RAPPORT DE STAGE TN10

Apprentissage profond appliqué aux données transcriptomiques : *self-supervised learning*

Période allant du **07/02/2022 au 22/07/2022** – Semestre **P22**

Réalisé par : M. Thomas LAURENT, GI06

Suiveur de stage : M. Elmokhtar ALAYA

Maîtres de stage : M. Blaise HANCZAR, Mme Victoria BOURGEAIS

Établissement : Université de Technologie de Compiègne

Entreprise : Laboratoire IBISC, Bâtiment IBGBI – 2ème étage
23, Boulevard de France, 91037 Évry-Courcouronnes

Remerciements

Je tiens personnellement à remercier les personnes qui ont contribué à ma bonne intégration ainsi qu'au bon déroulement de mon stage au sein du laboratoire IBISC.
Je remercie tout d'abord mes tuteurs de stage, M. Blaise HANCZAR et Mme Victoria BOURGEAIS, pour m'avoir encadré tout au long de ce stage. Une de mes craintes avant ce stage était de me retrouver seul dans mon sujet de recherche, mais j'ai au contraire été agréablement surpris par l'accompagnement que j'ai reçu. Leur disponibilité pour répondre à mes questions et les orientations qui m'ont été données m'ont permis d'être serein et efficace dans mon travail.
Dans ce sens, je remercie également M. Aurélien BEAUDE et Mme Alice LACAN avec qui il a été un plaisir de partager le bureau. En leur présence, j'ai pu m'épanouir dans un environnement de travail studieux, mais détendu, et je les remercie pour le temps qu'ils m'ont accordé lorsque j'avais besoin de leurs conseils.
Je remercie Mme Murielle BOURGEOIS pour m'avoir accompagné administrativement au cours de mon stage, pour son implication dans l'organisation d'événements tout au long de ces six mois ainsi que pour sa bonne humeur qui contribue grandement à l'ambiance positive qui règne dans le laboratoire.
Je remercie enfin les stagiaires et doctorants avec qui j'ai eu le plaisir de partager du temps, pendant les pauses et en fin de journée. Je pense en particulier à Khang, Tina, François, Constance, Priyanka, Henes et Loic ; je n'oublierai pas leur bonne humeur et leur disponibilité qui m'ont permis de passer du bon temps durant et aussi en dehors des heures de travail.

Résumé technique

Le stage TN10 est un stage de 6 mois que j'ai effectué dans le cadre de ma formation d'ingénieur en génie informatique, filière intelligence artificielle et science des données, à l'Université de Technologie de Compiègne. Ce stage marque la fin du cursus ingénieur, il intervient en effet sur le dernier semestre de la formation. Il permet ainsi la mise en application des connaissances acquises tout au long des études, et en particulier les compétences dans un domaine correspondant à la filière choisie par le stagiaire.

J'ai effectué mon stage TN10 au laboratoire IBISC (Informatique, BioInformatique, Systèmes Complexes) de l'Université d'Évry Paris-Saclay. Ma mission durant ce stage était d'adapter des méthodes d'apprentissage profond *self-supervised* (initialement prévues pour des tâches dans le domaine de l'image) pour des tâches utilisant des données issues de la transcriptomique.

J'ai débuté mon stage par la réalisation d'un état de l'art des méthodes *self-supervised*. Suite à cela, j'ai pu sélectionner une des méthodes les plus performantes dans le domaine de l'image que j'ai ensuite implémentée et adaptée pour qu'elle puisse fonctionner avec des données transcriptomiques. Il s'est ensuivi une phase d'expérimentation afin d'ajuster les paramètres du modèle développé puis de tester les performances de la méthode sur les données transcriptomiques.

Table des matières

Présentation du laboratoire et de l'équipe d'accueil

Histoire du Laboratoire IBISC

Le laboratoire IBISC (Informatique, Biologie Intégrative et Systèmes Complexes) a été créé en 2006, à la demande du CNRS. Le laboratoire est issu de la fusion de deux laboratoires localisés à Évry-Courcouronnes : le LAMI (Laboratoire des Méthodes Informatiques) et le LSC (Laboratoire des Systèmes Complexes). IBISC est une Équipe d'Accueil (EA 4526) sous la tutelle de l'UEVE (université d'Evry-Val-d'Essonne) et avec le soutien de l'ENSIIE (École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise) et du GENOPOLE.

Présentation des équipes

Les thématiques de recherche abordées au sein du laboratoire IBISC traitent de la modélisation, la conception, la simulation et la validation des systèmes complexes. Les systèmes considérés sont aussi bien des systèmes biologiques que des systèmes artificiels (robots, drones, véhicules intelligents).
IBISC compte, en juillet 2022, 51 enseignants chercheurs permanents, 7 chercheurs associés, 42 doctorants, 9 post-doctorants, soit un effectif total de plus de 100 personnes. En plus de cela, 41 stagiaires ont été employés sur le semestre de printemps 2022.
Ce personnel est organisé autour de 4 équipes de recherche, à savoir :

- L'équipe COSMO (Communications Spécifications Modèles) qui étudie les propriétés fondamentales des systèmes informatiques et biologiques et, plus généralement, le comportement de systèmes dynamiques réactifs, décentralisés et ouverts. Une des principales orientations de recherche de l'équipe est la problématique applicative axée sur les problèmes de société (médecine personnalisée, véhicule du futur, internet du futur) et pour lesquels la définition de nouveaux cadres théoriques ou méthodologiques est nécessaire.
- L'équipe IRA2 (Interaction, Réalité virtuelle & Augmentée, Robotique Ambiante) qui travaille sur l'amélioration des interactions entre des personnes et des systèmes complexes artificiels (virtuels, augmentés, robotisés, applications informatiques).
- L'équipe SIAM (Signal, Image, AutoMatique) dont les recherches s'articulent autour des quatre étapes de l'étude générale d'un système que sont la perception, l'observation, la modélisation et la commande. Ces méthodes sont principalement appliquées sur des thématiques autour des véhicules et des systèmes biologiques.
- Enfin, l'équipe AROBAS (Algorithmique, Recherche Opérationnelle, Bioinformatique et Apprentissage Statistique) qui mène une recherche à la fois fondamentale et appliquée. Autour du thème de l'apprentissage statistique, l'équipe AROBAS s'intéresse notamment au domaine de l'apprentissage profond, essentiellement pour l'analyse de données génomiques. Les sujets de recherche portent par exemple sur la

problématiques de l'interprétation, sur l'apprentissage avec peu d'exemples, ou sur la réduction de dimension.

Encadrement du stage

Durant ce stage, j'ai été accompagné par deux encadrants, M. Blaise HANCZAR et Mme Victoria BOURGEAIS. Blaise est professeur des universités à l'université d'Évry, ses sujets de recherche s'organisent autour des dernières méthodes d'apprentissage profond avec des applications dans la médecine personnalisée. Victoria est actuellement en dernière année de thèse sur le sujet autour de l'interprétation des réseaux de neurones. En plus des discussions régulières que je pouvais avoir avec Victoria liées au fait que nous partagions le même bureau, une réunion hebdomadaire était organisée avec mes deux encadrants. C'était pour moi l'occasion de faire une courte présentation de mon travail de la semaine passée. Suite à cela, nous définissons les objectifs pour la semaine à venir.

Ma mission

Sujet initial et évolutions

Le sujet de mon stage porte sur l'analyse de données transcriptomiques (données d'expression des gènes) par apprentissage profond. Une contrainte posée par les données transcriptomiques est le manque de grands jeux de données disponibles dû à leur coût élevé d'acquisition. En effet, les jeux de données transcriptomiques publiques ne contiennent que très peu de patients, quelques milliers au mieux. En entraînant des réseaux de neurones avec les méthodes classiques d'apprentissage profond, on se heurte ainsi au problème de sur-apprentissage qui sera détaillé dans la suite de ce rapport.

Afin d'améliorer les performances, il faudrait pouvoir agrandir l'ensemble d'apprentissage. Dans de nombreux domaines, l'acquisition de données brutes est relativement aisée. Ces sont plutôt les données labellisées, c'est-à-dire dont on dispose de la donnée et de sa description, qui sont coûteuses à obtenir. À l'inverse, les données non labellisées, c'est-à-dire qu'on dispose de la donnée mais pas de sa description, sont relativement faciles d'accès. Pour le domaine de l'image, les réseaux sociaux donnent aujourd'hui accès à des millions de nouvelles images chaque jour, cependant ces images sont dénouées d'une quelconque description, on ne sait pas quel est leur contenu tant qu'un opérateur humain ne les a pas classifiées. Le domaine de la transcriptomique suit ce même modèle : prélever et séquencer un échantillon chez un patient est aujourd'hui une procédure rapide et peu coûteuse. Néanmoins, le séquençage ne nous apprend rien sur les éventuelles pathologies du patient ; pour labelliser l'échantillon, par exemple en déterminant s'il est cancéreux, il faudra avoir recours à des analyses en laboratoire, ce qui a nécessairement un coût.

Le milieu de la recherche étudie donc des méthodes d'apprentissage automatique qui permettent d'exploiter ces données non labellisées, elles sont appelées méthodes d'apprentissage non supervisé. Ces méthodes se divisent en plusieurs sous classes, notamment la classe des méthodes d'apprentissage *self-supervised*, dont le principe sera explicité dans la suite de ce rapport.

Un précédent stagiaire avait exploré des méthodes d'apprentissage *self-supervised*, sans obtenir de résultats probants. L'état de l'art évoluant rapidement, de nouvelles méthodes d'apprentissage ont depuis émergé, et ma mission était d'évaluer ces nouvelles méthodes. Mon premier objectif était ainsi de réaliser un nouvel état de l'art couvrant les principales techniques d'apprentissage *self-supervised*. Suite à cela, le but était de se baser sur cet état de l'art pour choisir une méthode prometteuse et de l'implémenter sur le problème de l'analyse des données transcriptomiques, principalement sur des tâches de prédiction de cancer. La mission était ensuite de mesurer l'impact de la méthode ainsi développée au niveau des performances sur les différentes tâches ciblées. Ce sujet initial a été respecté, car les résultats obtenus au fur et à mesure de l'avancement du stage ne nous ont pas contraints à dériver des orientations préalablement définies. Cependant, il convient de relever que certaines tâches sur lesquelles il était prévu d'évaluer la méthode d'apprentissage *self-supervised* ont finalement été mises de côté à causes des difficultés liées à l'entraînement d'un réseau de neurones sur celles-ci, ce qui ne permettait pas de mesurer assez préciser les performances de la méthode d'apprentissage *self-supervised*.

Introduction à l'apprentissage profond

Les réseaux de neurones classiques sont communément utilisés afin d'effectuer une prédiction, concernant un problème de régression ou de classification, à partir d'une donnée d'entrée. Ainsi, il est possible de prédire le prix d'une maison (il s'agit alors d'un problème de régression, le prix d'une maison pouvant être considéré comme une variable continue) en fonction de caractéristiques comme la surface, le nombre de pièces, les coordonnées géographiques ou encore la présence ou non de balcons dans la maison considérée ; de même, la prédiction peut être un score de confiance sur la présence ou non d'un chat (le problème est dit de classification, la certitude de la présence du chat étant traduite par une prédiction égale à 1 et la certitude de son absence par une prédiction égale à 0) sur une image donnée en entrée au réseau de neurones.
Les caractéristiques d'entrée, *input features* en anglais, forment l'*input layer* du réseau de neurones, et la prédiction finale est enveloppée dans l'*output layer*. Entre ces deux couches uniques, un nombre variable d'*hidden layers*, ou couches cachées, modifient successivement les valeurs d'entrée du réseau de manière non réellement interprétable par l'être humain jusqu'à atteindre l'*output layer* où apparaît alors la prédiction souhaitée. Généralement, tous les neurones d'une couche sont liés à tous ceux de la couche suivante, formant ainsi un réseau entièrement connecté, mais cela n'est en aucun cas obligatoire.

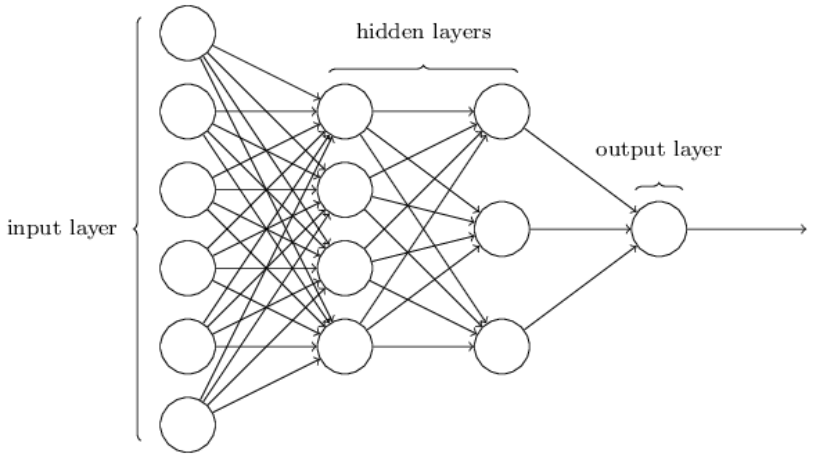

Figure 1 : Schéma d'un réseau de neurones classique (Karim Rezaul)

Chaque cercle sur la Figure 1 ci-dessus représente un neurone dont l'activation, hormis pour ceux de l'*input layer* qui constituent les *input features* que l'on fournit au réseau, est obtenue de la manière décrite sur la figure suivante. Chaque feature en entrée d'un neurone se voit appliquer un facteur dénommé poids, en anglais *weight*, puis l'ensemble des produits résultants est sommé. À cette somme de produits est ajoutée un biais, en anglais *bias*, puis une fonction dite d'activation est appliquée au tout, cette fonction servant à contenir les valeurs entre 0 et 1 à l'intérieur du réseau pour assurer sa stabilité. L'image de la somme des *weights* et du *biais* par cette fonction d'activation constitue ainsi la valeur résultat placée dans un neurone de la couche suivante, et ce, pour chaque neurone de chaque couche.

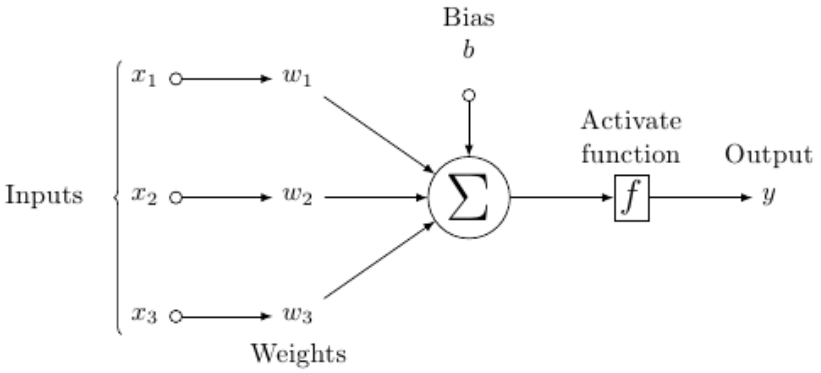

Figure 2 : Schéma d'un neurone (Gonzalo Medina)

Tout l'enjeu des réseaux de neurones est de parvenir à leur faire apprendre les *weights* et *biais* appliqués à chaque neurone de façon à ce qu'ils puissent effectuer des prédictions satisfaisantes, par exemple pour les caractéristiques d'une maison ou une image d'entrée inconnues.

Un MLP, pour *multilayer perceptron*, est un réseau de neurones entièrement connecté comportant au minimum trois couches : l'*input layer*, une *hidden layer*, et l'*output layer*. Les MLP se distinguent des *linear perceptrons* (eux ne comportant pas de couche cachée) par le fait qu'ils peuvent distinguer les données qui ne sont pas linéairement séparables. Les MLP sont parfois appelés réseaux de neurones "*vanilla"*, en particulier lorsqu'ils n'ont qu'une seule couche cachée. À partir de deux ou trois couches cachées, on pourra qualifier les MLP de méthode d'apprentissage profond, ou *deep learning* en anglais.

Entrainement d'un réseau de neurones

Classiquement, on entraine un réseau de neurones d'une manière supervisée, avec des données dites labellisées. Par exemple, prenons l'exemple de la classification chat / chien à partir d'images. On va fournir à l'entrée du réseau de neurones l'image d'un chat (ou d'un chien) ; les pixels de l'image vont constituer l'*input layer*. On va ensuite indiquer au réseau un label, c'est-à-dire une étiquette associée à l'image qui indique sa catégorie : chat (ou chien). Grâce à cela, le réseau va pouvoir, automatiquement, ajuster les valeurs de ses paramètres (*weights* et *biaises*) afin que la valeur de son *output* (la prédiction du réseau) corresponde le plus souvent possible à la valeur du *label* (vérité terrain). Cet entraînement est un processus itératif, les valeurs des *weights* et de ses *biais* sont optimisées (mises à jour) étape par étape. L'entraînement du réseau est terminé une fois que les paramètres atteignent des valeurs minimisant une fonction appelée fonction de coût (fonction qui mesure la distance entre les *outputs* et les *labels*).

État de l'art du *self-supervised learning*

Un des enjeux de l'apprentissage supervisé est de disposer de suffisamment de données labellisées. En effet, un des problèmes rencontrés si le jeu de données est trop restreint est que le réseau va avoir tendance à apprendre par cœur les données intervenant dans l'entraînement et présenter de mauvaises performances une fois déployé en production quand il va rencontrer des données qui ne sont pas

intervenues dans son entraînement. On appelle ce phénomène le sur-apprentissage, *overfitting* en anglais.
On va alors chercher à obtenir plus de données labellisées pour contenir ce phénomène, mais ce n'est pas toujours possible, en particulier dans le domaine médical. Cependant, il est souvent bien plus aisé de se procurer de larges jeux de données non labellisés, et plusieurs méthodes d'apprentissage profond s'intéressent à la valorisation de ce type de jeu de données.
Le *self-supervised learning* (SSL), pour apprentissage auto-supervisé, est une méthode faisant intervenir des données non labellisées lors de l'entraînement du réseau de neurones.

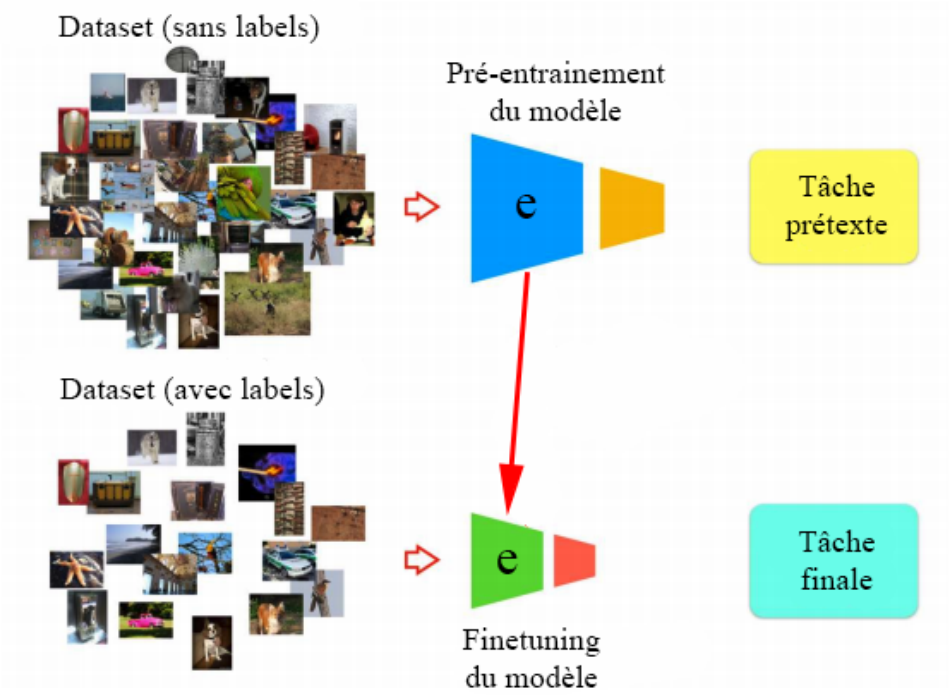

Figure 3 : Procédure d'entraînement self-supervised (Mehdi Noroozi, modifié)

L'objectif général des méthodes de SSL est d'entraîner un réseau de neurones dont le but est de trouver une bonne représentation des données, c'est-à-dire une représentation des données qui capture les concepts haut niveau liés au domaine des données. Ce type de réseau de neurones est souvent référencé sous le terme d'*encoder* dans la littérature.
Les méthodes de SSL utilisent généralement une tâche prétexte pour générer des *pseudo-label*s, c'est-à-dire que des labels sont assignés aux données de manière automatique, l'*encoder* s'entraînant ensuite à partir de ces *pseudo-labels*.
Une fois la représentation de données obtenues grâce à la méthode *self-supervised*, on va pouvoir récupérer le réseau de neurones ainsi "pré-entrainé".
On appose ensuite un MLP à la sortie du réseau et on entraîne le modèle ainsi construit de manière supervisée sur la tâche finale (par exemple une tâche de classification) : c'est l'étape de *fine-tuning*. L'apposition d'un MLP à la suite du réseau pré-entraîné permet de conserver la bonne représentation des données et les couches de neurones supplémentaires vont permettre de répondre aux spécificités de la tâche finale.
Les méthodes de SSL de l'état de l'art dans le domaine de l'image peuvent se regrouper en deux grandes catégories : les méthodes *contrastives* et les méthodes de *clustering*.

Méthodes *contrastives*

Les méthodes *contrastives* consistent à entraîner l'*encoder* en rapprochant des points similaires (appelés exemples positifs) et en éloignant les points dissimilaires (appelés exemples négatifs) dans l'espace latent (l'espace de représentation des données après passage dans l'*encoder)*.

$$Score(e(x),\ e(x+)) \gg Score(e(x),\ e(x-))$$

avec :

- $e()$ *l'encoder*
- $x+$ *un exemple positif de l'exemple* x
- $x-$ *un exemple négatif de l'exemple* x
- $Score()$ *une métrique qui mesure la similarité entre deux points*

Formule 1 : Objectif des méthodes contrastives

MoCo : Momentum Contrast for Unsupervised Visual Representation Learning

MoCo [1] est une méthode de *contrastive learning* proposée par Ke et al. (FAIR) en 2020. L'architecturé de MoCo utilise deux *encoders* distincts. Le premier est qualifié de *student encoder*, ses paramètres sont mis à jour classiquement par la méthode de rétropropagation du gradient.

$$\theta k \leftarrow m * \theta k + (1 - m) * \theta q$$

avec :

- $m \in [0,\ 1)$ *le coefficient de momentum*
- θk *les paramètres du momentum encoder*
- θq *les paramètres du student encoder*

Formule 2 : Mise à jour des paramètres du momentum encoder

Le second *encoder*, qualifié de *momentum encoder* : ses paramètres sont mis à jour à partir des paramètres du réseau *student* selon la Formule 2.

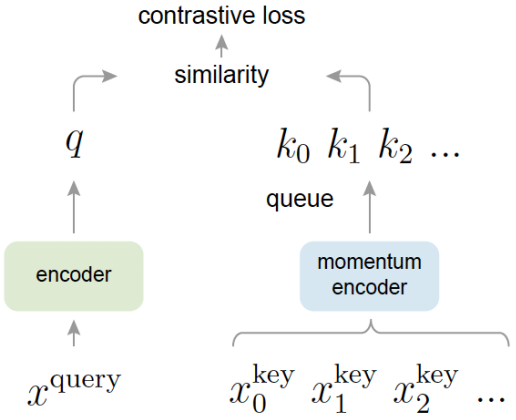

Figure 4 : Procédure d'entraînement de MoCo (Ke et al.)

L'entraînement de MoCo est schématisé sur la Figure 4. Premièrement, deux versions augmentées d'un *mini-batch* (un sous-ensemble du jeu de données) sont générées (en rognant les images par exemple). On se retrouve ainsi avec des paires d'images différentes, mais provenant de la même image initialement : ce sont les paires positives. Une des versions est passée à travers le *student encoder* pour obtenir la *query* (q) et l'autre à travers le *momentum encoder* pour obtenir une *key*. Ces représentations interviennent ensuite une fonction de coût *contrastive* qui cherche à rapprocher les images de la *query* à leur paire dans la *key* et à les éloigner

des *keys* des précédents mini-batchs (exemples négatifs) qui ont été conservés à l'aide d'une queue FIFO.

SimCLR : Simple framework for contrastive learning of visual representations

SimCLR [2] est une méthode de *contrastive learning* introduite par Chen et al. (Google Brain) en 2020. SimCLR repose sur une architecture composée de deux parties : un *encoder* et un petit réseau de neurones appelé *projection head*. L'encoder extrait une représentation des données, c'est donc cette partie qui sera récupérée pour la tâche de *fine-tuning*. La *projection head* projette les sorties de l'*encoder* dans un espace où est appliquée la fonction de coût *contrastive*.

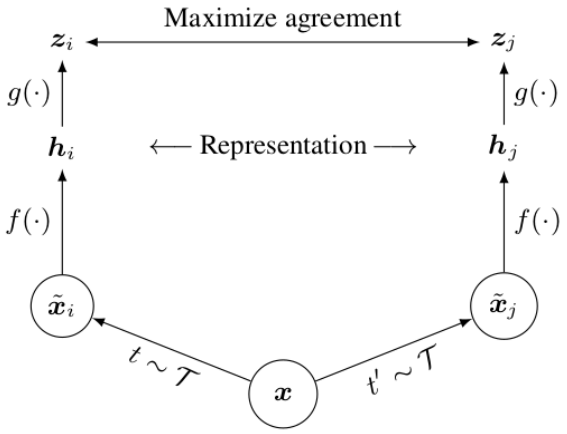

Figure 5 : Procédure d'entraînement de SimCLR (Chen et al.)

La Figure 5 illustre le fonctionnement de SimCLR : une image est augmentée de deux manières différentes pour en créer deux versions, chacune va être passée à travers l'*encoder* et la *projection head*. Les deux représentations ainsi obtenues vont intervenir dans la fonction de coût en tant qu'exemples positifs, et les exemples négatifs seront les augmentations des autres images du *mini-batch*.

Une seconde version de la méthode, SimCLRv2 [3] est venue ajouter une étape de *self-training* à la suite de la phase de *fine-tuning*.

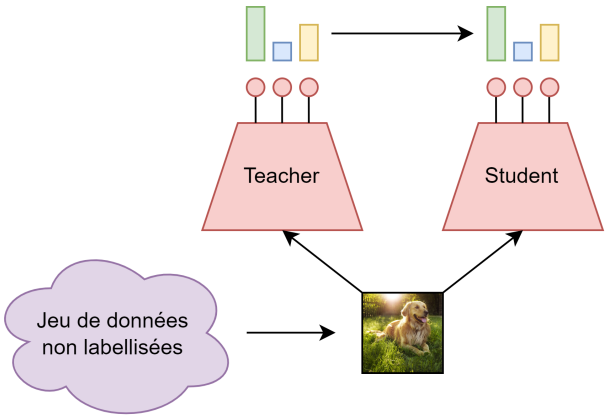

Figure 6 : Teacher-Student self-training

La Figure 6 illustre le principe du *self-training* : le réseau dit *Teacher* est le modèle obtenu à la suite du pré-entraînement et du *fine-tuning* du réseau. Le réseau *student* est un réseau de neurones en tous points similaire au *teacher*, à l'exception qu'il n'est pas entraîné. Il est alors entraîné à partir du jeu de données non labellisées, avec comme *label* les sorties du réseau *teacher*. Les expérimentations montrent que

cette étape supplémentaire de *self-training* induit une augmentation des performances sur la tâche finale. La spécificité du *self-training* est que le réseau *student* est entraîné à partir de *soft-labels* (un ensemble de scores d'appartenance aux différentes classes, par exemple 70% chien, 20% loup, 10% renard).

BYOL : Bootstrap your own latent: A new approach to self-supervised Learning

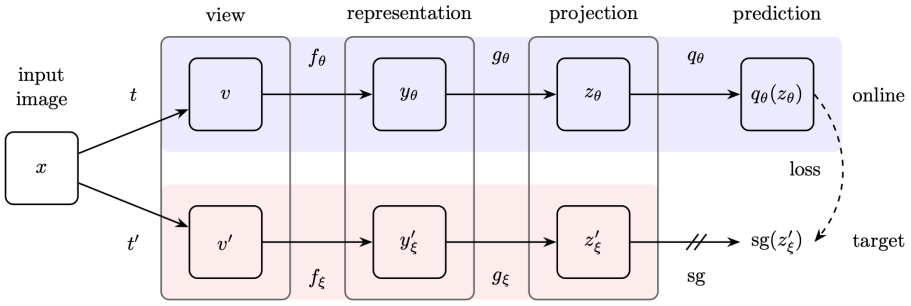

Figure 7 : Procédure d'entraînement de BYOL (Grill et al.)

BYOL [4] est une méthode *contrastive* présentée par Grill et al. (Google Brain) en 2020. À l'instar de MoCo, la méthode proposée par BYOL se base sur deux réseaux de neurones, un *encoder* ("online" sur la Figure 7) et un *momentum encoder* ("target" sur la Figure 7). Cependant, contrairement aux approches précédentes, BYOL est une méthode de *contrastive learning* ne faisant pas intervenir d'exemple négatif lors l'entraînement.
L'image d'entrée est transformée deux fois avec deux transformations différentes, puis ces représentations sont transmises à travers le réseau "online" et le réseau "target". Le réseau "online" diffère du réseau "target" avec l'ajout d'un réseau "prediction" qui a pour objectif de deviner la sortie du réseau "target". La fonction de coût calcule la différence entre les représentations du réseau "online" et du réseau "target". Les paramètres du réseau "online" sont alors mis à jour par rétropropagation du gradient et ceux du réseau "target" le sont également, à la manière d'un *momentum encoder* (voir la Formule 2).

Méthodes de *clustering*

Les méthodes de *clustering* sont des méthodes de SSL qui implémentent une étape de *clustering* (partitionnement) des données dans leur pipeline. Le partitionnement des données peut ensuite servir à fournir des *pseudo-labels*, c'est-à-dire qu'on génère des *labels* automatiquement en fonction des *clusters* auxquels appartiennent les points. Ces *pseudo-labels* servent ensuite à entraîner le réseau de neurones de manière classique, comme pour l'apprentissage supervisé.

DeepCluster : Deep Clustering for Unsupervised Learning of Visual Features

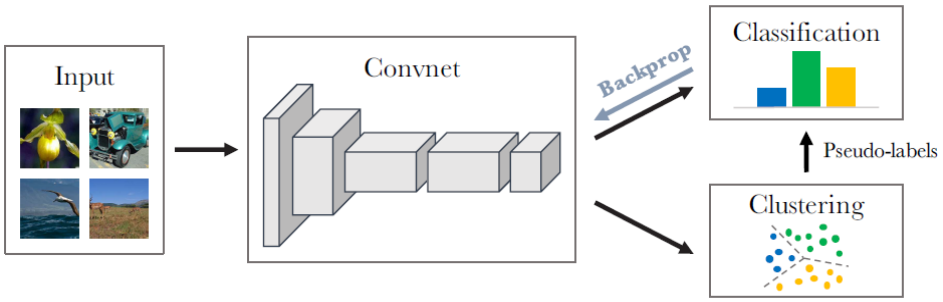

Figure 8 : Procédure d'entraînement de DeepCluster (Caron et al.)

DeepCluster [5] est une méthode de SSL par *clustering* proposée par Caron et al. (FAIR) en 2019. L'algorithme proposé par le papier est basique et intuitif. Les images d'un *mini-batch* sont augmentées selon deux routines d'augmentations afin d'en obtenir deux versions distinctes, tout comme pour les méthodes de *contrastive learning*. Ces deux versions passent ensuite par un *encoder* (un Convnet dans la Figure 8) pour ainsi obtenir leurs paires de représentation dans l'espace latent. L'algorithme K-means [6] est appliqué aux points de l'espace latent provenant d'une des deux routines d'augmentation, on obtient ainsi un *clustering* en k clusters. On déduit ensuite des *pseudo-labels* de ce *clustering*, en assignant en tant que classe le *cluster* auxquels appartiennent les points. Les points provenant de l'autre routine d'augmentation passent quant à eux par une couche supplémentaire du réseau de neurones (ayant une vocation de classification) afin d'obtenir un vecteur de dimension k. Une fonction de coût calcule l'erreur entre les prédictions de classification fournies par le réseau de neurones et les *pseudos-labels* issus du *clustering*. Les paramètres du réseau sont alors mis à jour et l'entraînement continue avec l'itération suivante de l'algorithme.

SwAV : Unsupervised Learning of Visual Features by Contrasting Cluster Assignments

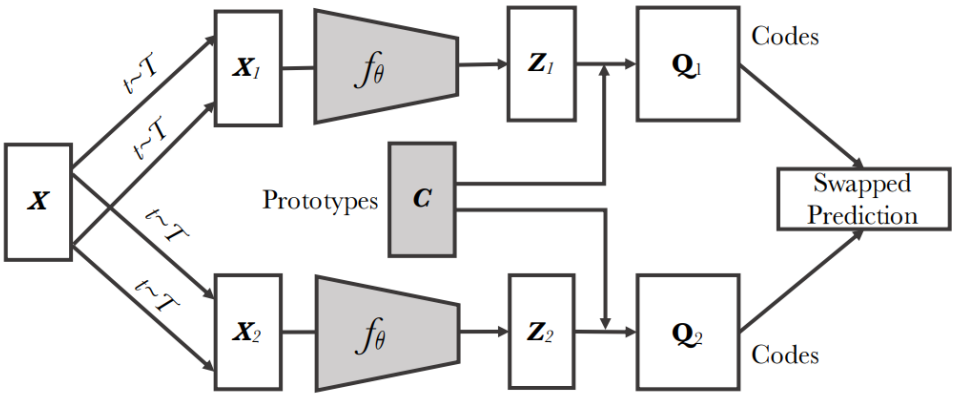

Figure 9 : Procédure d'entraînement de SwAV (Caron et al.)

SwAV [7] est une évolution de DeepCluster, proposée par Caron et al. (FAIR) en 2021. L'approche proposée diffère de DeepCluster dans le fait que l'étape de *clustering* ne passe pas par un algorithme séparé du réseau de neurones (k-means pour DeepCluster). En effet, le *clustering* est implémenté par une simple couche de neurones de dimension k apposée à la suite de l'*encoder*, avec k le nombre de "prototypes" (terminologie utilisée dans le papier pour parler des *clusters*).

L'*encoder* se divise en deux structures de réseaux de neurones : un ResNet50 [8] et un réseau entièrement connecté, noté Z sur la Figure 10.

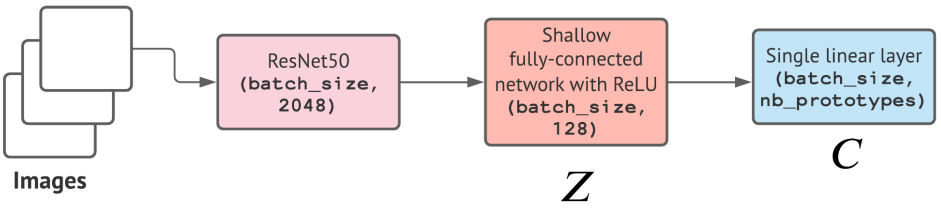

Figure 10 : Architecture du réseau SwAV

La matrice C (Figure 10) d'assignation aux prototypes est ensuite optimisée par l'algorithme itératif Sinkhorn-Knopp [9] afin qu'elle suive une contrainte d'équipartition au niveau de ses lignes et ses colonnes. Cela assure le fait que les prototypes soient assignés le même nombre de fois au sein d'un *mini-batch*. On espère ainsi que les vecteurs qui sont éloignés dans l'espace latent Z (possédant donc des caractéristiques distinctes) ne soient pas assignés au même prototype. En effet, la solution triviale consistant à une assignation constante au même prototype doit être évitée. La matrice Q des "codes" est ainsi obtenue après optimisation par l'algorithme Sinkhorn-Knopp. Le coût associé à un *mini-batch* peut alors être calculé : ce coût est une addition de deux termes symétriques (Formule 3).

$$L(Z1,\ Z2)\ =\ l(Z1,\ Q2)\ +\ l(Z2,\ Q1)$$

$avec$:

- $l(Z,\ Q)\ l'entropie\ croisée\ entre\ Z\quad C\ et\ Q$
- $(Z/Q)1\ les\ matrices\ issues\ de\ l'augmentation\ 1$
- $(Z/Q)2\ les\ matrices\ issues\ de\ l'augmentation\ 2$

Formule 3 : Calcul du coût d'entraînement dans l'algorithme SwAV

Ainsi, pour réduire le coût, SwAV doit correctement prédire le prototype assigné à l'augmentation 1 de l'image d'entrée à partir du vecteur latent associé à l'augmentation 2 de l'image d'entrée.

Outils et jeux de données

Afin de faire tourner des algorithmes nécessitant une puissance de calcul élevée, par exemple l'entraînement des réseaux de neurones, le laboratoire IBISC héberge des serveurs de calculs. Ces serveurs sont organisés en clusters de CPUs (*central processing units*) and GPUs (*graphics processing units)*. Les GPUs à ma disposition étaient deux cartes Nvidia GeForce GTX TITAN X donc la puissance et la mémoire vive de 12 Go me permettaient d'entraîner mes modèles en quelques dizaines de secondes. L'interface de développement, accessible simplement via un navigateur, est propulsée par un serveur Jupyter avec un noyau Python 3.

L'intégralité du développement durant mon stage a été réalisé en Python : c'est en effet le langage le plus populaire dans le domaine du machine learning et la science de données, en particulier pour les applications ne demandant pas de calcul en temps réel. De plus, de nombreuses bibliothèques Python (ensemble organisé de fonctions pré-définies permettant de travailler sur un thème particulier) sont disponibles. J'ai particulièrement utilisé Pytorch, qui est une bibliothèque Python open source dédiée à l'apprentissage machine développée par Meta. Cette bibliothèque fournit des fonctions et structures permettant de concevoir et entraîner

des réseaux de neurones. J'ai également utilisé les biblothéques Scikit-Learn, Numpy et Pandas, pour ne citer qu'elles.

Durant ce stage, j'ai travaillé sur des données transcriptomiques : ce sont des données provenant de quantifications de l'ARN messager issu de la transcription du génome dans un tissu et des conditions données. Ces données permettent ainsi de connaître le niveau d'expression des gènes d'un organisme vivant, donnent des informations sur la façon dont les gènes sont régulés et révèlent des détails sur la biologie d'un organisme. Certains gènes sont "annotés", c'est-à-dire qu'on connaît leur rôle dans le fonctionnement d'un organisme. Mais certains ne le sont pas et la transcriptomique peut également être utilisée pour déduire les fonctions de gènes précédemment non annotés.
Plus généralement, l'analyse du transcriptome a permis l'étude de la manière dont l'expression des gènes change dans différents organismes et a joué un rôle déterminant dans la compréhension des maladies humaines. Une analyse de l'expression génique dans son intégralité permet de détecter de tendances globales et coordonnées qui ne peuvent être discernées par des tests plus ciblés.

Plusieurs méthodes permettent de prélever des données transcriptomiques, en particulier les méthodes *microarray* et *RNA-Seq*.
Dans la méthode *microarray*, un échantillon de cellules est prélevé et cet échantillon est purifié afin d'isoler l'ARN messager. On induit ensuite le mécanisme de transcriptase inverse dans le but d'obtenir des brins d'ADN complémentaire qui représentent ainsi la partie codante de la région du génome ayant été transcrit en cet ARN messager. Ces brins d'ADN complémentaires sont ensuite déposés sur la plaque *microarray* : cette plaque est constituée de différentes lames qui comportent des brins d'ADN ayant le rôle de sonde, car ils réagissent à la présence de brins d'ADN complémentaires en émettant de la lumière. On peut ainsi quantifier des ARN messagers précis en fonction des brins d'ADN sondes qui constituent la *microarray* en passant la plaque dans une machine dotée de capteurs optiques qui vont mesurer l'intensité de la lumière émise.
Dans la méthode *RNA-Seq* (pour *RNA sequencing*, ou séquencage de l'ARN en français), l'ARN messager est isolé et de ADN complémentaire est synthétisé à partir des brins d'ARN, comme pour la méthode *microarray*. Cependant, la méthode *RNA-Seq* utilise le séquençage parallèle massif (une approche à haut débit du séquençage de l'ADN) pour séquencer les brins d'ADN complémentaire afin de les rendre lisibles par un ordinateur. Les quantifications des ARN messagers peuvent alors se faire informatiquement. La méthode *RNA-Seq* est plus précise que la méthode *microarray* et son coût d'acquisition est similaire, si bien que cette technologie est très majoritaire dans les nouvelles études.

Durant mon stage, j'ai travaillé autour de deux jeux de données. Le premier jeu de données "MicroArray" est un jeu de données provenant d'une étude pan-cancer, c'est-à-dire une étude visant à trouver les similarités et différences parmi les altérations génomiques et cellulaires trouvés dans différents types de tumeurs. Cette étude a compilé les profils d'expressions de gènes publiquement accessibles d'environ 40000 profils génomiques. Cela combine différents jeux de données d'expression de gène contenant divers tissus et protocoles expérimentaux et intègre des données provenant de patient ou de lignée cellulaires. Le jeu de données est accessible depuis la base de données ArrayExpress, et j'ai utilisé une version

pré-traitée (contrôle de la qualité et normalisation) présentant les expressions de 54675 gènes. Seuls les profils provenant des types de tissus les plus présents (plus de 400 occurrences) sont conservés. Pour chaque profil, les informations disponibles sont :

- le statut (cancer / pas de cancer)
- le type de profil (patient / lignée cellulaire)
- le tissu

Le jeu de données n'est pas équilibré, comme le montre la Table 1 : tout d'abord entre les tissus, certains tissus sont plus présents que d'autre, mais aussi au sein d'un tissu, il y a généralement plus (3 fois plus en moyenne) d'exemples positifs (cancer) que d'exemples négatifs (pas de cancer).

Type de cancer	Taille	Patients	Lignée	Cancer	¬ Cancer
Leucémie	4283	3452	831	2336	1947
Moelle osseuse	3525	3374	151	3185	340
Sein	2171	1366	805	1863	308
Rein	657	423	234	400	257
Foie	727	312	415	601	126
Poumons	1415	749	666	818	597
Peau	835	554	281	454	381
Cerveau	869	468	401	819	50
Colon	1239	875	364	1112	127
Ovaires	573	427	146	533	40
Prostate	415	182	233	350	65
Total	16,709	12,182	4527	12,471	4238

Table 1 : Répartition des données MicroArray

Le second jeu de données que j'ai été amené à utiliser durant mon stage provient de *The Cancer Genome Atlas* (TCGA). TCGA est un consortium entre l'institut américain du cancer (*National Cancer Institute*, NCI) et l'institut américain de recherche sur le génome humain (*National Human Genome Research Institute*, NHGRI) lancé en 2005. TCGA fournit un jeu de données multiomique (contient différentes "omes", tels que le génome, le protéome, le transcriptome, l'épigénome, le métabolome et le microbiome, qui contient des données patient avec différents types de cancer). Ce jeu de données est bien plus complet et homogène que le jeu de données MicroArray. J'ai utilisé spécifiquement les données *RNA-Seq* sur la tâche de prédiction de type de cancer. Ces données présentant les expressions de 57387 gènes. Ici aussi, le jeu de données n'est pas équilibré, avec certains tissus bien plus représentés que d'autres (Table 2).

Type de cancer	Patients	Description
BRCA	1089	*Breast invasive carcinoma*
UCEC	544	*Uterine Corpus Endometrial Carcinoma*
KIRC	531	*Kidney renal clear cell carcinoma*
LGG	529	*Brain Lower Grade Glioma*
LUAD	515	*Lung adenocarcinoma*
THCA	510	*Thyroid carcinoma*
HNSC	502	*Head and Neck squamous cell carcinoma*
LUSC	501	*Lung squamous cell carcinoma*
PRAD	496	*Prostate adenocarcinoma*
SKCM	471	*Skin Cutaneous Melanoma*
COAD	458	*Colon adenocarcinoma*
BLCA	408	*Bladder Urothelial Carcinoma*
OV	379	*Ovarian serous cystadenocarcinoma*
STAD	375	*Stomach adenocarcinoma*
LIHC	374	*Liver hepatocellular carcinoma*
CESC	305	*Cervical squamous cell carcinoma*
KIRP	289	*Kidney renal papillary cell carcinoma*
SARC	263	*Sarcoma*
LAML	151	*Acute Myeloid Leukemia*
Total cancer	8690	
Pas de cancer	662	
Total	9352	

Table 2 : Répartition des données TCGA

Mes réalisations

Travail préliminaire

Dans un premier temps, l'objectif était de prendre en main les données MicroArray à travers le développement d'un premier réseau de neurones basique pour la prédiction cancer / non cancer, et le jeu de données TCGA à travers le développement d'un réseau de neurones prédisant le type de cancer.
La première étape a été de préparer les données : il s'agit de les formater afin d'isoler les *input features*, dans notre cas le niveau d'expression de chacun des gènes, ainsi que les *outputs*, dans notre cas le statut cancer / non cancer du patient ou bien le type du cancer.
Le jeu de données doit ensuite être divisé en trois ensembles disjoints :

- le *training set* : ce sont les données qui vont servir pour l'entraînement du réseau
- le *validation set* : ces données vont servir à évaluer les performances du réseau en faisant varier ses hyperparamètres (paramètres fixés manuellement qui interviennent dans l'entraînement et la structure du réseau)
- le *test set* : ayant paramétré le réseau en fonction des performances sur le *validation set*, on ne peut utiliser celui-ci pour mesurer les performances de manière non biaisée, d'où la nécessité d'avoir un *test set*

Pour mes expérimentations, j'ai réparti les données selon le schéma suivant : 70% (*train*) - 15% (*validation*) - 15% (*test*). Le nombre précis d'exemples est présenté dans la Table 3.

	MicroArray	TCGA
Training set (70%)	14870	6546
Validation set (15%)	3187	1403
Test set (15%)	3187	1403

Table 3 : Répartition des exemples dans le training/validation/test set

L'étape suivante a été la conception de la structure du réseau, c'est-à-dire le nombre d'*hidden layers* et leur nombre de neurones. Une spécificité des données transcriptomiques est le grand nombre d'*input features* (54675 sur le jeu MicroArray, 57387 sur le jeu TCGA). C'est un nombre qu'on retrouve assez classiquement lorsqu'on travaille avec des images aux vues du nombre de pixels, mais moins classiquement lorsqu'on travaille avec des données tabulaires. Pour les images, on peut utiliser des réseaux de neurones convolutifs, à savoir des réseaux utilisant certaines couches avec des filtres de convolution qui vont avoir pour conséquence de réduire la dimension des données. Mais ce principe repose sur la spatialité qu'on retrouve avec les données images, mais que les données transcriptomiques ne présentent pas. Si on représente un exemple d'entraînement par un vecteur de dimension égale au nombre de gènes, alors les performances du réseau de neurones seront les mêmes, peu importe l'ordre des éléments dans le vecteur. Il faut cependant s'assurer que cet ordre soit constant lors d'un

entraînement, c'est-à-dire qu'un gène ait toujours la même position, sans quoi l'entraînement serait impossible.
Le problème lié à ce grand nombre d'*input features* et au fait qu'on utilise un réseau entièrement connecté est le nombre de paramètres liés aux connexions entre l'*input layer* et la première *hidden layer*. Empiriquement, aux vues de la puissance des GPUs du laboratoire, la dimension maximale pour que le temps d'entraînement reste raisonnable est de l'ordre du millier de neurones. Vient ensuite la question de la dimension des *hidden layers*. Ce point est très arbitraire, il n'y a pas de règle permettant de déterminer de manière certaine une bonne valeur pour la dimension. On recommande toutefois de choisir une valeur de dimension comprise entre la dimension de l'*input layer* et celle de l'*output layer*. Certaines formules parlent de prendre pour valeur la moyenne entre la dimension de l'*input layer* et celle de l'*output layer,* d'autres de choisir la racine carrée de la dimension de l'*input layer*. Cette seconde formule a servi de point de départ dans la démarche de recherche d'une bonne structure du réseau. Pour le nombre d'*hidden layers*, il faut garder en tête qu'une seule *hidden layer* permet déjà de traiter des données non linéairement séparables. En pratique, il est rare que le fait de multiplier le nombre d'*hidden layers* ait un impact significatif sur les performances. Il convient alors de prendre une seule *hidden layer* comme point de départ, et d'ensuite incrémenter ce nombre si cela entraîne une amélioration des performances. En prenant en compte ces facteurs, on peut a priori avoir une idée d'un intervalle d'hyperparamètres à tester pour la structure du réseau. Afin de fixer des valeurs, j'ai effectué une *grid search*, ou recherche par quadrillage ; cela consiste à tester toutes les combinaisons de valeurs pour chacun des hyperparamètres qu'on souhaite vérifier.

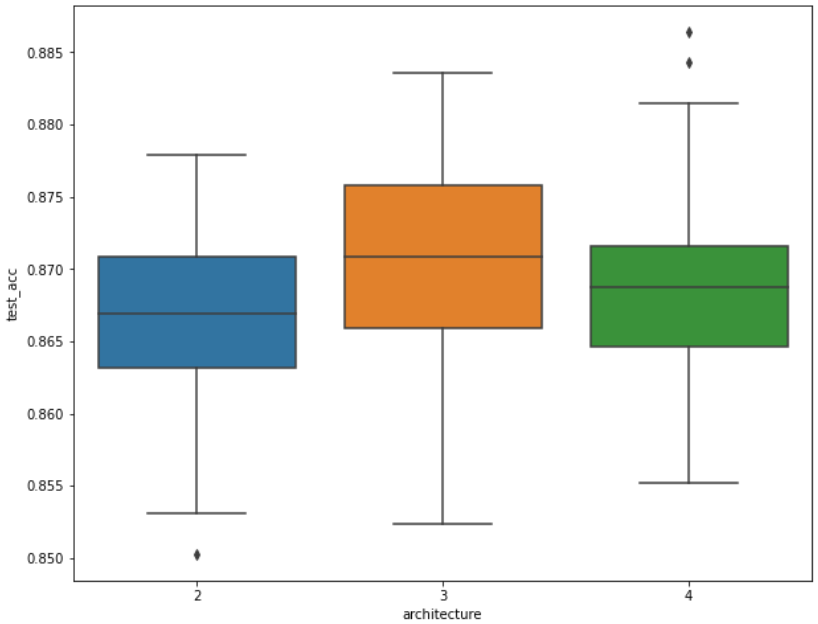

Figure 11 : Précision de classification "type de cancer" sur TCGA en fonction du nombre d'hidden layers (taille du training set réduite)

À la suite de cette *grid search*, j'ai abouti à une architecture comportant trois *hidden layers* (Figure 11), respectivement de dimension 512, 256 et 128.
Les paramètres liés à l'architecture du réseau de neurones ne sont pas les seuls à prendre en compte, il faut également fixer des hyperparamètres liés à l'entraînement du réseau, à commencer par la *learning rate*, ou taux d'apprentissage en français. Cette valeur intervient lors de l'optimisation des

valeurs des paramètres (*weights* et *biaises*) du réseau. En effet, après chaque itération d'entraînement, le gradient de l'erreur, calculée par la fonction de coût, permet de savoir la direction et la force avec lesquels optimiser les paramètres. La *learning rate* intervient en tant que coefficient multiplicateur du gradient ; elle va donc avoir un impact dans la rapidité du modèle à converger. Une *learning rate* trop élevé peut cependant conduire un modèle à diverger, et une *learning rate* trop faible peut amener le modèle à converger dans un minimum local, ainsi on risque de ne pas atteindre les performances optimales dans ces deux cas. Un second hyperparamètre à prendre en compte pour l'entraînement du modèle est le *mini-batch size*, communément raccourci en *batch size*. Ce paramètre définit le nombre d'exemples parmi l'ensemble d'entraînement à passer dans le réseau avant de calculer une étape d'optimisation des paramètres. Plus le *batch size* est grand, et plus le calcul du gradient de l'erreur d'entraînement va être précis. Les deux extrêmes sont ainsi de prendre un *batch* comportant l'entièreté de l'ensemble d'entraînement et de prendre un *batch* comportant un seul exemple d'entraînement. Dans le premier cas, le modèle met relativement beaucoup de temps à converger, car toutes les données d'entraînement doit être présentées au réseau avant de pouvoir l'optimiser ; cela peut causer une saturation de la mémoire, et de plus, l'introduction d'une certaine variance dans le calcul du gradient (en réduisant le *batch size*), peut être bénéfique à l'entraînement, en réduisant les chances de tomber dans un minimum local. Il faut alors prendre une valeur raisonnable, car un *batch size* trop faible introduit une trop grande variabilité. Un bon point de départ est classiquement une valeur autour de quelques dizaines d'exemples.
J'ai également effectué une *grid search* pour déterminer la valeur de ces hyperparamètres, ce qui m'a conduit à retenir une valeur de 0,0004 pour la *learning rate*, et une valeur de 256 pour le *batch size*.
Après avoir déterminé ces hyperparamètres en mesurant les performances sur le *validation set*, le *test set* a été mis à contribution pour effectuer une mesure non biaisée des performances. Les résultats sont les suivants :

- 0.937 de précision ($\sigma = 0.005$) pour la tâche de prédiction cancer / non-cancer sur les données MicroArray
- 0.963 de précision ($\sigma = 0.002$) pour la tâche de classification du type de cancer sur les données TCGA

Ces valeurs résultent de l'entraînement et de la mesure des performances du même réseau de neurones, et ce 50 fois. La performance finale du réseau n'est jamais exactement identique (même si les variations sont faibles), car l'entraînement du réseau dépend de l'initialisation des paramètres (*weights* et *biaises*) qui est aléatoire. Certaines initialisations sont, par hasard, meilleures que d'autres, conduisant à un entraînement donnant de meilleures performances finales. Il est ainsi important de répéter les expériences, pour obtenir une valeur de performance suffisamment précise.
Les performances obtenues sont très bonnes, car les jeux de données MicroArray et TCGA comportent suffisamment d'exemples pour que les performances ne soient pas bridées par un manque de données. Pour mieux s'en rendre compte, il est intéressant de tracer un graphique représentant les performances en fonction du nombre de d'exemples d'entraînement. En réduisant le jeu d'entraînement, on prend le soin de conserver les proportions initiales de chacune des classes, pour ne pas introduire de biais.

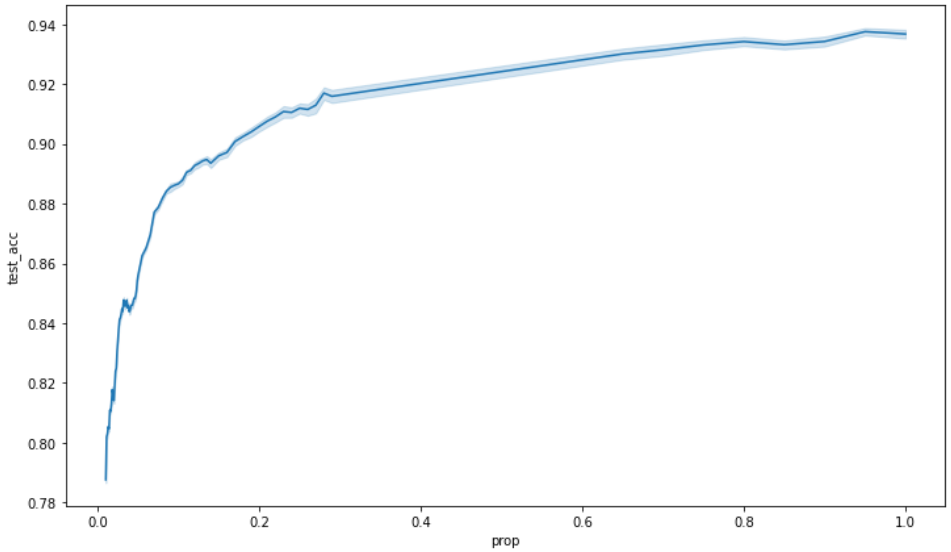

Figure 12 : Précision de classification "cancer / non cancer" sur MicroArray en fonction de la proportion de données conservées

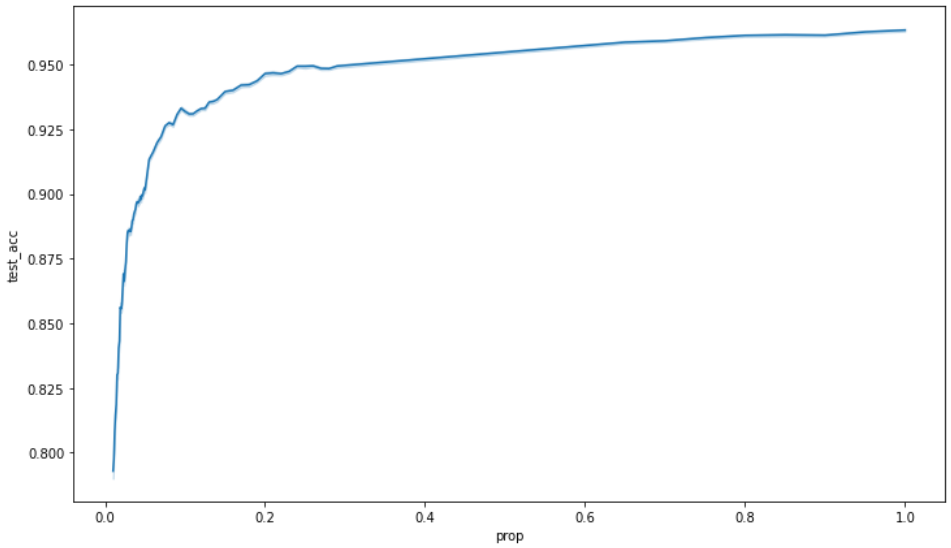

Figure 13 : Précision de classification "type de cancer" sur TCGA en fonction de la proportion de données conservées

Les Figures 12 et 13 présentent ainsi les mesures de précision pour les deux tâches de classification sur une centaine de tailles de *training set* différentes, avec au minimum seulement 1% des données conservées (cela correspond à 149 exemples pour MicroArray et 67 exemples pour TCGA) . On observe, pour les deux cas, que malgré avec un très faible nombre d'exemples, les performances restent acceptables. Mais on remarque également qu'à partir d'un certain nombre de données, la pente de la courbe diminue fortement. Au-dessus de ce nombre, ajouter des exemples supplémentaires continue à améliorer les performances, mais de manière bien moins significative qu'en dessous de ce nombre. C'est pour ces faibles tailles de jeu de données que l'apport des méthodes *self-supervised* est à priori le plus intéressant, car l'intégration de nouveaux exemples d'entraînement a le potentiel de résulter en une nette augmentation des performances.

Implémentation de la méthode SwAV

Le travail préliminaire présenté ci-dessus a permis d'obtenir des performances de base auxquelles se référer pour mesurer l'apport d'une approche d'entraînement *self-supervised*. La prochaine étape du travail est ainsi l'implémentation d'une de ces méthodes. Un stagiaire avait précédemment exploré les approches *self-supervised* de type *contrastive*, sans obtenir de résultat probant. Mon stage avait donc pour but de plutôt explorer les méthodes de *clustering*, avec pour espoir d'obtenir plus de succès. Pour choisir la méthode de *clustering* sur laquelle se baser, nous avons sélectionné la plus récente qui est également celle qui présente les meilleures performances sur des tâches de classification sur le jeu de données ImageNet [10] : la méthode SwAV.

Une implémentation Pytorch de la méthode SwAV développée par l'équipe de recherche est disponible sur GitHub : https://github.com/facebookresearch/swav.

Cette implémentation est cependant prévue pour reproduire les résultats présentés dans le papier, sur des tâches de vision par ordinateur. Ainsi, la structure générale de l'algorithme et certaines fonctions peuvent être directement récupérées, mais un certain travail d'adaptation du code pour les données transcriptomiques est nécessaire. La grosse différence se fait au niveau de la routine d'augmentation de données. Pour rappel, la phase d'apprentissage dans les méthodes *self-supervised* repose sur le principe de rapprocher deux versions différentes d'une même donnée. C'est lors de la génération de ces deux versions différentes que la routine d'augmentation intervient. Dans le domaine de l'image, on réalise classiquement deux rognages différents de l'image, rognages qui peuvent ensuite être transformés (changement de couleur / luminosité / contraste, déformations, rotations, symétries). Dans le domaine de la transcriptomique, ces augmentations sont incompatibles avec le type de la donnée. On peut néanmoins faire un parallèle entre le rognage des images et le masquage de certains gènes (Figure 14).

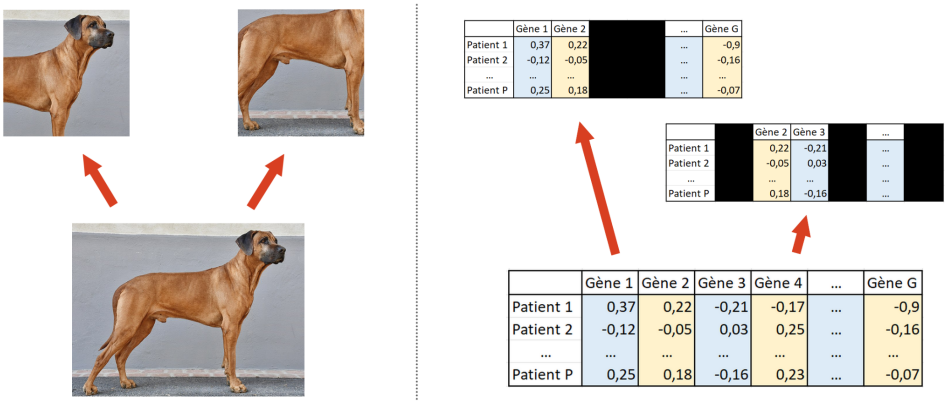

Figure 14 : Parallèle entre le rognage des images (gauche) et le masquage de gènes dans les données transcriptomiques (droite)

Nous verrons dans la suite de ce rapport que le masquage de gènes n'est pas la seule manière d'augmenter les données transcriptomique, mais c'était cette méthode qui a classiquement été utilisé pour les expérimentations que j'ai réalisées.

Point sur le *transfer learning*

Le transfert de la représentation apprise par SwAV pour servir de pré-initialisation à un second entraînement supervisé (*finetuning)* peut s'implémenter de différentes manières. Cette question rejoint le champ de recherche nommé *transfert learning*, ou apprentissage par transfert. En effet, le *transfer learning* est un ensemble de méthodes qui visent à transférer des connaissances d'une ou plusieurs tâches sources vers une ou plusieurs tâches cibles. Il est classique d'utiliser le *transfer learning* dans des problèmes de vision par ordinateur : il est en effet possible de récupérer des modèles ayant été entraînés pour une tâche de classification sur un jeu de données conséquent (ImageNet par exemple), puis de s'en servir comme initialisation pour un entraînement sur une tâche finale bien précise, par exemple la détection d'obstacle en milieu urbain. Grâce à ces méthodes, on observe généralement une hausse des performances, à condition que le choix des connaissances transférées soit judicieux. Un réseau de neurones entraîné sur ImageNet aura ainsi intégré des concepts généraux propres aux images : la compréhension des formes, des contours, des ombres, des différents plans d'une image, etc. Ces concepts parfaitement appris peuvent ainsi être valorisés dans diverses tâches dans le domaine de l'image. Dans ce cas, le principal de l'entraînement sera focalisé sur l'apprentissage des spécificités liées à la tâche cible, car l'étape d'apprentissage des concepts généraux de l'image aura été évitée grâce au transfert de connaissances de la tâche source.

Globalement, le *transfer learning* se matérialise par le transfert d'une où plusieurs couches du réseau de neurones source vers le réseau de neurones cible. Concrètement, plutôt qu'être aléatoirement initialisés, les paramètres (*weights* et *biaises*) sont initialisés avec les valeurs des paramètres (après entraînement) du réseau de neurones source. Il faut ainsi que les structures de réseaux source et cible soient compatibles ; si l'on souhaite transférer une couche, il faudra que sa dimension soit conservée.

Le *transfer learning* peut s'implémenter de différentes manières. Il faut d'abord décider du nombre de couches à transférer. Les travaux d'interprétation des réseaux de neurones ont montré que les concepts généraux sont identifiés par les premières *hidden layers* du réseau, et que les concepts plus spécifiques sont identifiés par les couches plus profondes. Il peut ainsi être intéressant de ne transférer que les premières *hidden layers* afin de ne pas apporter d'information superflue. Ensuite, on peut avoir recours au *freezing* (figeage) des couches transférées. En effet, si l'on considère que les couches transférées ont été optimalement entraînées pour capturer des concepts généraux, on ne souhaite pas que l'apprentissage sur la tâche cible vienne modifier les paramètres de ces couches, impactant ainsi négativement leur intérêt.

Cependant, ces principes sont propres à chacun des types de données, et ce qui vaut pour les images ne vaut pas forcément pour les données transcriptomiques.

J'ai ainsi expérimenté le fait de transférer une, deux ou bien les trois *hidden layers* pré-initialisée par SwAV. Il en ressort que le mieux est de transférer les trois *hidden layers* afin de maximiser les performances, bien que transférer uniquement les deux premières ne fasse pas de différence significative (Figure 15).

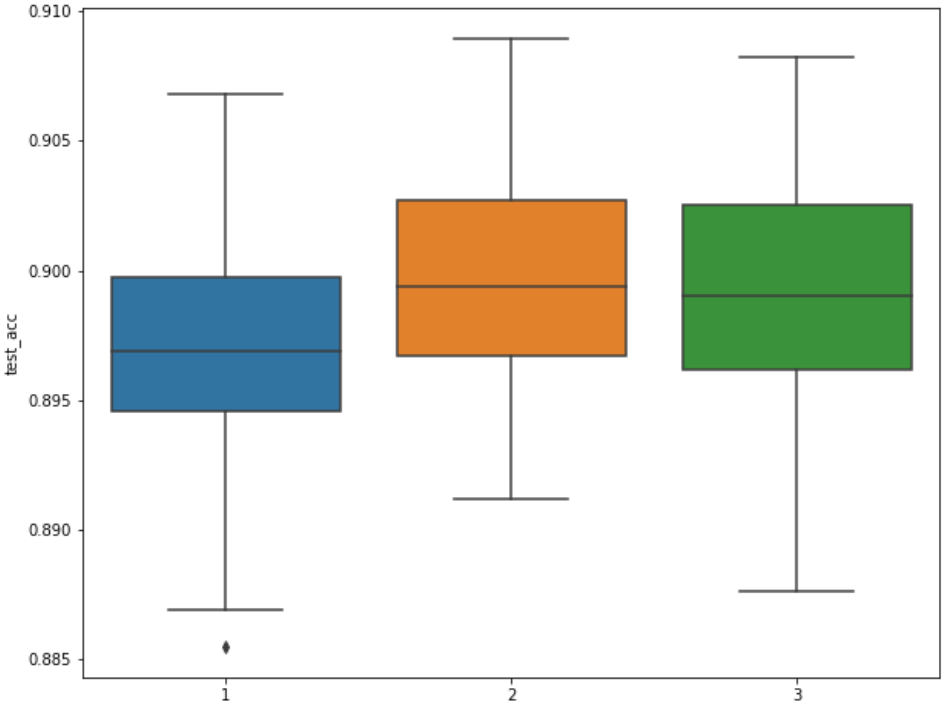

Figure 15 : Précision de classification "type de cancer" sur TCGA en fonction du nombre d'hidden layers tranférées (taille du training set réduite)

Les travaux de *transfer learning* réalisés à IBSIC sur les données MicroArray et TCGA ont montré que *freezing* des couches transférées est contre-productif, c'est un constat que j'ai également observé et qui est présenté par la Figure 16.

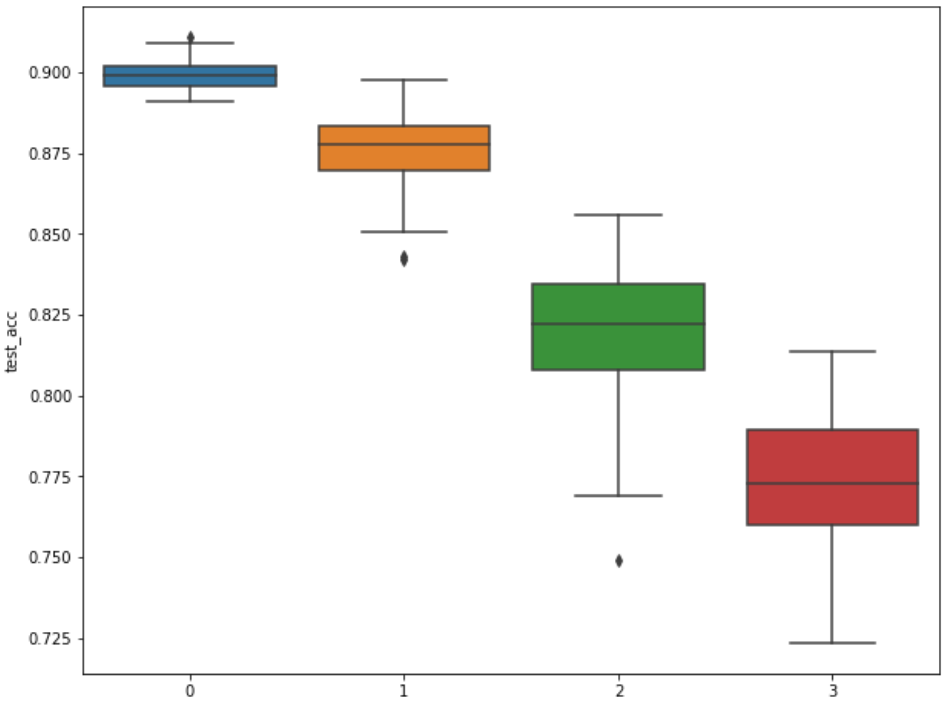

Figure 16 : Précision de classification "type de cancer" sur TCGA en fonction du nombre d'hidden layers figées (taille du training set réduite)

Pour résumer, les expérimentations que j'ai effectuées sur les données MicroArray et TCGA montrent que le mieux pour maximiser les performances sur la tâche cible est de transférer les trois *hidden layers* et de ne pas avoir recours au *freezing*.

Ce principe a été appliqué pour toutes mes expérimentations, qui sont présentées dans la suite de ce rapport.

Optimisation des hyperparamètres de la méthode SwAV

Pour rappel, les méthodes *self-supervised* reviennent à entraîner un réseau de neurones avec des données non labellisées, et l'état final sur réseau sert ensuite de pré-initialisation à en entraînement supervisé. La méthode SwAV implique des hyperparamètres, certains liés à l'entraînement du réseau de neurones, et d'autres qui sont des constantes qui interviennent dans de diverses fonctions intégrées dans l'algorithme SwAV.
Habituellement, on fixe les hyperparamètres d'un réseau de neurones en mesurant ses performances sur les données du jeu de validation. Le réseau de neurones entraîné par la méthode SwAV ne sert cependant que d'initialisation à un entraînement classique. Il n'est par exemple pas possible de mesurer les performances de classification du réseau entraîné par SwAV, car il n'a pas été entraîné pour ça. Il a été entraîné sur une tâche bien plus globale qui est de rapprocher les données similaires entre elles, dans le but d'obtenir une bonne représentation des données, qui pourra servir d'initialisation lors d'un entraînement supervisé sur une tâche finale (par exemple de classification cancer / non cancer), suivant la procédure de *transfer learning* décrite précédemment. Ainsi, cette représentation des données apprise par SwAV n'est pas évaluable directement, c'est plutôt la mesure des performances sur la tâche finale qui peut donner une évaluation de l'apport de la pré-initialisation apprise par SwAV. Si SwAV apprend une représentation des données bénéfique à un entraînement sur une tâche de classification cancer / non cancer, alors les performances sur la tâche finale avec *transfer learning* seront relativement supérieures à celles qui seraient obtenues avec une pré-initialisation moins bénéfique.

Nombre de prototypes

Un premier hyperparamètres à fixer est le nombre de prototypes, qui correspond pour rappel au nombre de *clusters*. On souhaite que le *clustering* appris par SwAV ait un rapport avec la tâche finale de classification. La situation idéale mais utopique serait que chacun des *clusters* corresponde à une classe de la tâche finale. Cependant, l'intuition qui en découle est qu'il faut au moins avoir autant de prototypes que de classes, afin de donner les moyens à SwAV de pouvoir tendre vers un *clustering* proche des classes réelles.
Le papier de SwAV indique ainsi que ce nombre de prototypes n'a pas de grande influence tant qu'il est suffisamment élevé. Les auteurs du papier ont ainsi utilisé un nombre de 3000 prototypes dans leurs expérimentations, pour un problème de classification avec 1000 classes. On peut ainsi prendre pour principe de base de choisir un nombre de prototypes égal à 3 fois le nombre de classes.

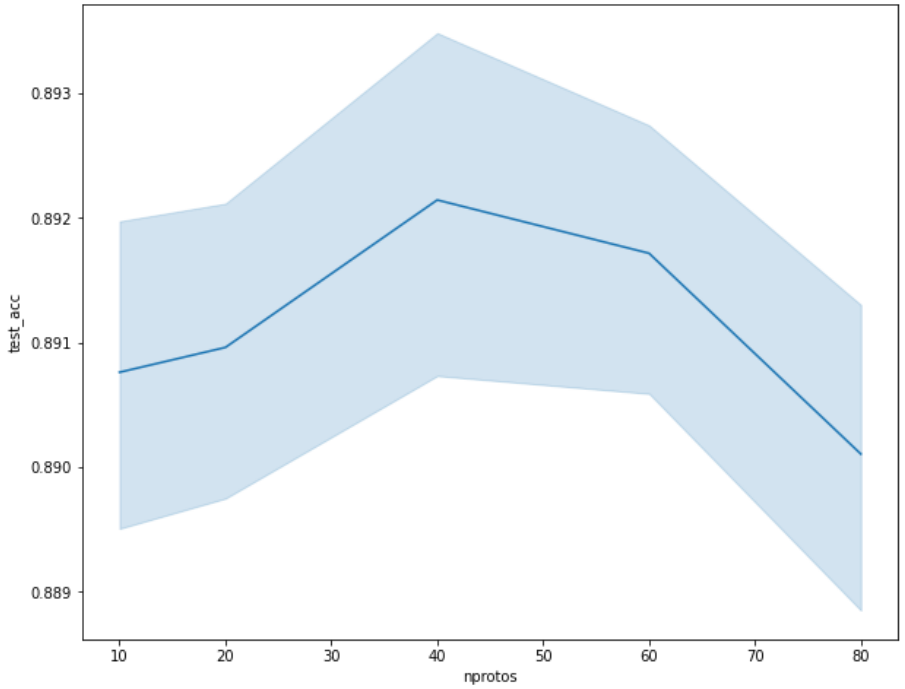

Figure 17 : Précision de classification "type de cancer" sur TCGA en fonction du nombre de prototypes (taille du training set réduite)

La Figure 17 montre qu'un nombre avoisinant les 50 prototypes est le mieux pour l'entraînement de SwAV sur les données TCGA. Pour rappel, le problème de classification *"type de cancer"* sur TCGA comporte 20 classes (19 types de cancers + 1 classe "pas de cancer"). On a bien un nombre de prototypes proche de 3 fois le nombre de classes, cependant on observe une baisse des performances quand le nombre de prototypes augmente, fait qui n'a pas pu être expliqué dans le cadre de ce stage.

De la même manière, pour les données MicroArray, un nombre de 16 prototypes a été sélectionné (Figure 18).

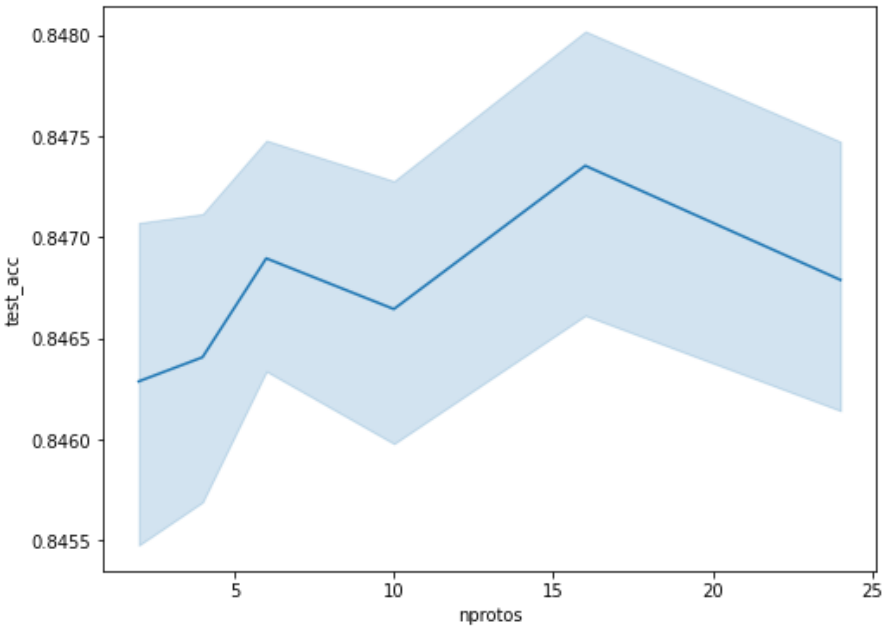

Figure 18 : Précision de classification "cancer / non cancer" sur MicroArray en fonction du nombre de prototypes (taille du training set réduite)

Soft versus *hard labelling*

Pour rappel, SwAV doit correctement prédire le prototype assigné à l'augmentation 1 de la donnée d'entrée à partir du vecteur latent associé à l'augmentation 2 de la donnée d'entrée. Au niveau de l'implémentation de l'algorithme, cette assignation au prototype prend la forme d'un vecteur normalisé de scores d'appartenance à chacun des k prototypes. Ce vecteur de score peut être plus ou moins lissé, c'est-à-dire que les scores vont être plus ou moins proches les uns des autres.

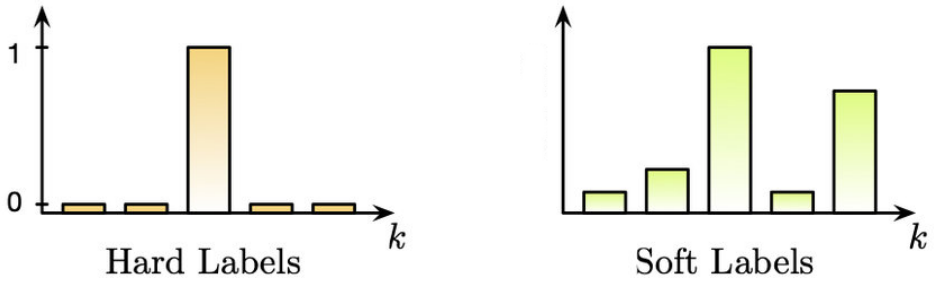

Figure 19 : Illustration du hard et du soft labelling (Engelmann et al.)

Deux situations peuvent alors se dégager, comme illustré Figure 19 :

- un score peut être proche de 1, les autres étant proches de 0 : c'est le *hard labelling*
- sinon, les scores peuvent avoir une distribution davantage homogène, c'est ce qu'on appelle le *soft labelling*

Le lissage de ce vecteur de scores peut être ajusté dans l'algorithme SwAV via le réglage d'une constante ε. Concrètement, cette constante joue le rôle de paramètre de régularisation dans l'algorithme Sinkhorn-Knopp. Les auteurs de SwAV montrent un léger avantage à utiliser un *soft labelling*.

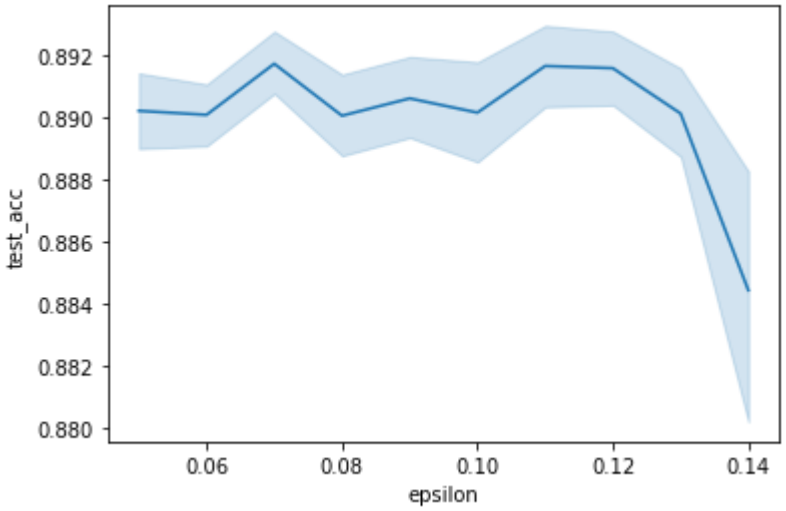

Figure 20 : Précision de classification "type de cancer" sur TCGA en fonction du paramètre ε (taille du training set réduite)

La Figure 20 présente les résultats obtenus sur TCGA ; au plus ε est grand, au plus le *labelling* est *soft*. Les performances de classification sont stables peu importe la valeur du paramètre ε, jusqu'à un certain point ($\varepsilon > 0.13$) ou les performances chutent rapidement. En effet, on peut vérifier empiriquement que pour de telles valeurs, le vecteur de score subit une régularisation telle que les scores sont tous égaux. De ce fait, l'algorithme SwAV ne peut pas apprendre une quelconque représentation des données, d'où la chute des performances.

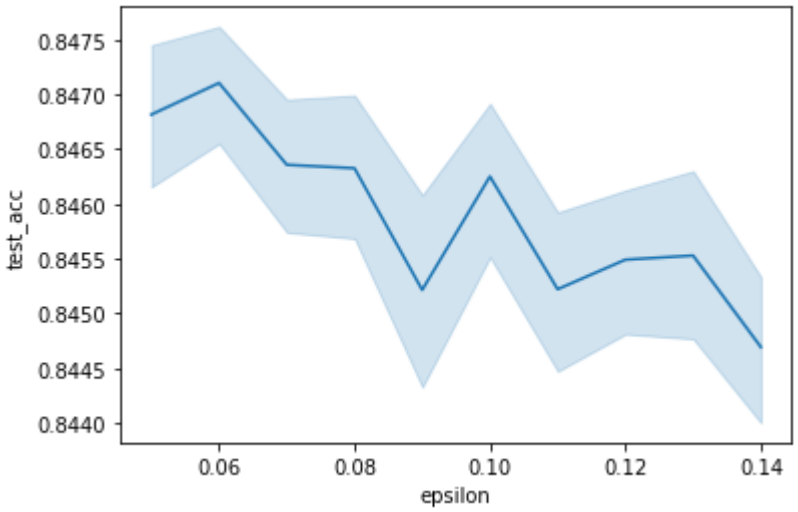

Figure 21 : Précision de classification "cancer / non cancer" sur MicroArray en fonction du paramètre ε (taille du training set réduite)

Sur les données MicroArray (Figure 21), on observe que les performances se détériorent au plus le *labelling* est *soft*.
Sur les deux de données, un *labelling hard* semble ainsi préférable afin de maximiser les performances.

Batch size SwAV

Comme pour n'importe quel réseau de neurones, le *batch size* est également un paramètre à prendre en compte pour l'entraînement de SwAV. Le fonctionnement de SwAV suppose de choisir un *batch size* au moins aussi grand que le nombre de prototypes : en effet, pour ne pas que l'algorithme ne s'effondre (situation menant à une solution triviale), il faut assurer au mieux une contrainte d'équirépartion au sein des prototypes. Plus explicitement, il faut qu'au niveau d'un *batch*, les exemples soient assignés aux prototypes de manière équitable, autrement dit les prototypes doivent être sélectionnés le même nombre de fois. Cette contrainte n'est évidement pas satisfaisable si le *batch size* est inférieur au nombre de prototypes.

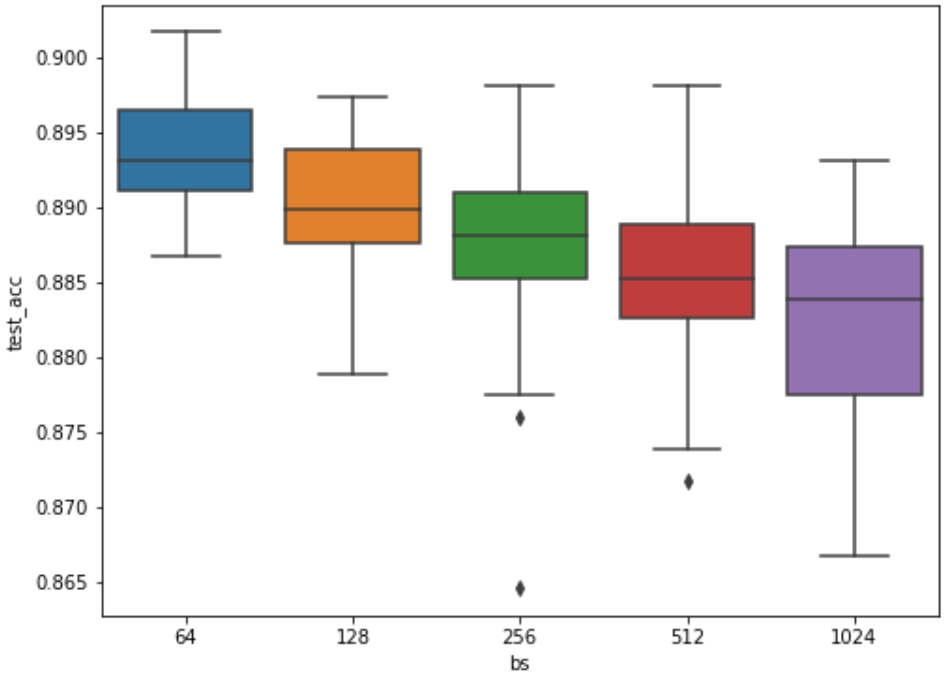

Figure 22 : Précision de classification "type de cancer" sur TCGA en fonction du batch size (taille du training set réduite)

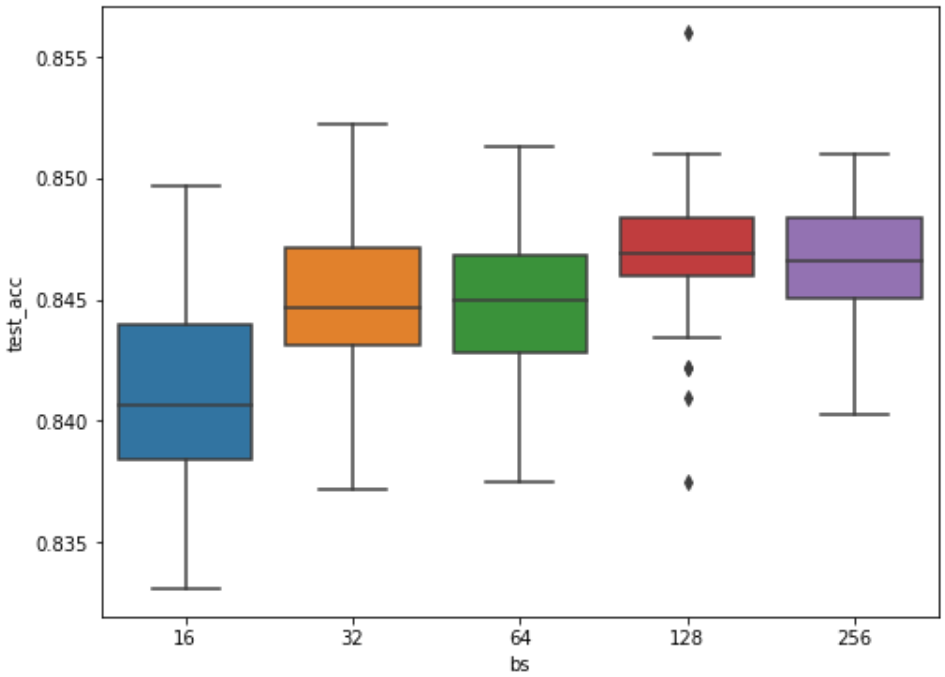

Figure 23 : Performances de classification "cancer / non cancer" sur MicroArray en fonction du batch size (taille du training set réduite)

Les expérimentations, dont les résultats sont présentés sur les Figures 22 et 23, montrent qu'un petit *batch size* est préférable sur le jeu de données TCGA, alors qu'un plus grand *batch size* donne de meilleurs résultats sur le jeu de données MicroArray.

Méthode d'augmentation

Comme expliqué dans la partie sur l'implémentation de SwAV, la méthode par défaut que j'ai utilisée pour l'augmentation des données transcriptomique est le masquage de gènes, qui est l'équivalent du rognage dans le domaine de l'image.
Concrètement, la procédure de masquage de gènes est réalisée au niveau d'un *batch* de données d'entraînement. On peut se représenter ce *batch* comme un tableau de taille M * N, avec M le nombre de patients dans un *batch* (*batch size*) et N le nombre de gènes. Pour ce *batch*, un sous-ensemble de gènes (de colonnes dans le tableau) est sélectionné aléatoirement. On peut faire varier la taille de ce sous-ensemble, en fonction du point auquel on souhaite masquer les données. Les valeurs présentes sur ces colonnes sont ensuite masquées : on peut penser à remplacer leur valeur par 0, mais il est plus judicieux de remplacer leur valeur par des nombres tirés aléatoirement à partir d'une distribution normale centrée réduite (les données étant également préalablement centrées et réduites).
Cependant, ce n'est pas le seul moyen imaginable pour augmenter ce type de données : on peut aussi ajouter du bruit gaussien, plus ou moins fortement. Ici, il suffit de générer un tableau de taille M * N rempli de valeurs tirées aléatoirement à partir d'une distribution normale centrée réduite. On applique ensuite un coefficient multiplicateur aux valeurs de ce tableau, puis on l'ajoute au tableau des données d'entraînement.

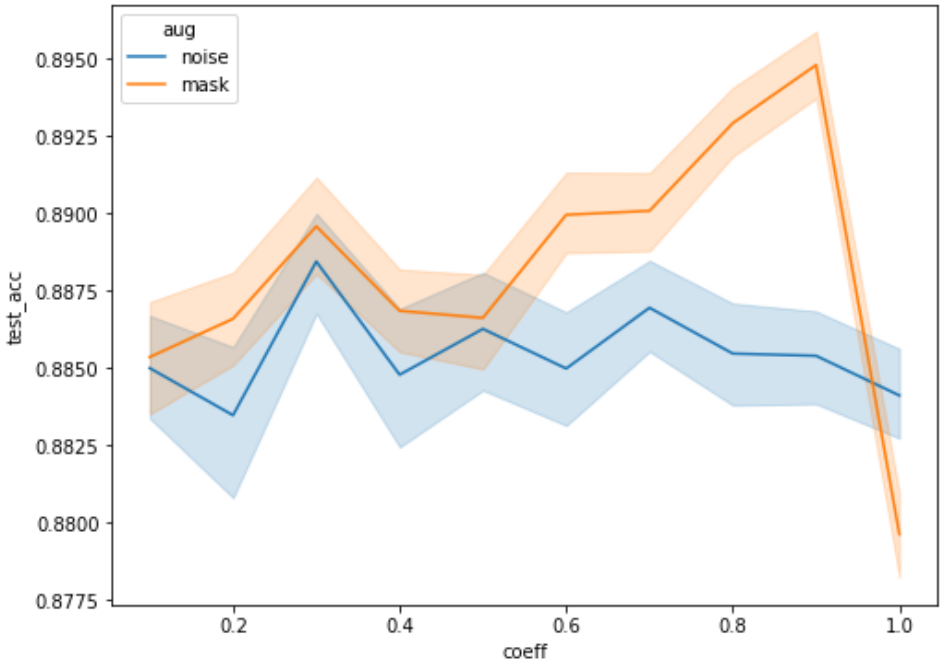

Figure 24 : Précision de classification "type de cancer" sur TCGA en fonction de la méthode d'augmentation (taille du training set réduite)

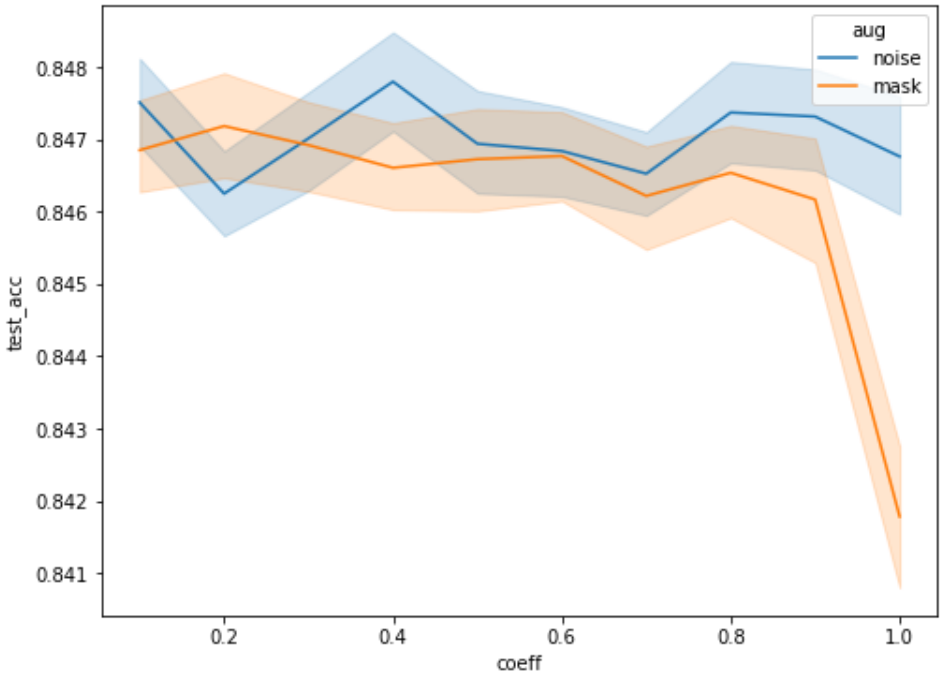

Figure 25 : Précision de classification "cancer / non cancer" sur MicroArray en fonction de la méthode d'augmentation (taille du training set réduite)

Ces deux manières d'augmenter les données ont été comparées expérimentalement sur TCGA (Figure 24) et MicroArray (Figure 25). La variable "coeff" de l'axe des abscisses joue un rôle différent en fonction de la méthode d'augmentation :

- pour la méthode avec masquage des données (courbes oranges), la variable indique la proportion de gènes masqués
- pour la méthode avec bruit gaussien (courbes bleues), la variable est le coefficient multiplicatif du tableau de bruit

Sur le jeu de données TCGA, les performances sont les meilleures avec la méthode d'augmentation utilisant le masquage des gènes, pour de grandes proportions de gènes masqués. On remarque que les performances s'effondrent lorsque tous les gènes sont masqués, et c'est logique, car dans ce cas SwAV s'entraîne intégralement sur des données générées aléatoirement.

Sur le jeu de données MicroArray, l'effet est moins marqué : les deux méthodes semblent équivalentes en termes de performances, et stables peu importe la valeur de la variable "coeff". On observe également la chute de performances pour la méthode avec masquage des données, en masquant tous les gènes.

Sur la fin de mon stage, j'ai exploré une troisième manière d'augmenter les données, à l'aide d'un auto-encodeur. Un auto-encodeur est une structure de réseau de neurones avec une *input layer* de même dimension que l'*output layer*, et avec des *hidden layer* de dimension inférieures, formant une structure en forme de sablier orienté horizontalement (Figure 26).

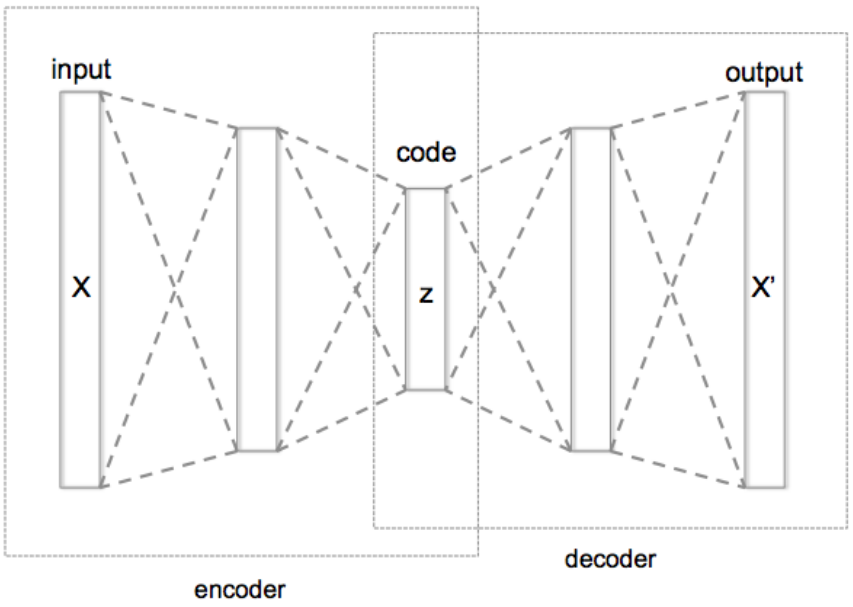

Figure 26 : Structure schématique d'un auto-encodeur (Chervinskii)

Conceptuellement, il s'agit d'un *encoder*, qui réduit la dimension des données, suivi d'un *decoder*, qui rétablit la dimension des données. Au centre, la représentation des données obtenue est appelée vecteur latent (z). Ce vecteur est d'une dimension classiquement largement inférieure à la dimension des données d'entrée, mais on espère idéalement qu'il contienne suffisamment d'information pour reconstruire la donnée. En effet, lors de l'entraînement d'un auto-encoder, le coût est fonction de l'écart entre les données d'entrée et de sortie, le but étant de réduire cet écart au possible.

Pour augmenter un exemple avec un auto-encoder, il est possible de faire passer cet exemple dans la partie *encoder* et d'ainsi obtenir sa représentation dans l'espace latent. On peut ensuite choisir aléatoirement deux points de l'espace latent plus ou moins proches de cette représentation, et passer ensuite ces deux points dans la partie *decoder*. On peut ainsi espérer obtenir deux données différentes, générées depuis un même exemple d'entraînement, ce qui respecte le principe de l'augmentation des données expliqué précédemment.

Dans les faits, j'ai pu correctement entraîner un auto-encoder à générer des données augmentées, mais les performances de classification obtenues n'ont pas été encourageantes. Il se peut que les données sortant de l'auto-encodeur soit trop différentes, et que la représentation apprise par SwAV ne profite pas lors de l'entraînement sur les données originales.

Mesure des performances

Après avoir expérimenté différentes valeurs pour les hyperparamètres de SwAV sur les jeux de données MicroArray (classification cancer / non cancer) et TCGA (classification type de cancer), j'ai sélectionné celles qui donnaient les meilleures performances, afin d'entraîner un modèle SwAV optimisé. Il s'agit maintenant de savoir si le pré-entraînement apporté par la méthode SwAV entraîne une hausse de performance significative, par rapport à un modèle classique, sans pré-entraînement.

Cette mesure de performance est réalisée en faisant varier le nombre de données labellisées, mais avec un nombre de données non labellisées fixé (l'entièreté du jeu de données). Par exemple, supposons qu'on conserve les *labels* d'uniquement 10% des exemples d'un *training set* contenant 1000 exemples :

- pour la méthode SwAV, on pré-initialise le modèle avec la méthode SwAV en utilisant les 1000 exemples (sans utiliser les *labels*), puis on effectue le *fine-tuning* du réseau avec les 100 exemples labellisés
- pour la méthode classique, ou "*baseline*", le modèle est initialisé aléatoirement, puis est entraîné classiquement avec les 100 exemples labellisés

Dans l'exemple ci-dessus, 10% des *labels* étaient conservés, mais pour mes mesures de performances, j'ai sélectionné 89 proportions différentes, allant de 1% à 100% de *labels* conservés. Cela permet de mesurer l'impact de la méthode SwAV selon la taille du jeu de données labellisé.

Pour chacune des 89 proportions, et pour chacune des deux méthodes (SwAV et *baseline*), 50 entraînements de modèles et mesures de performances sont réalisés afin d'avoir des résultats précis.

Résultats sur le jeu de données MicroArray

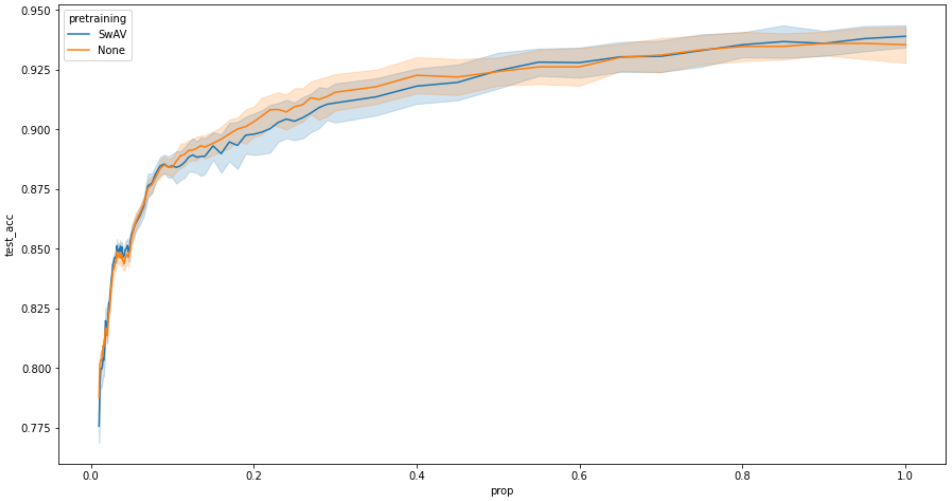

Figure 27 : Précision de classification "cancer / non cancer" sur MicroArray en fonction du pré-entraîement et de la proportion de labels conservés

La Figure 27 présente la comparaison des performances avec pré-entraînement SwAV (courbe bleue), et sans pré-entraînement (*baseline*, courbe orange). On remarque que les courbes sont souvent confondues, cela signifie que l'approche *self-supervised* n'apporte pas de gain de performance, et ce, peu importe le nombre de *labels*.

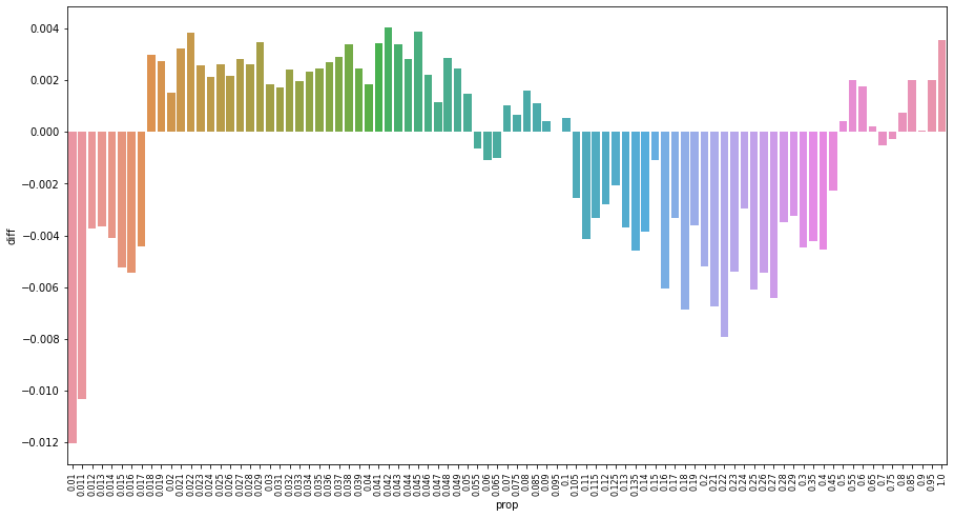

Figure 28 : Différence absolue entre la précision avec le pré-entraînement SwAV et la précision sans pré-entraînement (classification "cancer / non cancer" sur MicroArray)

La Figure 28 présente la différence absolue, pour chacune des 89 proportions tailles de jeu de données labellé, entre la performance avec le pré-entraînement SwAV et la baseline ($diff = précision\ SwAV - précision\ baseline$). On voit que les différences sont minimes, et sont parfois positives (avantage SwAV) mais aussi parfois négatives (avantage *baseline*).

Les résultats ainsi obtenus montrent que l'utilisation d'un pré-entraînement *self-supervised* avec l'algorithme SwAV sur le jeu de données MicroArray pour la tâche de classification "cancer / non cancer" ne présente pas d'intérêt.

Résultats sur le jeu de données TCGA

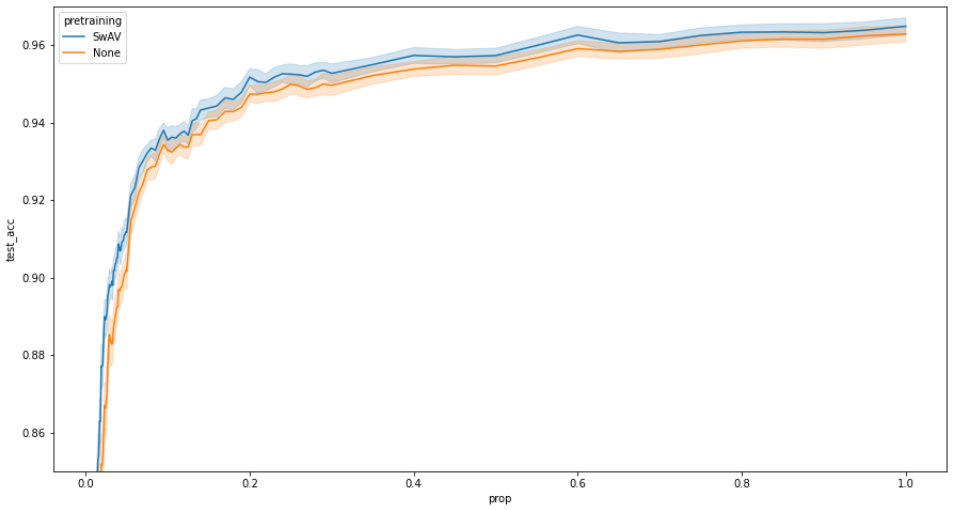

Figure 29 : Précision de classification "type de cancer" sur TCGA en fonction du pré-entraînement et de la proportion de labels conservés

Les résultats présentés par la Figure 29 sont quant à eux positifs. La courbe bleue, représentant les performances avec le pré-entraînement SwAV, est constamment au-dessus de la courbe orange représentant la *baseline*.

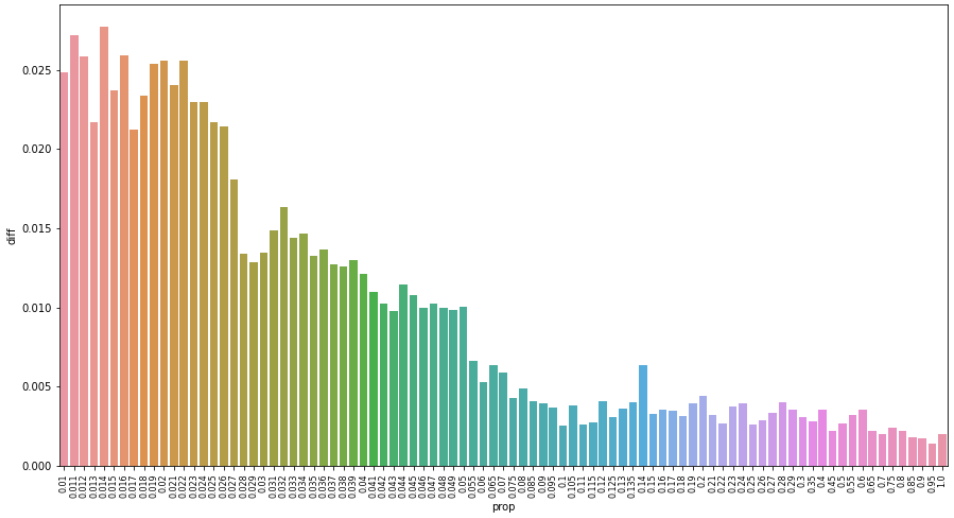

Figure 30 : Différence absolue entre la précision avec le pré-entraînement SwAV et la précision sans pré-entraînement (classification "type de cancer" sur TCGA)

Ces résultats sont confirmés par la Figure 30. La différence absolue est toujours positive, et d'autant plus importante lorsque la taille du jeu de données labellisées est faible, ce qui est un résultat attendu. Cette différence atteint 2,5 points sur certaines proportions, ce qui est n'est pas négligeable. Les tests de significativité statistique de Student (T-test) rapportent un p-valeur inférieure (souvent très largement) à 0.001 pour chacune des 89 tailles de jeu de données labellisées, ce qui montre que les écarts observés sont significatifs.
Les résultats ainsi obtenus montrent que l'utilisation d'un pré-entraînement *self-supervised* avec l'algorithme SwAV sur le jeu de données TCGA pour la tâche de classification "type de cancer" présente un intérêt significatif.
Une remarque très intéressante, est que même en gardant la totalité des *labels* (prop = 1), le pré-entraînement SwAV est meilleur qu'un pré-entraînement aléatoire, même si la différence est faible. L'approche *self-supervised* peut donc présenter un intérêt même lorsque le nombre de données labellisées semble suffisant.

Cette différence de résultats avec le jeu de données MicroArray peut s'expliquer par deux facteurs. Les données du jeu de données TCGA sont plus homogènes, car le jeu de données MicroArray est issu de la compilation de plusieurs études, chacune étant dotées de leur propre protocole expérimental. La tâche de classification est, elle aussi, différente. Sur TCGA, il s'agit de classification multiclasse : c'est également le cas dans les expériences présentées dans le papier de SwAV ; au contraire, sur le jeu de données MicroArray, la tâche de classification est binaire et on peut ici imaginer que l'approche par *clustering* de SwAV présente moins d'intérêt.

Conclusion

Mon stage au sein du laboratoire IBISC de l'Université d'Évry Paris-Saclay fût la dernière étape de mon cursus à Université de Technologie de Compiègne. Ce stage vient ainsi conclure une formation complète d'ingénieur, avec une spécialisation dans l'intelligence artificielle et la science des données. J'ai pu y mobiliser les différentes connaissances théoriques et compétences techniques qui me permettent aujourd'hui d'être prêt à entrer dans le monde du travail sur un poste d'ingénieur.

L'objectif de mon stage était de mesurer l'impact des méthodes d'apprentissage *self-supervised* pour les données transcriptomiques. J'ai commencé mon travail par la rédaction d'un état de l'art des méthodes *self-supervised*, ce qui m'a permis de cibler une méthode performante et adaptable au problème, à savoir la méthode SwAV. J'ai continué mes travaux en implémentant cette méthode pour les données transcriptomiques, suite à quoi j'ai pu déterminer les meilleurs réglages pour rendre la procédure la plus performante possible. Enfin, j'ai établi une procédure d'expérimentation qui a permis de montrer un impact positif significatif de l'algorithme SwAV sur un des deux jeux de données étudiés.

Effectuer un stage dans un laboratoire de recherche fût pour moi une expérience particulièrement enrichissante. J'ai aujourd'hui beaucoup plus de facilités à comprendre un article scientifique dans le domaine de l'intelligence artificielle, ce qui me permet maintenant de pouvoir me renseigner des dernières avancées dans le domaine par moi-même. J'ai également développé une rigueur scientifique indispensable en recherche, afin de présenter des résultats précis, non biaisés et reproductibles, qui pourront servir de base solide pour de futurs travaux.

Les résultats positifs que j'ai obtenus au terme de mon stage ouvrent une nouvelle piste de recherche qui pourra être approfondie au sein du laboratoire IBISC, par exemple à travers un sujet de thèse. Un papier scientifique rapportant les résultats sera également rédigé et soumis pour publication : il s'agit de montrer qu'une méthode *self-supervised*, initialement prévue pour des problématiques relatives au domaine de la vision par ordinateur, est adaptable pour une utilisation sur des données transcriptomiques, et présentent des résultats significatifs.

Plus personnellement, ce stage m'a permis de développer mes connaissances théoriques et techniques dans le domaine dans lequel je me suis spécialisé lors de mes études, à savoir l'intelligence artificielle. Au cours de ces 6 mois, j'ai particulièrement étudié l'apprentissage *self-supervised*, ce qui m'a permis d'être aujourd'hui vraiment renseigné et compétant dans ce domaine actuellement en vogue, car très prometteur et très promu par des acteurs influents. Mais j'ai pu plus généralement satisfaire mon désir d'explorer davantage le sujet de l'apprentissage profond, car j'ai travaillé durant 6 mois avec des personnes spécialisées dans le domaine, qui ont pu partager avec moi leurs conseils, connaissances et pratiques.

Références

[1] He, Kaiming, et al. "Momentum contrast for unsupervised visual representation learning." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020. *arXiv:1911.05722 [cs]* [en ligne]. Novembre 2019. Disponible à l'adresse : https://arxiv.org/abs/1911.05722. ArXiv: 1911.05722.

[2] Chen, Ting, et al. "A simple framework for contrastive learning of visual representations." *International conference on machine learning*. PMLR, 2020. *arXiv:2002.05709 [cs]* [en ligne]. Février 2020. Disponible à l'adresse : https://arxiv.org/abs/2002.05709. ArXiv: 2002.05709.

[3] Chen, Ting, et al. "Big self-supervised models are strong semi-supervised learners." *Advances in neural information processing systems* 33 (2020): 22243-22255. *arXiv:2006.10029 [cs]* [en ligne]. Juin 2020. Disponible à l'adresse : https://arxiv.org/abs/2006.10029. ArXiv: 2006.10029.

[4] Grill, Jean-Bastien, et al. "Bootstrap your own latent-a new approach to self-supervised learning." *Advances in neural information processing systems* 33 (2020): 21271-21284. *arXiv:2006.07733 [cs]* [en ligne]. Juin 2020. Disponible à l'adresse : https://arxiv.org/abs/2006.07733. ArXiv: 2006.07733.

[5] Caron, Mathilde, et al. "Deep clustering for unsupervised learning of visual features." *Proceedings of the European conference on computer vision (ECCV)*. 2018. *arXiv:1807.05520 [cs]* [en ligne]. Juillet 2018. Disponible à l'adresse : https://arxiv.org/abs/1807.05520. ArXiv: 1807.05520.

[6] MacQueen, J. "Classification and analysis of multivariate observations." *5th Berkeley Symp. Math. Statist. Probability.* 1967. Disponible à l'adresse : https://projecteuclid.org/ebooks/berkeley-symposium-on-mathematical-statistics-and-probability/Some-methods-for-classification-and-analysis-of-multivariate-observations/chapter/Some-methods-for-classification-and-analysis-of-multivariate-observations/bsmsp/1200512992

[7] Caron, Mathilde, et al. "Unsupervised learning of visual features by contrasting cluster assignments." *Advances in Neural Information Processing Systems* 33 (2020): 9912-9924. *arXiv:2006.09882 [cs]* [en ligne]. Juin 2020. Disponible à l'adresse : https://arxiv.org/abs/2006.09882. ArXiv: 2006.09882.

[8] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016. *arXiv:512.03385 [cs]* [en ligne]. Décembre 2015. Disponible à l'adresse : https://arxiv.org/abs/512.03385. ArXiv: 512.03385.

[9] Sinkhorn, Richard, and Paul Knopp. "Concerning nonnegative matrices and doubly stochastic matrices." *Pacific Journal of Mathematics* 21.2 (1967): 343-348. Disponible à l'adresse : https://projecteuclid.org/journals/pacific-journal-of-mathematics/volume-21/issue-2

/Concerning-nonnegative-matrices-and-doubly-stochastic-matrices/pjm/1102992505.full

[10] Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009. Disponible à l'adresse : https://ieeexplore.ieee.org/document/5206848