

Reconnaissance de pièces de monnaie

Maxime Lallemand, Arthur Mahy, Astrid Mehuys,
Amelia Moore, Bruno Ploumhans, Gilles Prieur

26 avril 2018

Table des matières

Introduction	3
1 Histoire de l'intelligence artificielle	3
1.1 Introduction et définition	3
1.2 Quelques grands précurseurs de l'IA	4
1.2.1 Des scientifiques et des ingénieurs	4
1.2.2 Un économiste	6
1.2.3 Des influences littéraires	6
1.3 Apparition des premiers ordinateurs	6
1.3.1 Les travaux de Simon et Newell	8
1.4 Aujourd'hui	9
1.4.1 Le Japon et l'IA	9
1.4.2 L'IA dans tous les domaines	10
2 Techniques de reconnaissance de pièces	11
2.1 Machine learning	11
2.1.1 Principe général	11
2.1.2 Application détaillée du principe	12
2.1.3 Différents type de machine learning	14
2.1.4 Le deep learning	14
2.2 Transformée de Hough	16
2.2.1 Introduction	16
2.2.2 Principe	16
2.2.3 Première approche	17
2.2.4 Approche trigonométrique	18
2.3 Détection d'ensembles d'une même couleur	20
3 Un algorithme en profondeur	21
3.1 Généralités sur les graphes	21
3.2 Composantes connexes	21
3.3 Parcours d'un graphe	22
3.4 Première passe : ensembles de couleur	24
3.5 Deuxième passe : pièces de monnaie	26
3.6 Filtrage heuristique	27
3.7 Complexité	28

Conclusion	29
A Détection de la valeur des pièces	29
A.1 Détection de la figure des pièces	29
A.2 Problème de la rotation	30
A.3 Caveat emptor	30
B Code de l'algorithme	30
C Bibliographie	35
C.1 Livres	35
C.2 Articles	35
C.3 Internet	35

Introduction

Ce TFE porte sur la reconnaissance de pièces sur une image, à savoir la détection des ensembles de pixels correspondant à des pièces de monnaie. Comme il s'agit d'une forme d'intelligence artificielle, nous allons commencer par une définition et un historique de l'intelligence artificielle. Ensuite, nous verrons différentes techniques pouvant résoudre ce problème de façon plus ou moins efficace. Finalement, nous traiterons tous les aspects d'une de ces méthodes et nous irons jusqu'à l'implémenter afin de compter le nombre des pièces présentes sur une image.

1 Histoire de l'intelligence artificielle

1.1 Introduction et définition

Commençons tout d'abord par définir l'intelligence artificielle. L'intelligence artificielle (IA, ou AI en anglais pour *Artificial Intelligence*) a pour but de mimer ou d'imiter une certaine forme d'intelligence réelle. Elle essaie de se substituer à la logique de la prise de décision d'un humain en mettant en œuvre différentes techniques.

L'intelligence artificielle est définie comme *l'ensemble de théories et de techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence*. Elle correspond donc à un ensemble de concepts et de technologies plus qu'à une discipline autonome constituée.

La notion d'intelligence artificielle est née dans les années 50 et a été mise en avant par deux chercheurs. Tout d'abord, le mathématicien **Alan Turing** a débuté ses recherches dans ce domaine car il se demandait si une machine pouvait penser. Dans son article Computing Machinery and Intelligence, il cherche à savoir s'il est possible d'apporter une forme d'intelligence aux machines. Turing a analysé le problème et a proposé une expérience connue aujourd'hui sous le nom de *Test de Turing*. Dans ce test, un sujet interagit à l'aveugle avec un autre humain, puis avec une machine programmée pour formuler des réponses sensées. Si le sujet n'est pas capable de faire la différence, alors la machine a réussi le test et, selon l'auteur, peut véritablement être considérée comme *intelligente*. Ce test permet donc de définir si une machine est consciente.

Ensuite, **Warren Weaver** a publié en 1949 un mémorandum. Celui-ci se base sur la traduction automatique des langues, ce qui signifie qu'une machine est capable de faire des tâches dites humaines. Il est principalement connu comme un des pionniers de la traduction automatique.

L'intelligence artificielle en tant que domaine de recherche a été créée lors d'une conférence qui se déroulait sur le campus de Dartmouth College pendant l'été 1956. Les organisateurs de la conférence de Dartmouth avaient prévu que diverses questions autour de l'idée d'une machine pensante seraient abordées :

- Comment simuler la pensée et le langage au travers de règles formelles ?
- Comment faire penser un réseau de neurones ?
- Comment doter une machine de capacité d'apprentissage automatique ?
- Comment doter une machine de créativité ?

Néanmoins, les discussions ont été assez limitées et la conférence de Dartmouth n'a servi qu'à forger un embryon de communauté de recherche autour des problématiques évoquées. **John McCarthy** a proposé le terme *intelligence artificielle* pour désigner la nouvelle discipline qui a connu son âge d'or durant les 15 années suivantes.

Si l'intelligence artificielle s'est fortement développée aux États-Unis, c'est grâce à certaines personnes de renom dont John McCarthy à Stanford, **Marvin Minsky** au MIT, **Allen Newell** et **Hebert Simon** à Carnegie Mellon ainsi que **Donald Michie** à l'université d'Edimbourg. La plupart d'entre eux ont reçu le prix Turing, c'est-à-dire le prix le plus prestigieux en matière d'informatique. Leur travaux sur l'intelligence artificielle seront détaillés plus loin.

1.2 Quelques grands précurseurs de l'IA

1.2.1 Des scientifiques et des ingénieurs

Konrad Zuse (1910–1995) est un ingénieur allemand et l'un des fondateurs du calcul programmable. Entre 1936 et 1938, il développe Z1, le premier calculateur mécanique fonctionnant dans un moteur électrique. Pendant ce

temps, il travaille sur un autre projet : il crée en 1937 le premier calculateur électro-mécanique programmable binaire à virgule flottante, le Z3. Il est opérationnel en 1941 c'est-à-dire après seulement 4 ans de travail.

Claude Shannon (1916–2002) est un ingénieur électricien et un mathématicien américain. Il est l'un des pères fondateurs de la *Théorie de l'information*. Il donne son nom à un schéma qui prend le nom de *Schéma de Shannon*. Cet outil est utilisé en sciences humaines et en communication. En tant que mathématicien, il utilise l'algèbre de Boole dans le but de mettre en place des circuits de commutation. Il ajoute sa pierre à l'édifice en apportant un outil théorique aux concepteurs de circuits logiques. Il se tourne ensuite vers l'informatique. Il commence à créer une machine à jongler, une souris parcourant les labyrinthes, une machine qui résout le Rubik's cube et une autre machine permettant de jouer aux échecs.

Allen Newell (1902-1992) a été chercheur en informatique ainsi qu'en psychologie cognitive au sein de deux organismes : RAND Corporation et Carnegie Mellon's School of Computer Science aux Etats Unis. Il participe à l'élaboration de différents programmes : *Information Processing Language* en 1956 et à deux programmes en Intelligence Artificielle *The Logic Theory Machine* (1956) et le *General Problem Solver* (1957). Pour ce dernier programme, il travaille en collaboration avec Herbert Simon.

John Mc Marthy (1927-2011) est l'un des pionniers de l'Intelligence Artificielle avec Marvin Minsky. Il met l'accent sur la logique symbolique. En 1948, John McCarthy obtient un Bachelor of Science en mathématiques au California Institute of Technology, puis un doctorat à Princeton en 1951. Sa thèse porte sur un certain type d'équations aux dérivées partielles, mais son passage à Princeton lui fait rencontrer Minsky avec qui il se découvre une passion commune pour l'idée de machine pensante. A l'âge de 28 ans, il élabore un algorithme qui jouera un rôle majeur dans la programmation en Intelligence Artificielle. Cette algorithme est utilisé dans la plupart des programmes d'échecs. En 1958, il met au point le langage List Processing, LISP. Il crée à l'Université Standord le laboratoire de l'Intelligence Artificielle.

1.2.2 Un économiste

Herbert Simon (1916-2001) était avant tout un économiste et sociologue américain. Il a utilisé pour la première fois les ordinateurs et en a conclu que l'ordinateur avait deux rôles, à savoir la reproduction de la pensée humaine et la systématisation de celle-ci. Sa rencontre avec Allen Newell a été primordiale car il s'est penché sur les activités intellectuelles humaines et a découvert qu'elles pouvaient être automatisées. Ils ont conçu le *General Program Solver* en 1957 et ont développé l'intelligence artificielle de différentes manières.

1.2.3 Des influences littéraires

Isaac Asimov (1920-1992) est un écrivain américain. Il a écrit des œuvres de science-fiction ainsi que des livres de vulgarisation scientifique. Il a aussi écrit une série d'histoires (série des Robots) sur les rapports conflictuels entre l'homme et la machine ainsi que sur le rôle des robots.

Hubert Dreyfus (1929-2017) est un professeur américain de philosophie à l'université de Californie. Il s'intéresse à différents sujets comme la phénoménologie, l'existentialisme, la philosophie de la psychologie, la littérature et bien sûr l'intelligence artificielle. Il critique notamment fortement Allen Newell et Herbert Simon dans son livre *Alchemy and Artificial Intelligence*.

1.3 Apparition des premiers ordinateurs

Les années 1940 et 1950 voient l'apparition des premiers véritables ordinateurs. Ils sont Turing complets et électroniques, donc (relativement) rapides. Les entrées et sorties se font par cartes perforées et impression papier. La programmation de telles machines est compliquée et très longue. Il n'existe que très peu d'ordinateurs, uniquement dans quelques universités ou grandes entreprises. Ces machines couteuses servent surtout à faire des calculs massifs (statistiques pour l'état, calculs scientifiques pour la recherche nucléaire, calculs balistiques pour l'armée, etc.). Par exemple, l'*UNIVAC I* (Universel Automatic Computer) est installé en 1951 au bureau du recensement américain.

Notons bien que l'informatique n'existe pas encore. Les spécialistes des ordinateurs sont en effet essentiellement des mathématiciens ou des électro-niciens. Les langages de programmation au sens moderne du terme n'existent pas encore : le premier langage évolué, *FORTRAN* (FORmula TRANslator) ne verra le jour qu'en 1954. L'apparition des ordinateurs semble néanmoins rendre possible le rêve de l'IA. Les deux approches de l'IA vont émerger dans les années 1940 à savoir le connexionnisme et le cognitivisme.

Le **connexionnisme** modélise les phénomènes mentaux ou comportementaux comme des processus émergents de réseaux d'unités simples interconnectées. Le plus souvent les connexionnistes modélisent ces phénomènes à l'aide de réseaux de neurones.

Quant au **cognitivisme**, il est le courant de recherche scientifique endossant l'hypothèse selon laquelle la pensée est analogue à un processus de traitement de l'information. Il considère que la pensée peut être décrite à un niveau abstrait comme manipulation de symboles, indépendamment du support matériel de cette manipulation (cerveau, machine électronique, etc.). Elle est définie en lien avec l'intelligence artificielle comme une manipulation de symboles ou de représentations symboliques effectuée selon un ensemble de règles. Cette approche établit un lien entre la pensée et le langage (système de symboles).

Le 20ème siècle voit de fait apparaître plusieurs théories comme la cybernétique et le cognitivisme pour modéliser l'esprit, le cerveau et le mode de fonctionnement de la pensée. De surcroît, dans le contexte de la guerre froide, la traduction automatique du russe en anglais ou de l'anglais au russe est cruciale. En effet, en 1954, un premier programme, écrit à l'université de Georgetown permet de traduire plusieurs dizaines de phrases simples. Le programme utilise 250 mots et seulement 6 règles de grammaire et tourne sur un IBM 701.

Des crédits sont rapidement alloués aux recherches sur la traduction automatique (aussi bien aux USA qu'en URSS). Les premiers travaux visent la traduction directe, presque mot à mot, à l'aide de dictionnaires bilingues et de règles simples. Les problèmes de polysémie (amateur, blanc) ou d'homonymie (mousse, avocat, ...) apparaissent cependant très rapidement.

1.3.1 Les travaux de Simon et Newell

Allen Newell, après un Bachelor of Science en physique à Stanford, rejoint Princeton en 1949 pour mener une thèse en mathématiques. Durant ses études, il a été fortement influencé par le mathématicien hongrois Georges Polya (1887-1985), qui avait introduit la notion d'heuristique pour la résolution de problèmes. Une heuristique (du grec *Eurisko, Je trouve*) est une méthode empirique de résolution de problèmes, dont la validité ou l'efficacité n'est pas prouvée. Par exemple, protéger la reine aux échecs, choisir la caisse où la file est la plus courte, ... Trouvant finalement les mathématiques trop abstraites, Newell accepte en 1950 un poste à la RAND Corporation de Santa Monica, pour mener des travaux plus concrets, sur l'aéronautique de défense notamment.

Simon est également consultant à la RAND (Research AND Development). La RAND, créée pour étudier la mise au point d'un satellite artificiel, va peu à peu étendre ses travaux à l'informatique, l'économie et la géopolitique. Les idées de Simon et de Newell convergent : La rationalité limitée de Simon implique que la prise de décisions repose sur des procédures permettant de palier aux manques d'information en tenant compte du contexte. Pour Newell, ces procédures sont des heuristiques.

Simon et Newell considèrent que la condition nécessaire et suffisante pour qu'une machine puisse faire preuve d'intelligence est qu'elle soit un système physique de symboles. Néanmoins, ils mettent au coeur de leurs travaux la notion d'heuristique : être intelligent, c'est aussi être capable de construire des heuristiques, de les tester, de les faire évoluer. Aidés par un programmeur de la RAND, Cliff Shaw, ils développent Logic Theorist en 1956, un programme de démonstration automatique de théorèmes (voir l'article The logic theory machine : A Complex Information Processing System, 1956). Pour faciliter la programmation du Logic Theorist, Newell, Simon et Shaw développent le langage IPL (Information Processing Language) en 1956. Logic Theorist est considéré comme le premier programme informatique relevant du domaine de l'IA.

Newell et Simon y démontrent une série de théorèmes et envoient un article avec leur nouvelle démonstration au Journal of Symbolic Logic. L'article est cependant refusé au motif que ce théorème est déjà démontré depuis longtemps. Simon écrira néanmoins dans son autobiographie : « nous avons in-

venté un programme informatique capable de penser de façon non-numérique et, de ce fait, avons résolu le vénérable problème de l'âme et du corps, en expliquant comment un système composé de matière pouvait exhiber les propriétés de l'esprit » (Models of My Life, H. Simon, 1991).

Notons que les travaux de Newell et Simon illustrent également les apports croisés entre l'informatique et l'intelligence artificielle. On a d'une part le développement de l'informatique rendu possible par des expériences en IA, et d'autre part les problèmes posés par les expériences en IA qui conduisent à produire des outils servant au développement de l'informatique.

1.4 Aujourd’hui

1.4.1 Le Japon et l'IA

Bénéficiant des techniques de l'intelligence artificielle, la transmission des savoirs vit au début des années 90 une révolution. Les concepts sont bouleversés. Ils s'enrichissent de l'apport du couple IA-Robotique qui crée ses propres spécificités.

Si les robots font déjà partie intégrante du paysage industriel japonais, où ils servent à réaliser des tâches dangereuses ou nécessitant une haute précision de manière automatique, une évolution majeure du marché se dessine avec l'émergence de la robotique de service et, dans une moindre mesure, de la robotique de loisir. Les Japonais sont depuis très longtemps les champions de la robotique et on ne compte plus leurs inventions. Les recherches actuelles concernent tout particulièrement les robots *humanoïdes*, qui tentent d'imiter les humains et leurs capacités.

Par exemple, les robots humanoïdes présentent plus de difficultés de développement, puisqu'il s'agit d'imiter des facultés humaines, ce qui n'est pas simple. Le Japon rencontre un problème très particulier avec une natalité très faible et donc un vieillissement très important de la population qui nécessite de plus en plus d'aide. C'est pour cette raison que Honda continue de travailler sur ASIMO, le robot humanoïde qui devrait aider les personnes âgées à rester à leur domicile, en les assistant dans de nombreuses tâches qu'ils ne peuvent assumer seuls. Par ailleurs, des chercheurs de l'université de Tohoku, associés à des chercheurs français, consacrent leurs travaux à la

résolution d'un problème très important : permettre à un robot humanoïde, HRP-2, de se déplacer sur des surfaces irrégulières. En effet, ce type de robot ne peut se déplacer que sur une surface dure et stable, ce qui limite fortement son champ de manœuvre. La résolution de ce problème représenterait une avancée considérable en robotique et donc également en intelligence artificielle.

1.4.2 L'IA dans tous les domaines

D'année en année, l'IA a été implantée dans de plus en plus de domaines d'application et le développement de celle-ci est au centre de beaucoup de recherches actuelles. Le développement des technologies informatiques et des techniques algorithmiques ont eu une telle croissance ces dernières années que les machines dépassent l'homme dans plusieurs domaine comme le jeu d'échecs ou encore le jeu de go. Toutefois, les sujets concernés par l'IA ont varié au cours du temps. Par exemple, dans les années 1950, la recherche d'un itinéraire était considéré comme un problème d'intelligence artificielle, alors qu'aujourd'hui, il existe différentes applications qui sur base d'algorithmes résolvent ces questions de recherche d'itinéraires et il ne s'agit plus d'intelligence artificielle.

Aujourd'hui toutes les grandes entreprises dans le monde de l'informatique (Google, Microsoft, Apple, IBM ou Facebook) portent une attention toute particulière aux problématiques de l'intelligence artificielle en tentant de l'appliquer à différents domaines précis. Chacun met ainsi en place des réseaux de neurones artificiels constitués de serveurs afin de traiter de lourds calculs au sein de gigantesques bases de données.

L'intelligence artificielle est aussi de plus en plus utilisée pour être au service des humains. Par exemple, la vision artificielle comme nous l'expliquerons dans ce travail, permet à la machine de déterminer précisément le contenu d'une image comme des pièces de monnaies pour ensuite la classer automatiquement selon l'objet, la couleur ou le visage repéré. La reconnaissance vocale a également le vent en poupe avec des assistants virtuels capables de transcrire les propos formulés en langage naturel puis de traiter les requêtes soit en répondant directement via une synthèse vocale, soit avec une traduction instantanée ou encore en effectuant une requête relative à la

commande. Apple était en la matière un des précurseurs avec son application Siri.

Nous pouvons donc affirmer que l'IA qui était au départ très restreinte a un potentiel infini. De jour en jour, les recherches progressent et les résultats de ce que l'intelligence artificielle est capable de réaliser croissent de manière exponentielle.

2 Techniques de reconnaissance de pièces

Après cet aperçu de l'intelligence artificielle et de son histoire, penchons-nous sur 3 méthodes permettant de reconnaître des pièces sur une image.

2.1 Machine learning

Premièrement, nous pouvons utiliser le machine learning. Le *machine learning* ou en bon français *apprentissage automatique* est l'élaboration d'algorithmes capable d'apprendre en étudiant des exemples. Il est la branche principale de l'intelligence artificielle. Le machine learning est particulièrement bien adapté à la reconnaissance d'image et ainsi à la détection de pièces de monnaie.

2.1.1 Principe général

Le machine learning est constitué de deux étapes principales.

1. Phase d'apprentissage : l'algorithme traite des milliers d'information d'une base de donnée. Il reçoit une entrée X dont il connaît la sortie Y . Lors de cette phase, l'algorithme va analyser les corrélations qui lient les entrées X aux sorties Y . Il va en déduire de lui-même les caractéristiques importantes de cette liaison.
2. Phase de prédiction : l'algorithme reçoit une nouvelle entrée X , il la compare à celles qui l'a analysées et ainsi, il doit prédire la sortie Y .

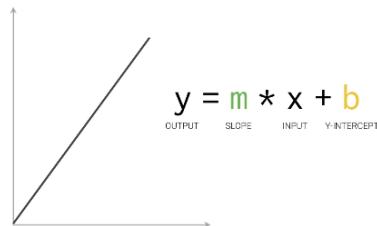
Dans notre cas, l'entrée X est l'image d'une pièce de monnaie et la sortie Y la valeur de la pièce. L'algorithme va s'entrainer à reconnaître une pièce de monnaie et à les distinguer les unes des autres en analysant des milliers d'images d'une base de donnée. Ainsi, il sera capable face à une nouvelle image d'en déduire la valeur de la pièce.

2.1.2 Application détaillée du principe

Le machine learning peut être détaillé en 7 étapes et ainsi être mieux expliqué.

1. Rassemblement de données : actuellement, il existe des milliers de base de donnée regroupant chacune d'elle des milliers d'images. Cette première étape consiste à rassembler les images qui nous intéressent.
2. Préparation des données : c'est l'encodage et la détermination de l'entrée X et de la sortie Y. On doit imaginer pouvoir interpréter les données par un repère et des axes X et Y.
3. Choix du modèle : le modèle est un algorithme mathématique avec un certain nombre de paramètres qui doivent être appris à partir de données. La partie essentielle du machine Learning est de trouver un modèle qui puisse correspondre au mieux à ces données.
4. Entrainement : le but de l'entraînement est de créer un modèle précis qui puisse répondre à notre question de façon fiable. On a besoin de données pour entraîner le module. On va prendre des caractéristiques de l'objet que l'on veut reconnaître (taille, forme couleur, ...).

Training



On place les paramètres sur droite $d \equiv y = mx + b$ où

$$\begin{cases} x \text{ est l'entrée} \\ m \text{ est la pente} \\ b \text{ est l'ordonnée à l'origine} \\ y \text{ est la valeur de } d \text{ en un point précis, c'est la sortie} \end{cases}$$

Pour modifier la position de la droite, on joue sur les paramètres m et b (m et b sont les différentes valeurs des caractéristiques). Pour que le système puisse assimiler toute l'information, on utilise des matrices. La matrice des m est la matrice des poids (*weights*) et la matrice des b est la matrice des biais (*biases*).

Training

$$\begin{aligned} \text{WEIGHTS} &= \begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \\ m_{3,1} & m_{3,2} \end{bmatrix} \\ \text{BIASES} &= \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \end{bmatrix} \end{aligned}$$

Il faut un système avec tous les paramètres m et tous les paramètres b et le but est de, en faisant varier les paramètres, entraîner le système à reconnaître et à faire des prédictions de plus en plus fiable.

5. Evaluation : après l'entraînement, une évaluation du système avec des nouvelles données nous aide à voir le pourcentage de fiabilité du module.
6. Paramètre et hyperparamètre : ce sont deux types de paramètre. Ces paramètres expriment les propriétés de «niveau supérieur» du modèle telles que la complexité et la rapidité d'apprentissage. Les hyperparamètres, ne pouvant pas être directement appris sur base de l'entraînement, sont généralement fixés avant-même que le processus d'apprentissage commence. De manière générale, on fixe la valeur de l'hyperparamètre, après avoir testé différentes valeurs de celui-ci et en décidant lesquels fonctionnent le mieux.
7. Prédiction : il ne reste plus qu'à faire tourner le système avec les données qui nous intéressent pour obtenir la reconnaissance des différentes pièces monnaies.

2.1.3 Différents type de machine learning

2.1.3.1 Apprentissage supervisé

L'apprentissage supervisé est la forme de machine learning la plus répandue. Elle consiste à entraîner l'algorithme sur des entrées X dont on connaît les sorties Y . Il y a eu donc au préalable une intervention d'un expert qui a lié les entrées X et les sorties Y entre elles.

2.1.3.2 Apprentissage non-supervisé

L'apprentissage non-supervisé consiste à faire aucun lien entre les entrées et sorties. L'algorithme doit trouver par lui-même la structure cachée entre les données. Ainsi, aucun expert n'est requis. Et la phase de prédiction est dès lors impossible : l'algorithme classe uniquement les données.

2.1.3.3 Apprentissage par renforcement

Par ce dernier apprentissage, l'algorithme modifie son comportement au vu des observations et des résultats qu'il tire de ses expériences. Chaque nouvelle action de l'algorithme lui permet de connaître mieux son environnement et ainsi d'optimiser davantage son apprentissage.

2.1.4 Le deep learning

2.1.4.1 Limitations du machine learning conventionnel

Dans des problèmes plus compliqués, on peut avoir plus d'une donnée en entrée x , ce qui donne une relation graphique de plus en plus complexe au fur et à mesure que l'on ajoute des entrées. On se rend rapidement compte que la simple droite ne pourra pas répondre à nos attentes en reconnaissance d'image dans beaucoup de situations. Une image est composée de millions de pixels et en machine learning il devient rapidement compliqué de gérer une telle quantité de données en entrée. C'est ici qu'intervient le deep learning.

2.1.4.2 Des neurones en réseau

Le deep learning est une façon de faire du machine learning. Il utilise des neurones artificiels pour trouver la sortie y . Un neurone (artificiel) est une construction mathématique qui va mettre en relation des entrées X avec une sortie Y . En ce sens, c'est juste une fonction à plusieurs variables.

Exemple On pourrait par exemple avoir un neurone qui, pour une série de n entrées x_i et de n poids w_i (avec $i \in [1, n]$), calculerait la somme $s = \sum_{i=1}^n x_i w_i$. Selon que cette somme serait $\geq S$ ou $< S$ où S est un seuil fixé, le neurone renverrait 1 ou 0 respectivement. On pourrait ainsi créer un système binaire.

Ce n'est bien sûr qu'un exemple, mais il montre bien la polyvalence d'un neurone. Dans des situations plus complexes, on peut combiner les neurones et créer alors un réseau de neurones. Pour augmenter la fiabilité des résultats, il faut passer par une phase d'entraînement avant d'entrer dans la phase de prédiction, comme dans le machine learning. Pour adapter au mieux la justesse des réponses, l'algorithme joue sur les poids et les seuils.

Un des gros inconvénient du deep learning est qu'au plus on ajoute de couches de neurones, au plus la phase d'entraînement est longue et au plus il devient difficile de trouver les bonnes valeurs de poids et seuil.

2.1.4.3 Application à la reconnaissance d'images

Sur une image brute, il y a beaucoup de pixels. Il est donc très difficile de pouvoir établir un réseau de neurones. Pour pouvoir reconnaître des images, il est plus simple de faire abstraction de l'image et de créer un algorithme intermédiaire qui aura pour but d'analyser toutes les caractéristiques intéressantes. Ce sont ensuite ces caractéristiques qui seront données à un réseau de neurones simple pour faire la reconnaissance. Cette approche est plus efficace. L'algorithme filtre déjà une partie de l'information intéressante pour la reconnaissance. Le seul problème de cette méthode est que la qualité de la reconnaissance va fortement dépendre du choix des caractéristiques par l'algorithme et du réseau de neurones.

Cependant, on peut faire un réseau profond, c'est un très gros réseau de neurones auxquels on peut directement donner l'image brute. Si l'on parvient à correctement entraîner ce réseau, on se rend compte qu'après l'entraînement, le réseau a lui-même découvert les caractéristiques à sélectionner dans les images. Il y a une architecture particulière à respecter pour que le réseau soit le plus efficace possible. Cette méthode n'a pu se développer fortement ces dernières années que grâce à la grande disponibilité des données. A l'heure actuelle, on peut faire des réseaux de deep learning de plus d'une centaine

de couches constituées de plusieurs millions de neurones. Les performances en reconnaissance d'images sont assez spectaculaires.

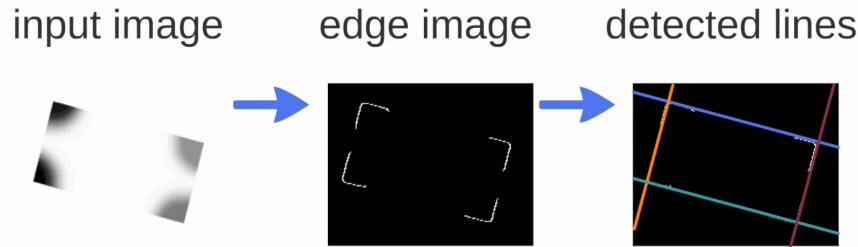
2.2 Transformée de Hough

2.2.1 Introduction

Après avoir vu ce qu'il est possible de faire dans le domaine du machine learning, penchons nous maintenant sur la *transformée de Hough*, qui est un algorithme traitant exclusivement de la reconnaissance d'images. La transformée (ou transformation) de Hough est une technique de reconnaissance de formes inventée en 1962 par **Paul Hough**. Son application la plus simple permet de détecter les lignes/droites présentes dans une image mais des modifications peuvent être apportées à cette technique pour détecter d'autres formes géométriques (cercles, ellipses...) : c'est la transformée généralisée de Hough développée par Richard Duda et Peter Hart en 1972. Cette technique est devenue un outil standard dans le domaine de la vision artificielle, et elle est notamment utilisée pour la détection de routes dans les images prises par satellite, pour la lecture de codes barres ou encore dans le logiciel *OpenCV*. Le principe fondamental étant le même que pour la reconnaissance de cercles, nous allons développer ici la reconnaissance de droites dans une image, ce qui rend l'algorithme plus simple à comprendre sans toutefois masquer sa puissance.

2.2.2 Principe

Le problème posé est donc celui de la recherche et de la détection de lignes qui seraient éventuellement présentes dans une image à analyser. Le principe de la transformée de Hough est qu'il existe un nombre infini de lignes qui passent par un point, dont la seule différence est la pente a , l'orientation, l'angle... Le but de la transformée est de déterminer lesquelles de ces lignes sont les plus présentes dans l'image analysée.

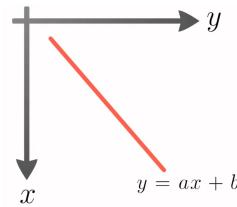


2.2.3 Première approche

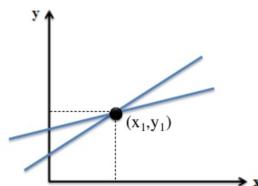
2.2.3.1 Représentation d'une droite

La formule la plus simple représentant une droite dans le plan est son équation cartésienne et réduite $y = ax + b$ (avec $a, b \in \mathbb{R}$) où

$$\begin{cases} a \text{ est la pente de la droite} \\ b \text{ est l'ordonnée à l'origine} \end{cases}$$



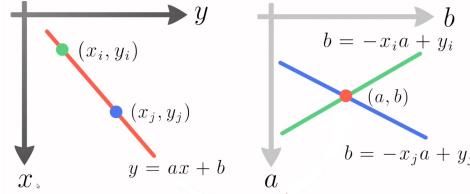
Toutes les droites passant par un point de coordonnées (x_1, y_1) fixées ont la forme $y_1 = ax_1 + b$ pour différentes valeurs de a et b .



2.2.3.2 Principe détaillé

On va changer le repère en remplaçant x par a et y par b . Chaque droite dans l'espace (x, y) sera transformée en un point dans l'espace (a, b) , espace

de Hough. Inversément, chaque point dans l'espace (x, y) sera transformé en une droite d'équation $b = -ax + y$ dans l'espace de Hough.



Pour un point A , toutes les droites passant par ce point correspondent à une seule droite a dans l'espace de Hough. Pour un point B , toutes les droites passant par ce point correspondent à une seule droite b dans l'espace de Hough. Ces 2 faisceaux de droites dans l'espace (x, y) ont en commun l'unique droite qui relie A et B . Cette droite correspond au point qui est l'intersubsection des 2 droites a et b . Finalement, cette intersection contient les paramètres de la droite recherchée.

Tous les points situés sur une même droite c dans l'espace (x, y) sont représentés par des droites passant toutes par un même point C dans l'espace de Hough. Les coordonnées de ce point $C(x_c, y_c)$ contiennent les paramètres recherchés de la droite $c \equiv y = x_c x + y_c$. Afin de déterminer quelles droites sont les plus présentes, on utilise un accumulateur où chaque point peut voter pour une droite particulière et les droites recevant le plus de votes sont conservées. On ne va cependant pas développer ce principe ici.

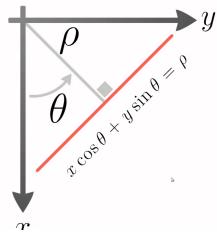
Remarquons que la représentation $y = ax + b$ pose un problème pour les droites verticales. Voilà pourquoi la représentation suivante, qui règle ce problème, est la plus utilisée.

2.2.4 Approche trigonométrique

2.2.4.1 Représentation polaire d'une droite

Une droite peut aussi être représentée par la formule $y \sin \theta + x \cos \theta = \rho$ (avec $\theta \in [0; \pi/2]$ et $\rho \in [0; d]$ où d est la longueur de la diagonale de l'image) où

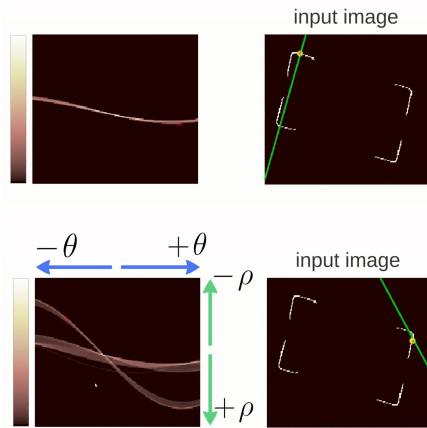
$$\begin{cases} \theta \text{ est l'angle entre l'axe } x \text{ et le vecteur } \vec{\rho} \\ \rho \text{ est la distance la plus courte entre l'origine du repère et la droite} \end{cases}$$

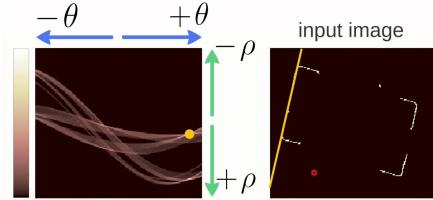


2.2.4.2 Principe détaillé

On change également le repère en remplaçant x par θ et y par ρ . Chaque droite dans l'espace (x, y) est alors transformée en un point dans l'espace (ρ, θ) . Chaque point dans l'espace (x, y) est alors transformé en une courbe, une sinusoïde dans l'espace (ρ, θ) . Les points d'intersubsection des sinusoïdes dans l'espace (ρ, θ) sont utilisés pour trouver les droites dans l'espace (x, y) .

La transformée de Hough fait ensuite appel à un algorithme qui regarde tous les points et imagine toutes les valeurs possibles pour ρ et θ . On observe, dans l'espace (ρ, θ) , des pics pour les valeurs qui reviennent le plus souvent et l'algorithme va définir les droites dans l'espace (x, y) et donc les lignes détectées dans l'image.





2.3 Détection d'ensembles d'une même couleur

Troisièmeme, nous pouvons exploiter les propriétés du problème afin de trouver un algorithme spécialisé certes mais relativement simple et très efficace. Remarquons que pour détecter des pièces de monnaie sur une image, il est possible d'exploiter le contraste entre les pièces elles-mêmes et le matériau sur lequel elles sont placées. Dans l'image suivante, par exemple, les pièces sont jaune ou rouge foncé alors que le fond est blanc.



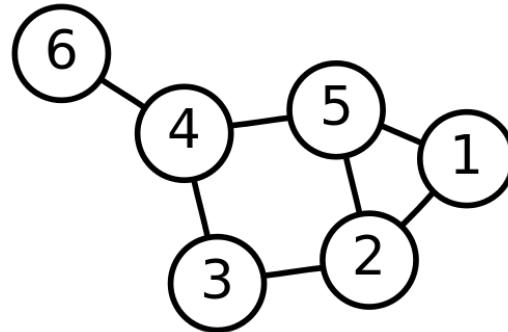
Nous allons donc développer un algorithme permettant d'utiliser la couleur du fond pour isoler les pièces. Et nous allons l'appliquer à cette image afin d'en extraire les pièces.

3 Un algorithme en profondeur

L'algorithme se décompose en 2 passes principales, suivies de 2 filtres heuristiques. Avant de parler de ces éléments, nous devons introduire du vocabulaire sur les graphes et présenter quelques algorithmes simples.

3.1 Généralités sur les graphes

Un graphe est un ensemble de points appelés *sommets* (*vertices* en anglais) et de connections appelées *arêtes* (*edges* en anglais). Voici par exemple un graphe de 6 sommets et de 7 arêtes :



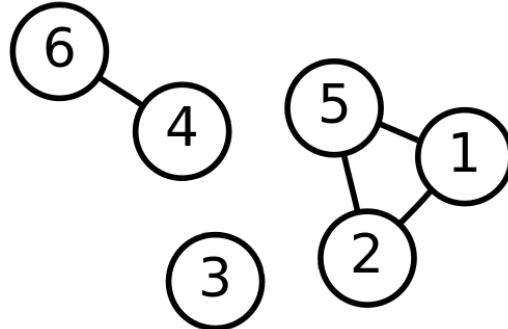
Dans ce cas-ci, les points sont 1, 2, 3, 4, 5, 6 et les connections sont $(1, 2), (1, 5), (2, 3), (2, 5), (3, 4), (4, 5), (4, 6)$.

D'un point de vue mathématique, le graphe G peut se définir ainsi :

$$\begin{aligned}G &= (V, E), \\V &= \{1, 2, 3, 4, 5, 6\}, \\E &= \{(1, 2), (1, 5), (2, 3), (2, 5), (3, 4), (4, 5), (4, 6)\}.\end{aligned}$$

3.2 Composantes connexes

Le graphe ci-dessus est un exemple de graphe *connexe*, c'est-à-dire un graph dont tout point est relié à tout point. Il existe aussi des graphes non connexes. Les différentes parties connexes du graphe sont alors appelées des *composantes connexes*. Le graphe ci-dessous est ainsi composé de 3 composantes connexes. Il est clair que le graphe est non connexe car le point 1 n'est par exemple pas relié au point 3.



3.3 Parcours d'un graphe

Voyons maintenant comment on peut détecter les différentes composantes connexes d'un graphe. Pour ce faire, commençons par remarquer la transitivité de la définition d'une composante connexe. En effet, s'il existe un chemin de u à v et de u à w , il existe forcément un chemin allant de v à w et inversément. Il suffit en effet de partir de v , de se rendre à u et de rejoindre w depuis u . Cette propriété peut sembler anodine, mais elle affirme que pour détecter une composante connexe, il suffit de partir d'un sommet quelconque et de parcourir le graphe en catalogant tous les sommets atteignables.

Un algorithme classique pour parcourir les graphes est le *depth-first search* (DFS) ou encore *parcours en profondeur*, appelé ainsi parce qu'il va aussi loin qu'il peut avant de revenir en arrière.

On part d'un sommet u . Si le sommet u n'a pas encore été visité, on le marque comme visité. Ensuite, pour chaque sommet v adjacent à u , on reprend l'algorithme récursivement. En C++, cela donne :

```

1 const int N; // nombre de sommets (numérotés de 0 à N - 1)
2
3 // Pour chaque sommet, on garde une liste des sommets adjacents
4 vector<int> adjacent_vertices[N];
5
6 // Au départ, aucun des sommets n'a été visité
7 bool visited[N] = {false};
8
9 void dfs(int u) {
10    if (!visited[u]) { // Si u n'a pas été visité
11        visited[u] = true;
12        // Pour chaque sommet v adjacent

```

```

13     for( int v : adjacent_vertices[u] ) {
14         dfs(v); // On continue récursivement
15     }
16
17     // Cataloguer u d'une façon ou d'une autre
18     /* ... */
19 }
20 }
```

Listing 1 – DFS

Cet algorithme ne parcourt qu'une seule composante connexe. Afin de détecter toutes les composantes connexes, on peut essayer de lancer l'algorithme depuis chaque sommet. Si l'algorithme n'a pas encore été lancé depuis un sommet u , c'est que u appartient à une nouvelle composante connexe.

```

1 // Pour chaque sommet u
2 for( int u = 0; u < N; ++u) {
3     if( !visited[u] ) {
4         // Nouvelle composante connexe
5         // donc on appelle l'algorithme.
6         dfs(u);
7     }
8 }
```

Listing 2 – Composantes connexes

Ce code parcourt bien tous les sommets mais il ne permet toujours pas de déterminer la composante connexe d'un sommet donné. Résolvons ce problème : plutôt que d'enregistrer si un sommet a été visité ou non, on peut directement enregistrer à quelle composante connexe il appartient.

```

1 // Au départ, aucun des sommets n'a été visité (cc[u] = -1 ∀u)
2 int cc[N];
3
4 void dfs( int u, int id) {
5     if( cc[u] == -1) { // Si u n'a pas été assigné
6         cc[u] = id;
7         // Pour chaque sommet v adjacent
8         for( int v : adjacent_vertices[u] ) {
9             dfs(v, id); // On continue récursivement
10        }
11    }
12 }
```

```

13 // On initialise à -1
14 for( int u = 0; u < N; ++u) cc[u] = -1;
15
16 // Composante connexe actuelle
17 int id = 0;
18 // Pour chaque sommet u
19 for( int u = 0; u < N; ++u) {
20     if(cc[u] == -1) {
21         // Nouvelle composante connexe
22         // donc on appelle l'algorithme.
23         dfs(u, id++);
24     }
25 }
26 }
```

Listing 3 – Composantes connexes

3.4 Première passe : ensembles de couleur

Appliquons maintenant ces idées à l'image. Pour ce faire, il faut savoir qu'une image est essentiellement une matrice de pixels, chacun composé de trois composantes (r, g, b) (red, green, blue) avec r, g, b discrets $\in [0, 255]$. Nous allons donc utiliser la structure de graphe implicite de cette matrice en prenant comme sommets les différents pixels et en stipulant qu'un pixel (i, j) est adjacent aux pixels $(i, j + 1), (i, j - 1), (i + 1, j), (i - 1, j)$. Si nous calculons maintenant les composantes connexes de ce graphe, nous obtiendrons une unique composante contenant tout le graphe. C'est normal, le graphe est connexe d'après notre définition et nous n'avons d'ailleurs pas pris en compte les couleurs des pixels. Afin de pouvoir détecter des composantes connexes, nous aimerais isoler les zones de couleurs différentes. Nous devons donc connecter les pixels adjacents de couleurs très proches par des arêtes.

Nous cherchons donc une fonction f prenant deux pixels $p(r, g, b)$ et $p'(r', g', b')$ comme arguments et renvoyant 1 ou 0 selon que leurs couleurs sont proches ou non. C'est l'étape cruciale de l'algorithme. En effet, une fonction mal choisie ne permettra pas de détecter de bonnes composantes connexes. Cependant, comme nous allons le voir dans ce qui va suivre, nous voulons uniquement détecter la composante connexe la plus grande, c'est-à-dire celle du fond sur lequel les pièces reposent. Nous allons donc nous contenter d'une

fonction prenant la plus petite différence absolue entre chaque paire de composantes correspondantes. En pratique, cela donne :

$$f(p, p') = \begin{cases} 1 & \text{si } \max(|r - r'|, |g - g'|, |b - b'|) \leq s \\ 0 & \text{sinon} \end{cases}$$

où s est un paramètre fixé au début de l'algorithme. La lettre s a été choisie pour le terme anglais *sharpness* et nous allons l'appeler coefficient de finesse. Pour de faibles valeurs de s , beaucoup de composantes connexes seront détectées. Si s vaut 0, chaque pixel sera une composante connexe. Pour de grandes valeurs de s , moins de composantes connexes seront identifiées. Si s vaut 255 ou plus, l'image ne sera qu'une grande composante connexe.

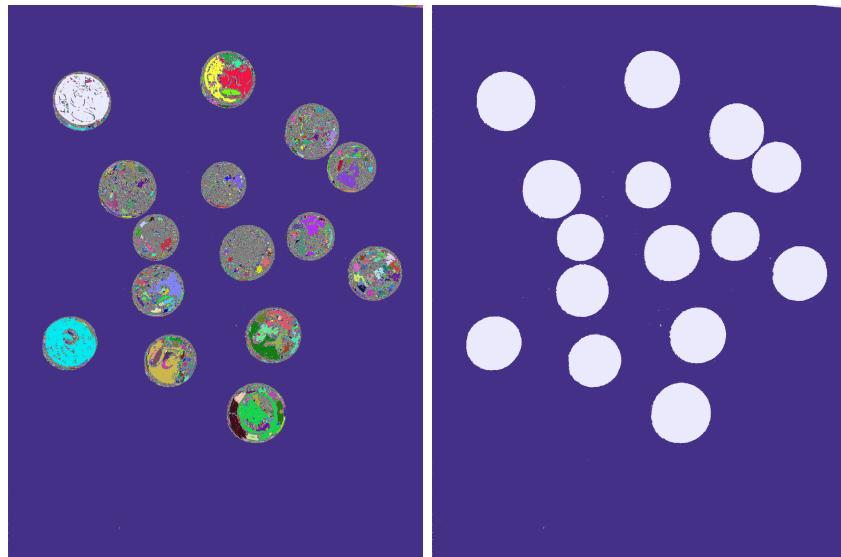
La valeur à choisir en pratique dépend principalement de la résolution de l'image. En effet, les images à très haute résolution comme celle des pièces de monnaie plus haut (environ 3000x4000 pixels) nécessitent de très faibles valeurs de s à cause de la diffusion de la lumière alors qu'il convient de choisir des valeurs plus hautes pour des résolutions plus faible afin de s'assurer que le fond est bien une unique composante connexe. Voyons donc ce qui se passe pour $s = 2$ (des couleurs aléatoires ont été choisies pour faciliter la visualisation des composantes connexes) :



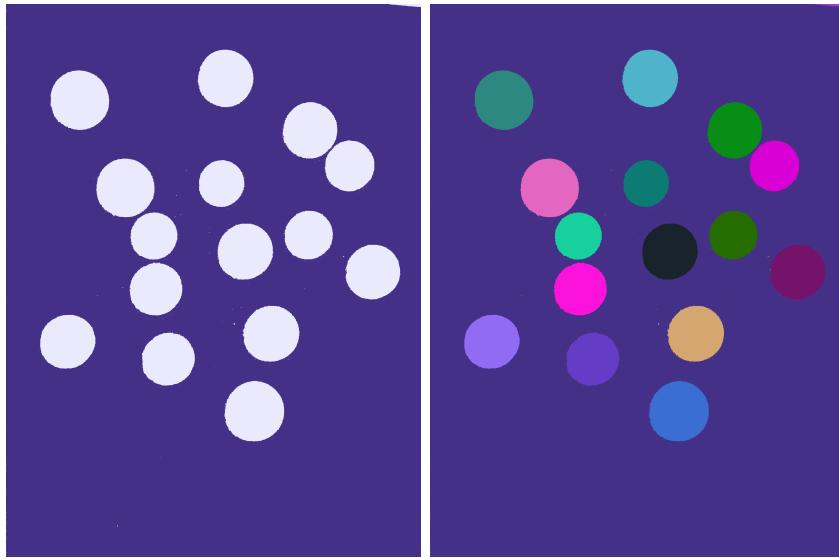
Remarquons que le fond de l'image est correctement détecté, et c'est tout ce qui importe. Nous allons maintenant voir comment exploiter cette information pour compter le nombre de pièces.

3.5 Deuxième passe : pièces de monnaie

Une fois que nous avons réussi à détecter le fond, nous pouvons séparer le graphe en deux parties : le fond et les pièces. Ceci peut être fait avec un algorithme trivial que nous ne détaillerons pas ici. Voici le résultat que l'on obtient :



Maintenant, il ne reste plus qu'à détecter les pièces individuelles. Pour ce faire, il suffit calculer les composantes connexes du sous-graphe contenant les pièces, toujours avec un DFS. Pour ce faire, nous connectons les pixels adjacents s'ils appartiennent au même sous-graphe. Nous ne détaillons pas le code mais une version commentée se trouve en annexe. Nous n'accordons alors plus aucune importance à la couleur, la distinction entre le fond et les pièces ayant déjà été faite par la première passe. Le résultat obtenu est alors le suivant. De nouveau, des couleurs générées aléatoirement ont été utilisées pour aider à la visualisation :



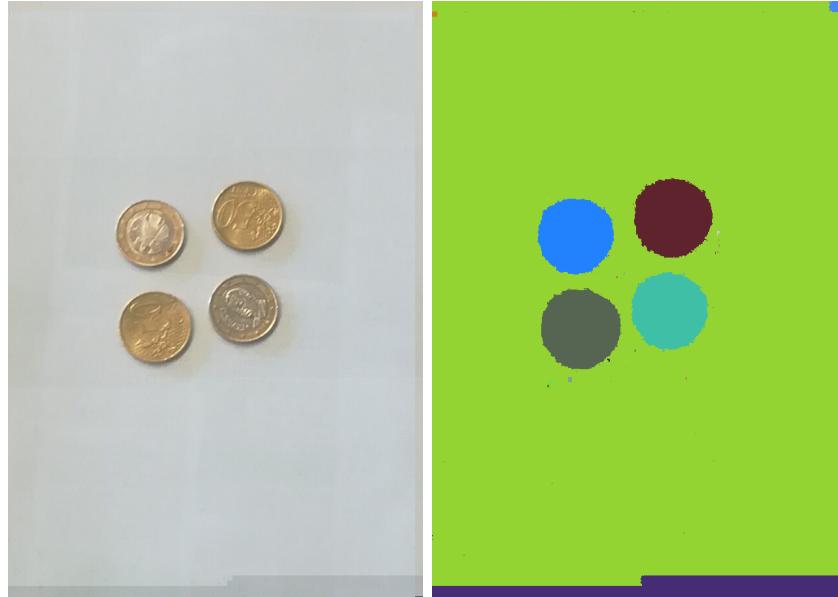
C'est bien le résultat attendu, il suffit maintenant de compter le nombre de composantes connexes détectées parmi les pièces. Sur cette image, la réponse est évidemment... 1202 !? L'image a l'air correcte à priori, comment se fait-il que l'algorithme détecte 1202 pièces et non 15 ? Une analyse détaillée de l'image révèle de toutes petites taches de couleur et lorsqu'on demande au programme d'afficher la taille des différentes composantes connexes on se rend compte que certaines ont une taille de 20 pixels ou moins.

3.6 Filtrage heuristique

Afin de déterminer quelles composantes connexes correspondent ou non à des pièces de monnaie, nous allons appliquer deux heuristiques. Premièrement, nous pouvons éliminer les composantes connexes d'une taille en pixels inférieure à $t\%$ de l'image, avec t que nous appellerons coefficient de tolérance. Une valeur de $t = 0.1$ fonctionne assez bien. Cela peut sembler peu mais pour une image de 12000000 pixels, cela veut dire que les pièces doivent faire au moins 1200 pixels, ce qui n'est pas négligeable. Avec cette modification, notre algorithme détecte correctement les 15 pièces de monnaie.

Deuxièmement, nous pouvons éliminer les composantes connexes qui n'ont pas la forme d'une ellipse. Bien qu'il n'ait pas fallu recourir à cette option pour cette image-ci, cela se révèle souvent nécessaire en pratique. Nous

avons par exemple essayé d'appliquer notre algorithme à l'image ci-dessous. Le résultat est assez clair :



Une approximation qui fonctionne assez bien en pratique est de calculer les valeurs minimale et maximale de x et de y pour tous les points appartenant à une composante connexe donnée. Nous pouvons alors calculer une approximation de l'aire sur base de ces valeurs en partant du principe que la forme repérée est une ellipse :

$$A = \frac{\pi}{4} \cdot |x_{min} - x_{max}| \cdot |y_{min} - y_{max}|$$

. Une fois cette aire calculée, nous pouvons comparer cette valeur à la taille en pixels de la composante connexe et nous assurer que l'erreur est bien inférieure à un certain pourcentage. Nous avons choisi 20% mais une valeur plus faible aurait sans doute fait l'affaire.

3.7 Complexité

La complexité d'un algorithme est définie comme la quantité de ressources (temps ou mémoire) nécessaire à son exécution. La notation $\mathcal{O}(\dots)$ exprime la complexité asymptotique d'un algorithme dans le pire cas, où \dots est fonction d'un certain nombre de paramètres.

Si nous prenons w la largeur de l'image à analyser et h sa hauteur, notre algorithme tourne en un temps $\mathcal{O}(wh)$. Posons $n = wh$ par simplicité. Chacun des DFS prend $\mathcal{O}(n+4n) = \mathcal{O}(n)$. En effet, chaque sommet est visité une fois ($\mathcal{O}(n)$) et pour chaque sommet il faut tester les 4 potentiels sommets adjacents ($\mathcal{O}(4n)$). Les heuristiques sont aussi en $\mathcal{O}(n)$ car elles nécessitent de parcourir tous les pixels exactement une fois. Notre algorithme tourne donc en $\mathcal{O}(n)$ et il est trivial de constater qu'il utilise $\mathcal{O}(n)$ en mémoire. Il est d'ailleurs impossible d'obtenir une complexité inférieure à partir du moment où on souhaite lire toute l'image, car la lecture prend au moins $\mathcal{O}(n)$ en temps et en mémoire. La partie la plus lente de notre programme est sans doute le chargement et la sauvegarde des images, car la bibliothèque utilisée pour ça doit suivre des algorithmes bien plus compliqués pour décoder et surtout encoder efficacement l'image. Pour garantir une complexité de $\mathcal{O}(n)$, nous pourrions utiliser un format de fichier non compressé comme les *bitmaps* (.bmp) mais cela ne semble pas nécessaire en pratique et nous ne l'avons donc pas fait.

Conclusion

Nous avons donc parlé de l'histoire de l'intelligence artificielle. Nous avons ensuite considéré plusieurs techniques pour résoudre le problème de la reconnaissance de pièces de monnaie. Finalement, nous avons développé un algorithme en particulier et nous avons déterminé que cet algorithme est très efficace : il a un temps d'exécution et une utilisation de mémoire optimaux et produit de bons résultats. Cerise sur le gâteau, nous proposons en annexe une piste d'amélioration afin de compter la valeur exacte de chacune des pièce.

A Détection de la valeur des pièces

A.1 Détection de la figure des pièces

Afin de détecter la valeur exacte de chaque pièce, nous pourrions maintenir une liste des différents symboles apparaissant sur les pièces de monnaie couramment utilisées dans le monde et voir duquel de ces symboles s'approchent le plus ceux des pièces détectées. Pour ce faire, nous pourrions prendre les valeurs des pixels le long de cercles pris à intervalles réguliers. Par exemple, nous pourrions décider de prendre 10 cercles concentriques de

rayons d'une longueur de 10%, 20%, ..., 100% du rayon de chacune des pièces et de parcourir les pixels qui s'y trouvent. Nous pourrions ainsi comparer le signal obtenu et déterminer à quel symbole correspond une pièce de monnaie donnée en utilisant par exemple la méthode des moindres carrés. Malheureusement, cet algorithme ne tient pas compte de la rotation des pièces et garder traces de toutes les pièces de monnaie en usage dans tout le monde à tous les angles possibles ne semble pas faisable.

A.2 Problème de la rotation

Il existe une solution théorique relativement simple à ce problème. Il suffit en effet d'appliquer la transformée de Fourier à chacun des signaux et d'éliminer ensuite le déphasage afin d'aligner les différents signaux obtenus avant de reprendre l'algorithme.

A.3 Caveat emptor

Estimant que ces améliorations sortent de l'ampleur de notre TFE, nous n'avons pas creusé ces idées et nous ne les avons pas implémentées. Nous n'avons donc pas pu vérifier l'efficacité de l'algorithme suggéré ci-dessus et nous sommes peut-être passés à côté de certains problèmes essentiels avec cette nouvelle méthode. Bien qu'elle semble fonctionner d'un point de vue théorique, elle est donc à prendre avec des pincettes.

B Code de l'algorithme

Voici le code que nous avons utilisé afin de compter le nombre de pièces de monnaie et de générer les différentes images présentées plus haut. Il nécessite la bibliothèque *CImg*, elle-même faisant usage de *libpng* et de *libjpeg*. Ce code compile sous Debian avec les options suivantes :

```
1| g++ -pthread -lpng -ljpeg -lX11 -std=gnu+14 main.cpp
```

```
1 #define cimg_use_jpeg
2 #define cimg_use_png
3 #include "CImg.h"
4 using namespace cimg_library;
5
6 #include <bits/stdc++.h>
```

```

7 using namespace std;
8
9 CImg<unsigned char> img;
10 int w, h, sharpness;
11
12 vector<vector<int>> cc;
13 vector<vector<int>> chunks;
14
15 int dx[4] = { 0, 0, 1, -1 };
16 int dy[4] = { 1, -1, 0, 0 };
17
18 // Pour ne pas sortir de la grille
19 bool is_valid(int x, int y) {
20     return 0 <= x && x < h && 0 <= y && y < w;
21 }
22
23 // Pour tester si deux pixels ont des couleurs similaires
24 bool is_adj(int x1, int y1, int x2, int y2) {
25     bool ok = true;
26     for(int i = 0; i < 3; ++i) {
27         int diff = abs(img(y1, x1, i) - img(y2, x2, i));
28         ok = ok && diff <= sharpness;
29     }
30     return ok;
31 }
32
33 // Première passe
34 void dfs(int x, int y, int c) {
35     if(cc[x][y] != -1)
36         return;
37     cc[x][y] = c;
38     for(int i = 0; i < 4; ++i) {
39         int nx = x + dx[i];
40         int ny = y + dy[i];
41         if(is_valid(nx, ny) && is_adj(x, y, nx, ny))
42             dfs(nx, ny, c);
43     }
44 }
45
46 // Deuxième passe
47 void flood_fill(int x, int y, int c) {
48     if(chunks[x][y] != -1)
49         return;
50     chunks[x][y] = c;
51     for(int i = 0; i < 4; ++i) {

```

```

52     int nx = x + dx[ i ];
53     int ny = y + dy[ i ];
54     if( is_valid(nx, ny))
55         flood_fill(nx, ny, c);
56 }
57 }
58
59 int main( int argc, char** argv) {
60     if(argc <= 2) {
61         cout << "Arguments: <sharpness> <tolerance> [filename]" <<
62             endl;
63         return 1;
64     }
65     // Chargement l'image
66     string filename(argc == 3 ? "image.png" : argv[3]);
67     cout << "Loading image " << filename << endl;
68     CImg<unsigned char> src(filename.c_str());
69     img = src;
70     w = src.width();
71     h = src.height();
72     {
73         stringstream ss(argv[1]);
74         ss >> sharpness;
75     }
76     // Affichage de quelques informations
77     cout << "image width and height: " << w << "x" << h << endl;
78     cout << "sharpness: " << sharpness << endl;
79
80     // Première passe
81     cc.assign(h, vector<int>(w, -1));
82     vector< pair<int, int> > sizes;
83     int c = 0;
84     for( int i = 0; i < h; ++i)
85         for( int j = 0; j < w; ++j) {
86             if(cc[ i ][ j ] == -1) {
87                 sizes.push_back(make_pair(0, c));
88                 dfs(i, j, c++);
89             }
90             ++sizes[ cc[ i ][ j ] ].first;
91         }
92     cout << c << endl;
93
94     // Remplissage par des couleurs aléatoires
95     srand(time(0));

```

```

96     vector< array<int , 3> > colors(c);
97     for( int i = 0; i < c; ++i)
98         for( int j = 0; j < 3; ++j)
99             colors[ i ][ j ] = rand() %256;
100
101    for( int i = 0; i < h; ++i)
102        for( int j = 0; j < w; ++j) {
103            for( int k = 0; k < 3; ++k)
104                img(j , i , k) = colors[cc[ i ][ j ]][ k ];
105        }
106
107    img . save _png( "out _many colors . png" );
108
109 // Deuxième passe
110 if(c > 1) {
111     sort(sizes . rbegin() , sizes . rend()); // Reverse sort
112     colors . resize(2);
113
114     for( int i = 2; i < c; ++i)
115         for( int j = 0; j < 3; ++j)
116             colors [ sizes [ i ]. second ][ j ] = colors [ sizes [ 1 ]. second ][ j ];
117
118     for( int i = 0; i < h; ++i)
119         for( int j = 0; j < w; ++j)
120             for( int k = 0; k < 3; ++k)
121                 img(j , i , k) = colors [ cc[ i ][ j ]][ k ];
122
123     img . save _png( "out _2 colors . png" );
124
125 // Détection des pièces
126 chunks . assign(h, vector<int>(w, -1));
127     for( int i = 0; i < h; ++i)
128         for( int j = 0; j < w; ++j)
129             if(cc[ i ][ j ] == sizes [ 0 ]. second)
130                 chunks[ i ][ j ] = 0;
131
132             sizes . clear();
133             sizes . push_back(make_pair(0, 0));
134             c = 1;
135             for( int i = 0; i < h; ++i)
136                 for( int j = 0; j < w; ++j) {
137                     if(chunks[ i ][ j ] == -1) {
138                         sizes . push_back(make_pair(0, c));
139                         flood_fill(i , j , c++);
140                 }

```

```

141     ++sizes[ chunks[i][j] ].first ;
142 }
143
144 // Filtrages heuristiques
145 double tolerance ;
146 {
147     stringstream ss(argv[2]) ;
148     ss >> tolerance ;
149 }
150 tolerance /= 100.0;
151
152     vector<int> minX(c, 1e9), maxX(c, 0), minY(c, 1e9), maxY(c,
153     0) ;
154     for(int i = 0; i < h; ++i) {
155         for(int j = 0; j < w; ++j) {
156             minX[ chunks[i][j] ] = min(minX[ chunks[i][j] ], i) ;
157             minY[ chunks[i][j] ] = min(minY[ chunks[i][j] ], j) ;
158             maxX[ chunks[i][j] ] = max(maxX[ chunks[i][j] ], i) ;
159             maxY[ chunks[i][j] ] = max(maxY[ chunks[i][j] ], j) ;
160         }
161     }
162
163     int coins = 0;
164     for(int i = 1; i < c; ++i) {
165         double expected_size = M_PI*abs(minX[i]-maxX[i]) / 4.0 * abs(
166             minY[i]-maxY[i]) / 4.0;
167         if(sizes[i].first >= w*h*tolerance && (expected_size -
168             sizes[i].first)/expected_size < 0.2)
169             ++coins;
170     }
171
172
173     colors.resize(c);
174     for(int i = 1; i < c; ++i)
175         for(int j = 0; j < 3; ++j)
176             colors[i][j] = rand()%256;
177
178     for(int i = 0; i < h; ++i)
179         for(int j = 0; j < w; ++j) {
180             for(int k = 0; k < 3; ++k)
181                 img(j, i, k) = colors[ chunks[i][j] ][k];
182         }

```

```

183     img.save_png( "out_chunks.png" );
184 }
185 return 0;
186 }
```

Listing 4 – Notre algorithme

C Bibliographie

C.1 Livres

- Rafael C. Gonzalez et Richard E. Woods. (2008). *Digital Image Processing - Third Edition*. Pearson.

C.2 Articles

- Yosra Hasnaoui, Balkiss Hamad et Khaled Fayala. *Détection des droites par la transformée de Hough*. École Supérieure des Sciences et Techniques de Tunis.

C.3 Internet

- <https://www.quora.com/What-are-hyperparameters-in-machine-learning>.
- <https://www.lesechos.fr/idees-debats/cercle/cercle-161999-demystifier-la-machine-learning-2038392.php>.
- <https://www.youtube.com/watch?v=nKW8Ndu7Mjw>.
- <https://www.youtube.com/watch?v=trWrEWfhTVg>.
- <https://www.youtube.com/watch?v=h0e2HAPTGF4>.
- <https://openclassrooms.com/courses/initiez-vous-au-machine-learning/qu-est-ce-que-le-machine-learning>.
- <https://www.youtube.com/watch?v=trWrEWfhTVg>.
- https://fr.wikipedia.org/wiki/Apprentissage_automatique.
- <https://openclassrooms.com/courses/initiez-vous-au-machine-learning/qu-est-ce-que-le-machine-learning>.
- <https://www.quantmetry.com/single-post/2015/10/28/Une-petite-histoire-du-Machine-Learning>.

- <http://loic.knuchel.org/blog/2013/11/22/le-machine-learning-cest-quoi-exactement/>.
- <https://markdown.data-ensta.fr/s/machine-learning-introduction>.
- <https://www.futura-sciences.com/tech/definitions/informatique-intelligence-artificielle-555/>.
- https://fr.wikipedia.org/wiki/Intelligence_artificielle.
- <https://youtu.be/4zHbI-fFI1I>.