figure 18 : Produit de convolution d'un Conv1D

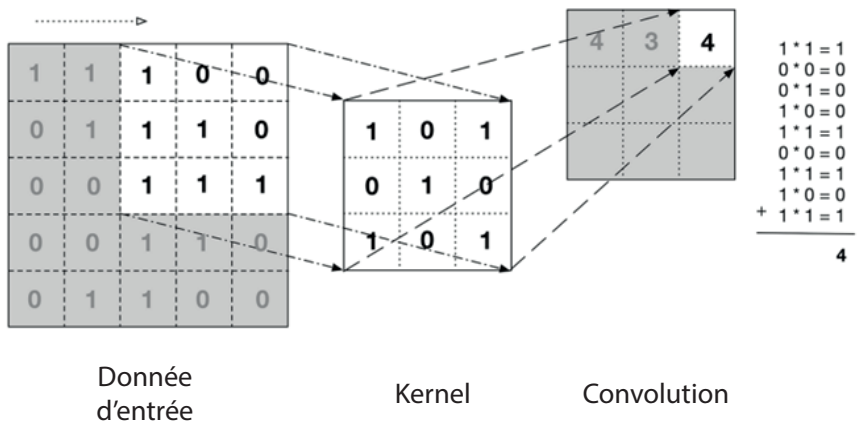


Figure 19 : Produit de convolution d'un Conv2D

Hn+1. Et l'opération se répète ainsi jusqu'à la fin de la séquence. Le réseau maintient et met à jour les informations au fil du traitement de la séquence. En termes de représentation, il est donc préférable de mettre en exergue la temporalité. Je peux « dérouler » le réseau afin de montrer le neurone au fil du temps (figure 16). Le neurone à un instant t est appelé unité. Le neurone dans sa globalité est plutôt nommé cellule. Cela vient notamment du fait que les **variantes du RNN sont bien plus complexes et que le mot neurone n'est plus en adéquation avec le modèle montré sur la figure 15.**

Par ailleurs, toutes les informations de la séquence sont plus ou moins gardées. À titre d'exemple, dans une phrase, plus les mots sont éloignés du dernier mot lu et moins ils ont d'importance. Pour faire une prédiction, c'est le dernier mot qui aura le plus de poids. Ceux qui sont beaucoup plus éloignés sont donc oubliés. Cette mémoire à court terme permet au réseau de capturer les dépendances temporelles locales dans les données. Cependant, ce fonctionnement est le principal défaut du RNN. Le modèle devient instable. Il apprend difficilement les caractéristiques à plus long terme. Les gradients deviennent si petits ou si grands en valeur que les poids ne peuvent être réajustés convenablement. Les gradients explosent ou disparaissent.

Le LSTM[19], une variante du réseau de neurones récurrents, pallie ces problèmes. Il prend en plus d'un segment de la séquence et de l'état caché de l'unité précédente un troisième élément qui n'est autre qu'un nouvel état conservant des informations sur du plus long terme, l'état cellulaire (figure 17). Grâce à ces propriétés, les LSTM sont largement utilisés dans des applications telles que la traduction automatique, la reconnaissance vocale et l'analyse vidéo.

19 Hochreiter, Sepp. Schmidhuber, Jürgen. «Long Short-Term Memory». Neural Computation 9(8):1735-1780, 1977. doi:10.1162/neco.1997.9.8.1735

e Deux réseaux de neurones convolutifs (CNN)

Le réseau de neurones convolutifs est adapté à l'analyse des données comme les images. Il se trouve qu'il **filtre les données** et diminue la taille de l'image une ou plusieurs fois pour classifier.

L'algorithme consiste en un glissement des filtres de petite taille, des kernels, sur l'ensemble de l'image d'entrée et produit une nouvelle matrice (carte de caractéristiques) issue de l'application des filtres grâce à un procédé mathématique, le produit convolutif. Les filtres sont ajustés en jouant sur des poids pour qu'à la fin, la matrice fasse ressortir les caractéristiques spécifiques à chaque classe. Les CNN analysent ainsi l'image à différentes échelles grâce à des couches successives de convolution et de **sous-échantillonnage**, de pooling en anglais, permettant de déceler des motifs de plus en plus abstraits. Toutes les unités de traitement d'une couche ne sont pas interconnectées aux unités des couches suivantes. En parallèle, les unités des dernières couches qui sont entièrement connectées pour établir la classification finale, sont appelées **fully connected layer**.

La performance du CNN dépend de plusieurs hyperparamètres clés: la taille des filtres, leur nombre, le padding et le stride. Le padding est l'opération qui consiste à ajouter des pixels autour des bords de l'image d'entrée pour pouvoir appliquer les filtres et préserver les informations. Le stride est le pas de déplacement du filtre sur l'image lors de la convolution. Plus le stride est grand, plus la dimension de sortie est réduite, car moins de convolutions sont effectuées.

De nombreuses architectures CNN ont été développées par les chercheurs, comme AlexNet, VGGNet et ResNet, chacune apportant ses propres innovations pour améliorer les performances de reconnaissance. Les réseaux de neurones convolutifs sont très utilisés pour la reconnaissance d'objets, la classification d'images et de vidéos et se retrouvent dans de nombreuses applications quotidiennes comme la reconnaissance faciale sur smartphones ou les systèmes de surveillance.

Conv1d

Le Conv1D est une variante des CNN adaptée aux séquences temporelles unidimensionnelles (comme le son ou les signaux). Les **filtres glissent le long d'une seule dimension temporelle** (figure18). Bien qu'il traite des données séquentielles comme le LSTM, l'approche est fondamentalement différente : l'algorithme se concentre sur la détection de motifs locaux.

Conv2d

Le Conv2D est spécialement utile pour les données bidimensionnelles comme les images (figure 19). Contrairement au Conv1D qui examine une séquence, le Conv2D opère sur deux dimensions de l'image, la hauteur et la largeur. Les kernels sont donc des matrices 2D qui se déplacent sur toute la surface de l'image, permettant de détecter des caractéristiques spatiales comme les bords, les textures et les formes complexes. Cette architecture est particulièrement efficace pour la reconnaissance de motifs spatiaux, car elle prend en compte le contexte environnant de chaque pixel dans les deux dimensions.

f La classification audio

Les algorithmes présentés acceptent des données standardisées, de natures spécifiques. Au format Waveform Audio File Format (WAV), les fichiers sont sauvegardés sans perte à l'enregistrement. Les fréquences d'échantillonnage et les profondeurs de bits des différents fichiers doivent être identiques. Par ailleurs, les modèles d'apprentissage fonctionnent sur des données en virgule flottante.

Il est possible d'enregistrer directement avec les paramètres adéquats. Mais il est tout à fait envisageable de modifier les propriétés à la suite de l'enregistrement. Toutefois, les formats compressés comme le mp3 ne peuvent être convertis correctement au format WAV par manque de données.

L'intelligence artificielle enveloppe un grand nombre d'algorithmes aux architectures très différentes. Il faut donc que je fixe mes objectifs, je souhaite faire une classification audio des matériaux de construction. Je ne connais pas les caractéristiques exactes des signaux sonores pour moi-même les différencier. Je ne peux pas utiliser un système expert. Je peux en revanche classer les séquences sonores de sons d'impact en fonction des matériaux. Mes données sont labélisées. J'en déduis que le machine learning supervisé est pertinent. Il n'y a pas de méthode infaillible pour déterminer quel modèle se prête le mieux à ma requête. Cependant, au vu de la complexité des données, je fais le choix d'explorer les modèles de deep learning, au travers des CNN 1d et 2d et du LSTM.

PROBLÉMATIQUE

Les métiers d'études techniques en architecture reposent largement sur l'emploi de dispositifs numériques. Les techniques de sondage renommées sont réalisées par des machines, car elles sont gages de fiabilité dans la mesure où leur technologie est sophistiquée. Les dispositifs sont néanmoins onéreux et demandent une expertise spécifique. Par ailleurs, la connaissance des matériaux de construction implique une série de recherches, en commençant la datation de l'édifice à l'aide de documents d'archives par exemple, et se poursuivant par la formulation d'hypothèses. Pour confirmer la nature des matériaux, il est souvent nécessaire de conduire des sondages destructifs, ne laissant pas le bâti réellement intact.

En parallèle, l'intelligence artificielle s'est développée. Elle donne la possibilité d'effectuer un grand nombre de calculs rapidement, de transformer la donnée pour en extraire des informations efficacement. Il est indéniable qu'elle a bousculé les méthodes de travail traditionnelles par l'automatisation de certaines tâches (Di Gui, 2021. Shanaka Kristombu Baduge, 2022). Bien que l'intelligence artificielle soulève également des questions éthiques et sociétales, celles-ci ne seront pas abordées dans ce mémoire. Le dessein de mon travail est d'explorer les possibles.

J'ai constaté que des modèles algorithmiques d'une architecture de CNN et de LSTM peuvent classifier des sons dans des domaines d'application autres que celui de la construction, comme le recyclage ou la musique. L'efficacité de la classification des matériaux des objets par l'analyse acoustique de leurs sons d'impact par un CNN a été démontrée dans la littérature scientifique (Dimiccoli, 2021).

Ces techniques informatiques de Deep Learning sont à moindre coût. Le savoir est accessible sur internet. Des logiciels libres et open source sont disponibles. Toute un chacun dispose des ressources pour déployer un nouveau modèle algorithmique grâce au réseau informatique mondial.

C'est pourquoi je propose d'appliquer les techniques existantes dans le domaine de la construction. Je suggère d'élaborer une méthode de sondage non invasive pour identifier rapidement les matériaux de construction sur site. La solution serait économique pour l'utilisateur qui possède déjà un

accès à internet et un microphone, comme celui intégré à un téléphone. Cette initiative encouragerait une démarche de connaissance du bâti, non seulement pour un intérêt technique mais également patrimonial. D'une part, la documentation et la préservation de notre patrimoine architectural seraient consolidées. D'autre part, cette solution simplifierait les phases préliminaires de projet en permettant une évaluation des matériaux sans recourir à des analyses coûteuses ou destructives.

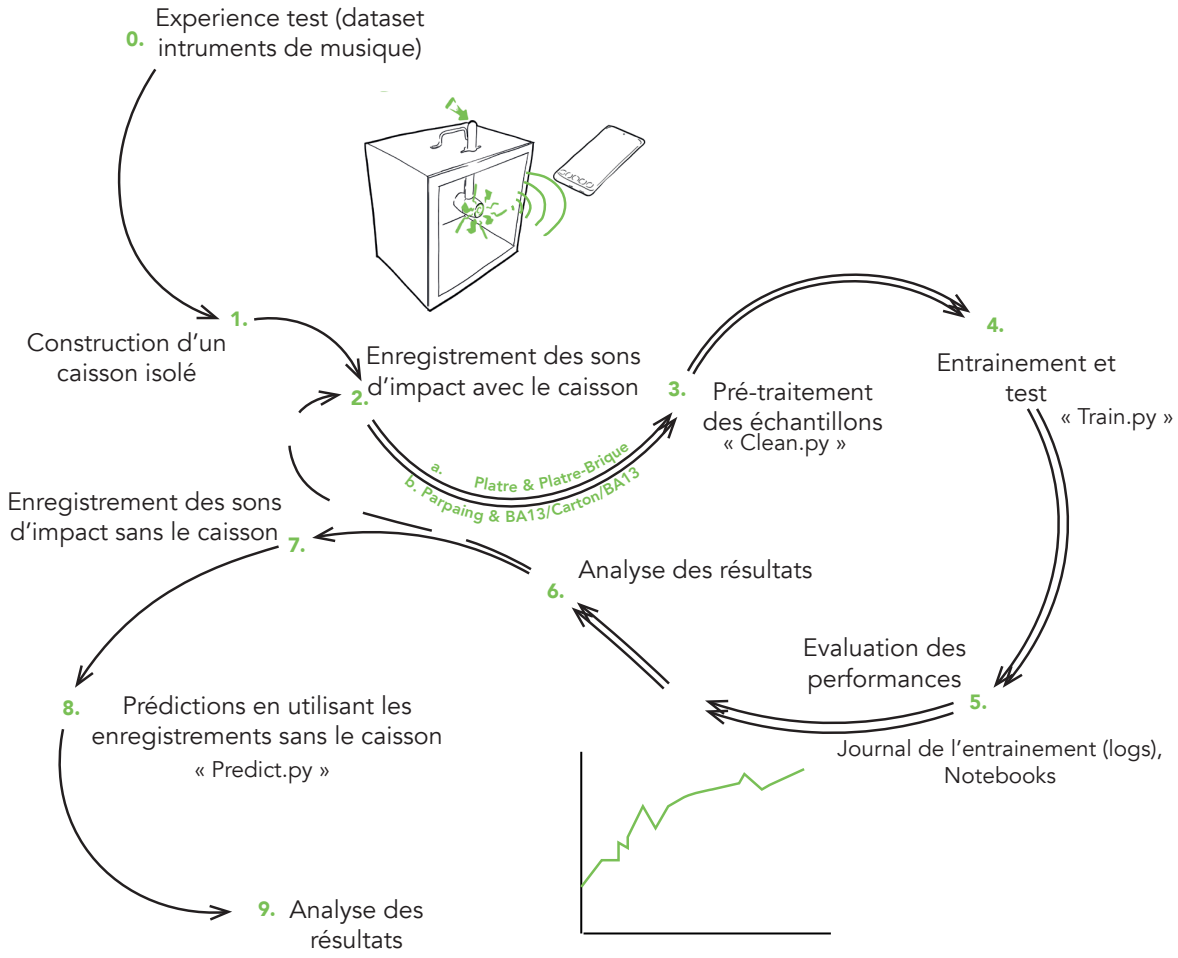
Comment identifier les matériaux de construction par un sondage non destructif et économique ? Des réseaux de neurones peuvent-ils classifier les matériaux dans un contexte bâti par le biais de sons d'impact ? Quelles en sont leurs performances ?

Comment les réseaux LSTM, Conv1D et Conv2D peuvent-ils identifier les matériaux de construction à partir de leurs signatures acoustiques d'impact?

Les recherches précédentes, exposées dans la partie sur l'état de l'art, m'ont amenée à faire l'hypothèse que le LSTM, le Conv1D et le Conv2D sont capables de classifier des matériaux de construction de manière fiable grâce à des sons d'impact. Cet objectif serait réalisable en construisant moi-même les données (dataset) d'enregistrements sonores avec mon téléphone. Il faudrait ensuite entraîner et tester les réseaux de neurones avec un ordinateur connecté à Internet. Le dispositif serait donc simple à mettre en place et surtout portable.

La deuxième hypothèse est que les performances des modèles des trois architectures d'IA varient en fonction des données soumises aux algorithmes. En effet, les bruits externes pourraient réduire leurs performances. La modification de l'instrument de percussion pourrait également affecter les résultats, d'où l'importance de mettre en place un cadre rigoureux lors de l'expérience. Le corpus est constitué de séquences sonores d'une seconde de sons d'impact par un objet sur des murs de plusieurs bâtiments.

Je cherche donc à mesurer la capacité d'une intelligence artificielle à distinguer les matériaux de murs à partir d'un corpus de spectrogrammes de sons d'impact d'un objet sur des murs dont je connais la composition. J'examinerai le nombre de réussites et d'échecs de la classification pour chaque série de tests puis l'évolution de l'apprentissage du modèle.


0. Experience test (dataset intruments de musique)
1. Construction d'un caisson isolé
2. Enregistrement des sons d'impact avec le caisson
a. Platre & Platre-Brique
b. Parpaing & BA13/Carton/BA13
3. Pré-traitement des échantillons « Clean.py »
4. Entrainement et test « Train.py »
5. Evaluation des performances
Journal de l'entrainement (logs), Notebooks
6. Analyse des résultats
7. Enregistrement des sons d'impact sans le caisson
8. Prédictions en utilisant les enregistrements sans le caisson « Predict.py »
9. Analyse des résultats

MÉTHODOLOGIE

Les recherches antérieures m'ont servi de tremplin pour explorer mon hypothèse. J'ai d'abord recherché quels algorithmes étaient les plus susceptibles de répondre à mes besoins. De nombreux programmes sont disponibles en ligne, et j'ai très vite trouvé mon bonheur. J'ai tout de même dû créer mes propres données. J'ai donc procédé comme suit.

La composition exacte de tous les murs qui composent le corpus devait être connue. J'ai donc choisi des logements dont les habitants étaient témoins de la construction de leur maison, certains des matériaux employés. Les édifices étaient tous situés en ville.

Le passage fréquent des voitures, mais également les mobiliers intérieurs, génèrent des sons parasites. L'objectif était d'avoir des sons relativement propres, sans réverbération et avec peu de bruits environnants. Il fallait également que les percussions soient identiques. Un seul outil de percussion, en bois, devait porter toujours la même force, la même direction et le même sens d'un enregistrement à un autre. J'ai donc concocté un caisson isolé muni d'un marteau pour pouvoir effectuer l'expérience dans des conditions optimales. Limiter toutes les variations de la puissance des coups et des bruits dans les lieux a permis de réduire les paramètres qui ont une incidence sur l'expérience.

J'ai réalisé des fiches recensant tous les enregistrements, prenant en compte les caractéristiques des murs qui composent le corpus, l'outil de percussion et surtout les conditions d'enregistrement (figure 20). Cela m'a permis d'avoir des informations sur les données, qui pourraient également être utilisées à d'autres fins dans le cadre d'autres travaux. La totalité des informations a été ensuite notée dans un tableau. Les enregistrements ont été classés selon leur matériau dans des dossiers séparés. Le plus important était de garder une trace de tout ce qui a été fait et de chaque paramètre pour déterminer ce qui aurait pu expliquer certains résultats. C'est pourquoi la tenue d'un journal m'a permis de noter les difficultés, les résultats et les changements adoptés dans mon processus.

FICHE - ENREGISTREMENT N°		
Lieu		Date et heure
CARACTERISTIQUE MUR ET OUTIL		
Matériau de l'outil de percussion □ Bois □ Articulation interphalangienne □ Métal □ Plastique Autre :	Type □ Porteur □ Cloison □ Int/int □ Int/ext	Composition / matériau Epaisseur
CONTEXTE		
Niveau sonore / Intensité sonore □ Fort □ Moyen □ Faible	Bruits de fond	Résonance / Réverbération □ Fort □ Faible □ Non
DATASET		
Type de set	□ Entrainement □ Test	

Figure 20 : Fiche d'enregistrement

J'ai donc effectué une expérience avec une approche itérative. D'une part, j'ai construit un dataset de sons d'impact. D'autre part, j'ai entraîné des algorithmes d'apprentissage profond et j'ai testé les modèles en question pour valider ou non mon hypothèse dans un premier temps.

En cas de réussite, des paramètres ont été modifiés et de nouvelles phases de test ont été menées afin de vérifier si les modèles étaient capables de généraliser suffisamment pour reconnaître les matériaux de construction:

- Avec et sans caisson isolé. Cela permet de comprendre l'influence des bruits sur la classification des matériaux.
- La force et/ou la direction du marteau.

En l'absence d'effet des deux paramètres ci-dessus sur les performances du modèle, il pourrait être possible d'abandonner le caisson. Cela voudrait dire que nous pouvons tous exploiter le modèle à n'importe quel moment et à n'importe quel endroit. L'instrument serait donc facilement portable.

Les coups ont été enregistrés sur l'application native d'un Samsung S21 Ultra, modèle SM-G998B/DS. Ils ont été capturés au format wav. Il y a des éléments non négligeables pour entraîner une intelligence artificielle. Les données doivent toutes être d'une durée identique, d'un format et des caractéristiques standardisées. Pour le son, il faut que le fichier soit en format wav. Dans l'idéal, le son est enregistré en format wav. Mais dans mon cas, j'ai été contrainte de convertir les fichiers post-enregistrement.

J'ai ensuite eu recours à différents programmes pour :

- Préparer et nettoyer les données. L'intérêt est d'uniformiser les signaux sonores. Il est également important de trouver un mode de représentation que les modèles sont en mesure de comprendre. Il y a aussi des manières de représenter pour optimiser la classification. Pour ma part, j'ai choisi le type de données le plus communément appliqué. La fréquence d'échantillonnage est ainsi de 1600 Hz et la durée est de 1 seconde. C'est le mel-spectrogramme qui sera analysé par l'algorithme d'apprentissage profond.
- Diviser les sons dans un dataset de deux ensembles: un ensemble d'entraînement et un ensemble de test. L'algorithme vérifie ses prédictions pour les sets d'entraînement pour se réajuster. Le dataset de test permet de définir l'exactitude du modèle en vue de sa mise à jour.
- Entraîner et tester les réseaux de neurones.
- Analyser, interpréter et représenter les résultats. Une courbe d'évolution de l'exactitude des modèles donne à voir la variation des performances. Il arrive que l'intelligence artificielle sur-apprenne et que ses performances diminuent.
- Faire plusieurs ajustements / itérations de l'expérience !

Le résultat attendu correspond à un mot : le nom du matériau reconnu par le son d'impact du mur.

PARTIE 3

EXPÉRIMENTATION

LES MURS, INSTRUMENTS POUR LA RECONNAISSANCE, UNE MÉTHODE ITÉRATIVE

L'expérimentation se segmente en plusieurs étapes. Les projets de programmation existants m'ont mise sur la voie d'une classification d'instruments de musique. À travers une approche itérative, plusieurs tentatives m'ont conduite à concentrer mes efforts sur l'entraînement de trois modèles aux architectures différentes. Pour ce faire, la construction d'une boîte isolée acoustiquement et de jeux de données pour les phases d'entraînement et de test ont été mises au point.

1 Le choix de l'algorithme d'apprentissage profond

Il existe des modèles d'intelligence artificielle à profusion. Étant une néophyte en informatique, j'ai sur-le-champ consulté des tutoriels ou des extraits de codes pour mieux appréhender l'entraînement d'un réseau de neurones avant de me lancer dans l'écriture d'un algorithme.

J'ai d'abord fait des recherches sur Whisper, une technologie de reconnaissance automatique de la parole (ASR) mise au point par OpenAI. Spécialisé dans la reconnaissance et transcription vocale, le code peut être édité afin que le modèle identifie le nombre de personnes d'un dialogue et associe une voix à un nom. Bien que Whisper soit open source, il est nécessaire d'avoir une expertise pour pouvoir le moduler selon les besoins de la recherche. Le modèle ne s'accordait pas à mes objectifs. Un temps supplémentaire pour fine-tuner (peaufiner) le modèle aurait diminué les chances de réussite de l'expérience.

Faute de connaître des personnes dans le milieu, la plateforme Youtube m'a été d'une précieuse aide. J'ai donc scruté plusieurs vidéos et sites web. John Nordby présente un exemple de classification audio avec le jeu de données « Urbansound8k » dans « Audio Classification with Machine Learning (EuroPython 2019) ».[20] Jason Brownlee nous enseigne le Machine Learning grâce à son site internet[21]. Il rend le codage accessible à un public beaucoup plus large que celui des spécialistes en intelligence artificielle. Enfin, un cours sur les transformers, une architecture d'apprentissage profond, est donné par Sanchit Gandhi, Matthijs Hollemans, Maria Khalusova et Vaibhav Srivastav sur la plateforme Hugging Face[22].

J'ai finalement décidé de reprendre un code réalisé en 2020 par Seth Adams, un ingénieur spécialisé dans le Machine Learning. Une série de 3 vidéos et un pipeline pour le prototypage d'algorithmes de classification avec

20 Nordby, Jon. «Audio Classification with Machine Learning» EuroPython, Bâle, 2019. Disponible sur : https://www.youtube.com/watch?v=uCGROOUO_wY.

21 Brownlee, Jason. « Machine Learning Mastery ». Disponible sur : https://www.machinelearningmastery.com/.

22 Gandhi et al. « Introduction aux données audio - Hugging Face Audio Course », consulté le 12 mars 2024. Disponible sur : https://huggingface.co/learn/audio-course/fr/chapter1/audio_data.

TensorFlow 2.3 ont été mis à disposition de tous[23]. Seth Adams, à travers ses explications, montre comment fonctionne la classification audio, quelles sont les variables, les éléments structurant l'apprentissage profond pour le son et plus encore. Il nous proposait initialement de classifier des sons de dix instruments de musique avec un Conv1D, un Conv2D et un LSTM. La justesse des prédictions tournait aux alentours de 80 à 90 %.

Son projet est stocké sur la plateforme Github, une plateforme collaborative de stockage de code disponible en ligne pour les communautés open-source et les développeurs. Différentes versions du code sont sauvegardées pour assurer un suivi de toutes les collaborations et des changements. La dernière version de son projet se décline en plusieurs dossiers (figure 21).

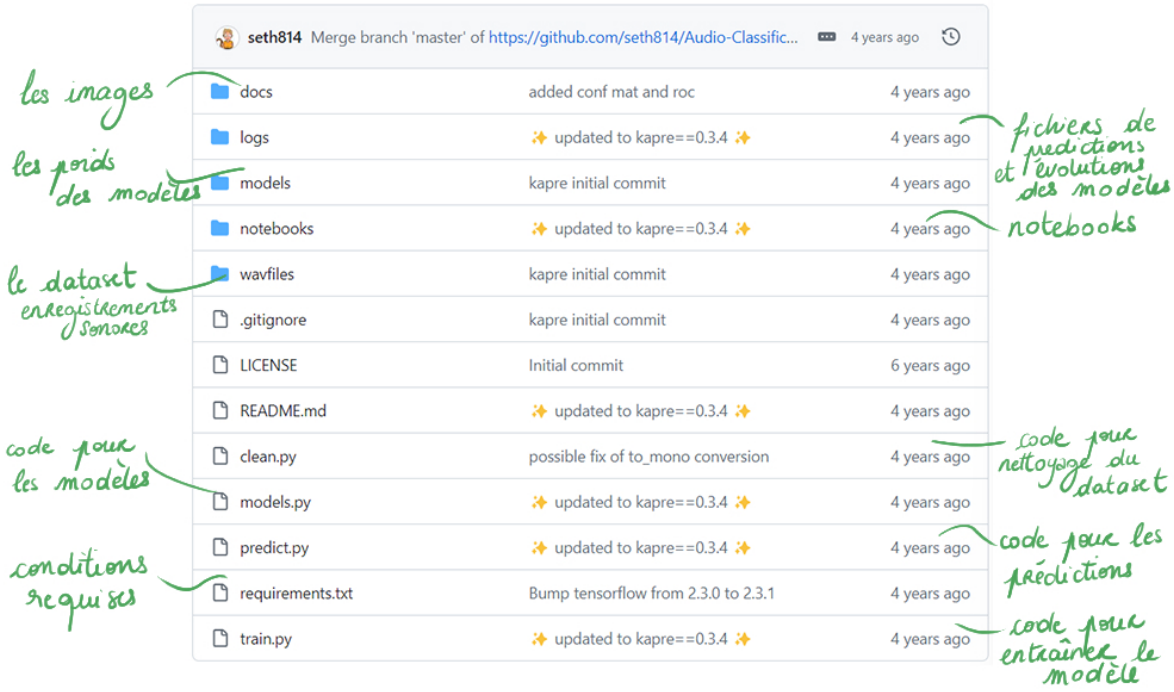


Figure 21 : Répertoire en ligne sur Github

Tous ces éléments peuvent être scindés en plusieurs catégories :

- « License », « Readme.md », « Requirements.txt » sont des fichiers que l'on retrouve souvent dans les dossiers de logiciels. Toutes les conditions nécessaires sont énumérées dans « Requirements.txt ». Il est donc essentiel d'en prendre connaissance au risque d'aboutir à un processus entraînant des erreurs. « Readme », comme son nom l'indique est à lire

23 Adams, Seth. « Deep learning for audio classification », YouTube. Avril 2020. consulté le 27 octobre 2024, http://www.youtube.com/playlist?list=PL7Zd_5jZT2cerrNmV3HR3-RHocb6fWe4z.

pour préparer le terrain.

- Le répertoire « wavfiles » contient tous les enregistrements sonores bruts.
- Des éléments tels que « clean.py », « models.py », « train.py » et « predict.py », sont des programmes en langage python qui respectivement nettoient les données, spécifient les caractéristiques des modèles, entraînent et testent les algorithmes de « models » et accomplissent une prédiction.
- « models » abrite la mise à jour des algorithmes d'apprentissage profond, plus particulièrement, les poids des modèles.
- « logs » est un dossier destiné au suivi des performances du modèle, qui sert principalement à fournir des éléments d'évaluation. Les fichiers créés suite à l'entraînement et au test des réseaux de neurones y sont donc stockés.
- « notebooks » recèle des pages web qui sont en réalité des bloc-notes de programmes à exécuter dans un environnement. Un environnement est un ensemble de matériels et de logiciels informatiques nécessaires à l'exécution du code. On peut y trouver des programmes pour évaluer l'évolution des performances des modèles, pour construire une matrice de confusion et pour visualiser les fichiers sonores sous différentes formes.

Les fichiers provenant des programmes issus de « notebooks » sont alors répertoriés dans le dossier « docs ».

Python est un des langages de programmation les plus populaires utilisés pour le codage. Il est principalement choisi pour sa polyvalence et sa flexibilité.

Des collections de code pré-écrit pour la programmation sont créées par des développeurs pour optimiser le flux de travail. Ces **bibliothèques** (ou libraries en anglais) comportent des fonctions ou encore des modules qui effectuent des tâches, pour faire en sorte de ne pas partir de zéro.

TensorFlow est un framework. Il constitue un ensemble de composants pour créer la structure d'un programme. Il est composé de plusieurs bibliothèques open-source pour le machine learning développées par Google. De nombreux outils sont proposés pour faciliter la création, l'entraînement et le déploiement des modèles en langage python.

2 Une classification d'instruments de musique

Une vérification du code s'impose d'abord. Pour cela, je télécharge l'intégralité du dossier d'origine. Les algorithmes sont façonnés pour la classification d'instruments de musique. Le répertoire « wavfiles » contient dix dossiers. Ce sont les dix classes. Les 30 fichiers wav de chaque dossier durent entre 3 et 26 secondes. Pour modifier et exécuter le code à ma guise, je choisis Google Colab.

Google Colab est une plateforme, lancée en 2017, disponible sur internet permettant d'écrire et d'exécuter du code Python dans un environnement réseau. Elle permet d'exploiter des capacités de calcul et de la mémoire d'ordinateurs et de serveurs par réseau. Elle permet aussi de stocker le code dans un fichier Jupyter notebook (ipynb) et de le partager. Un fichier ipynb est l'équivalent d'un journal, d'un bloc-note dans lequel sont notées des lignes de codes et des lignes de textes, souvent didactiques, organisées par cellule. Cela en fait une plateforme interactive facile à prendre en main.

Le nom du fichier Jupyter Notebook est tiré du nom d'une plateforme semblable à Google Colab créée en 2014.

Dans cette partie, la structure de l'algorithme et son fonctionnement seront détaillés. Je débute par l'installation des bibliothèques et l'importation des modules. Je poursuis avec le nettoyage des données. Je définis les paramètres des modèles. J'effectue une session d'entraînement. Enfin, j'analyse les résultats.

a L'importation des modules

J'ouvre en priorité la page Google Colab. Je commence par installer et importer les modules requis pour la programmation à partir de bibliothèques. La plateforme est liée à la plateforme de stockage et de partage de fichiers de Google. Je peux ainsi importer toutes les données et scripts Python lors des prochaines opérations. La première chose à faire est de vérifier que les versions de bibliothèques préinstallées pour former l'environnement sont compatibles avec le code des algorithmes. Je regarde le fichier « requirement.txt ». Les versions des bibliothèques installées dans l'environnement d'exécution ne correspondent pas toutes aux versions requises. Il faut notamment changer la version de Librosa. Après quelques

tentatives, un énième essai est couronné d'un succès.

Quelques couleurs accompagnent le texte. Ce qui est en rose décrit une fonction, donnée par un ensemble d'instructions pour réaliser une ou plusieurs tâches bien précises. Sur la figure 22, « from google.colab import drive » signifie « Depuis la bibliothèque google colab, importer le module drive ». Sans cette ligne, la fonction « mount.drive » n'aurait pas été possible. Le texte en rouge correspond à une chaîne de caractères. Il peut s'agir de langage parlé, de lien URL, ou simplement de symbole. Dans mon cas, je cherche à connecter Google Drive à Google Colab. Un espace de stockage supplémentaire est ajouté à l'environnement. Mais par-dessus tout, j'accède enfin à tous les fichiers nécessaires à la réalisation de l'expérience.

IPython sert d'interface pour exécuter et interagir avec le code Python. Il est destiné à créer un environnement pour développer le modèle de Machine Learning.

Les bibliothèques et modules :

Librosa est une bibliothèque pour l'analyse audio. Elle permet notamment de charger des fichiers audios, de générer des spectrogrammes et de visualiser la donnée.

Kapre est un préprocesseur audio qui fonctionne avec Keras. Il permet de calculer des transformées de Fourier à court terme, le spectrogramme Mel par exemple.

Keras est un framework. Tout comme Tensorflow, il participe à la construction et l'entraînement des algorithmes de Deep Learning. Son utilisation est plus simple et plus intuitive, mais moins flexible et moins appropriée aux tâches complexes. Elle s'intègre par ailleurs à Tensorflow grâce au module tf.keras.

NumPy est une bibliothèque qui sert à créer et transformer des matrices multidimensionnelles. Wavio est un module en lien avec NumPy. Il permet de lire des fichiers WAV et d'écrire une matrice numpy en fichier WAV.

SciPy est une bibliothèque qui dépend également de NumPy. Elle fournit divers algorithmes mathématiques pour manipuler et visualiser les données.

```
from google.colab import drive
drive.mount('/content/drive')

from IPython.display import Audio
from scipy.io import wavfile
!pip install wavio
import wavio
!pip install kapre
import kapre
!pip install git+https://github.com/librosa/librosa
```

```
Collecting wavio
  Downloading wavio-0.0.8-py3-none-any.whl (9.4 kB)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.10/d
Installing collected packages: wavio
Successfully installed wavio-0.0.8
Collecting kapre
```

Figure 22 : Importation des modules

b Le nettoyage des données

L'importation des modules est suivie de l'étape du nettoyage des données grâce au programme « clean.py ». Les enregistrements sonores, parfois longs, sont divisés en plusieurs séquences d'une seconde. Les données courtes sont plus faciles à manipuler par le réseau de neurones et demandent moins de ressources que des séquences longues. Cela permet aussi d'avoir une donnée la plus pure possible pour que le futur entraînement soit axé sur l'analyse des fréquences plutôt que sur l'aspect global du morceau d'origine. Pour que le processus se déroule correctement, il faut penser à renseigner les nouvelles adresses dans l'espace dédié aux données fournies aux fonctions, que l'on dit arguments.

La figure 23 fait ressortir le fonctionnement du programme, en un tour d'horizon des détails du code python. Les opérations sont nombreuses. Je décrirai donc seulement les grandes lignes pour expliquer la démarche de l'expérience.

La structure du code est en trois parties. Chaque fichier python doit pouvoir regrouper une ou plusieurs fonctions pour la réalisation d'une grande tâche en fonctionnant indépendamment des autres fichiers python.

1 Importation des modules

```
clean.py
import matplotlib.pyplot as plt
from scipy.io import wavfile
import argparse
import os
from glob import glob
import numpy as np
import pandas as pd
from librosa.core import resample, to_mono
from tqdm import tqdm
import wavio
```

2 Définition des fonctions

a Enveloppe sonore

```
def envelope(y, rate, threshold):
    mask = []
    y = pd.Series(y).apply(np.abs)
    y_mean = y.rolling(window=int(rate/20),
                       min_periods=1,
                       center=True).max()
    for mean in y_mean:
        if mean > threshold:
            mask.append(True)
        else:
            mask.append(False)
    return mask, y_mean
```

b Sous échantillonnage

```
def downsample_mono(path, sr):
    obj = wavio.read(path)
    wav = obj.data.astype(np.float32, order='F')
    rate = obj.rate
    try:
        channel = wav.shape[1]
        if channel == 2:
            wav = to_mono(wav.T)
        elif channel == 1:
            wav = to_mono(wav.reshape(-1))
    except IndexError:
        wav = to_mono(wav.reshape(-1))
        pass
    except Exception as exc:
        raise exc
    wav = resample(wav, orig_sr=rate, target_sr=sr)
    wav = wav.astype(np.int16)
    return sr, wav
```

c Sauvegarde fichiers

```
def save_sample(sample, rate, target_dir, fn, ix):
    fn = fn.split('.wav')[0]
    dst_path = os.path.join(target_dir.split('.')[0], fn+'_{}.wav'.format(str(ix)))
    if os.path.exists(dst_path):
        return
    wavfile.write(dst_path, rate, sample)
```

d Création répertoire

```
def check_dir(path):
    if os.path.exists(path) is False:
        os.mkdir(path)
```

e Découpage des fichiers audio

```
def split_wavs(args):
    src_root = args.src_root
    dst_root = args.dst_root
    dt = args.delta_time

    wav_paths = glob('{}/**'.format(src_root), recursive=True)
    wav_paths = [x for x in wav_paths if '.wav' in x]
    dirs = os.listdir(src_root)
    check_dir(dst_root)
    classes = os.listdir(src_root)
    for _cls in classes:
        target_dir = os.path.join(dst_root, _cls)
        check_dir(target_dir)
        src_dir = os.path.join(src_root, _cls)
        for fn in tqdm(os.listdir(src_dir)):
            src_fn = os.path.join(src_dir, fn)
            rate, wav = downsample_mono(src_fn, args.sr)
            mask, y_mean = envelope(wav, rate, threshold=args.threshold)
            wav = wav[mask]
            delta_sample = int(dt*rate)

            # cleaned audio is less than a single sample
            # pad with zeros to delta_sample size
            if wav.shape[0] < delta_sample:
                sample = np.zeros(shape=(delta_sample,), dtype=np.int16)
                sample[:wav.shape[0]] = wav
                save_sample(sample, rate, target_dir, fn, 0)
            # step through audio and save every delta_sample
            # discard the ending audio if it is too short
            else:
                trunc = wav.shape[0] % delta_sample
                for cnt, i in enumerate(np.arange(0, wav.shape[0]-trunc, delta_sample)):
                    start = int(i)
                    stop = int(i + delta_sample)
                    sample = wav[start:stop]
                    save_sample(sample, rate, target_dir, fn, cnt)
```

f Graphique enveloppe sonore

```
def test_threshold(args):
    src_root = args.src_root
    wav_paths = glob('{}/**'.format(src_root), recursive=True)
    wav_path = [x for x in wav_paths if args.fn in x]
    if len(wav_path) != 1:
        print('audio file not found for sub-string: {}'.format(args.fn))
        return
    rate, wav = downsample_mono(wav_path[0], args.sr)
    mask, env = envelope(wav, rate, threshold=args.threshold)
    plt.style.use('ggplot')
    plt.title('Signal Envelope, Threshold = {}'.format(str(args.threshold)))
    plt.plot(wav[np.logical_not(mask)], color='r', label='remove')
    plt.plot(wav[mask], color='c', label='keep')
    plt.plot(env, color='m', label='envelope')
    plt.grid(False)
    plt.legend(loc='best')
    plt.show()
```

3 Ajout et analyse des arguments

```
if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Cleaning audio data')
    parser.add_argument('--src_root', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/wavfiles',
                        help='directory of audio files in total duration')
    parser.add_argument('--dst_root', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/clean',
                        help='directory to put audio files split by delta_time')
    parser.add_argument('--delta_time', '-dt', type=float, default=1.0,
                        help='time in seconds to sample audio')
    parser.add_argument('--sr', type=int, default=16000,
                        help='rate to downsample audio')

    parser.add_argument('--fn', type=str, default='3a3d0279',
                        help='file to plot over time to check magnitude')
    parser.add_argument('--threshold', type=str, default=20,
                        help='threshold magnitude for np.int16 dtype')
    args, _ = parser.parse_known_args()

    #test_threshold(args)
    split_wavs(args)
```

Figure 23 : Code du fichier clean.py

C'est pourquoi la première partie d'un fichier python consiste toujours à **importer** des modules, comme vu précédemment. La deuxième partie est vouée à la **définition des fonctions.** Dans la troisième partie, j'indique les paramètres et la localisation des données. Plus exactement, je **définis les arguments**. Ainsi, le répertoire wavfiles est renseigné. J'écris également l'endroit où je souhaite enregistrer les nouvelles données.

La deuxième partie est la plus longue. Certaines fonctions sont prédéfinies, natives ou réalisées par des tiers sous forme de fichier ou module. Pour ma part, il faut confectionner un code sur mesure. Je définis des fonctions grâce à la fonction def. Chaque bloc de code d'une série d'instructions est alors nommé. Il suffit de réemployer le nom du bloc pour pouvoir utiliser l'intégralité des instructions qu'il contient. De cette manière, je n'ai plus à réécrire les mêmes lignes de code à plusieurs reprises. Je gagne alors en temps comme en organisation, en m'évitant de me perdre dans un document trop long et illisible.

Dans la première sous-partie **(a),** je définis la courbe qui décrit l'évolution de l'amplitude du son au cours du temps. C'est **l'enveloppe sonore**. Le défi pour les programmes, dans une séquence sonore, est de se concentrer sur les éléments importants pour la classification. Or, le son s'évanouit. Le silence apparaît. Dans un fichier, cet espace mort se traduit par de très faibles informations qui n'ont aucun lien avec le signal sonore émis. Le modèle est susceptible d'oublier ou de mémoriser des données qui faussent les résultats. C'est pour cette raison que je cherche à effacer ces données en les remplaçant par des valeurs nulles. Cette section est donc la première étape menant à la suppression du silence.

Je sous-échantillonne ensuite les audios **(b)**. La piste stéréo de taux d'échantillonnage de 44 100 Hz se transforme en piste mono de taux d'échantillonnage de 16 000 Hz.

Les sous-parties **(c)** et **(d)** servent à enregistrer et créer un répertoire stockant les fichiers nettoyés, respectivement .

La fonction **(e)** regroupe toutes les fonctions définies précédemment. D'abord, je récupère les fichiers sources, situés dans le répertoire « wavfiles ». Je fais appel à la fonction **(d)**. Puis, je sous-échantillonne **(b)** et je supprime le silence en me référant à l'enveloppe sonore **(a)** . En outre, je découpe la séquence en plusieurs séquences d'une seconde. Enfin, j'enregistre les nouvelles données **(c)** dans un nouvel endroit intitulé « clean ».

1 Importation des modules

```
from tensorflow.keras import layers
from tensorflow.keras.layers import TimeDistributed, LayerNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
import kapre
from kapre.composed import get_melspectrogram_layer
import tensorflow as tf
import os
```

2 Définition des fonctions

a Définition Conv1D

```
def Conv1D(N_CLASSES=10, SR=16000, DT=1.0):
    input_shape = (int(SR*DT), 1)
    i = get_melspectrogram_layer(input_shape=input_shape,
                                 n_mels=128,
                                 pad_end=True,
                                 n_fft=512,
                                 win_length=400,
                                 hop_length=160,
                                 sample_rate=SR,
                                 return_decibel=True,
                                 input_data_format='channels_last',
                                 output_data_format='channels_last')
    x = LayerNormalization(axis=2, name='batch_norm')(i.output)
    x = TimeDistributed(layers.Conv1D(8, kernel_size=(4), activation='tanh'), name='td_conv_1d_tanh')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), name='max_pool_2d_1')(x)
    x = TimeDistributed(layers.Conv1D(16, kernel_size=(4), activation='relu'), name='td_conv_1d_relu_1')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), name='max_pool_2d_2')(x)
    x = TimeDistributed(layers.Conv1D(32, kernel_size=(4), activation='relu'), name='td_conv_1d_relu_2')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), name='max_pool_2d_3')(x)
    x = TimeDistributed(layers.Conv1D(64, kernel_size=(4), activation='relu'), name='td_conv_1d_relu_3')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), name='max_pool_2d_4')(x)
    x = TimeDistributed(layers.Conv1D(128, kernel_size=(4), activation='relu'), name='td_conv_1d_relu_4')(x)
    x = layers.GlobalMaxPooling2D(name='global_max_pooling_2d')(x)
    x = layers.Dropout(rate=0.1, name='dropout')(x)
    x = layers.Dense(64, activation='relu', activity_regularizer=l2(0.001), name='dense')(x)
    o = layers.Dense(N_CLASSES, activation='softmax', name='softmax')(x)
    model = Model(inputs=i.input, outputs=o, name='1d_convolution')
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

b Définition Conv2D

```
def Conv2D(N_CLASSES=10, SR=16000, DT=1.0):
    input_shape = (int(SR*DT), 1)
    i = get_melspectrogram_layer(input_shape=input_shape,
                                 n_mels=128,
                                 pad_end=True,
                                 n_fft=512,
                                 win_length=400,
                                 hop_length=160,
                                 sample_rate=SR,
                                 return_decibel=True,
                                 input_data_format='channels_last',
                                 output_data_format='channels_last')
    x = LayerNormalization(axis=2, name='batch_norm')(i.output)
    x = layers.Conv2D(8, kernel_size=(7,7), activation='tanh', padding='same', name='conv2d_tanh')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), padding='same', name='max_pool_2d_1')(x)
    x = layers.Conv2D(16, kernel_size=(5,5), activation='relu', padding='same', name='conv2d_relu_1')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), padding='same', name='max_pool_2d_2')(x)
    x = layers.Conv2D(16, kernel_size=(3,3), activation='relu', padding='same', name='conv2d_relu_2')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), padding='same', name='max_pool_2d_3')(x)
    x = layers.Conv2D(32, kernel_size=(3,3), activation='relu', padding='same', name='conv2d_relu_3')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), padding='same', name='max_pool_2d_4')(x)
    x = layers.Conv2D(32, kernel_size=(3,3), activation='relu', padding='same', name='conv2d_relu_4')(x)
    x = layers.Flatten(name='flatten')(x)
    x = layers.Dropout(rate=0.2, name='dropout')(x)
    x = layers.Dense(64, activation='relu', activity_regularizer=l2(0.001), name='dense')(x)
    o = layers.Dense(N_CLASSES, activation='softmax', name='softmax')(x)
    model = Model(inputs=i.input, outputs=o, name='2d_convolution')
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

c Définition LSTM

```
def LSTM(N_CLASSES=10, SR=16000, DT=1.0):
    input_shape = (int(SR*DT), 1)
    i = get_melspectrogram_layer(input_shape=input_shape,
                                 n_mels=128,
                                 pad_end=True,
                                 n_fft=512,
                                 win_length=400,
                                 hop_length=160,
                                 sample_rate=SR,
                                 return_decibel=True,
                                 input_data_format='channels_last',
                                 output_data_format='channels_last',
                                 name='2d_convolution')
    x = LayerNormalization(axis=2, name='batch_norm')(i.output)
    x = TimeDistributed(layers.Reshape((-1,)), name='reshape')(x)
    s = TimeDistributed(layers.Dense(64, activation='tanh'),
                        name='td_dense_tanh')(x)
    x = layers.Bidirectional(layers.LSTM(32, return_sequences=True),
                             name='bidirectional_lstm')(s)
    x = layers.concatenate([s, x], axis=2, name='skip_connection')
    x = layers.Dense(64, activation='relu', name='dense_1_relu')(x)
    x = layers.MaxPooling1D(name='max_pool_1d')(x)
    x = layers.Dense(32, activation='relu', name='dense_2_relu')(x)
    x = layers.Flatten(name='flatten')(x)
    x = layers.Dropout(rate=0.2, name='dropout')(x)
    x = layers.Dense(32, activation='relu',
                     activity_regularizer=l2(0.001),
                     name='dense_3_relu')(x)
    o = layers.Dense(N_CLASSES, activation='softmax', name='softmax')(x)
    model = Model(inputs=i.input, outputs=o, name='long_short_term_memory')
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Figure 24 : Code du fichier model.py

La dernière section **(f)** sert à visualiser l'opération (figure 25).

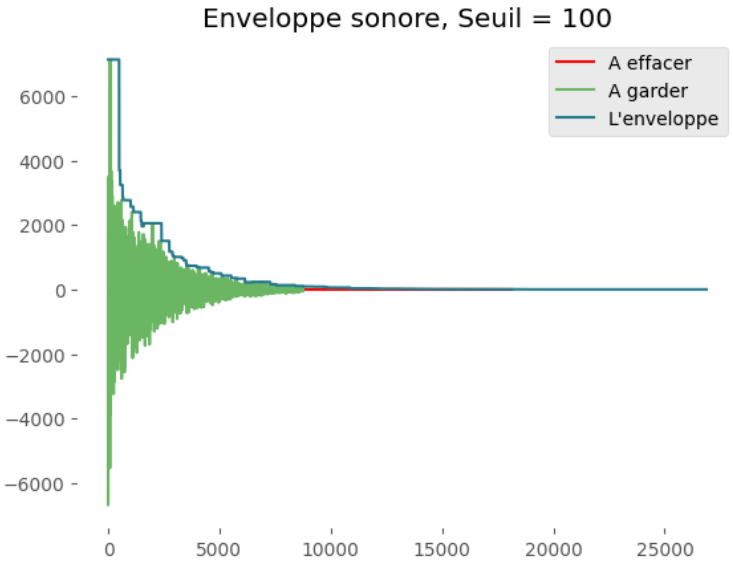


Figure 25 : Enveloppe sonore

Tous les éléments ne signifient rien sans l'unique ligne à la toute fin. En effet, la dernière ligne « split_wavs » est le code qui enclenche tout le processus. Maintenant que la première étape est réalisée, je peux passer à l'entraînement du modèle. Mais avant, j'analyse les modèles confectionnés par Seth Adams.

c La définition des fonctions pour le Deep Learning

Ce code pour la définition des architectures en réseaux de neurones suit la même structure que celui du nettoyage des données, en dehors du fait que je n'ai pas besoin d'arguments pour cette tâche-ci. Ne demeurent donc que l'importation des bibliothèques et modules ainsi que la définition des trois modèles : un Conv1D, un Conv2D et un LSTM (figure 24).

J'ai déjà expliqué les spécificités de chaque modèle dans une partie précédente. Cependant, il y a certains détails, communs aux trois fonctions, qui n'ont pas été évoqués.

Les spectrogrammes sont calculés en première couche des réseaux. Keun Woo Choi, le créateur de la bibliothèque Kapre, a montré que le temps de computation des spectrogrammes était négligeable pour un CNN. Cette technique dispense, à chaque introduction de nouveaux sons, de passer par une étape de prétraitement pour obtenir des spectrogrammes et de devoir à nouveau tout stocker sur le disque dur. Utiliser Kapre facilite les itérations.

1 Importation des modules

a
```
import tensorflow as tf
from tensorflow.keras.callbacks import CSVLogger, ModelCheckpoint
from tensorflow.keras.utils import to_categorical
import os
from scipy.io import wavfile
import pandas as pd
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from models import Conv1D, Conv2D, LSTM
from tqdm import tqdm
from glob import glob
import argparse
import warnings
```

2 Génération des batches par epoch

Initialisation

b
```
class DataGenerator(tf.keras.utils.Sequence):
    def __init__(self, wav_paths, labels, sr, dt, n_classes,
                 batch_size=32, shuffle=True):
        self.wav_paths = wav_paths
        self.labels = labels
        self.sr = sr
        self.dt = dt
        self.n_classes = n_classes
        self.batch_size = batch_size
        self.shuffle = True
        self.on_epoch_end()
```

Nombre batch

c
```
    def __len__(self):
        return int(np.floor(len(self.wav_paths) / self.batch_size))
```

Génération batch

d
```
    def __getitem__(self, index):
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        wav_paths = [self.wav_paths[k] for k in indexes]
        labels = [self.labels[k] for k in indexes]

        # generate a batch of time data
        X = np.empty((self.batch_size, int(self.sr*self.dt), 1), dtype=np.float32)
        Y = np.empty((self.batch_size, self.n_classes), dtype=np.float32)

        for i, (path, label) in enumerate(zip(wav_paths, labels)):
            rate, wav = wavfile.read(path)
            X[i,] = wav.reshape(-1, 1)
            Y[i,] = to_categorical(label, num_classes=self.n_classes)

        return X, Y
```

Mélange

e
```
    def on_epoch_end(self):
        self.indexes = np.arange(len(self.wav_paths))
        if self.shuffle:
            np.random.shuffle(self.indexes)
```

3 Entrainement

Variables

f
```
def train(args):
    src_root = args.src_root
    sr = args.sample_rate
    dt = args.delta_time
    batch_size = args.batch_size
    model_type = args.model_type
    params = {'N_CLASSES':len(os.listdir(args.src_root)),
              'SR':sr,
              'DT':dt}
    models = {'conv1d':Conv1D(**params),
              'conv2d':Conv2D(**params),
              'lstm':  LSTM(**params)}
    assert model_type in models.keys(), '{} not an available model'.format(model_type)
    csv_path = os.path.join('/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/logs', '{}_history.csv'.format(model_t
```

Division des données pour former les datasets

g
```
    wav_paths = glob('{}/**'.format(src_root), recursive=True)
    wav_paths = [x.replace(os.sep, '/') for x in wav_paths if '.wav' in x]
    classes = sorted(os.listdir(args.src_root))
    le = LabelEncoder()
    le.fit(classes)
    labels = [os.path.split(x)[0].split('/')[-1] for x in wav_paths]
    labels = le.transform(labels)
    wav_train, wav_val, label_train, label_val = train_test_split(wav_paths,
                                                                  labels,
                                                                  test_size=0.1,
                                                                  random_state=0)
```

Génération des datasets et sauvegarde des modèles

h
```
    assert len(label_train) >= args.batch_size, 'Number of train samples must be >= batch_size'
    if len(set(label_train)) != params['N_CLASSES']:
        warnings.warn('Found {}/{} classes in training data. Increase data size or change random_state.'.format(len(set(label_
    if len(set(label_val)) != params['N_CLASSES']:
        warnings.warn('Found {}/{} classes in validation data. Increase data size or change random_state.'.format(len(set(labe

    tg = DataGenerator(wav_train, label_train, sr, dt,
                       params['N_CLASSES'], batch_size=batch_size)
    vg = DataGenerator(wav_val, label_val, sr, dt,
                       params['N_CLASSES'], batch_size=batch_size)
    model = models[model_type]
    cp = ModelCheckpoint('/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/models/{}.h5'.format(model_type), monitor
                         save_best_only=True, save_weights_only=False,
                         mode='auto', save_freq='epoch', verbose=1)
    csv_logger = CSVLogger(csv_path, append=False)
    model.fit(tg, validation_data=vg,
              epochs=30, verbose=1,
              callbacks=[csv_logger, cp])
```

4 Ajout et analyse des arguments

```
if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Audio Classification Training')
    parser.add_argument('--model_type', type=str, default='lstm',
                        help='model to run. i.e. conv1d, conv2d, lstm')
    parser.add_argument('--src_root', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/clean',
                        help='directory of audio files in total duration')
    parser.add_argument('--batch_size', type=int, default=16,
                        help='batch size')
    parser.add_argument('--delta_time', '-dt', type=float, default=1.0,
                        help='time in seconds to sample audio')
    parser.add_argument('--sample_rate', '-sr', type=int, default=16000,
                        help='sample rate of clean audio')
    args, _ = parser.parse_known_args()

    train(args)
```

Figure 26 : Code du fichier train.py

Les matrices (l**es spectrogrammes mel de décibels**) produites sont de **128 x 100** valeurs. Les données de petite taille réduisent le temps de traitement ainsi que les ressources nécessaires. Cela empêche aussi que le modèle ne s'égare en se concentrant sur des éléments futiles.

Il faut aussi n**ormaliser les données pour que le modèle apprenne mieux**. Le modèle transforme les données pour qu'elles couvrent une plage de valeurs similaires. Les données sont mises à une échelle semblable.

Avant les ultimes couches de neurones intégralement connectés se trouve la couche **dropout**. En s'entraînant, les modèles tendent à l'overfitting. Ils « sur-apprennent » en se moulant trop fidèlement aux données d'entraînement. Ils perdent en exactitude lorsqu'on leur soumet des données inconnues. Ils ne peuvent se généraliser à ces nouvelles données.

Le dropout revient à désactiver des neurones aléatoirement à chaque itération d'entraînement. Cette forme de régularisation assure une indépendance entre les neurones. Cela rend le modèle plus robuste.

Nous avons à présent saisi le fonctionnement des modèles. Il est temps de passer à l'entraînement.

d L'entraînement des réseaux de neurones

L'entraînement comprend la création des jeux de données d'entraînement et de test ainsi que la mise à jour des paramètres du modèle. Tout comme pour les codes précédents, j'importe les modules. Je cherche à **générer les jeux de données** (ou datasets), à **lancer l'entraînement** et enfin à **sauvegarder les poids et les biais**.

La deuxième partie concerne la génération des batches (taille des lots) des différents jeux de données. Séparer les datasets en plusieurs lots, c'est utiliser moins de ressources du ou des processeurs, les cerveaux de l'ordinateur, à un instant t.

L'entraînement a lieu sur le **dataset d'entraînement.** Une fois qu'un lot est traité, le modèle est ajusté. Une fois que tous les lots sont passés en revue, un epoch (époque) est réalisé. À la fin de chaque **epoch** un test est effectué sur un autre jeu de données, le **dataset de test**. Une évaluation de l'ajustement du modèle sur ce nouveau jeu permet de repérer un éventuel overfitting.

Le jeu de données pour l'entraînement constitue 90% des données, tandis que celui pour le test est de 10%. Les hyperparamètres doivent également être pris en compte et modifié en fonction de la quantité de données d'entrée. Dans ce cas :

- Le nombre d'échantillons : 1152 (Nombre de fichiers variant entre 37 et 210)
- La taille du batch: 16
- Le nombre d'epochs : 30
- Nombre de batches : 1152/16 = 72

Dans la zone des arguments, je peux choisir entre les trois algorithmes: le LSTM, le Conv1D et le Conv2D. Sont également vérifiées la longueur et la largeur des données d'entrée.

Dès que toutes les variables et fonctions sont définies, l'entraînement peut se lancer grâce à la dernière ligne de code « train ».

Je peux voir en direct l'avancement de l'entraînement. Sur la figure 27, la perte et l'exactitude pour chaque entraînement et test sont notées. Les chiffres sont précieusement conservés dans des fichiers au format .csv du dossier « logs ». Ils me permettent d'examiner les résultats pour moi, mais surtout pour le modèle, qui est réajusté grâce à la descente de gradient. Un autre fichier en format .h5 est créé dans le dossier « models ». Ce dernier recueille les valeurs des paramètres du ou des modèles entraînés.

```
✓ 3 min [7]  !python /content/drive/MyDrive/MEMOIRE/Classification_essais/train.py

2024-04-23 22:07:58.787030: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to r
2024-04-23 22:07:58.787108: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to re
2024-04-23 22:07:58.788712: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to
2024-04-23 22:07:58.797510: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU ins
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-04-23 22:08:00.385204: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
Epoch 1/30
20/20 [==============================] - ETA: 0s - loss: 2.1009 - accuracy: 0.2625
Epoch 1: val_loss improved from inf to 1.69765, saving model to /content/drive/MyDrive/MEMOIRE/Classification_essais/models/lstm.h5
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.
  saving_api.save_model(
20/20 [==============================] - 91s 4s/step - loss: 2.1009 - accuracy: 0.2625 - val_loss: 1.6977 - val_accuracy: 0.4688
Epoch 2/30
20/20 [==============================] - ETA: 0s - loss: 1.2300 - accuracy: 0.6625
Epoch 2: val_loss improved from 1.69765 to 1.10093, saving model to /content/drive/MyDrive/MEMOIRE/Classification_essais/models/lstm.h5
20/20 [==============================] - 7s 341ms/step - loss: 1.2300 - accuracy: 0.6625 - val_loss: 1.1009 - val_accuracy: 0.7500
Epoch 3/30
20/20 [==============================] - ETA: 0s - loss: 0.7738 - accuracy: 0.8281
Epoch 3: val_loss improved from 1.10093 to 0.87569, saving model to /content/drive/MyDrive/MEMOIRE/Classification_essais/models/lstm.h5
20/20 [==============================] - 3s 157ms/step - loss: 0.7738 - accuracy: 0.8281 - val_loss: 0.8757 - val_accuracy: 0.8125
```

Figure 27 : Aperçu du journal d'entrainement du LSTM

e Les résultats

Il est temps d'observer et d'analyser les chiffres pour en dégager une conclusion. Les données tabulaires, initialement sous forme de virgule, de l'exactitude et de la perte au fil des epoch du Conv1D, Conv2D et LSTM, sont traduites sous forme de tableau. Des graphiques (figure 28) montrent l'évolution de l'exactitude durant les phases d'entraînement et de test en fonction du nombre d'itération.

Le modèle en Conv1D augmente rapidement en exactitude au cours des 10 premiers epoch, puis se stabilise autour de 88%. Il y a une différence minime entre les courbes d'entraînement et de test. L'écart entre les deux courbes révèle la dose de surapprentissage du modèle. Il n'y a donc pas d'overfitting. Les données inconnues sont aussi bien traitées que les données connues. Le modèle en Conv2D présente un comportement similaire. Au 20ème epoch, l'exactitude est plafonnée à 95%. Elle décroît et atteint 87% au bout du 30ème epoch. La performance de test varie davantage. La sensibilité est plus importante au dataset de test que pour le Conv1d. L'exactitude du modèle en LSTM dépasse celle des deux autres modèles, même si elle présente la même tendance. Elle atteint 98 %. Cependant, elle baisse de 10% lors du test. Le modèle souffre d'overfitting.

J'en conclus que parmi les trois modèles, le LSTM est le plus exact au 30ème epoch avec une exactitude de 98%, suivi du Conv1D et du Conv2D. L'architecture semble en effet être plus adaptée à l'analyse des données séquentielles. Le modèle en Conv1d est celui qui a le moins souffert d'overfitting. La généralisation à des données inconnues est de meilleure qualité. À l'opposé, le LSTM a sur-appris malgré d'excellents résultats.

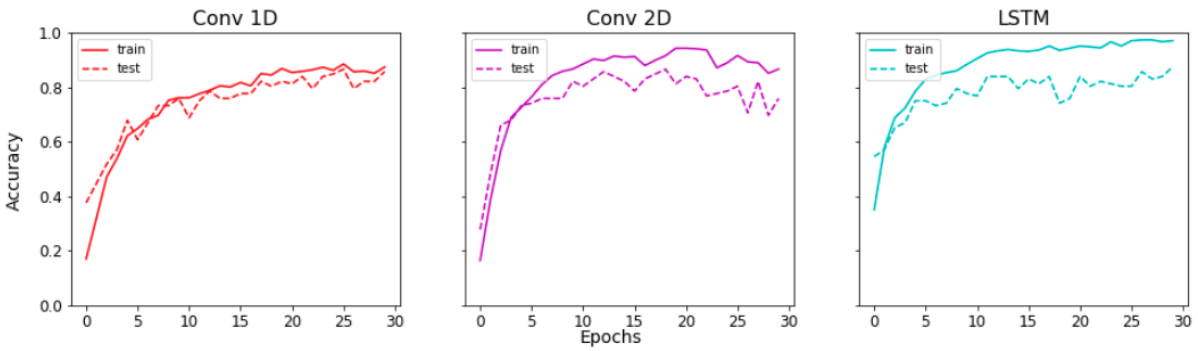


Figure 28 : Évolution de l'exactitude des modèles pour la classification d'instrument de musique

3 Un caisson isolé pour la prise de son

C'est suite à une longue réflexion sur l'automatisation du déclenchement du coup et du démarrage de l'enregistrement que je me suis finalement résolue à imaginer un mécanisme manuel (figure 29).

La première étape consiste à couper et assembler les planches en bois avec des vis. Un trou avec une encoche a été créé sur le côté haut du caisson pour pouvoir insérer le marteau et créer un système de levier avec le rond métallique. Le marteau a été percé pour que la barre en fer puisse se glisser et servir d'appui. Enfin, ce sont deux couches d'isolants acoustiques qui ont été collées à l'intérieur des parois. Le caisson est de 40 x 40 x 25cm.

Le téléphone est posé sur la base du caisson. J'appuie manuellement sur la touche de démarrage de l'enregistrement avant de plaquer la boîte contre le mur et de porter le coup.

Le matériel :

- Des planches en bois
- Un marteau
- De la mousse en polyuréthane
- Une poignée
- Un rond métallique
- Une scie à onglet
- Une ponceuse
- Une perceuse
- Une visseuse et des vis
- Un cutter
- De la colle
- De l'adhésif double face

Figure 29 : Photographie du caisson acoustique

4 La construction des jeux de données : les sons d'impact avec le caisson

Cette étape paraît anodine. Et pourtant, il s'agit d'une des plus longues et fastidieuses. Le contenu du dataset est important. C'est le support d'apprentissage pour le programme. Il faut accorder une attention particulière à tous les éléments pour optimiser l'apprentissage et être doté d'un soupçon de maîtrise d'une partie de l'entraînement.

Le titre « construction du jeu de données » sous-entend plusieurs processus. Celui d'enregistrer, mais également celui de pré-prétraiter, en amont du fichier python pour nettoyer les données, et celui de prétraiter. La segmentation « manuelle » s'ajoute à la procédure, à défaut de pouvoir tout automatiser.

Je me rends dans un logement dont je connais la composition des murs, munie du caisson acoustique. Je brandis mon téléphone et le pose soigneusement au devant du dispositif, microphone tourné vers le point d'impact. Le téléphone est à une distance d'environ cinq centimètres du mur. Je démarre l'enregistrement sur l'application WaveEditor. Je plaque le caisson contre le mur. Le marteau percute le mur à l'aide du levier. Je translate la boîte d'une vingtaine de centimètres. Puis retape le mur. Et ainsi de suite. Cela me permet d'avoir en trente ou quarante minutes un ou deux enregistrements d'une vingtaine de coups de plus ou moins bonne qualité. J'effectue la même action sur plusieurs murs au sein du logement. Je remplis les fiches d'enregistrement en conséquence.

Les premiers essais ont été peu fructueux. D'une part, je n'avais pas mis en place une procédure efficace. Je réalisais un ou deux enregistrements par mur. Je produisais moi-même du son lors des percussions. J'ai compris que cette méthode demande en effet un certain nombre d'impératifs. Être stable durant le coup. Être silencieux. Rester suffisamment longtemps au même endroit sans bouger pour avoir un son assez propre. Malgré tout, le caisson destiné à atténuer le bruit peut lui-même en générer, et les voix restent audibles.

Une fois tous les sons enregistrés, les fichiers WAV sont copiés dans un disque dur d'ordinateur. Une préparation préliminaire des données peut débuter : la segmentation. Pour cela, j'utilise le logiciel libre Audacity pour éditer la piste sonore.

La piste sonore contient plusieurs sons d'impact. Mon objectif est de ne conserver que ces sons grâce à la découpe audio. L'enregistrement de trente minutes évolue en une dizaine de fichiers WAV distincts. Certains signaux sonores ne sont pas gardés à cause des bruits émis par la boîte. Ce travail prend plusieurs heures. Bien que je n'aie pas chronométré précisément cette tâche, il faut compter trois heures pour aboutir à soixante pistes audios.

Je décide de créer deux classes:

- Mur en plâtre : 29 enregistrements,
- Mur en brique recouvert de plâtre : 35 enregistrements.

Au total, le répertoire compte 64 enregistrements.

Ces nouveaux fichiers stockés sur le disque dur sont ensuite téléversés dans Google Drive, la plateforme de stockage et de partage de fichiers sur un serveur en ligne. En même temps, je copie le Jupyter Notebook et modifie le fichier « clean.py ». Je corrige les liens des répertoires sources et de destination. Le répertoire source est alors composé de deux dossiers, « Platre» et « Platre_Brique ». Le répertoire de destination est créé conformément. Je vérifie, en me servant du graphique, qu'un fichier est correctement découpé. Je modifie quelques variables : la dimension de la fenêtre glissante qui permet de créer l'enveloppe sonore (dans la sous-partie a) et le seuil en-deçà duquel le signal est supprimé (dans la partie 3). Je lance l'exécution et le tour est joué.

Cela ne s'arrête pas là. La construction des jeux de données s'est faite de manière itérative. Suite à des essais sur les nouvelles données, j'ai agrandi le nombre de classes en reprenant la même procédure :

- Mur en parpaing : 56 enregistrements,
- Une cloison alvéolaire en BA13 et carton : 57 enregistrements.

La somme de tous les enregistrements est de 177. Le nombre d'échantillons est donc beaucoup plus petit que pour la classification des instruments de musique.

J'ai suivi les mêmes étapes, en commençant par l'enregistrement avec le caisson acoustique, en passant par la segmentation avec Audacity et en clôturant avec le nettoyage des signaux sonores.

Ces spectrogrammes sont une représentation des données en sortie de première couche. Ils ne sont cependant pas stockés sur le disque dur à moins d'écrire quelques lignes pour créer ces données. Dans le cadre de l'expérience, le stockage des spectrogrammes n'est pas nécessaire. Ce qu'il est intéressant d'observer est la suppression d'une partie du signal. Ce qui est en noir révèle les valeurs nulles.

Lorsque toutes les données sont créées, je peux construire le dataset. C'est dans le fichier « train.py » que je génère le dataset d'entraînement et de test.

5 Une classification de sons d'impact

Une première expérience confirme que la classification d'instruments de musique par un LSTM, un Conv1D et un Conv2D est fiable. J'ai plusieurs datasets pour entraîner et tester les modèles. Je peux désormais effectuer une classification de matériaux de construction.

Je transforme légèrement le code pour les modèles et l'entraînement afin de les adapter aux nouvelles données. Je modifie donc le nombre de classes, la taille des batches et le nombre d'epoch des fichiers « model.py » et « train.py ». Je change plusieurs fois les valeurs des hyperparamètres jusqu'à aboutir à un processus qui fonctionne :

- La taille du batch, autrement dit le nombre d'échantillons d'un lot: 8
- Le nombre d'epochs : 30
- Nombre de batches : 177/8 = 22

Je lance ensuite l'entraînement (figure 30).

a Résultats

Sur la figure 31, l'exactitude (accuracy) des modèles lors de l'entraînement frappe aux yeux. Elle augmente rapidement et atteint les 100% entre le 6ème et le 13ème epoch.

Pour le Conv1D, l'exactitude atteint 100% au bout du 13ème epoch. L'exactitude du modèle sur les données de test reste inférieure. Elle oscille beaucoup. Au bout du 30ème epoch, le modèle n'est toujours pas stable. Un surapprentissage est visible.

Pour le Conv2D, l'exactitude durant l'entraînement et le test augmente quasi-simultanément. Elle atteint 100% au bout du 10ème epoch. Sa stabilité

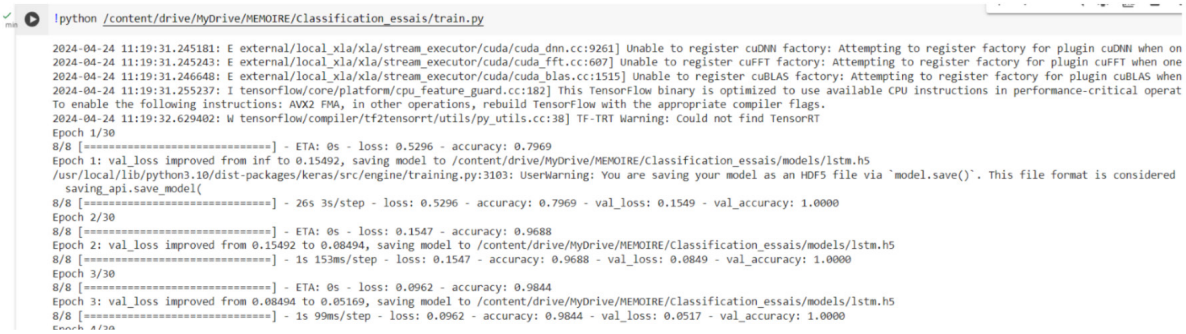


Figure 30 : Aperçu du journal d'entrainenement du modèle LSTM pour la classification des sons d'impact

laisse croire à une bonne généralisation sur de nouvelles données.

Enfin, le modèle qui a une architecture LSTM plafonne à 100% d'exactitude dès le 6ème epoch. La courbe de l'exactitude de test suit la même trajectoire mais reste légèrement instable. Un overfitting a lieu.

Pour conclure, le Conv2d semble être l'architecture qui offre la meilleure stabilité. Les données apprises sont généralisables. Les Conv1D et LSTM ont tendance à augmenter le surapprentissage.

En variant le nombre de classes, des différences ont émergé. Pour classifier les matériaux d'un dataset de sons d'impact de cloisons en plâtre et de murs en brique recouvert de plâtre, les trois modèles sont aussi efficaces les uns que les autres. Ils ont tous une exactitude de 100% sur les datasets d'entraînement et de test. J'ai ensuite ajouté aux jeux de données des sons d'impact de mur en parpaing. Pour reconnaître un de ses trois matériaux, un Conv1D est plus approprié. Enfin, les sons d'impact de cloison en carton alvéolaire entre des plaques de plâtre ont complété les jeux de données. Cette fois-ci, c'est le modèle en Conv2D qui a le mieux appris.

b Le problème du surapprentissage

Le plus important est que le modèle puisse s'adapter à toutes les données qu'on pourrait lui présenter. Pourrait-on faire une reconnaissance des matériaux dans un contexte plus bruyant ? Pour préparer la suite de l'expérience, je repasse par l'étape de l'enregistrement. Les murs frappés sont les mêmes. En revanche, cette fois, j'abandonne le caisson acoustique. J'opère tout manuellement. Rien n'est contrôlé. Un nouveau dossier s'installe à mon répertoire : « Pour prédiction ». Toutes les données dont j'aimerais prédire les matériaux y sont rangées. Elles sont au nombre de 10.

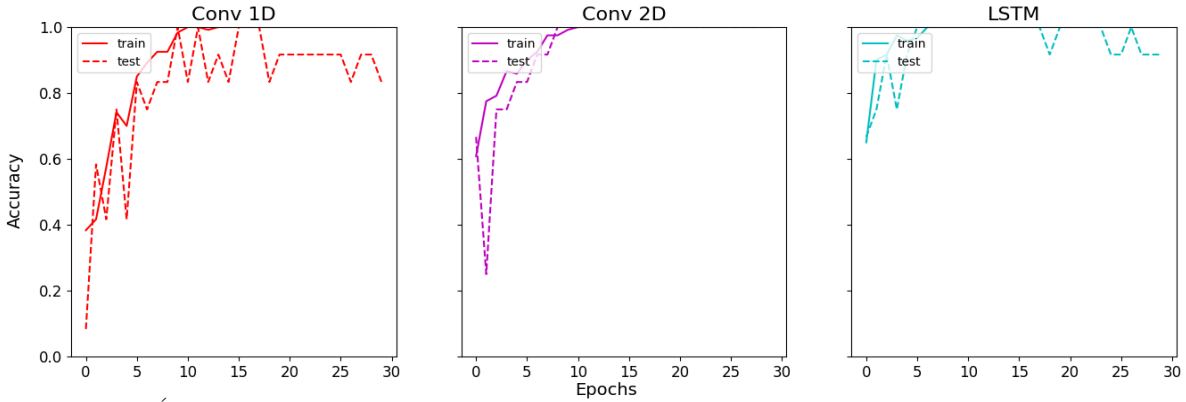


Figure 31 : Évolution de l'exactitude des modèles entraînés sur quatre types de matériaux

1 Importation des modules

```python
from tensorflow.keras.models import load_model
from clean import downsample_mono, envelope
from kapre.time_frequency import STFT, Magnitude, ApplyFilterbank, MagnitudeToDecibel
from sklearn.preprocessing import LabelEncoder
import numpy as np
from glob import glob
import argparse
import os
import pandas as pd
from tqdm import tqdm
```

2 Définition des fonctions

```python
def make_prediction(args):

    model = load_model(args.model_fn,
        custom_objects={'STFT':STFT,
                        'Magnitude':Magnitude,
                        'ApplyFilterbank':ApplyFilterbank,
                        'MagnitudeToDecibel':MagnitudeToDecibel})
    wav_paths = glob('{}/**'.format(args.src_dir), recursive=True)
    wav_paths = sorted([x.replace(os.sep, '/') for x in wav_paths if '.wav' in x])
    classes = sorted(os.listdir(args.src_dir))
    labels = [os.path.split(x)[0].split('/')[-1] for x in wav_paths]
    le = LabelEncoder()
    y_true = le.fit_transform(labels)
    results = []

    for z, wav_fn in tqdm(enumerate(wav_paths), total=len(wav_paths)):
        rate, wav = downsample_mono(wav_fn, args.sr)
        mask, env = envelope(wav, rate, threshold=args.threshold)
        clean_wav = wav[mask]
        step = int(args.sr*args.dt)
        batch = []

        for i in range(0, clean_wav.shape[0], step):
            sample = clean_wav[i:i+step]
            sample = sample.reshape(-1, 1)
            if sample.shape[0] < step:
                tmp = np.zeros(shape=(step, 1), dtype=np.float32)
                tmp[:sample.shape[0],:] = sample.flatten().reshape(-1, 1)
                sample = tmp
            batch.append(sample)
        X_batch = np.array(batch, dtype=np.float32)
        y_pred = model.predict(X_batch)
        y_mean = np.mean(y_pred, axis=0)
        y_pred = np.argmax(y_mean)
        real_class = os.path.dirname(wav_fn).split('/')[-1]
        print('Actual class: {}, Predicted class: {}'.format(real_class, classes[y_pred]))
        results.append(y_mean)

    np.save(os.path.join('/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/logs', args.pred_fn), np.array(results))
```

3 Ajout et analyse des arguments

```python
if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Audio Classification Training')
    parser.add_argument('--model_fn', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/models/lst
                        help='model file to make predictions')
    parser.add_argument('--pred_fn', type=str, default='y_pred',
                        help='fn to write predictions in logs dir')
    parser.add_argument('--src_dir', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/wavfiles',
                        help='directory containing wavfiles to predict')
    parser.add_argument('--dt', type=float, default=1.0,
                        help='time in seconds to sample audio')
    parser.add_argument('--sr', type=int, default=16000,
                        help='sample rate of clean audio')
    parser.add_argument('--threshold', type=str, default=20,
                        help='threshold magnitude for np.int16 dtype')
    args, _ = parser.parse_known_args()

    make_prediction(args)
```

Figure 32 : Code du fichier predict.py

```
1/1 [==============================] - 0s 24ms/step
File name: Platre brique-1.wav, Predicted class: Platre_carton alveo_platre
1/1 [==============================] - 0s 25ms/step
File name: Platre brique-2.wav, Predicted class: Platre_Brique
1/1 [==============================] - 0s 34ms/step
File name: Platre-1.wav, Predicted class: Platre_carton alveo_platre
1/1 [==============================] - 0s 34ms/step
File name: Platre-2.wav, Predicted class: Platre_carton alveo_platre
1/1 [==============================] - 0s 48ms/step
File name: Platre-3.wav, Predicted class: Platre
1/1 [==============================] - 0s 49ms/step
File name: Platre-4.wav, Predicted class: Platre_carton alveo_platre
1/1 [==============================] - 0s 35ms/step
File name: Platre-brique-2.wav, Predicted class: Platre_Brique
1/1 [==============================] - 0s 47ms/step
File name: Platre-brique-3.wav, Predicted class: Platre
1/1 [==============================] - 0s 39ms/step
File name: Platre-brique-4.wav, Predicted class: Platre_carton alveo_platre
1/1 [==============================] - 0s 49ms/step
File name: Platre.wav, Predicted class: Platre
100% 50/50 [00:27<00:00,  1.83it/s]
```

Figure 33 : Aperçu des prédictions par le modèle Conv2D

Je n'entraîne pas les modèles à nouveau sur ces données. Je cherche à montrer si, oui ou non, les modèles tels qu'ils sont sont capables de reconnaître le bon matériau. Pour cela, je déploie les modèles pour pouvoir effectuer des prédictions sans les entraîner (figure 32).

J'importe les modules. Je définis la fonction « make_prediction ». Je charge le modèle. La donnée d'entrée est transformée pour qu'elle reprenne la même forme que les données des datasets. Pour cela, les mêmes tâches que le fichier "clean.py" sont effectuées. Les données sont cependant stockées temporairement. L'algorithme procède au nettoyage des données et à l'application de la transformée de Fourier, au calcul d'un spectrogramme de décibels. La prédiction peut ensuite se réaliser. Puis, je définis et vérifie tous les arguments et enfin, je lance le processus grâce à la commande "make_prediction".

Sur les 10 échantillons, 4 prédictions sont exactes (figure 33). Les performances ne sont pas aussi bonnes que les chiffres précédents les laissent entendre.

Cela veut donc dire que les algorithmes ont sur-appris. Plusieurs facteurs pourraient en être à l'origine. Le premier est que le nombre d'échantillons est trop faible. En effet, les algorithmes donnent en général de meilleurs résultats sur un grand nombre de données. Plus il y en a et mieux c'est. Avoir plus de données permettrait de mieux représenter l'ensemble des sons d'impact selon leur matériau. Je pourrais donc réitérer les enregistrements dans de nouveaux lieux.

La deuxième hypothèse est que les données ne sont pas assez variées. Il pourrait manquer des échantillons de sons d'impact réalisés sans caisson acoustique pour l'entraînement. Réitérer la phase d'enregistrement sans caisson pourrait améliorer les performances.

Si augmenter les données et changer de contexte sonore n'est pas suffisant, le code pourrait être perfectionné. La couche de dropout des réseaux de neurones n'est vraisemblablement pas suffisante. Je peux également augmenter informatiquement les données. Les nouvelles données naissent de l'altération de celles existantes. Du bruit, l'atténuation ou le renforcement du contraste peuvent être appliqués.

PARTIE 4

BILAN

LES PRÉMISSES D'UNE RECHERCHE PLUS ABOUTIE

L'expérience a permis de valider une partie de l'hypothèse de la recherche. Classifier les matériaux de construction par le biais des sons d'impact grâce au réseau de neurones est possible. Pourtant, quand le contexte d'enregistrement des sons pour une prédiction diffère du contexte d'enregistrement pour l'entraînement, les résultats sont nuancés.

Mon travail comporte des limites, mais également quelques potentiels qu'il est important d'explorer. Retracer les grandes lignes de cette expérience permet de mettre en exergue des pistes d'amélioration.

1 L'analyse des résultats

En croisant les résultats de la classification audio des instruments de musique et les résultats de la classification audio des matériaux de construction, je constate que les courbes d'évolution de l'exactitude des modèles suivent la même trajectoire. Les réseaux de neurones convolutifs et récurrents peuvent s'adapter à plusieurs types de données. Même si l'on joue avec le nombre de classes, les modèles sont ajustés avec brio. J'insiste sur le fait que la même procédure a été suivie pour les deux expériences. Les données brutes sont nettoyées. La durée des fichiers audio est identique, le taux d'échantillonnage, la profondeur de bits également. Finalement, la principale variable sur laquelle je me concentre sont les fréquences au fil du temps. Ce que je peux donc assurer est que les modèles peuvent s'adapter à différents types de données dans le cas où les échantillons sont d'une durée d'une seconde, de 16000 hertz de taux d'échantillonnage et que la profondeur de bits est de 16 bits. Si les valeurs des variables sont plus élevées, je suppose que le temps de l'entraînement serait plus long.

Par ailleurs, le nombre de paramètres des modèles est relativement faible, autour de 400 000. Ils pourraient facilement être déployés sur un ordinateur équipé d'un processeur de 4 cœurs, d'une mémoire vive de 16 Go, d'une carte graphique telle que NVIDIA GTX 1650 et d'un espace de stockage de 10 Go.

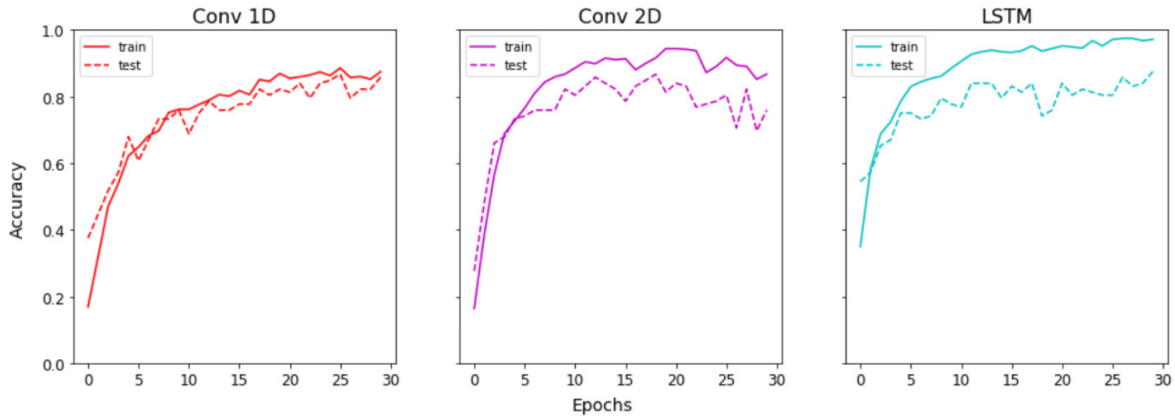


Figure 28 : Évolution de l'exactitude des modèles pour la classification d'instruments de musique

Je remarque d'une part que les algorithmes d'apprentissage profond semblent mieux ajuster les poids des modèles sur des datasets plus petits. D'autre part, dans le cas de la classification des matériaux, les signaux sonores ont tous des enveloppes très similaires, avec une attaque et un déclin, sans maintien du signal. Les sons sont enregistrés dans un environnement isolé acoustiquement. Les poids des modèles sont correctement ajustés pour les datasets. Le Conv2d est l'architecture aboutissant au modèle d'une justesse impeccable.

Cependant, puisque les datasets sont très petits, les résultats de l'expérience sur les quatre types de matériaux de construction révèlent un surapprentissage. Les données ne peuvent être généralisées à des données résultant de coups d'impact dans un milieu non isolé. La variation de la puissance du coup sur les murs et l'absence de caisson acoustique diminuent les chances d'identifier le matériau. Il pourrait manquer de diversité de situations dans les jeux de données.

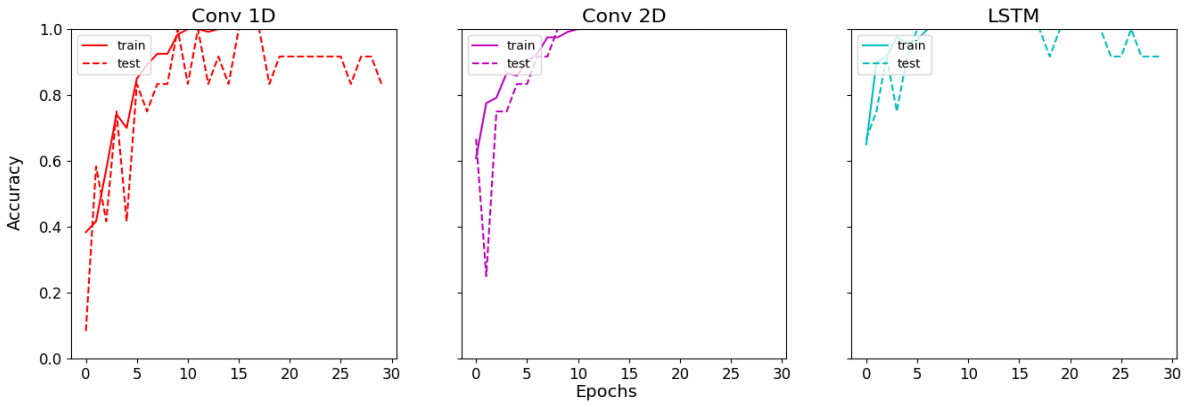


Figure 30 : Évolution de l'exactitude des modèles entraînés sur quatre types de matériaux

2 Les pistes d'amélioration

D'autres améliorations de la procédure à toutes les phases peuvent être proposées, à commencer par la prise de son.

La première piste d'amélioration concerne la méthode d'enregistrement des sons. Une application peut être développée et déployée sur le smartphone pour mieux maîtriser le moment et la durée de la prise de son. À titre d'exemple, sur l'appareil que j'utilise, un Android, Audiorecord permet de réaliser mon propre logiciel en Javascript. C'est une boîte à outils, un framework tout comme Tensorflow, mais dédié au langage Java. Il permet d'utiliser les fonctions natives d'Android. L'environnement de développement serait alors Android Studio. L'objectif serait de créer un retardateur pour démarrer l'audio quelques secondes plus tard et un minuteur qui stoppe l'enregistrement automatiquement. Malgré le temps supplémentaire qu'implique le fait de développer le programme en amont et d'appuyer sur un bouton pour lancer le processus lors de la création des sons, plusieurs heures de travail me seraient épargnées.

Je ne suis pas en mesure de proposer une amélioration précise du code Python pour les différentes étapes. Simplement, au 20 décembre 2024, les algorithmes ne sont plus à jour, à moins de réactualiser le code, il n'est plus possible d'effectuer les tâches. Une autre option serait d'exploiter localement les ressources pour avoir plus de flexibilité quant au développement de l'environnement algorithmique. J'ai également trouvé un travail sur la reconnaissance algorithmique qui propose d'identifier les échantillons qui ne font pas partie d'une classe d'échantillons étiquetés lors de la prédiction.

À propos du problème de la généralisation, une solution serait d'entraîner les réseaux de neurones sur des données aux paramètres qui varient. Des données générées sans caisson acoustique pourraient être ajoutées au jeu de données. L'amplitude du son pourrait également varier de séquences sonores en séquences sonores dans un second temps. De cette manière, le modèle pourrait plus facilement généraliser. Il faudrait remanier un paramètre à la fois pour comprendre quel modèle et quelle architecture sont les plus adéquats.

Par ailleurs, il serait intéressant de réitérer l'expérience avec plus de données et de comparer le taux d'apprentissage de l'algorithme. Étendre le nombre de matériaux donnera de plus larges possibilités. L'hypothèse est que l'apprentissage reprendrait la même tendance que celle de la courbe

d'évolution pour les 10 instruments de musique. Mais à terme, les solutions seront meilleures.

Sur le plan technique et informatique, il est possible de jouer avec tous ces paramètres. Sur le plan architectural et constructif, de nombreuses questions émergent.

Comment faire pour les murs composites ? Pour ces édifices composés de matériaux de différentes époques, juxtaposés, entremêlés ? Suffit-il d'augmenter le nombre de classes, la quantité d'échantillons ?

Après avoir trouvé des solutions pour la généralisation, j'aurais pu enrichir le nombre de classes pour classifier des murs plus complexes. Pour identifier les épaisseurs de murs. Pour repérer des montants métalliques (certes pas aussi fidèlement qu'un Ferroscan). Je suis de l'avis que cette méthode de reconnaissance des matériaux a du potentiel. Cette expérience n'est qu'un fragment de toutes les possibilités qu'offre le Deep Learning. Ce n'est plus un modèle qu'il faut créer, mais moults modèles. Un système pour répondre à tous ces problèmes. Nous pourrions collectionner des sons dès la construction des bâtiments pour créer un large dataset. Plusieurs algorithmes se nourriraient de ces échantillons. À la manière des médecins. Certains algorithmes seraient généralistes, précisant si le mur est homogène ou composite, d'autres seraient les spécialistes dans un domaine. Par exemple, l'un pourrait être spécialisé dans les murs composites, décelant les matériaux de manière plus précise.

Toujours est-il qu'il existe encore des parts d'ombre, des ambiguïtés, des exemples uniques. Des données échapperont toujours au système. Les algorithmes ont leurs limites. Nous devons nous résoudre à les accepter et tirer parti de leurs forces et de leurs faiblesses.

CONCLUSION

Le projet sur l'existant est valorisé principalement par les réglementations. De ce fait, sonder le « déjà là », les édifices, est essentiel. Les investigateurs dans le secteur du bâtiment opèrent leur diagnostic avec des outils sophistiqués et onéreux, tels que le ferroscan et le spectromètre. Les dispositifs aux prix abordables sont en revanche moins fiables. De là est née l'idée d'associer sondages et intelligences artificielles. La classification des sons a bien été réalisée (Dimiccoli, 2022). L'efficacité d'un algorithme de machine learning s'est confirmée.

J'ai présenté, au cours de cette recherche, une méthode de sondage en unissant analyses fréquentielles des sons au cours du temps et programmation d'algorithmes en réseaux de neurones LSTM, Conv1D et Conv2D. Une reconnaissance des matériaux de construction se pratique à travers la classification des matériaux. Mon expérimentation se fonde donc sur la classification des matériaux de construction par le biais de sons d'impact. Plusieurs itérations du processus ont été effectuées. Les algorithmes ont réalisé une classification d'instruments de musique. J'ai construit un caisson acoustique et un jeu de données constitué de sons de percussions sur des murs classés par matériaux. Une classification de ses sons a suivi. Cette expérience prouve que les algorithmes sont capables de générer des modèles d'une justesse de 100% dans un cadre expérimental, selon le nombre de classes et l'architecture du modèle. Les algorithmes ajustent rapidement les paramètres sur les données d'entraînement.

Cependant, dans des conditions différentes, les modèles sont plus inexacts. Une généralisation à l'ensemble des données possibles est infirmée. Cela est notamment dû au faible nombre d'échantillons et à des prises de sons dans des conditions sonores particulières. Pour résoudre ce défaut, augmenter le nombre d'échantillons améliorerait les performances. Des enregistrements sans caisson peuvent être introduits dans le dataset. Enfin, seuls quatre matériaux font partie du corpus. De nouveaux matériaux pourraient former de nouvelles classes. Comment se déroule alors la reconnaissance des matériaux sur les murs plus complexes? Jusqu'où pourrions-nous pousser l'expérience?

TABLE DES FIGURES

BIBLIOGRAPHIE

OUVRAGES

Rocchesso, Davide. Fontana, Federico. *The Sounding Object*. Italie, Firenze : Mondo estremo, 2003. 399 p.

TRAVAUX DE RECHERCHE

R.P. Wildes. W.A. Richards. «Recovering material properties from sound.» *Natural Computation*, 1988. Consulté le 2 janvier 2024

Krotkov, Eric. Klatzky, Roberta. Zumel, Nina. «Analysis and synthesis of the sounds of impact based on Shape-Invariant Properties of Materials.» Université de Carnegie Mellon. Pittsburgh, Pennsylvanie. 1996.

L. Giordano, Bruno. «Material categorization and hardness scaling in real and synthetic impact sounds», *The sounding object.* Italie, Firenze : Mondo estremo, 2003. pp.73-93

Cavaco, Sofia. Rodeia, José.« Classification of Similar Impact Sounds », in *Image and Signal Processing,* éd. par Abderrahim Elmoataz et al., vol. 6134, Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. pp. 307-14, https://doi.org/10.1007/978-3-642-13681-8_36.

Guo Di, Liu Huaping, Fang Bin, Sun Fuchun et Yang Wuqiang, « Visual Affordance Guided Tactile Material Recognition for Waste Recycling », *IEEE Transactions on Automation Science and Engineering*, 19-4, octobre 2022, p. 2656-2664.

Son Hyojoo, Kim Changmin, Hwang Nahyae, Kim Changwan et Kang Youngcheol, « Classification of major construction materials in construction environments using ensemble classifiers », *Advanced Engineering Informatics*, 28-1, 1 janvier 2014, p. 1-10. https://doi.org/10.1016/j.aei.2013.10.001.

Shi Fengmin, Guo Jie, Zhang Haonan, Yang Shan, Wang Xiying et

Guo Yanwen. « GLAVNet: Global-Local Audio-Visual Cues for Fine-Grained Material Recognition », in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA: IEEE, 2021), 14428-37, https://doi.org/10.1109/CVPR46437.2021.01420.

Dimiccoli Mariella, Patni Shubhan, Hoffmann Matej et Moreno-Noguer Francesc. « Recognizing object surface material from impact sounds for robot manipulation ». In 2022 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan: IEEE, 2022 .9280-87. doi:10.1109/IROS47612.2022.9981578.

Van der Wielen, Audrey. Nguyen, Frédéric. Courard, Luc. «Détermination des propriétés de couches minces dans le béton à l'aide d'un géoradar commercial à hautes fréquences: approche pic-à-pic et analyse fréquentielle du coefficient de réflexion.». A*nnales Du bâtiment Et Des Travaux Publics*, 4(01). Université de Liège: 2014. https://orbi.uliege.be/bitstream/2268/165565/1/ArticleAvdWielen_GPR.pdf.

James, Amala et al. « Rebar corrosion detection, protection, and rehabilitation of reinforced concrete structures in coastal environments: A review », *Construction and Building Materials 224*, 10 novembre 2019. 1026-39. https://doi.org/10.1016/j.conbuildmat.2019.07.250.

Costas P. Providakis, Maria G. Mousteraki, et Georgia C. Providaki, « Operational Modal Analysis of Historical Buildings and Finite Element Model Updating Using α Laser Scanning Vibrometer », *Infrastructures* 8, no 2. Février 2023. https://doi.org/10.3390/infrastructures8020037.

Xavier Marsault et Hong Minh-Chau Nguyen, « Les GANs : stimulateurs de créativité en phase d'idéation », in *SHS Web of Conferences*, éd. par K. Jacquot et H. Lequay, vol. 147, SHS Web of Conferences (Lyon, France, 2022), 06003, https://doi.org/10.1051/shsconf/202214706003.

Shanaka Kristombu Baduge et al., « Artificial intelligence and smart vision for building and construction 4.0: Machine and deep learning methods and applications », *Automation in Construction 141*, 1 septembre 2022. https://doi.org/10.1016/j.autcon.2022.104440.

Hochreiter, Sepp. Schmidhuber, Jürgen. «Long Short-Term Memory». *Neural Computation* 9(8):1735-1780, 1977. doi:10.1162/neco.1997.9.8.1735

Adams, Seth. « Deep learning for audio classification »,

SITES EN LIGNE

« Son analogique / Son numérique | AFSI », *AssoConnec*t, consulté le 2 avril 2024, https://www.afsi.eu/articles/12926-son-analogique-son-numerique.

Brownlee, Jason. « Machine Learning Mastery ». Disponible sur : https://www.machinelearningmastery.com/.

Gandhi et al. « Introduction aux données audio - Hugging Face Audio Course », consulté le 12 mars 2024. Disponible sur : https://huggingface.co/learn/audio-course/fr/chapter1/audio_data.

VIDÉOS

Nordby, Jon. «Audio Classification with Machine Learning» EuroPython, Bâle, 2019. Disponible sur : https://www.youtube.com/watch?v=uCGROOUO_wY.

YouTube. Avril 2020. consulté le 27 octobre 2024, http://www.youtube.com/playlist?list=PL7Zd_5jZT2cerrNmV3HR3-RHocb6fWe4z.

GLOSSAIRE

Algorithme

Suite de procédés de calculs à partir de données d'entrées pour aboutir un résultat.

Bibliothèque

- LIBRARY EN ANGLAIS

Library en anglais. Ensemble de codes définissant des fonctions qui peuvent réutilisées dans d'autres fichiers codés.

Computation

- ANGLICISME

Calcul.

Dataset

- ANGLICISME

Jeu/ensemble de données.

Deep Learning

- ANGLICISME

Apprentissage profond en français. Sous domaine du Machine Learning. Processus de transformations multiples de la données en plusieurs couches cachées structurées en réseau.

Descente de gradient

Algorithme d'optimisation reposant sur l'ajustement des paramètres des algorithmes de Machine Learning pour produire un modèle avec la plus faible perte.

Environnement virtuel

Ensemble des composantes matérielles et logicielles sur lesquels sont executés les programmes. Ressources telles que la mémoire, les processeurs, le stockage, les systèmes d'exploitations, etc.

Fonction de perte

- OU FONCTION DE COÛT

Calculs de l'écart une valeur prédite et une valeur réelle pour mesurer la performance d'un modele

GitHub

- NOM PROPRE

Une plateforme de développement de logiciel en ligne. Permet de stocker et de collaborer sur des projets informatique.

Gradient

Vecteur. Pointe vers la direction la montée la plus raide de la fonction de perte.

Itération

Action de renouveler.

Jupyter Notebook

- NOM PROPRE

Un éditeur de «bloc-note» pour la programmation et la visualisation de données.

Keras

- NOM PROPRE

Une interface de programmation d'application. Puise les sources de code d'un framework comme Tensorflow.

Machine Learning

- ANGLICISME

Apprentissage automatique en français. Une branche de l'intelligence artificielle. Technique informatique reposant sur l'analyse et l'optimisation des performances d'une machine à travers l'expérience.

Matplotlib

- NOM PROPRE

Une bibliothèque Python pour la visualisation graphique de données.

Modèle

Toute construction mathématique qui transforment des données. Ensemble de paramètres et de structure nécessaire pour faire des prédictions.

NumPy

- NOM PROPRE

Une bibliothèque Python pour la création et la transformation de matrices.

Processeur (CPU)

Unité centrale de calcul. Système capable d'executer des calculs.

Python

Langage de programmation.

Réseau de neurones

> NEURAL NETWORK EN ANG.

Un type de machine learning. Un programme ou un modèle contenant une ou plusieurs couches cachées.

Réseau de neurones convolutifs (CNN)

> CONVOLUTIONAL NEURAL NETWORK EN ANGLAIS

Une architecture de réseau de neurones transformant la donnée grâce à un procédé mathématique lié à l'application de filtres.

Réseau de neurones récurrents (RNN)

> RECURRENT NEURAL NETWORK EN ANGLAIS

Une architecture de réseau de neurones caractérisée par la réalisation de mêmes opérations de manière récurrente.

TensorFlow

> NOM PROPRE

Un ensemble de bibliothèques (un framework) particulièrement utilisé pour le machine learning.

ANNEXES

Structure du modèle avec une couche LSTM

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 1, 16000)]	0	
melbands (Melspectrogram)	(None, 128, 100, 1)	296064	input[0][0]
freq_norm (Normalization2D)	(None, 128, 100, 1)	0	melbands[0][0]
permute (Permute)	(None, 100, 128, 1)	0	freq_norm[0][0]
reshape (TimeDistributed)	(None, 100, 128)	0	permute[0][0]
td_dense_tanh (TimeDistributed)	(None, 100, 64)	8256	reshape[0][0]
bidirectional_lstm (Bidirection	(None, 100, 64)	24832	td_dense_tanh[0][0]
skip_connection (Concatenate)	(None, 100, 128)	0	td_dense_tanh[0][0] bidirectional_lstm[0][0]
dense_1_relu (Dense)	(None, 100, 64)	8256	skip_connection[0][0]
max_pool_1d (MaxPooling1D)	(None, 50, 64)	0	dense_1_relu[0][0]
dense_2_relu (Dense)	(None, 50, 32)	2080	max_pool_1d[0][0]
flatten (Flatten)	(None, 1600)	0	dense_2_relu[0][0]
dropout (Dropout)	(None, 1600)	0	flatten[0][0]
dense_3_relu (Dense)	(None, 32)	51232	dropout[0][0]
softmax (Dense)	(None, 10)	330	dense_3_relu[0][0]

Total params: 391,050
Trainable params: 391,050
Non-trainable params: 0

Structure du modèle avec des couches Conv2D

Layer (type)	Output Shape	Param #
input (InputLayer)	[(None, 1, 16000)]	0
melbands (Melspectrogram)	(None, 128, 100, 1)	296064
freq_norm (Normalization2D)	(None, 128, 100, 1)	0
conv2d_tanh (Conv2D)	(None, 128, 100, 8)	400
max_pool_2d_1 (MaxPooling2D)	(None, 64, 50, 8)	0
conv2d_relu_1 (Conv2D)	(None, 64, 50, 16)	3216
max_pool_2d_2 (MaxPooling2D)	(None, 32, 25, 16)	0
conv2d_relu_2 (Conv2D)	(None, 32, 25, 16)	2320
max_pool_2d_3 (MaxPooling2D)	(None, 16, 13, 16)	0
conv2d_relu_3 (Conv2D)	(None, 16, 13, 32)	4640
max_pool_2d_4 (MaxPooling2D)	(None, 8, 7, 32)	0
conv2d_relu_4 (Conv2D)	(None, 8, 7, 32)	9248
flatten (Flatten)	(None, 1792)	0
dropout (Dropout)	(None, 1792)	0
dense (Dense)	(None, 64)	114752
softmax (Dense)	(None, 10)	650

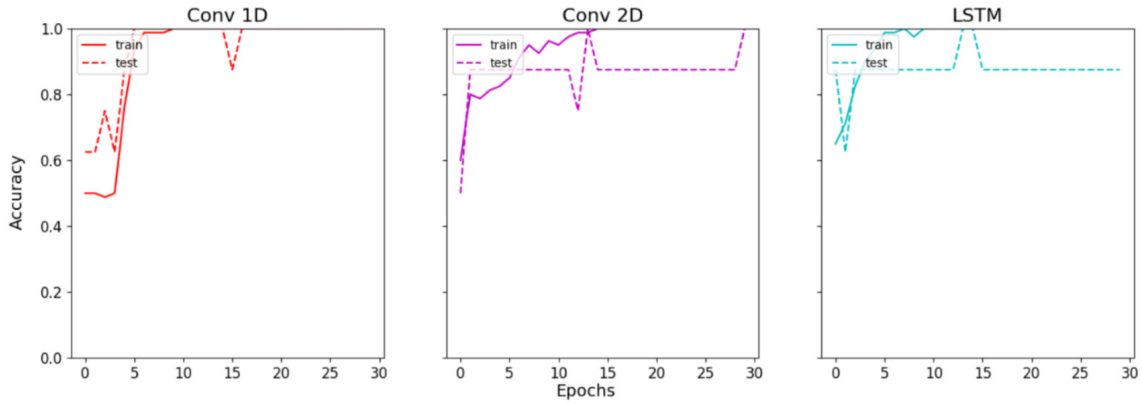
Total params: 431,290
Trainable params: 431,290
Non-trainable params: 0

Structure du modèle avec des couches Conv1D

Layer (type)	Output Shape	Param #
input (InputLayer)	[(None, 1, 16000)]	0
melbands (Melspectrogram)	(None, 128, 100, 1)	296064
freq_norm (Normalization2D)	(None, 128, 100, 1)	0
permute (Permute)	(None, 100, 128, 1)	0
td_conv_1d_tanh (TimeDistrib	(None, 100, 125, 8)	40
max_pool_2d_1 (MaxPooling2D)	(None, 50, 62, 8)	0
td_conv_1d_relu_1 (TimeDistr	(None, 50, 59, 16)	528
max_pool_2d_2 (MaxPooling2D)	(None, 25, 29, 16)	0
td_conv_1d_relu_2 (TimeDistr	(None, 25, 26, 32)	2080
max_pool_2d_3 (MaxPooling2D)	(None, 12, 13, 32)	0
td_conv_1d_relu_3 (TimeDistr	(None, 12, 10, 64)	8256
max_pool_2d_4 (MaxPooling2D)	(None, 6, 5, 64)	0
td_conv_1d_relu_4 (TimeDistr	(None, 6, 2, 128)	32896
global_max_pooling_2d (Globa	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 64)	8256
softmax (Dense)	(None, 10)	650

Total params: 348,770
Trainable params: 348,770
Non-trainable params: 0

Courbes d'évolution de l'exactitude des modèles entraînés sur trois types de matériaux



Code du fichier «clean.py»

]:
```
clean.py
import matplotlib.pyplot as plt
from scipy.io import wavfile
import argparse
import os
from glob import glob
import numpy as np
import pandas as pd
from librosa.core import resample, to_mono
from tqdm import tqdm
import wavio


def envelope(y, rate, threshold):
    mask = []
    y = pd.Series(y).apply(np.abs)
    y_mean = y.rolling(window=int(rate/20),
                       min_periods=1,
                       center=True).max()
    for mean in y_mean:
        if mean > threshold:
            mask.append(True)
        else:
            mask.append(False)
    return mask, y_mean


def downsample_mono(path, sr):
    obj = wavio.read(path)
    wav = obj.data.astype(np.float32, order='F')
    rate = obj.rate
    try:
        channel = wav.shape[1]
        if channel == 2:
            wav = to_mono(wav.T)
        elif channel == 1:
            wav = to_mono(wav.reshape(-1))
    except IndexError:
        wav = to_mono(wav.reshape(-1))
        pass
    except Exception as exc:
        raise exc
    wav = resample(wav, orig_sr=rate, target_sr=sr)
    wav = wav.astype(np.int16)
    return sr, wav


def save_sample(sample, rate, target_dir, fn, ix):
    fn = fn.split('.wav')[0]
    dst_path = os.path.join(target_dir.split('.')[0], fn+'_{}.wav'.format(str(ix)))
    if os.path.exists(dst_path):
        return
    wavfile.write(dst_path, rate, sample)


def check_dir(path):
    if os.path.exists(path) is False:
        os.mkdir(path)


def split_wavs(args):
    src_root = args.src_root
    dst_root = args.dst_root
    dt = args.delta_time
```

```
    wav_paths = glob('{}/**'.format(src_root), recursive=True)
    wav_paths = [x for x in wav_paths if '.wav' in x]
    dirs = os.listdir(src_root)
    check_dir(dst_root)
    classes = os.listdir(src_root)
    for _cls in classes:
        target_dir = os.path.join(dst_root, _cls)
        check_dir(target_dir)
        src_dir = os.path.join(src_root, _cls)
        for fn in tqdm(os.listdir(src_dir)):
            src_fn = os.path.join(src_dir, fn)
            rate, wav = downsample_mono(src_fn, args.sr)
            mask, y_mean = envelope(wav, rate, threshold=args.threshold)
            wav = wav[mask]
            delta_sample = int(dt*rate)

            # cleaned audio is less than a single sample
            # pad with zeros to delta_sample size
            if wav.shape[0] < delta_sample:
                sample = np.zeros(shape=(delta_sample,), dtype=np.int16)
                sample[:wav.shape[0]] = wav
                save_sample(sample, rate, target_dir, fn, 0)
            # step through audio and save every delta_sample
            # discard the ending audio if it is too short
            else:
                trunc = wav.shape[0] % delta_sample
                for cnt, i in enumerate(np.arange(0, wav.shape[0]-trunc, delta_sample)):
                    start = int(i)
                    stop = int(i + delta_sample)
                    sample = wav[start:stop]
                    save_sample(sample, rate, target_dir, fn, cnt)


def test_threshold(args):
    src_root = args.src_root
    wav_paths = glob('{}/**'.format(src_root), recursive=True)
    wav_path = [x for x in wav_paths if args.fn in x]
    if len(wav_path) != 1:
        print('audio file not found for sub-string: {}'.format(args.fn))
        return
    rate, wav = downsample_mono(wav_path[0], args.sr)
    mask, env = envelope(wav, rate, threshold=args.threshold)
    plt.style.use('ggplot')
    plt.title('Signal Envelope, Threshold = {}'.format(str(args.threshold)))
    plt.plot(wav[np.logical_not(mask)], color='r', label='remove')
    plt.plot(wav[mask], color='c', label='keep')
    plt.plot(env, color='m', label='envelope')
    plt.grid(False)
    plt.legend(loc='best')
    plt.show()


if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Cleaning audio data')
    parser.add_argument('--src_root', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/wavfiles',
                        help='directory of audio files in total duration')
    parser.add_argument('--dst_root', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/clean',
                        help='directory to put audio files split by delta_time')
    parser.add_argument('--delta_time', '-dt', type=float, default=1.0,
                        help='time in seconds to sample audio')
    parser.add_argument('--sr', type=int, default=16000,
                        help='rate to downsample audio')

    parser.add_argument('--fn', type=str, default='3a3d0279',
                        help='file to plot over time to check magnitude')
    parser.add_argument('--threshold', type=str, default=20,
                        help='threshold magnitude for np.int16 dtype')
    args, _ = parser.parse_known_args()

    #test_threshold(args)
    split_wavs(args)
```

Code du fichier «model.py»

```
]: from tensorflow.keras import layers
from tensorflow.keras.layers import TimeDistributed, LayerNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
import kapre
from kapre.composed import get_melspectrogram_layer
import tensorflow as tf
import os


def Conv1D(N_CLASSES=10, SR=16000, DT=1.0):
    input_shape = (int(SR*DT), 1)
    i = get_melspectrogram_layer(input_shape=input_shape,
                                 n_mels=128,
                                 pad_end=True,
                                 n_fft=512,
                                 win_length=400,
                                 hop_length=160,
                                 sample_rate=SR,
                                 return_decibel=True,
                                 input_data_format='channels_last',
                                 output_data_format='channels_last')
    x = LayerNormalization(axis=2, name='batch_norm')(i.output)
    x = TimeDistributed(layers.Conv1D(8, kernel_size=(4), activation='tanh'), name='td_conv_1d_tanh')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), name='max_pool_2d_1')(x)
    x = TimeDistributed(layers.Conv1D(16, kernel_size=(4), activation='relu'), name='td_conv_1d_relu_1')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), name='max_pool_2d_2')(x)
    x = TimeDistributed(layers.Conv1D(32, kernel_size=(4), activation='relu'), name='td_conv_1d_relu_2')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), name='max_pool_2d_3')(x)
    x = TimeDistributed(layers.Conv1D(64, kernel_size=(4), activation='relu'), name='td_conv_1d_relu_3')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), name='max_pool_2d_4')(x)
    x = TimeDistributed(layers.Conv1D(128, kernel_size=(4), activation='relu'), name='td_conv_1d_relu_4')(x)
    x = layers.GlobalMaxPooling2D(name='global_max_pooling_2d')(x)
    x = layers.Dropout(rate=0.1, name='dropout')(x)
    x = layers.Dense(64, activation='relu', activity_regularizer=l2(0.001), name='dense')(x)
    o = layers.Dense(N_CLASSES, activation='softmax', name='softmax')(x)
    model = Model(inputs=i.input, outputs=o, name='1d_convolution')
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model


def Conv2D(N_CLASSES=10, SR=16000, DT=1.0):
    input_shape = (int(SR*DT), 1)
    i = get_melspectrogram_layer(input_shape=input_shape,
                                 n_mels=128,
                                 pad_end=True,
                                 n_fft=512,
                                 win_length=400,
                                 hop_length=160,
                                 sample_rate=SR,
                                 return_decibel=True,
                                 input_data_format='channels_last',
                                 output_data_format='channels_last')
    x = LayerNormalization(axis=2, name='batch_norm')(i.output)
    x = layers.Conv2D(8, kernel_size=(7,7), activation='tanh', padding='same', name='conv2d_tanh')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), padding='same', name='max_pool_2d_1')(x)
    x = layers.Conv2D(16, kernel_size=(5,5), activation='relu', padding='same', name='conv2d_relu_1')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), padding='same', name='max_pool_2d_2')(x)
    x = layers.Conv2D(16, kernel_size=(3,3), activation='relu', padding='same', name='conv2d_relu_2')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), padding='same', name='max_pool_2d_3')(x)
    x = layers.Conv2D(32, kernel_size=(3,3), activation='relu', padding='same', name='conv2d_relu_3')(x)
    x = layers.MaxPooling2D(pool_size=(2,2), padding='same', name='max_pool_2d_4')(x)
    x = layers.Conv2D(32, kernel_size=(3,3), activation='relu', padding='same', name='conv2d_relu_4')(x)
    x = layers.Flatten(name='flatten')(x)
    x = layers.Dropout(rate=0.2, name='dropout')(x)
    x = layers.Dense(64, activation='relu', activity_regularizer=l2(0.001), name='dense')(x)
    o = layers.Dense(N_CLASSES, activation='softmax', name='softmax')(x)
    model = Model(inputs=i.input, outputs=o, name='2d_convolution')
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
def LSTM(N_CLASSES=10, SR=16000, DT=1.0):
    input_shape = (int(SR*DT), 1)
    i = get_melspectrogram_layer(input_shape=input_shape,
                                 n_mels=128,
                                 pad_end=True,
                                 n_fft=512,
                                 win_length=400,
                                 hop_length=160,
                                 sample_rate=SR,
                                 return_decibel=True,
                                 input_data_format='channels_last',
                                 output_data_format='channels_last',
                                 name='2d_convolution')
    x = LayerNormalization(axis=2, name='batch_norm')(i.output)
    x = TimeDistributed(layers.Reshape((-1,)), name='reshape')(x)
    s = TimeDistributed(layers.Dense(64, activation='tanh'),
                        name='td_dense_tanh')(x)
    x = layers.Bidirectional(layers.LSTM(32, return_sequences=True),
                             name='bidirectional_lstm')(s)
    x = layers.concatenate([s, x], axis=2, name='skip_connection')
    x = layers.Dense(64, activation='relu', name='dense_1_relu')(x)
    x = layers.MaxPooling1D(name='max_pool_1d')(x)
    x = layers.Dense(32, activation='relu', name='dense_2_relu')(x)
    x = layers.Flatten(name='flatten')(x)
    x = layers.Dropout(rate=0.2, name='dropout')(x)
    x = layers.Dense(32, activation='relu',
                         activity_regularizer=l2(0.001),
                         name='dense_3_relu')(x)
    o = layers.Dense(N_CLASSES, activation='softmax', name='softmax')(x)
    model = Model(inputs=i.input, outputs=o, name='long_short_term_memory')
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Code du fichier «train.py»

```
]: import tensorflow as tf
from tensorflow.keras.callbacks import CSVLogger, ModelCheckpoint
from tensorflow.keras.utils import to_categorical
import os
from scipy.io import wavfile
import pandas as pd
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from models import Conv1D, Conv2D, LSTM
from tqdm import tqdm
from glob import glob
import argparse
import warnings


class DataGenerator(tf.keras.utils.Sequence):
    def __init__(self, wav_paths, labels, sr, dt, n_classes,
                 batch_size=32, shuffle=True):
        self.wav_paths = wav_paths
        self.labels = labels
        self.sr = sr
        self.dt = dt
        self.n_classes = n_classes
        self.batch_size = batch_size
        self.shuffle = True
        self.on_epoch_end()


    def __len__(self):
        return int(np.floor(len(self.wav_paths) / self.batch_size))


    def __getitem__(self, index):
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        wav_paths = [self.wav_paths[k] for k in indexes]
```

```
        labels = [self.labels[k] for k in indexes]

        # generate a batch of time data
        X = np.empty((self.batch_size, int(self.sr*self.dt), 1), dtype=np.float32)
        Y = np.empty((self.batch_size, self.n_classes), dtype=np.float32)

        for i, (path, label) in enumerate(zip(wav_paths, labels)):
            rate, wav = wavfile.read(path)
            X[i,] = wav.reshape(-1, 1)
            Y[i,] = to_categorical(label, num_classes=self.n_classes)

        return X, Y


    def on_epoch_end(self):
        self.indexes = np.arange(len(self.wav_paths))
        if self.shuffle:
            np.random.shuffle(self.indexes)


def train(args):
    src_root = args.src_root
    sr = args.sample_rate
    dt = args.delta_time
    batch_size = args.batch_size
    model_type = args.model_type
    params = {'N_CLASSES':len(os.listdir(args.src_root)),
              'SR':sr,
              'DT':dt}
    models = {'conv1d':Conv1D(**params),
              'conv2d':Conv2D(**params),
              'lstm':  LSTM(**params)}
    assert model_type in models.keys(), '{} not an available model'.format(model_type)
    csv_path = os.path.join('/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/logs', '{}_history.csv'.format(model_t

    wav_paths = glob('{}/**'.format(src_root), recursive=True)
    wav_paths = [x.replace(os.sep, '/') for x in wav_paths if '.wav' in x]
    classes = sorted(os.listdir(args.src_root))
    le = LabelEncoder()
    le.fit(classes)
    labels = [os.path.split(x)[0].split('/')[-1] for x in wav_paths]
    labels = le.transform(labels)
    wav_train, wav_val, label_train, label_val = train_test_split(wav_paths,
                                                                  labels,
                                                                  test_size=0.1,
                                                                  random_state=0)

    assert len(label_train) >= args.batch_size, 'Number of train samples must be >= batch_size'
    if len(set(label_train)) != params['N_CLASSES']:
        warnings.warn('Found {}/{} classes in training data. Increase data size or change random_state.'.format(len(set(label_
    if len(set(label_val)) != params['N_CLASSES']:
        warnings.warn('Found {}/{} classes in validation data. Increase data size or change random_state.'.format(len(set(labe

    tg = DataGenerator(wav_train, label_train, sr, dt,
                       params['N_CLASSES'], batch_size=batch_size)
    vg = DataGenerator(wav_val, label_val, sr, dt,
                       params['N_CLASSES'], batch_size=batch_size)
    model = models[model_type]
    cp = ModelCheckpoint('/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/models/{}.h5'.format(model_type), monitor
                         save_best_only=True, save_weights_only=False,
                         mode='auto', save_freq='epoch', verbose=1)
    csv_logger = CSVLogger(csv_path, append=False)
    model.fit(tg, validation_data=vg,
              epochs=30, verbose=1,
              callbacks=[csv_logger, cp])

if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Audio Classification Training')
    parser.add_argument('--model_type', type=str, default='lstm',
                        help='model to run. i.e. conv1d, conv2d, lstm')
    parser.add_argument('--src_root', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/clean',
                        help='directory of audio files in total duration')
    parser.add_argument('--batch_size', type=int, default=16,
                        help='batch size')
    parser.add_argument('--delta_time', '-dt', type=float, default=1.0,
                        help='time in seconds to sample audio')
    parser.add_argument('--sample_rate', '-sr', type=int, default=16000,
                        help='sample rate of clean audio')
    args, _ = parser.parse_known_args()

    train(args)
```

Code du fichier «predict.py»

```
]: from tensorflow.keras.models import load_model
from clean import downsample_mono, envelope
from kapre.time_frequency import STFT, Magnitude, ApplyFilterbank, MagnitudeToDecibel
from sklearn.preprocessing import LabelEncoder
import numpy as np
from glob import glob
import argparse
import os
import pandas as pd
from tqdm import tqdm


def make_prediction(args):

    model = load_model(args.model_fn,
        custom_objects={'STFT':STFT,
                        'Magnitude':Magnitude,
                        'ApplyFilterbank':ApplyFilterbank,
                        'MagnitudeToDecibel':MagnitudeToDecibel})
    wav_paths = glob('{}/**'.format(args.src_dir), recursive=True)
    wav_paths = sorted([x.replace(os.sep, '/') for x in wav_paths if '.wav' in x])
    classes = sorted(os.listdir(args.src_dir))
    labels = [os.path.split(x)[0].split('/')[-1] for x in wav_paths]
    le = LabelEncoder()
    y_true = le.fit_transform(labels)
    results = []

    for z, wav_fn in tqdm(enumerate(wav_paths), total=len(wav_paths)):
        rate, wav = downsample_mono(wav_fn, args.sr)
        mask, env = envelope(wav, rate, threshold=args.threshold)
        clean_wav = wav[mask]
        step = int(args.sr*args.dt)
        batch = []

        for i in range(0, clean_wav.shape[0], step):
            sample = clean_wav[i:i+step]
            sample = sample.reshape(-1, 1)
            if sample.shape[0] < step:
                tmp = np.zeros(shape=(step, 1), dtype=np.float32)
                tmp[:sample.shape[0],:] = sample.flatten().reshape(-1, 1)
                sample = tmp
            batch.append(sample)
        X_batch = np.array(batch, dtype=np.float32)
        y_pred = model.predict(X_batch)
        y_mean = np.mean(y_pred, axis=0)
        y_pred = np.argmax(y_mean)
        real_class = os.path.dirname(wav_fn).split('/')[-1]
        print('Actual class: {}, Predicted class: {}'.format(real_class, classes[y_pred]))
        results.append(y_mean)

    np.save(os.path.join('/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/logs', args.pred_fn), np.array(results))


if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Audio Classification Training')
    parser.add_argument('--model_fn', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/models/lst
                        help='model file to make predictions')
    parser.add_argument('--pred_fn', type=str, default='y_pred',
                        help='fn to write predictions in logs dir')
    parser.add_argument('--src_dir', type=str, default='/content/drive/MyDrive/MEMOIRE/Audio-Classification-master/wavfiles',
                        help='directory containing wavfiles to predict')
    parser.add_argument('--dt', type=float, default=1.0,
                        help='time in seconds to sample audio')
    parser.add_argument('--sr', type=int, default=16000,
                        help='sample rate of clean audio')
    parser.add_argument('--threshold', type=str, default=20,
                        help='threshold magnitude for np.int16 dtype')
    args, _ = parser.parse_known_args()

    make_prediction(args)
```