

---

Rapport de Stage  
**Self-Supervised Deep Learning sur  
Données Transcriptomiques**

---

Flora CARRÉ

Stage effectué du 6 avril 2021 au 1er octobre 2021

*Sous la direction de :*

Blaise HANCZAR  
Victoria BOURGEAIS

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Laboratoire IBISC . . . . .	1
1.2	Sujet du stage . . . . .	1
1.3	Etat de l'Art . . . . .	2
<b>2</b>	<b>Modèles</b>	<b>3</b>
2.1	Perceptrons multi-couches . . . . .	3
2.2	Modèle Supervisé de Référence . . . . .	5
2.3	Self-supervised : Prévision de données masquées . . . . .	5
2.3.1	Modèle de prédiction de données masquées . . . . .	5
2.3.2	Fine-tuning . . . . .	7
2.3.2.1	Fine-tuning avec données étiquetées . . . . .	7
2.3.2.2	Fine-tuning avec données étiquetées et données prédites non étiquetées . . . . .	7
2.4	Contrastive Learning . . . . .	8
<b>3</b>	<b>Résultats</b>	<b>11</b>
3.1	Données . . . . .	11
3.1.1	MicroArray . . . . .	11
3.1.2	TCGA . . . . .	11
3.2	Modèle Supervisé de Référence . . . . .	12
3.2.1	Implémentation . . . . .	12
3.2.2	Courbe de référence . . . . .	13
3.3	Self-supervised : Prévision de données masquées . . . . .	14
3.3.1	Modèle de prédiction de données masquées . . . . .	14
3.3.1.1	Implémentation . . . . .	14
3.3.1.2	Résultats . . . . .	14
3.3.2	Fine-tuning . . . . .	15
3.3.2.1	Fine-tuning avec données étiquetées . . . . .	15
3.3.2.1.1	Implémentation . . . . .	15
3.3.2.1.2	Résultats . . . . .	15
3.3.2.2	Fine-tuning avec données étiquetées et données prédites non étiquetées . . . . .	19
3.3.2.2.1	Implémentation . . . . .	20
3.3.2.2.2	Résultats . . . . .	20
3.4	Contrastive Learning . . . . .	23
3.4.1	Modèle de contrastive learning . . . . .	23
3.4.1.1	Implémentation . . . . .	23
3.4.1.2	Résultats . . . . .	24
3.4.2	Fine-tuning . . . . .	25
<b>4</b>	<b>Conclusion</b>	<b>30</b>
<b>Annexes</b>		<b>iii</b>

# Introduction

## 1.1 Laboratoire IBISC

Le laboratoire IBISC (Informatique, BioInformatique, Systèmes Complexes) est un laboratoire de l'Université d'Évry Paris-Saclay. Il est implanté sur deux sites : IBGBI et Pelvoux. Dans chaque site, se trouvent deux équipes de recherche.

Sur le site Pelvoux, on trouve l'équipe IRA2 (Interaction, Réalité virtuelle & Augmentée, Robotique Ambiante) et l'équipe SIAM (Signal, Image, AutoMatique).

Sur le site IBGBI (Institut de Biologie Génétique et BioInformatique), on trouve l'équipe AROBAS (Algorithmique, Recherche Opérationnelle, Bioinformatique et Apprentissage Statistique) et l'équipe COSMO(COmmunications Spécifications MOdèles).

J'ai pris part, pendant ce stage, à l'équipe AROBAS. Cette équipe s'oriente autour de trois axes.

Le premier axe est "algorithmique et recherche opérationnelle". Il se concentre sur la conception d'algorithmes dans le domaine de l'optimisation, de la théorie des jeux et de la recherche opérationnelle, comme par exemple de tuiles artificielles d'ADN pouvant faire des calculs.

Le deuxième axe est la bio-informatique. Il concerne les méthodes pour l'analyse des ARN non-codants et la prédiction de structure d'ARN et de leurs interactions.

Le troisième axe, celui correspond à mon sujet de stage, concerne l'apprentissage statistique, avec notamment l'application de l'apprentissage profond dans le domaine de la santé. Les principaux problèmes relèvent du faible nombre d'exemples d'apprentissage disponibles et de l'interprétation des réseaux de neurones et de leurs prédictions.

## 1.2 Sujet du stage

L'apprentissage profond, approche de l'apprentissage automatique, est une avancée majeure de l'intelligence artificielle de ces dernières années. Il consiste à apprendre un réseau de neurones à réaliser une tâche de prédiction à partir d'un ensemble de données d'apprentissage. On le retrouve principalement en analyse d'image ou en traitement du langage naturel. Un de ses enjeux futurs est son application à la santé.

Dans le domaine de la santé, les recherches se concentrent plus spécifiquement sur la prédiction de phénotypes comme des diagnostics, des pronostiques, ou encore des réponses aux traitements. Les données utilisées sont alors des données omiques, par exemple d'expression de gènes ou de protéines. Je me concentre dans ce stage sur des données transcriptomiques.

Pour avancer dans le deep learning, un verrou majeur à lever est l'apprentissage de réseau de neurones à partir de jeux d'apprentissage de petite taille, ce qui est un grand problème lorsque l'on travaille sur des données médicales. Alors que les réseaux de neurones en image ou langage naturel sont construits à partir de très grands jeux de données contenant des centaines de milliers, voire des millions d'exemples, les jeux de données médicales, notamment transcriptomiques, n'en contiennent que beaucoup moins, quelques milliers au mieux. Cette grande différence s'explique par le coût de leur acquisition, évidemment beaucoup plus élevé pour des données médicales. À cause de ce faible nombre d'exemples, l'apprentissage des réseaux de neurones se heurte à des problèmes de sur-apprentissage où le réseau apprend par cœur les données et non leur concept sous-jacent.

Pour remédier à ce problème, ce stage consiste à explorer différentes approches du self-supervised learning (Figure 1.1). Ces approches consistent à utiliser un jeu de données secondaires de grande taille contenant notamment des données non étiquetées, pour pouvoir apprendre, grâce à une tâche prétexte, une bonne représentation de celles-ci dans les couches cachées du réseau neurones. Cette représentation des données sera ensuite utilisée afin de rendre la tâche de prédiction voulue plus facile et donc éviter le sur-apprentissage des données.

Le but de ce stage est d'adapter des approches de self-supervised deep learning actuellement performantes dans le domaine de l'analyse d'image, pour une utilisation sur des données transcriptomiques afin d'améliorer les performances de prédictions obtenues en supervised deep learning.

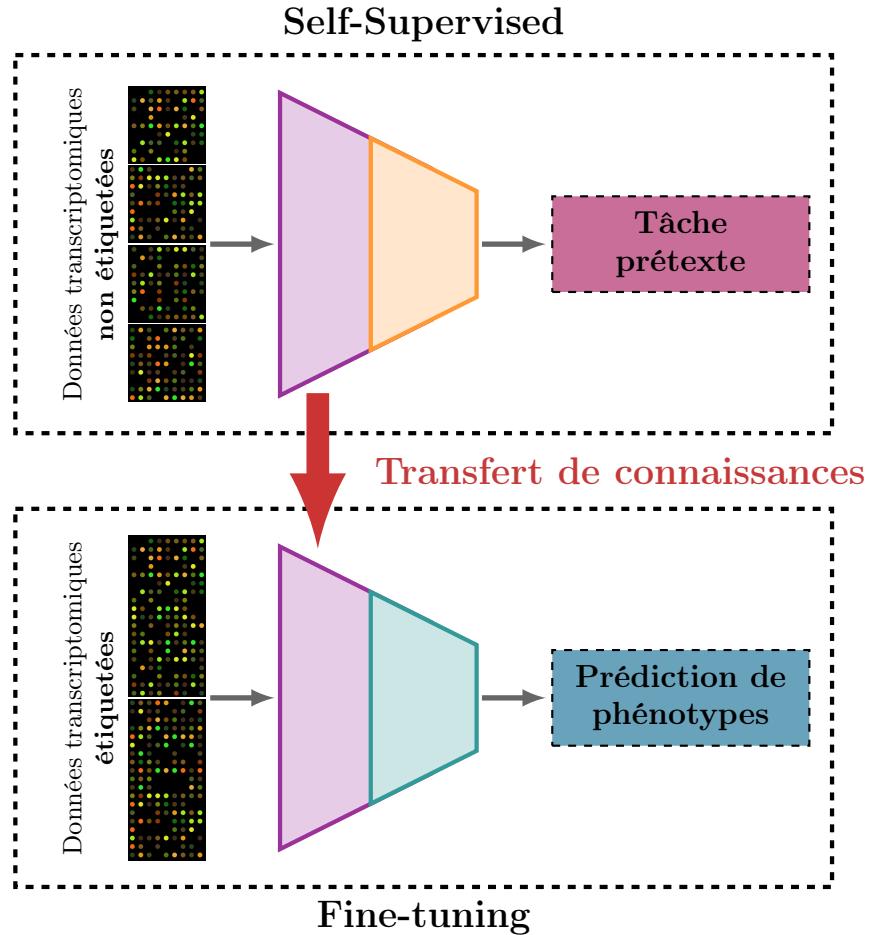


FIGURE 1.1 – Self-Supervised Learning

### 1.3 Etat de l’Art

L’application du machine learning sur les données transcriptomiques s’est répandue ces dernières années. Plusieurs méthodes sont mises en pratique avec succès pour des prédictions sur ces données, comme RF (Forêt aléatoire) ou ENET (Elastic Net) [1, 14, 12], mais il n’y a encore que peu de papiers sur l’apprentissage profond, jugé parfois moins performant que l’apprentissage automatique [15].

Pour appliquer le deep learning sur les données transcriptomiques, nous allons nous inspirer à des techniques déjà existantes. Due à la faible quantité de données comparée à leur dimension, nous nous intéressons donc au self-supervised deep learning, une branche de l’apprentissage profond. De nombreuses approches existent, notamment dans le domaine de l’analyse d’images. Le papier [9] en annonce quelques-unes, avec notamment différentes tâches prétextes usuelles. Une tâche prétexte est ce que va apprendre à réaliser le modèle sans besoin de données étiquetées par des humains. Le self-supervised learning utilise des tâches prétextes comme la prédiction de la rotation [6], prédiction d’une partie cachée [13] ou prédiction de colorations [18, 10].

Lorsqu’on s’intéresse à des données tabulaires, on se retrouve avec moins de tâches prétextes à disposition. Alors que l’on comprend rapidement qu’une rotation d’image ne va pas en changer le sens, c’est plus difficile de savoir quelle transformation appliquer sur des données tabulaires sans en changer le sens. On retrouve tout de même la prédiction de données masquées [17].

Une autre technique du self-supervised learning, toujours plus appliquée dans le domaine de l’image grâce à la diversité des tâches prétextes pouvant être effectuées, est le contrastive learning. C’est une technique consistant en la génération de différentes vues d’un même exemple que l’on va vouloir rapprocher entre elles tout en les éloignant des autres exemples. Un bon exemple d’utilisation de cette technique en image est SimCLR [3, 4]. Un article récent propose une application de cette technique sur des données tabulaires avec SCARF [2].

# Modèles

## 2.1 Perceptrons multi-couches

Avant de commencer une présentation précise des différentes méthodes utilisées, commençons par une introduction au concept de deep learning. Il s'inscrit dans le domaine de l'intelligence artificielle, qui a pour but premier de réaliser des tâches faciles pour un humain mais difficiles à expliquer et donc à réaliser par une machine. Un exemple simple de ce genre de tâche est d'identifier un objet sur une image. L'idée du deep learning est d'apprendre de l'expérience, à partir de données dont on connaît le résultat de prédiction. Le concept compliqué à décrire est décomposé en de plus simples, construits les uns par-dessus les autres : nous parlons alors d'apprentissage profond [7].

**Perceptron multi-couches** Une forme de deep learning la plus simple, et aussi celle utilisée durant ce stage, est le perceptron multi-couches, un réseau de neurones constitué de plusieurs couches pleinement connectées. Son but est d'approximer une fonction  $f^*$  reliant une entrée  $x$  à une sortie  $y$ . Le réseau de neurones apprend les valeurs de paramètres  $\theta$  de la fonction  $y = f(x; \theta)$  qu'il définit, pour donner la meilleure approximation de la fonction  $f^*$ .

**Couches** On construit le réseau de neurones avec un ensemble de fonctions, appelées les unes après les autres. Ce sont les différentes couches. L'ensemble de ces fonctions forme la fonction  $f$  définissant le réseau. Le nombre de couches est ce que l'on appelle la profondeur du réseau. La dernière couche est nommée la couche de sortie, elle donne la prédiction finale du réseau de neurones, celle que l'on veut être proche de l'objectif voulu. Les autres couches, pour lesquelles les données d'apprentissage ne donnent aucune indication sur les sorties, sont appelées les couches cachées.

**Perceptron** Pour compléter l'idée d'un réseau de neurones, chaque couche est constituée d'une multitude de petites unités, ressemblant dans leur fonctionnement aux neurones biologiques. Ce sont les perceptrons. Chaque perceptron est une fonction prenant en entrée un vecteur  $a$  et renvoyant un scalaire  $z$  en sortie, tel que  $z = a * w + b$ , avec  $w$  le poids et  $b$  le biais. Tous les perceptrons d'une couche peuvent agir en parallèle et l'ensemble de leurs sorties forme l'entrée de la couche suivante.

**Activation** Tel quel, un perceptron, et au final le réseau de neurones, est linéaire. Pour casser cette linéarité et permettre d'approximer des fonctions plus complexes, on utilise des fonctions d'activation. Chaque sortie d'un perceptron est passée dans cette fonction d'activation avant de passer aux autres couches. La fonction d'activation la plus courante cassant la linéarité est la fonction ReLu, Rectified Linear Unit, définie par  $relu(z) = \max(0, z)$ . D'autres fonctions d'activation sont plus utilisées au niveau de la couche de sortie. C'est le cas de softmax, définie par  $softmax(z)^i = \exp(z^i) / \sum_j \exp(z^j)$ , servant le plus souvent à la sortie d'un classificateur pour représenter la distribution de probabilité sur plusieurs classes.

Une fois l'architecture du réseau définie, c'est-à-dire les hyperparamètres choisis (nombre de couches, d'unités par couche et fonctions d'activation), le réseau de neurones est prêt à apprendre. Les différents poids et biais sont initialisés aléatoirement. Les données sont préalablement divisées en deux ensembles, un d'entraînement et un de test. Les données d'entraînement peuvent passer dans le réseau. On calcule alors la fonction de coût qui quantifie la différence entre la sortie donnée par le réseau et la sortie attendue des données d'entraînement. Ce coût permet de mettre à jour les différents poids du réseau de neurones, grâce à un algorithme de descente de gradient. Le gradient est calculé par propagation arrière du réseau, en commençant par le coût et en repassant à l'envers dans toutes les unités du réseau pour déterminer l'influence de chacune et calculer les changements à effectuer pour atteindre la sortie désirée.

**Coût** Le coût  $C$  est calculé entre la sortie du réseau de neurones  $\hat{y}$  et la sortie attendue par les données  $y$ . Il existe de multiples fonction de coût, mais on n'en utilise ici que deux : la cross-entropie (CE), définie par  $C(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^n y_i^c \log(\hat{y}_i^c)$ , et l'erreur quadratique moyenne (MSE), définie par  $C(y, \hat{y}) = \sum_{i=1}^N (y_i - \hat{y}_i)^2$ , avec  $n$  le nombre de classes, soit la taille de la sortie, et  $N$  la taille du batch. La cross-entropie est couramment utilisée lorsque la tâche de prédiction est une classification, alors que la MSE est plutôt utilisée pour une prédiction de données.

**Optimiseurs** Il existe plusieurs algorithmes de descente de gradient, utilisant tous une descente de gradient stochastique ou par mini-batch (SGD). Ce type de descente permettant de passer les données par petits groupes, les batchs, au lieu de toutes les passer ensemble. Ces algorithmes sont appelés les

optimiseurs. La SGD met à jour les poids et biais  $w$  grâce aux gradients  $\nabla_w C$  calculé à partir du coût, suivant  $w = w - \alpha \nabla_w C(y, \hat{y})$ , avec  $\alpha$  le taux d'apprentissage. Nous en utilisons ici deux : SGD avec momentum et Adam. **SGD avec momentum** est une variante simple de la SGD. Le momentum accumule le gradient de la précédente propagation arrière pour qu'il soit utilisé dans le calcul du suivant. La direction du gradient présent est alors constituée du gradient présent et celui de la fois précédente, permettant donc de l'ajuster et d'avancer plus rapidement vers l'optimum. Le paramètre du momentum est la proportion du présent gradient à prendre en compte par rapport au gradient précédent. **Adam** est une variante d'un troisième optimiseur, RmsProp, qui normalise le gradient obtenu avant de l'utiliser. Adam reprend cette idée et y ajoute le momentum.

Pour améliorer encore les performances d'un réseau de neurones, plusieurs éléments y sont ajoutés. Ils servent essentiellement à éviter le sur-apprentissage des données.

**Early Stopping** Le plus simple est l'early stopping. En observant la fonction de coût sur un ensemble de données dit de validation, faisant partie de l'ensemble d'entraînement et servant à tester le réseau en cours d'apprentissage, l'early stopping arrête, comme son nom l'indique, l'entraînement avant que l'on atteigne le nombre maximum d'époques, une époque représentant un ensemble d'itérations pour un passage complet de toutes les données. Le réseau de neurones revient alors à son état au moment où le coût de validation est au plus bas, au lieu de laisser le coût d'entraînement baisser et le réseau apprendre par cœur les données. Cela permet d'entraîner le réseau à convergence, avant que celui-ci ne sur-apprenne.

**Régularisation L1 L2** Une régularisation sous forme de pénalisation du coût peut être aussi appliquée. Intervenant alors dans la fonction de coût, cette pénalisation se présente sous forme de la norme 1 ou 2, selon la régularisation L1 ou L2, de l'ensemble de poids  $W$  du réseau, soit  $\Omega_{L1}(W) = \|W\|_1$  ou  $\Omega_{L2}(W) = \frac{1}{2} \|W\|_2^2$ . Elle est ajoutée au calcul du coût, pondérée par un paramètre  $r$ , soit  $\hat{C}(W) = r\Omega(W) + C(W)$ . Cela permet de réduire l'erreur de généralisation, où le réseau n'apprend pas le concept sous-jacent des données, sans réduire l'erreur d'entraînement.

**Dropout** Une autre régularisation est le dropout. Il permet d'entraîner un ensemble de sous-réseau en désactivant aléatoirement un pourcentage, le paramètre du dropout, d'unités dans les couches cachées.

**Batch Normalization** Pour aider le réseau à mieux apprendre, on utilise aussi la normalisation du batch. A chaque batch, les sorties de fonction d'activation sont normalisées afin de contrôler la distribution des activations des neurones.

**Métriques** Pour tester les performances du réseau de neurones, on fait appel à différentes métriques, calculées à partir de la sortie du réseau en considérant les données de test, qui n'ont pas été données à l'entraînement.

**Accuracy** L'accuracy, tout d'abord, intervient dans un problème de classification. Elle permet de mesurer le pourcentage de données qui ont été correctement classifiées, la classe finale d'une donnée étant celle avec la probabilité la plus importante.

**AUC** Il y a ensuite l'AUC, aire sous la courbe ROC. Cette métrique, calculée en testant la bonne classification des données pour différents seuils, et non simplement la meilleure classe donnée par la sortie, témoigne de la qualité de la classification des données et donc de la confiance du réseau.

**Brier Score** Enfin, une dernière métrique est le score Brier. Utilisé dans un problème de classification binaire, il permet aussi de quantifier l'accuracy des probabilités de classification et revient à calculer la MSE sur des sorties de classifications.

**Implémentation** Tous les modèles de réseaux de neurones suivants ont été implémentés en langage Python, à l'aide de la bibliothèque Tensorflow et son API Keras.

## 2.2 Modèle Supervisé de Référence

Le premier modèle créé a été celui du réseau de neurones supervisé. C'est le modèle le plus simple. Il a pour tâche de prédire la classe des données. Son entraînement se fait directement à partir d'un jeu de données. Il constitue la courbe de référence pour toutes les expérimentations à suivre. Chaque nouveau modèle sera comparé au modèle supervisé dans le but d'améliorer ses performances.

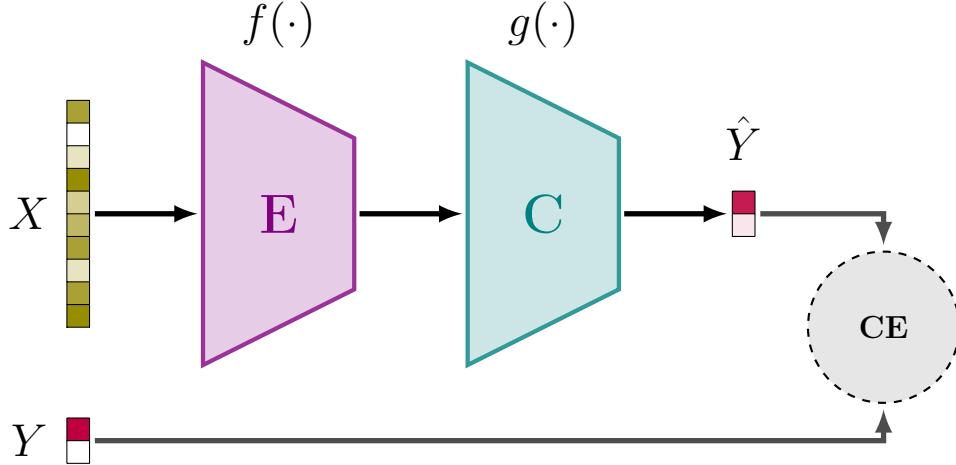


FIGURE 2.1 – Modèle Supervisé

---

**Algorithm 1:** Supervised model

---

```

input : batch size  $N$ , training data  $x \in \mathbb{R}^n$  and label  $y$ , encoder  $f(\cdot)$ , classifier  $g(\cdot)$ 
1 for sampled minibatch  $\{x_k, y_k\}_{k=1}^N$  do
2   forall  $k \in [1, N]$  do
3      $\tilde{x}_k = f(x_k)$ 
4      $\hat{y}_k = g(\tilde{x}_k)$ 
5      $L = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^n y_i^c \log(\hat{y}_i^c)$ 
6   update networks  $f(\cdot)$  and  $g(\cdot)$  to minimize  $L$ 

```

---

Le modèle supervisé est constitué de deux sous-modèles : l'encodeur E et le classificateur C qui se suivent : la sortie de l'encodeur devient l'entrée du classificateur. La figure 2.1 schématisé le modèle et l'algorithme 1 le décrit.

L'encodeur permet de construire une bonne représentation des données qui rend la tâche à réaliser plus facile. Le classificateur réalise cette tâche à partir de cette représentation et permet donc de trouver la classe à laquelle appartient chaque donnée. Il est toujours utilisé lorsque l'on souhaite classifier les données, et donne donc la sortie du modèle, la prédiction.

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^n y_i^c \log(\hat{y}_i^c) \quad (2.1)$$

**Cross-entropie** Le modèle supervisé est entraîné avec la fonction de coût cross-entropie CE, définie par la formule 2.1. Ce coût est adapté à une sortie de classification contenant les probabilités d'appartenance à chaque classe.

## 2.3 Self-supervised : Prévision de données masquées

### 2.3.1 Modèle de prédiction de données masquées

Le modèle de prédiction de données masquées est un modèle du self-supervised learning. Contrairement au modèle supervisé vu précédemment, le label des données, et donc ce que doit prédire le réseau de neurones, est créé par la machine et non par un humain. Cette prédiction de données masquées est donc la tâche prétexte de ce modèle self-supervised.

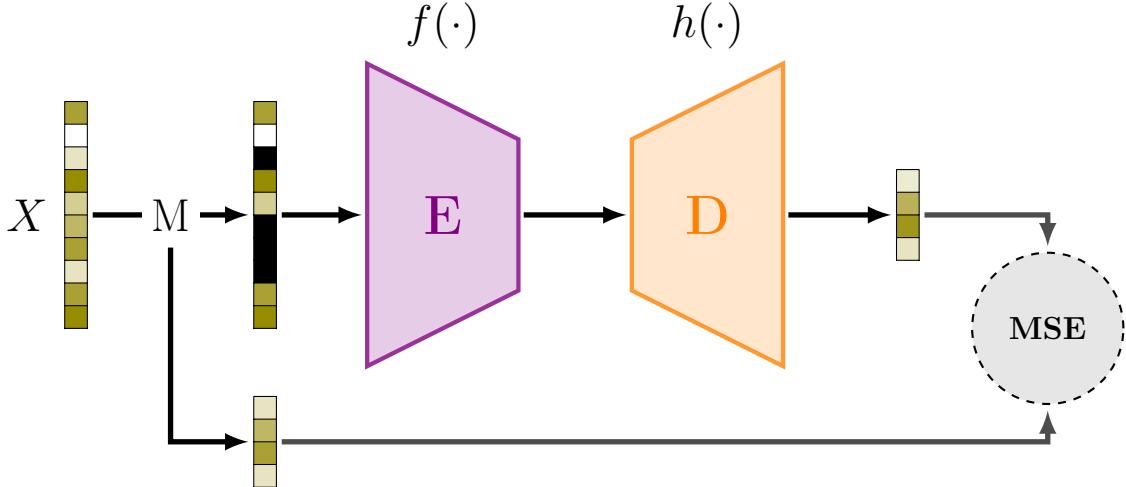


FIGURE 2.2 – Modèle self-supervised de prédiction de données masquées

---

**Algorithm 2:** Masked data prediction model

---

```

input : batch size  $N$ , mask  $M$  of size  $|M|$ , training data  $x \in \mathbb{R}^n$ , encoder  $f$ , decoder  $h$ 
output: pre-trained encoder  $f$ 
1 for sampled minibatch  $\{x_k, y_k\}_{k=1}^N$  do
2   forall  $k \in [1, N]$  do
3     with the mask  $M$ , mask the data  $x_k$  to obtain the masked data  $\bar{x}_k \in \mathbb{R}^n$  and the data
        under the mask  $\bar{y}_k \in \mathbb{R}^{|M|}$ 
4      $\tilde{x}_k = f(\bar{x}_k)$ 
5      $\hat{y}_k = h(\tilde{x}_k)$ 
6    $L_2 = \sum_{i=1}^N (y_i - \hat{y}_i)^2$ 
7   update networks  $f$  and  $h$  to minimize  $L_2$ 
8 return encoder  $f$ , predictor  $h$ 

```

---

Le modèle est donc constitué de plusieurs points-clés, schématisés dans la Figure 2.2 et décrits dans l’algorithme 2. Les données sont tout d’abord en partie masquées avec un masque fixe et les variables sous le masque deviennent la tâche à prédire. Ces données masquées passent ensuite dans l’encodeur puis dans le décodeur qui va prédire les données masquées à partir de la représentation donnée par l’encodeur. La fonction de coût utilisée pour comparer les données masquées à celles prédites est la MSE, erreur quadratique moyenne, adaptée pour une tâche de prédiction de données.

**Masque** Le masque est fixe durant tout l’entraînement du modèle. D’une certaine taille choisie, il est créé une seule fois pour que les variables à prédire reste les mêmes et donc que la tâche à réaliser soit la même durant tout l’apprentissage. On masque ici les données à zéro. Les données fournies en entrée au modèle ont la même dimension que les données originelles non masquées et les données sous masque en sortie ont pour dimension la taille du masque choisie.

**Modèle** Une fois les données masquées, elles peuvent donc passer dans l’encodeur, qui est en tout point identique à celui du modèle supervisé, puis dans le décodeur qui donne en sortie les prédictions pour toutes les variables masquées. Ce modèle n’est pas un autoencodeur. Il prend en entrée des données masquées et non intégrées, et ne recherche pas à reconstruire la totalité de l’entrée, juste une partie, celle masquée, à laquelle il n’a pas eu accès.

$$L_2(y, \hat{y}) = \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (2.2)$$

**MSE** La fonction de coût ensuite utilisée pour la propagation arrière et la mise à jour des poids du réseau, est l’erreur quadratique moyenne, donnée en formule 2.2. Comme une bonne valeur de cette erreur reste vague, contrairement à une mesure comme l’accuracy qui constitue en un pourcentage, elle est comparée à l’erreur obtenue si le réseau prédisait pour chaque variable masquée, juste sa moyenne. Pour des données ayant été centrées-réduites, la valeur d’une telle MSE est de 1.

Le modèle doit alors obtenir une MSE de test inférieure à 1 pour que l'on considère qu'il apprend véritablement les données.

**Fine-tuning** Une fois le modèle entraîné, l'encodeur, qui a dû apprendre une bonne représentation des données en s'entraînant à les prédire, est extrait et est utilisé dans un autre modèle, celui de fine-tuning, pour réaliser la tâche originelle de prédiction de phénotype. Le fine-tuning est l'action de entraîner à nouveau un modèle afin de l'ajuster de manière plus précise. Deux modèles de fine-tuning ont pu être testés.

### 2.3.2 Fine-tuning

#### 2.3.2.1 Fine-tuning avec données étiquetées

---

**Algorithm 3:** Finetuning of mask data prediction model

---

```

input : batch size  $N$ , training data  $x$  and label  $y$ , pre-trained encoder  $f$ , classifier  $g$ 
1 for sampled minibatch  $\{x_k, y_k\}_{k=1}^N$  do
2   forall  $k \in [1, N]$  do
3      $\tilde{x}_k = f(x_k)$ 
4      $\hat{y}_k = g(\tilde{x}_k)$ 
5      $L = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^n y_i^c \log(\hat{y}_i^c)$ 
6   update networks  $f$  and  $g$  to minimize  $L$ 

```

---

Ce premier modèle de fine-tuning est le plus simple. L'algorithme 3 en décrit les étapes. Ce modèle est en réalité d'architecture identique à celle du modèle supervisé vu précédemment. Il est constitué du même encodeur, même classificateur, et la même fonction de coût, c'est-à-dire la cross-entropie, est utilisée pour mettre à jour le modèle. La différence avec le modèle supervisé, est que l'encodeur, que l'on récupère du modèle self-supervised, a été pré-entraîné. Ces poids ne sont pas aléatoires mais permettent déjà de réaliser une bonne représentation des données.

**Gel des couches** Un nouveau paramètre rentre tout de même en jeu dans ce modèle et sera utilisé dans tous les autres modèles de fine-tuning qui suivront. Ce paramètre est le nombre de couches à geler de l'encodeur. Ces dernières sont gelées par ordre de couches. L'encodeur ayant été pré-entraîné, on peut geler ces couches pour que celles-ci ne soient plus modifiables par la mise à jour du réseau de neurones de fine-tuning. Cela permet de tester le pré-entraînement de l'encodeur, et de voir si l'encodeur, même gelé, réalise une bonne représentation des données que le classificateur peut bien utiliser pour réaliser sa tâche de prédiction.

#### 2.3.2.2 Fine-tuning avec données étiquetées et données prédites non étiquetées

Ce deuxième modèle de fine-tuning du modèle de prédiction des données masquées est une variante du premier. Inspiré par la fonction de coût utilisée lors du fine-tuning du modèle VIME [17], ce modèle est construit autour d'une fonction de coût différente de la classique cross-entropie. En effet, elle comprend, en plus, une partie utilisant des données non étiquetées, prédites par le modèle dont l'encodeur est extrait, avec la fonction de coût MSE. L'algorithme 4 et la figure 2.3 en décrivent le déroulement.

**Données non étiquetées** En plus des données étiquetées présentes dans les jeux de données, des données non étiquetées sont créées. Ces données proviennent directement du modèle de prédiction de données masquées. Elles sont construites en reprenant les données masquées et en les démasquant en remplaçant les valeurs masquées par celles prédites à la sortie du modèle self-supervised. Ce modèle self-supervised est d'ailleurs entièrement gelé, et ne changera donc pas lors du fine-tuning.

$$L_3(y, \hat{y}, \hat{y}') = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^n y_i^c \log(\hat{y}_i^c) + \lambda * \sum_{i=1}^N (\hat{y}_i - \hat{y}'_i)^2 \quad (2.3)$$

**Loss** Les données non étiquetées sont, si elles ont bien été prédites, assez proches des données étiquetées correspondantes et donc leurs prédictions de

phénotype doivent l'être aussi. On utilise la fonction de coût MSE pour calculer l'écart entre ces prédictions. Celle-ci est ensuite multipliée par un paramètre  $\lambda$  qui permet de pondérer cette MSE pour la rendre du même ordre de grandeur que la cross-entropie entre les prédictions des données étiquetées et

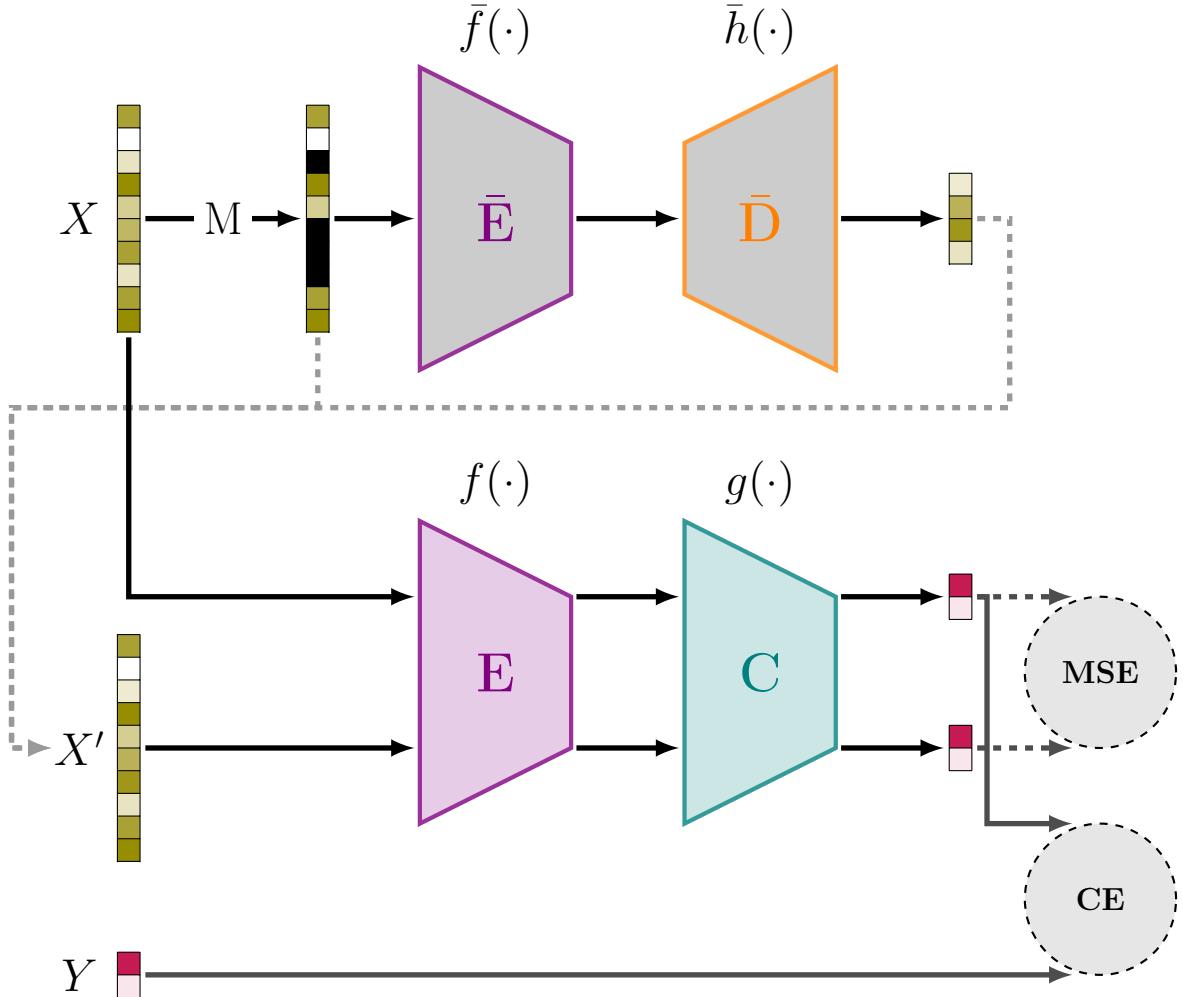


FIGURE 2.3 – Fine-tuning avec données non étiquetées du modèle self-supervisé de prédiction de données masquées

leurs étiquettes. Les deux fonctions de coût  $\lambda * MSE$  et  $CE$  sont enfin sommées pour former la fonction de coût du réseau de neurones (Formule 2.3).

**Architecture** Ce modèle diffère du précédent modèle de finetuning par la création des données non étiquetées et par la fonction de coût. L’encodeur et le classificateur reste, eux, de la même architecture que le modèle supervisé. Il faut quand même noter que l’encodeur gelé servant à la création des données non étiquetées et l’encodeur servant au niveau du fine-tuning sont indépendants, ce dernier étant seulement gelé en fonction du paramètre du nombre de couches à geler. Il peut donc être mis à jour, alors que l’encodeur au niveau de la prédiction des données masquées ne sera jamais modifié.

## 2.4 Contrastive Learning

Après avoir vu le modèle utilisant la tâche prétexte de la prédiction de données masquées, nous passons à une autre approche, récente, du self-supervised learning : le contrastive learning. Avec peu de données étiquetées, cette technique a été utilisée à plusieurs reprises avec succès, notamment avec SimCLR [3, 4], dont le présent modèle s’inspire.

Cette technique repose sur une comparaison de différentes vues d’une même donnée entre elles et avec les autres données. Pour chaque exemple, deux vues sont générées grâce à des augmentations ou corruptions. Ces vues passent ensuite dans l’encodeur, toujours identique à celui du modèle supervisé, puis dans le projecteur. Le projecteur a pour but de projeter les représentations données par l’encoder dans un espace donné pour pouvoir comparer les différentes projections de vues entre elles. Sur toutes ces projections, on applique la contrastive loss, qui compare les vues deux à deux. Ce modèle contrastif

---

**Algorithm 4:** Finetuning of mask data prediction model with unlabeled data

---

**input** : batch size  $N$ , mask  $M$  of size  $|M|$ , training data  $x \in \mathbb{R}^n$  and label  $y$ , pre-trained encoder  $f$  and frozen  $\bar{f}$ , trained predictor  $h$ , classifier  $g$

- 1 **for** sampled minibatch  $\{x_k, y_k\}_{k=1}^N$  **do**
- 2   **forall**  $k \in [1, N]$  **do**
- 3     with the mask  $M$ , mask the data  $x_k$  to obtain the masked data  $\bar{x}_k \in \mathbb{R}^n$
- 4      $\tilde{x}_k = \bar{f}(\bar{x}_k)$
- 5      $\tilde{y}_k = h(\tilde{x}_k)$
- 6     with the masked data  $\bar{x}_k \in \mathbb{R}^n$  and the predicted masked data  $\tilde{y}_k \in \mathbb{R}^{|M|}$  reconstruct the whole data  $x'_k \in \mathbb{R}^n$
- 7      $x_k = f(x_k)$
- 8      $\hat{y}_k = g(x_k)$
- 9      $x'_k = f(x'_k)$
- 10     $\hat{y}'_k = g(x'_k)$
- 11     $L_3 = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^n y_i^c \log(\hat{y}_i^c) + \lambda * \sum_{i=1}^N (\hat{y}_i - \hat{y}'_i)^2$
- 12    update networks  $f$  and  $h$  to minimize  $L_3$

---

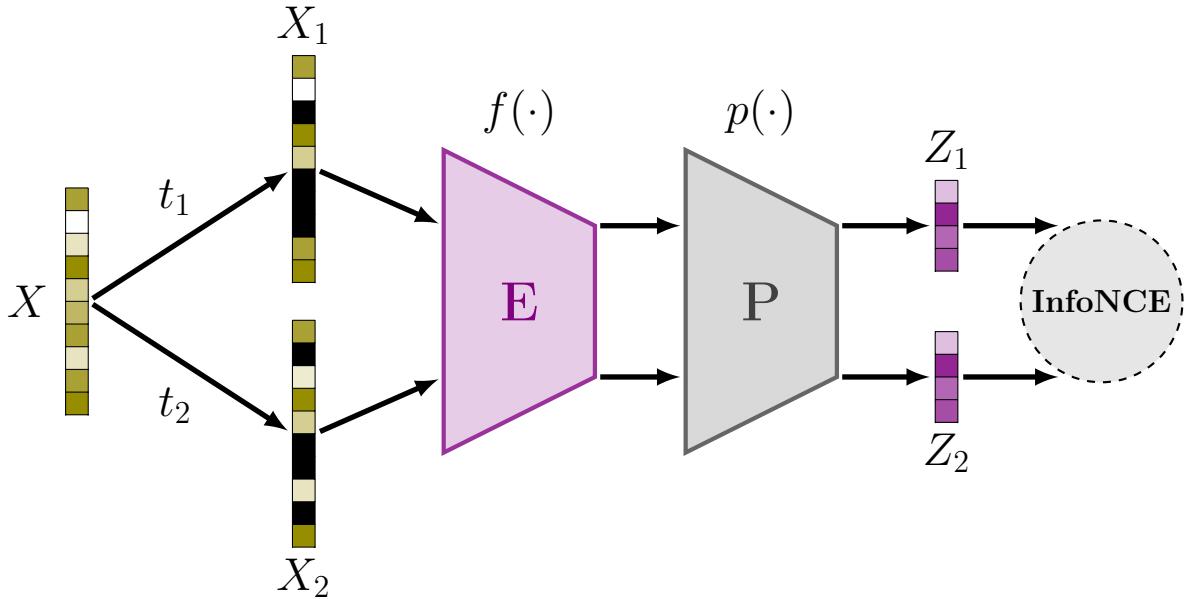


FIGURE 2.4 – Contrastive Learning Model

est décrit dans l'algorithme 5 ainsi que dans la Figure 2.4.

**Augmentations** Ce modèle repose sur la génération des différentes vues avec les augmentations. Les augmentations possibles peuvent être très diverses en fonction du type de données. Une bonne augmentation permet de réduire le bruit et autres artefacts de la donnée qui rendrait la tâche trop simple et donc l'éloignerait d'une bonne représentation. Une bonne augmentation ne doit cependant pas réduire les informations essentielles des données pour toujours bien trouver une bonne représentation utilisable pour la tâche en aval, à réaliser lors du fine-tuning.

Dans le domaine de l'image, les augmentations possibles sont nombreuses : rognage, rotation, recolorisation, masque... Cependant, comme les tâches prétextes, pour les données tabulaires comme les données transcriptomiques, il est plus compliqué de savoir ce que peut être une bonne augmentation.

On reste ici sur l'idée de masque. Les augmentations sont utilisées deux fois pour créer deux vues, ou une seule fois, pour n'en créer qu'une, la deuxième vue étant alors l'exemple lui-même.

**Masque zéro** La première augmentation testée pour créer de nouvelles vues d'un exemple rejoint la tâche prétexte du modèle self-supervised de prédiction de données masquées. En effet, cette augmentation consiste en un masque à zéro des données. Ce masque est aléatoire pour chaque exemple. Il est d'une taille constante, ce sont les variables masquées qui diffèrent.

---

**Algorithm 5:** Contrastive Learning Model

---

```

input : batch size  $N$ , temperature  $\tau$ , augmentation  $t_1$  and  $t_2$ , training data  $x \in \mathbb{R}^n$  encoder  $f$ ,
        projector  $p$ 
output: pre-trained encoder  $f$ 
1 for sampled minibatch  $\{x_k, y_k\}_{k=1}^N$  do
2   forall  $k \in [1, N]$  do
    // First augmentation
    3    $\tilde{x}_{1,k} = t_1(x_k)$ 
    4    $h_{1,k} = f(\tilde{x}_{1,k})$ 
    5    $z_{1,k} = p(h_{1,k})$ 
    // Second augmentation
    6    $\tilde{x}_{2,k} = t_2(x_k)$ 
    7    $h_{2,k} = f(\tilde{x}_{2,k})$ 
    8    $z_{2,k} = p(h_{2,k})$ 
    9   forall  $i \in [1, N]$  and  $j \in [1, N]$  do
10     $s_{i,j} = z_{1,i}^T \cdot z_{2,j} / (\|z_{1,i}\|_2 \|z_{2,j}\|_2)$ 
11     $L_4 = \frac{1}{N} \sum_{i=1}^N -\log\left(\frac{\exp(s_{i,j})/\tau}{\frac{1}{N} \sum_{k=1}^N \exp(s_{i,k})/\tau}\right)$ 
12    update networks  $f$  and  $g$  to minimize  $L_4$ 
13 return encoder  $f$ 

```

---

**Masque distribution** La deuxième augmentation testée est toujours un masque que l'on applique sur les données, étant donné le nombre limité d'augmentations disponibles pour des données tabulaires. Elle a été inspirée par la technique de masquage des données utilisée pour le contrastive learning de SCARF [2]. Le masque est ici toujours complètement aléatoire. On choisit donc aléatoirement, pour chaque exemple, les variables à masquer (du nombre de la taille de masque) et on les remplace par des valeurs présentes dans le batch (et donc d'autres exemples) de la même variable. Cela crée donc un masque moins abrupt que zéro en reprenant des valeurs de la distribution des variables et qui existent vraiment dans l'espace de données.

$$L_4(z_1, z_2) = \frac{1}{N} \sum_{i=1}^N -\log\left(\frac{\exp(s_{i,j})/\tau}{\frac{1}{N} \sum_{k=1}^N \exp(s_{i,k})/\tau}\right) \quad (2.4)$$

avec  $s_{i,j} = z_{1,i}^T \cdot z_{2,j} / (\|z_{1,i}\|_2 \|z_{2,j}\|_2)$

laire des projections normalisées avec la norme 2. On met ensuite à l'échelle cette similarité en la divisant la température  $\tau$ . On souhaite alors que les paires positives (projection venant du même exemple) aient une similarité de 1 et les paires négatives, une similarité de 0. On calcule donc la cross-entropie entre ces similarités normalisées et leur valeur objective, ce qui donne la contrastive loss, InfoNCE.

Ce calcul de la loss peut être fait différemment en fonction de quelles paires négatives nous considérons pour chaque projection. On peut considérer toutes les paires, y compris celle formée entre projections provenant de la même augmentation. On peut ne considérer que les paires entre projections provenant d'augmentations différentes. On peut enfin considérer une seule paire négative, choisie aléatoirement dans le batch, pour chaque paire positive ce qui équilibre leur nombre.

La température permet d'établir l'intervalle dans lequel se placent les similarités afin de bien pouvoir calculer la contrastive loss.

**Fine-tuning** Le fine-tuning du modèle de contrastive learning se fait de la même façon que le fine-tuning avec données étiquetées de modèle de prédiction des données masquées. L'encodeur pré-entraîné est récupéré et on y ajoute à la suite le classificateur, pour former une architecture identique à celle du modèle supervisé (Algorithme 3).

**Contrastive Loss : Paires Négatives et Température** Ce modèle utilise une fonction de coût particulière : la contrastive loss. Elle se base sur un calcul de similarité entre les différentes projections de vues. Cette similarité est calculée en faisant le produit scalaire des projections normalisées avec la norme 2. On met ensuite à l'échelle cette similarité en la divisant la température  $\tau$ . On souhaite alors que les paires positives (projection venant du même exemple) aient une similarité de 1 et les paires négatives, une similarité de 0. On calcule donc la cross-entropie entre ces similarités normalisées et leur valeur objective, ce qui donne la contrastive loss, InfoNCE.

# Résultats

## 3.1 Données

Tous les tests des différents modèles décrits ont été effectués sur deux jeux de données transcriptomiques différents : MicroArray et TCGA.

### 3.1.1 MicroArray

**Puce à ADN** Le premier jeu de données sur lequel les travaux ont pu être accomplis est le jeu de données dit MicroArray. Ces données proviennent d'une technologie appelée puce à ADN (ou microarray en anglais). Cette technologie permet d'analyser les taux d'expression des gènes grâce à une puce contenant plusieurs milliers de petits morceaux d'ADN appelés probes, auxquels se fixe l'ADN du sujet à observer. Par une technique de fluorescence, on peut ainsi voir les spots qui s'éclairent plus ou moins et en les comparant à un sujet de référence, connaître le niveau d'expression de ces probes.

**Jeu de données** Le jeu de données récupéré provient de la publication [16]. Ces données sont constituées de l'ensemble de ces niveaux d'expression ainsi que plusieurs informations concernant le sujet observé dont l'étiquette que nous récupérons : le phénotype cancer ou sain. Ce jeu est issu d'un rassemblement de plusieurs expériences avec des puces à ADN. Au final, le jeu de données contient 27887 patients avec chacun 54675 probes et un label : cancer/sain. Il y a 18437 patients cancéreux pour 9450 patients sains.

**Réduction du nombre de probes** Dans l'optique probable d'utiliser un autre jeu de données MicroArray de plus de 100000 patients en plus de ce jeu de données, nous avons sélectionné les probes en commun de ces deux jeux de données. Le nombre de probes a donc été réduit de 54675 à 22268. Ce jeu de données supplémentaires, utilisé dans l'article [5], n'a finalement pas été utilisé. Les données y étaient trop différentes de celles de notre jeu de données MicroArray. En effet, un réseau de neurones (le modèle supervisé ci-dessous) ayant pour tâche de reconnaître de quel jeu appartient chaque donnée a obtenu une accuracy parfaite de 100%. Les données sont donc beaucoup trop différentes pour pouvoir en tirer une même représentation.

**Pré-traitement** Avant d'utiliser ces données pour l'entraînement des différents modèles, nous les pré-traitons. Tout d'abord, les données sont normalisées par centrage-réduction, ce qui permet de réduire l'intervalle de données sans perdre d'informations comme des maximums locaux.

Ensuite, les labels cancer/sain sont encodés en one-hot, c'est-à-dire en un vecteur de la taille du nombre de classes, ici 2, où la bonne classe est mise à 1 et les autres à 0. Ce vecteur sera la tâche de prédiction du réseau de neurones.

Enfin, nous séparons le jeu de données en deux, avec préservation de la proportion des classes : le jeu de données d'entraînement, possédant alors 22887 exemples, et le jeu de données de test, servant à tester le modèle une fois entraîné, et possédant donc les 5000 exemples restants. De plus, lors de l'entraînement du réseau, le jeu de données d'entraînement est divisé en deux pour l'entraînement à 80% et la validation à 20%.

### 3.1.2 TCGA

**RNA-Seq** Le deuxième jeu de données, qui permet de vérifier les résultats acquis avec le premier, est le jeu de données dit TCGA. Ces données proviennent d'une autre technique permettant d'analyser l'expression des gènes d'un patient : le séquençage de l'ARN (ou RNA-seq en plus court). Cette technique produit alors les données transcriptomiques sous forme de comptage de brins d'ARN présents chez le sujet observé.

**Jeu de données** Ces données, disponibles sur le site [8], sont étiquetées avec des informations sur le sujet, et contrairement à MicroArray dont les labels n'étaient que le phénotype cancéreux ou sain, les étiquettes du jeu de données TCGA que l'on a retenues sont le type de cancer avec 11 types différents de cancer et une classe supplémentaire pour les sujets sains. Les données sélectionnées constituent un jeu de données de 6464 patients avec chacun 56602 probes, le tout réparti dans 12 classes (Table 3.1).

Abreviation	Nom de la classe	Nombre de patients
LGG	Cancer du cerveau	511
THCA	Cancer de la thyroïde	502
KIRC	Cancer du rein	538
LUAD	Adénocarcinome pulmonaire	533
LIHC	Cancer du foie	371
UCEC	Cancer du corps utérin	551
LUSC	Carcinome épidermoïde pulmonaire	502
NT	Sain	482
PRAD	Adénocarcinome de la prostate	498
OV	Cancer ovarien	374
BRCA	Cancer du Sein	1102
HNSC	Carcinome épidermoïde de la tête et du cou	500

TABLE 3.1 – Répartitions des 12 classes pour les données TCGA

**Pré-traitement** Le jeu de données a été pré-normalisé au format FPKM, un format normalisant les données de comptage en fonction de la longueur du probe et du compte total de probes lus. Pour être utilisées pour l’entraînement de nos réseaux de neurones, on leur applique, comme dans l’article [11], une transformation par logarithme 2, puis on les normalise par centrage-réduction.

Pour les différentes classes, le même traitement que les données MicroArray est appliqué. Un encodage one-hot crée un vecteur de taille 12 où la classe à laquelle appartient le sujet est à 1 et les autres à 0.

Enfin, on sépare le jeu de données en celui d’entraînement avec 5464 patients et en celui de test avec les 1000 patients restants, avec préservation de la proportion de chaque classe. Comme le jeu de données précédent, le jeu de données d’entraînement est divisé en deux pour l’entraînement à 80% et la validation à 20%.

Maintenant que les données ont été présentées, nous allons pouvoir observer les performances de chacun de nos modèles, en commençant par le plus simple : le modèle supervisé.

## 3.2 Modèle Supervisé de Référence

Les premiers résultats concernent le modèle supervisé avec tout d’abord les différents tests qui ont permis d’atteindre les paramètres choisis pour le modèle, et ensuite la courbe de référence avec laquelle les résultats des modèles self-supervised vont être comparés.

### 3.2.1 Implémentation

Le modèle supervisé est constitué d’un encodeur et classificateur qu’il reste à définir.

**Encodeur** Tout d’abord, l’encodeur, que l’on retrouve non seulement dans le modèle supervisé, mais aussi dans tous les modèles suivants, est constitué de trois couches avec normalisation du batch, fonction d’activation ReLu et dropout avec un taux de 0.1 pour chaque. La première couche contient 1000 neurones et une régularisation L1 et L2 aux taux respectifs 1e-6 et 1e-6 y est appliquée. La deuxième couche est identique à la troisième. Elles sont toutes les deux constituées de 700 neurones avec une régularisation L1 et L2 aux taux respectifs de 1e-7 et 1e-7. Les tests effectués pour arriver à ces paramètres sont disponibles dans l’Annexe A.1.

**Classificateur** Ensuite, les données passent dans le classificateur. Celui-ci est constitué d’une couche cachée et de la couche de sortie. La couche cachée possède 32 neurones avec normalisation du batch, fonction d’activation ReLu et dropout à 0.1, sous le même schéma que les couches de l’encodeur. Elle n’a cependant pas de régularisation. La couche de sortie est composée d’autant de neurones que de classes possibles pour les données. La fonction d’activation est ici softmax, plus adaptée à une sortie probabiliste comme une classification. Cette fonction d’activation justifie l’emploi de la cross-entropie pour fonction de coût pour la même tâche.

**Early stopping** De plus, pour déterminer quand arrêter d’entraîner le modèle pour obtenir une performance optimale sur l’ensemble de données de test, on utilise l’early stopping se basant sur le coût obtenu

avec les données de validations, avec une patience de 10. Ces paramètres d'early stopping sont utilisés à l'identique dans tous les modèles suivants.

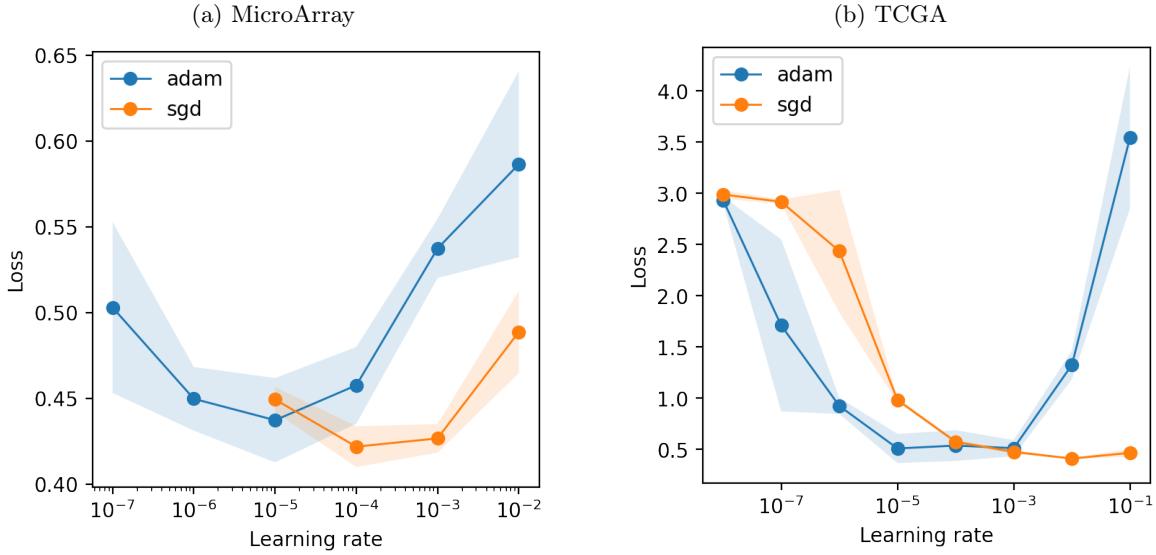


FIGURE 3.1 – Modèle supervisé : Loss en fonction du taux d'apprentissage pour deux optimiseurs : SGD avec momentum à 0.85 et Adam, pour les deux jeux de données : (a) MicroArray et (b) TCGA. Les points pour les taux d'apprentissage ne figurant pas sur la figure ont un coût beaucoup plus élevé ( $> 1$ ).

**Optimiseur et taux d'apprentissage** Une fois les précédents hyperparamètres déterminés, on teste quel est le meilleur optimiseur avec le taux d'apprentissage donnant le coût sur le jeu de données de test le plus faible, un coût faible signifiant que l'on s'approche plus de l'objectif de prédiction. Deux optimiseurs sont testés : Adam et SGD avec momentum à 0.85 pour des taux d'apprentissage entre  $1e-7$  et  $1e-2$ .

Le jeu de données MicroArray obtient des performances optimales et stables lorsqu'on utilise l'optimiseur SGD, descente de gradient stochastique, avec un taux d'apprentissage à  $1e-3$ , comme présenté sur la figure 3.2. Le jeu de données TCGA, quant à lui, obtient de meilleures performances avec l'optimiseur Adam avec un taux d'apprentissage à  $1e-4$ .

Ce test de la loss en fonction du taux d'apprentissage et de l'optimiseur est un test récurrent qui est effectué à chaque nouveau modèle et jeu de données pour déterminer les meilleurs paramètres d'apprentissage pour le réseau de neurones et lui permettre d'accomplir sa tâche de prédiction au mieux.

### 3.2.2 Courbe de référence

Maintenant que le réseau de modèle est bien construit et que tous ces paramètres d'apprentissage ont été choisis, on peut observer le comportement du modèle en changeant le nombre d'exemples d'apprentissage à disposition.

**Accuracy en fonction de la taille des données d'entraînement** La courbe présentée en figure 3.2 est la courbe de référence avec laquelle les autres modèles self-supervised vont être comparés. C'est l'accuracy en fonction de la taille des données d'entraînement que l'obtient avec le modèle supervisé. On voit bien que cette accuracy diminue au fur à mesure que le nombre d'exemples diminue. En zoomant sur les tailles comprises entre 50 et 1000, on observe que l'accuracy y est beaucoup plus faible qu'en ayant utilisé toutes les données et donc toutes les étiquettes. On descend de 10 points de pourcentage pour les données MicroArray et de 3 points pour le jeu de données TCGA. Ce sont ces accuracy que l'on souhaite améliorer pour obtenir une méthode de prédiction de phénotypes qui marcherait sur un jeu de données de petite taille, beaucoup plus courant en réalité.

Les courbes représentées ici sont celles de l'accuracy. Les autres métriques observées, comme la cross-entropie, l'aire sous la courbe ROC ou encore le score Brier, montrent la même baisse de performance du réseau quand le nombre d'exemples d'entraînement diminue (Annexe A.2).

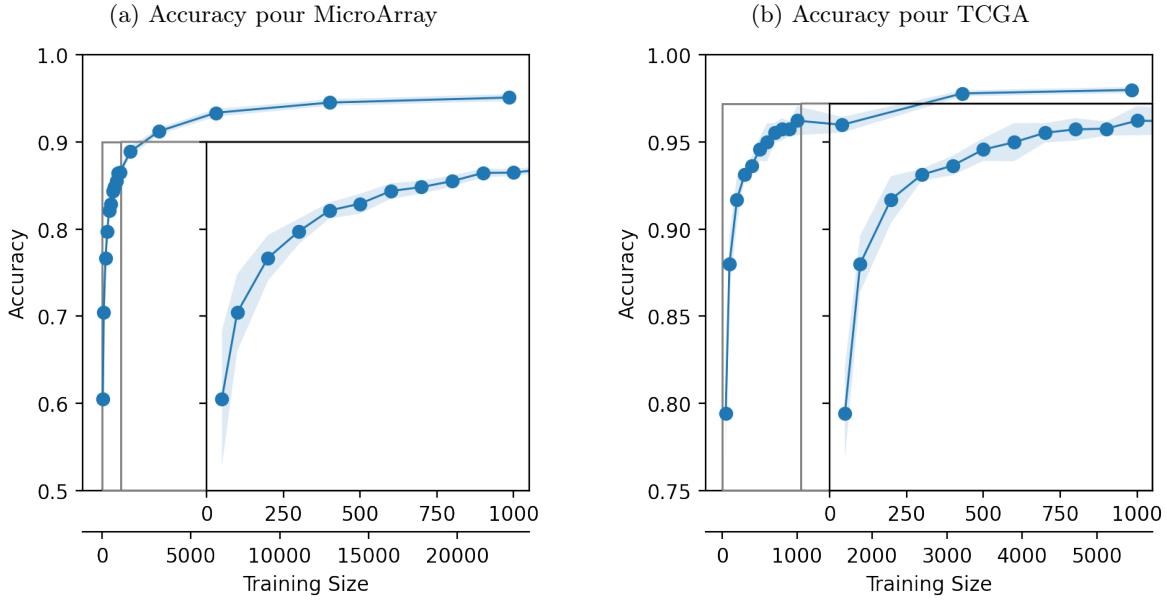


FIGURE 3.2 – Modèle supervisé : Accuracy en fonction de la taille des données d’entraînement pour les données (a) MicroArray et (b) TCGA. Le zoom effectué montre la courbe de référence, comprise entre 50 et 1000.

### 3.3 Self-supervised : Prévision de données masquées

En vue d’améliorer les performances observées sur le réseau de neurones supervisé, nous allons tester le premier modèle utilisant la technique du self-supervised learning. Il s’agit du modèle de prédiction de données masquées. La comparaison avec le modèle supervisé est réalisée lorsque l’on observe le fine-tuning de ce modèle. Nous allons donc tout d’abord nous intéresser au modèle self-supervised.

#### 3.3.1 Modèle de prédiction de données masquées

##### 3.3.1.1 Implémentation

Voyons premièrement comment le modèle de prédiction de données masquées est construit.

**Architecture** Ce modèle est constitué d’un encodeur, qui est en tout point identique à celui du modèle supervisé, et d’un décodeur. Les données transcriptomiques utilisées obtiennent de meilleures performances quand le nombre de couches reste limité. De ce fait, le décodeur est très simple. Il est constitué uniquement de la couche de sortie du modèle, une couche dense comprenant autant de neurones que la taille du masque, sans activation particulière, juste la fonction identité.

**Optimiseur** L’optimiseur utilisé pour réaliser l’entraînement de ce modèle est indépendant du jeu de données. On utilise ici l’optimiseur Adam. Le taux d’apprentissage utilisé est cependant différent selon les jeux de données. Il faut donc le tester pour choisir celui pour entraîner le modèle.

Ces tests répétitifs sont disponibles dans l’annexe (Annexe B.1) et désignent les paramètres suivants. Le jeu de données MicroArray utilise un taux d’apprentissage de 5e-5, tandis que le jeu de données TCGA utilise un taux d’apprentissage de 1e-5.

##### 3.3.1.2 Résultats

Une fois les paramètres choisis, on peut véritablement tester le modèle.

**MSE en fonction de la taille du masque** On le teste alors en fonction des différents paramètres restants : la taille de masque. On obtient alors la courbe de l’erreur quadratique moyenne, fonction de coût utilisée, en fonction de la taille du masque (Figure 3.3). On observe un comportement différent entre les deux jeux de données testés.

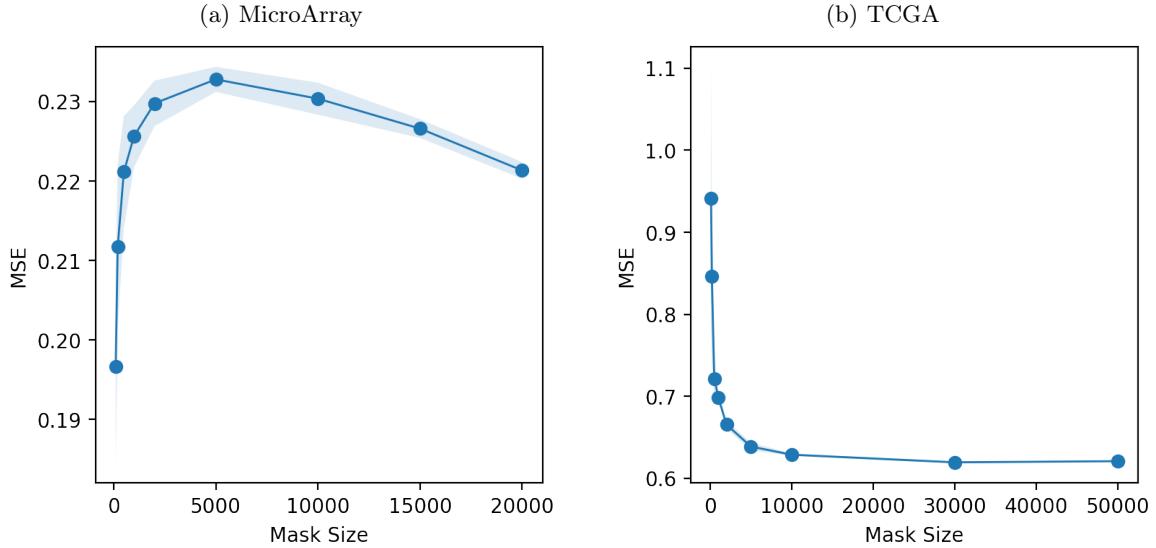


FIGURE 3.3 – Modèle self-supervised de prédiction de données masquées : MSE en fonction de la taille de masque pour les données (a) MicroArray et (b) TCGA.

Le jeu MicroArray nous montre ce qui paraît le plus logique. La MSE grandit avec la taille du masque. Il est en effet plus compliqué de prédire un masque plus grand avec en plus, moins de données à disposition vu que les données masquées ne sont plus source d'information. On remarque tout de même que la courbe décroît une fois passé le masque de taille 5000, indiquant que les plus grands masques sont plus faciles à prédire. Toutefois l'erreur reste assez basse, autour de 0.22, et dans le même ordre de grandeur quelle que soit la taille du masque avec une variation de seulement plus ou moins 0.02.

Le jeu TCGA montre, lui, un tout autre résultat. Pour ces données-ci, on voit la MSE diminuer au fur à mesure que le masque grandit. Le modèle aurait donc toujours plus de mal à prédire de petits masques que de grands. L'erreur quadratique moyenne est en plus assez haute, tournant autour de 0.7, allant de proche de 1 pour les plus petits masques à proche de 0.6 pour les plus grands.

On observe tout de fois que l'erreur quadratique moyenne reste en dessous du seuil de 1 auquel on considère que le modèle produit aléatoirement les résultats. Ce modèle self-supervised apprend donc bien à prédire les données masquées, et son fine-tuning permettra de voir si la représentation apprise grâce à cette tâche prétexte permet d'améliorer les performances de prédiction du phénotype.

### 3.3.2 Fine-tuning

#### 3.3.2.1 Fine-tuning avec données étiquetées

Le premier fine-tuning testé est celui qui reprend simplement le modèle supervisé. Nous allons voir l'implémentation de ce modèle puis ses performances en comparaison avec celles du modèle supervisé.

##### 3.3.2.1.1 Implémentation

S'entraînant avec les données étiquetées exactement comme le modèle supervisé et reprenant le même modèle, les paramètres d'apprentissage, tel que l'optimiseur et le taux d'apprentissage sont aussi les mêmes que le modèle supervisé. Nous avons SGD avec momentum à 0.85 et taux d'apprentissage à 1e-3 pour le jeu de données MicroArray et l'optimiseur Adam avec taux d'apprentissage à 1e-4 pour le jeu de données TCGA.

##### 3.3.2.1.2 Résultats

Les courbes présentes dans les différentes figures 3.4, 3.5, 3.6, 3.7, 3.8 montrent différentes métriques du fine-tuning des encodeurs extraits des modèles montrés précédemment dans la MSE en fonction de la taille du masque (Figure 3.3).

Ces différentes métriques, soient l'accuracy, la cross-entropie, l'aire sous la courbe, le nombre d'époques et le temps moyen par époques, sont affichées en fonction de la taille des données d'entraînement de la

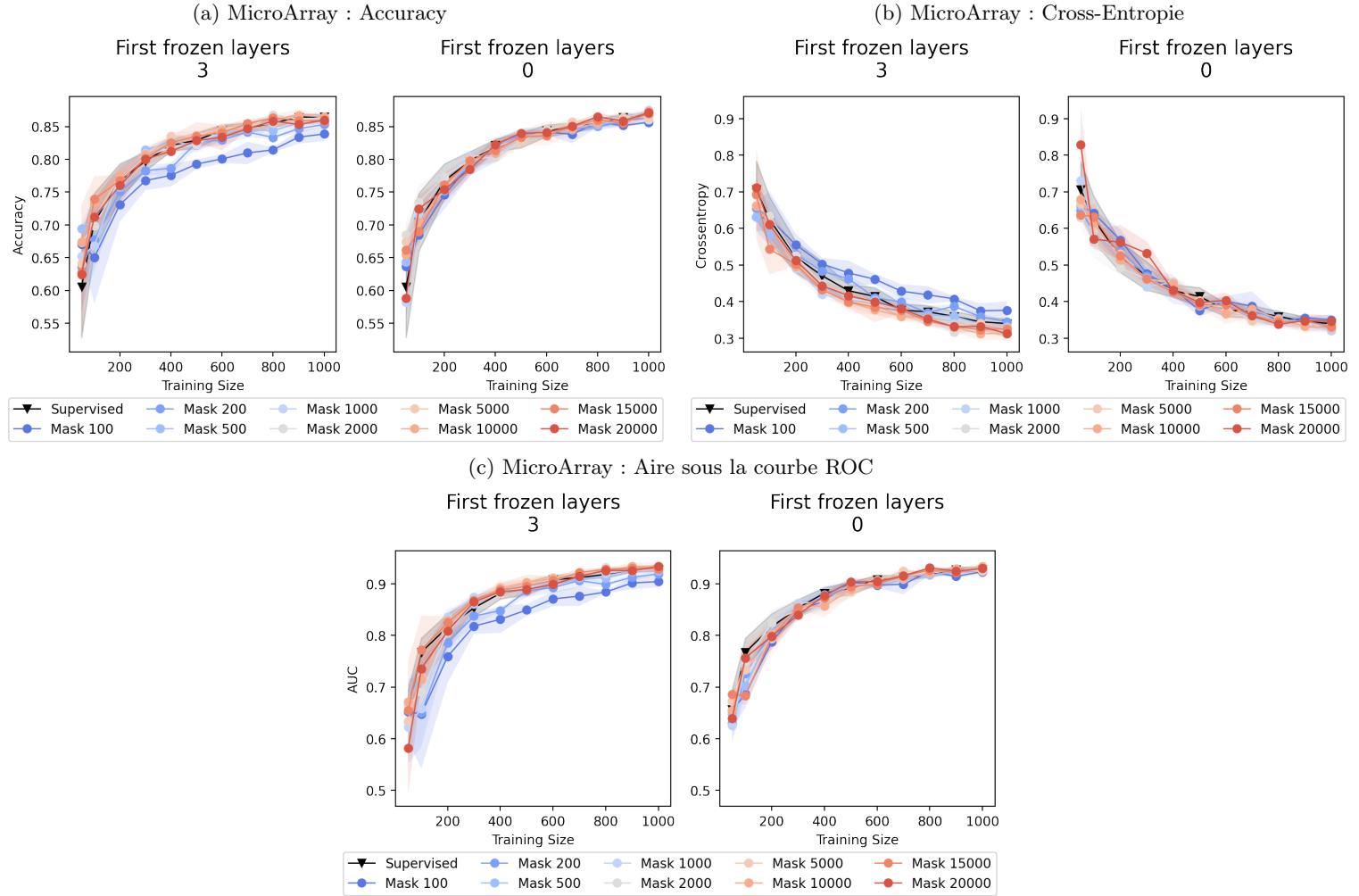


FIGURE 3.4 – Fine-tuning simple du modèle self-supervisé de prédiction de données masquées : Accuracy et cross-entropie en fonction de la taille des données d’entraînement pour les données MicroArray.

même façon que la courbe de référence du modèle supervisé. Cette courbe de référence étant d’ailleurs affichée en noir sur toutes les courbes afin de permettre la comparaison entre les résultats obtenus avec pré-training de l’encodeur et sans.

Les différentes couleurs de courbes représentent la taille des différents masques testés, dans un dégradé du bleu vers le rouge dans l’ordre croissant de la taille de masque. Les tailles de masques les plus faibles sont donc en bleu et celles les plus grandes en rouge. Dans les figures où les courbes sont en fonction de la taille du masque au lieu de la taille des données d’entraînement, les couleurs sont représentatives de la taille des données d’entraînement.

Le dernier paramètre pris en compte dans ces courbes est le nombre de couches gelées de l’encodeur. Nous avons donc à droite, avec les 0 premières couches gelées, les résultats obtenus en ré-entraînant la totalité de l’encodeur, et à gauche, avec les 3 premières couches gelées, les résultats obtenus en gelant totalement l’encodeur. Les résultats intermédiaires, avec 1 et 2 couches gelées, sont disponibles dans l’annexe B.2.

**Accuracy, Cross-Entropie et AUC** Intéressons-nous tout d’abord à l’accuracy, la cross-entropie et l’aire sous la courbe (AUC), trois métriques du réseau de neurones naturellement liées. Ce qu’il faut observer est l’allure des courbes colorées par rapport à la courbe noire. Nous souhaitons améliorer les performances du modèle supervisé, et donc avoir des courbes en fonction de la taille des données d’entraînement se trouvant dans une meilleure position que la courbe noire (au-dessus pour l’accuracy et l’AUC, en-dessous pour la cross-entropie) et qui empirerait moins lorsque le nombre d’exemples diminuerait.

**MicroArray** Lorsqu’on s’intéresse aux données MicroArray (Figure 3.4), ce n’est pas du tout le cas.

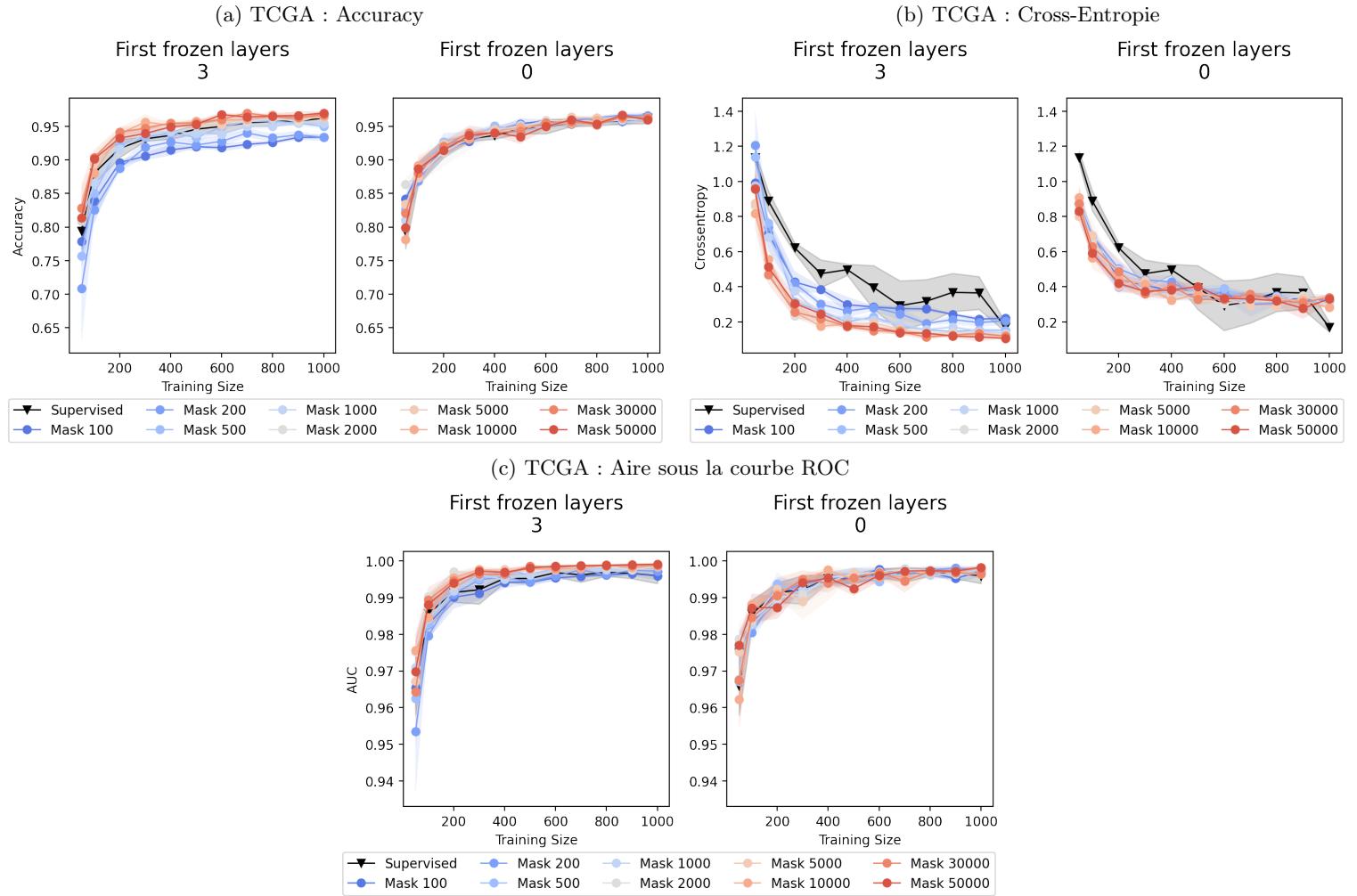


FIGURE 3.5 – Fine-tuning simple du modèle self-supervisé de prédiction de données masquées : Accuracy, cross-entropie et AUC en fonction de la taille des données d’entraînement pour les données TCGA

Quand on regarde les résultats pour le ré-entraînement complet de l’encodeur, nous ne remarquons aucune amélioration de l’accuracy, ni de la cross-entropie, ni de l’AUC. Dans ce cas-là, pré-entraîner le modèle ne permet pas de meilleures performances qu’un encodeur initialisé aléatoirement.

Quand on regarde ensuite les résultats pour le gel complet de l’encodeur, nous remarquons que pour des tailles de masques élevées, la cross-entropie et l’accuracy sont au même niveau que la courbe de référence. La cross-entropie n’est pas significativement inférieure à la courbe de référence, ce qui confirme la position de l’AUC au même niveau que la base line. Les tailles de masques plus faibles, en revanche, causent un moins bon entraînement de l’encodeur avec des valeurs des métriques moins bonnes que la base line.

**TCGA** Lorsqu’on regarde les mêmes courbes réalisées avec les jeu de données TCGA (Figure 3.5), les performances ne sont pas vraiment améliorées non plus.

Pour le ré-entraînement de l’encodeur, on observe que les courbes de l’accuracy et de l’AUC sont exactement au même niveau que la courbe du modèle supervisé. La cross-entropie a l’air d’être améliorée, sans être vraiment significative au vu des deux autres métriques. Une explication à cette baisse est que le modèle serait plus confiant dans sa prédiction pour les données qu’il prédit correctement mais resterait le même dans ses autres prédictions.

Dans le cas du gel complet de l’encodeur, on remarque les mêmes différences entre les tailles de masque où les plus faibles performent moins bien que le modèle de référence, et donc que les plus grandes tailles de masques. Et ce, bien qu’on remarque encore la baisse au niveau de la cross-entropie. Pour les tailles de masques plus grandes, en revanche, nous pouvons remarquer une très légère amélioration des trois métriques affichées, si légère qu’elle n’est pas vraiment significative.

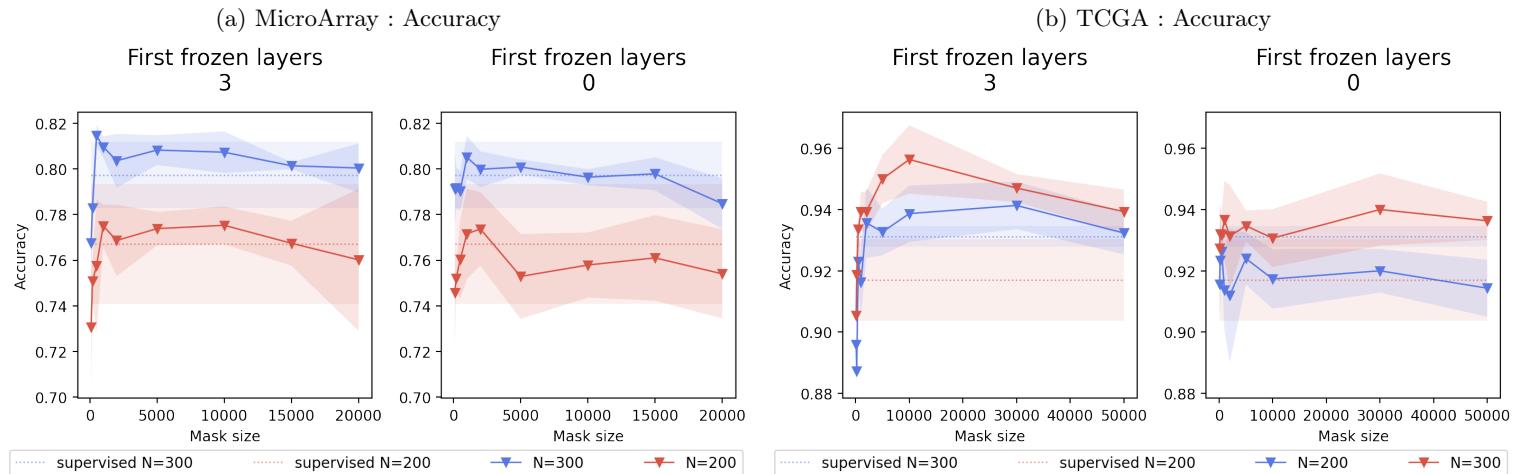


FIGURE 3.6 – Fine-tuning simple du modèle self-supervised de prédiction de données masquées : Accuracy en fonction de la taille de masque pour les données MicroArray et TCGA. Deux tailles de jeu d’entraînement sont présentées : 200 et 300.

**Taille du masque** L’influence de la taille du masque a déjà été observée avec les couleurs sur les figures précédentes 3.4 et 3.5, mais on peut l’observer plus précisément en affichant, non pas la courbe de l’accuracy en fonction de la taille des données d’entraînement, mais celle en fonction de la taille du masque (Figure 3.6). Nous pouvons alors remarquer une tendance sur l’influence de la taille du masque. En effet, en plus de nous confirmer nos observations des courbes précédentes, on peut remarquer que les courbes croissent rapidement jusqu’à une taille de masque représentant environ 10% de la dimension des exemples, et une fois ce niveau atteint, stagnent malgré l’augmentation massive de la taille des masques. Ou du moins quand l’encodeur est gelé. Lorsqu’il ne l’est pas, on remarque plutôt une courbe qui décroît à partir du 10% de variables masquées.

La taille du masque, pour permettre un bon pré-entraînement de l’encodeur, doit donc être au-dessus de 10% du nombre de variables par échantillons, et au-delà de ce seuil, le modèle n’apprend pas de meilleure représentation des données.

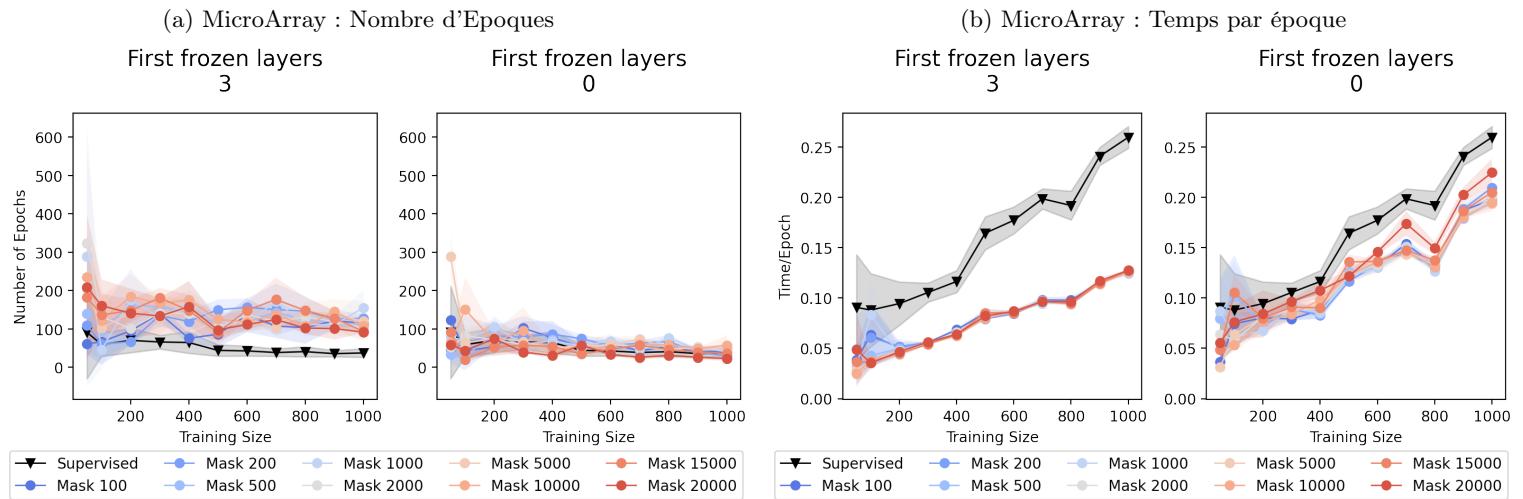


FIGURE 3.7 – Fine-tuning simple du modèle self-supervised de prédiction de données masquées : Nombre d’époques et temps moyen par époque en fonction de la taille des données d’entraînement pour les données MicroArray.

**Nombre d’époques** Un autre élément important indiquant la performance d’un réseau de neurones, notamment lors de l’utilisation d’un pré-entraînement du modèle, est le nombre d’époques qu’il faut au modèle pour finir l’apprentissage. Celui-ci est présenté en figure 3.7 (a) pour les données MicroArray et en figure 3.8 pour les données TCGA.

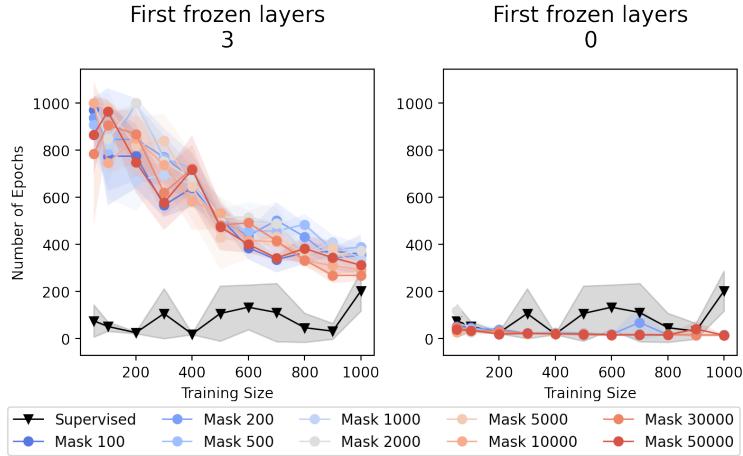


FIGURE 3.8 – Fine-tuning simple du modèle self-supervisé de prédiction de données masquées : Nombre d'époques en fonction de la taille des données d'entraînement pour les données TCGA.

Pour les données MicroArray, on remarque que le nombre d'époques nécessaire lorsqu'on gèle l'encodeur est supérieur à celui lors du ré-entraînement total. Il est donc plus difficile d'entraîner à convergence le réseau avec juste le classificateur que le réseau entier. Le nombre d'époques en ré-entraînant l'encodeur ne change pas par rapport à la courbe de référence du modèle supervisé. Le pré-entraînement de l'encodeur n'a donc pas permis de se rapprocher de la représentation finale des données utilisée dans ce cas-là.

Pour les données TCGA, on remarque qu'il est aussi plus difficile d'atteindre la convergence lorsque l'on entraîne seulement le classificateur et non tout le réseau. On remarque en plus que la taille des données d'entraînement influe aussi le nombre d'époques que met le classificateur à apprendre. Contrairement aux données MicroArray, on observe cette fois-ci une légère baisse du nombre d'époques nécessaire lorsque l'on ré-entraîne entièrement le modèle. Sans vraiment être sûrs que cette baisse soit significative, on peut quand même dire que dans le cas des données TCGA le modèle de prédiction des données masquées permet de nous rapprocher de la bonne représentation des données que forme l'encodeur.

Pour les deux jeux de données, la taille du masque n'influe pas le nombre d'époques nécessaire pour le fine-tuning du modèle.

**Temps moyen par époque** En plus du nombre d'époques, nous pouvons aussi observer le temps moyen par époque pour vérifier le bon fonctionnement du modèle. Les résultats entre les données TCGA et MicroArray n'étant pas différents dans la forme des courbes qu'ils créent, seuls ceux du jeu de données MicroArray (Figure 3.7) sont affichés. ~~Ge que nous informent ces courbes, c'est que le temps par époque du fine-tuning ne diffère pas en fonction de la taille du masque, et est proportionnel à la taille des données d'entraînement.~~ On remarque de plus que le temps du fine-tuning de ré-entraînement complet de l'encodeur est similaire à celui du modèle supervisé, alors qu'on voit que l'entraînement avec gel de l'encodeur prend un temps par époque beaucoup plus court. C'est logique, vu que la mise à jour de tous les poids de l'encodeur n'a alors pas à être faite.

**Conclusion** Au vu des bons résultats observés, surtout lorsque l'on gèle l'encodeur pré-entraîné, on peut dire que le modèle self-supervisé de prédiction de données masquées permet de créer une bonne représentation des données qui permet, même gelée, d'atteindre le niveau d'accuracy d'un modèle supervisé. Cependant, nous pointons aussi l'absence d'amélioration des performances. La représentation des données construite est assez bonne pour égaler les performances du supervised learning, mais pas assez pour les surpasser.

### 3.3.2.2 Fine-tuning avec données étiquetées et données prédites non étiquetées

Le premier fine-tuning n'ayant pas amélioré les résultats, nous en testons un autre découlant du premier. En plus de reprendre le modèle supervisé, ce fine-tuning utilise les données prédites par le modèle self-supervisé pour ajouter des données non étiquetées à notre ensemble de données. Ces données non étiquetées sont ensuite utilisées pour complexifier la fonction de coût en essayant de faire en sorte que les prédictions des données étiquetées et non étiquetées se rapprochent.

Les résultats observés pour ce fine-tuning ne présentent pas de différence entre les deux jeux de données. Ceux montrés pour cette partie appartiennent donc aux données MicroArray et les résultats pour le jeu TCGA sont disponibles dans l'annexe B.3.

### 3.3.2.2.1 Implémentation

Malgré ses différences, les paramètres d' entraînement, tel que l'optimiseur et le taux d'apprentissage restent les mêmes que ceux utilisés avec le modèle supervisé. Nous avons donc SGD avec momentum à 0.85 et taux d'apprentissage à 1e-3 pour le jeu de données MicroArray et l'optimiseur Adam avec taux d'apprentissage à 1e-4 pour le jeu de données TCGA.

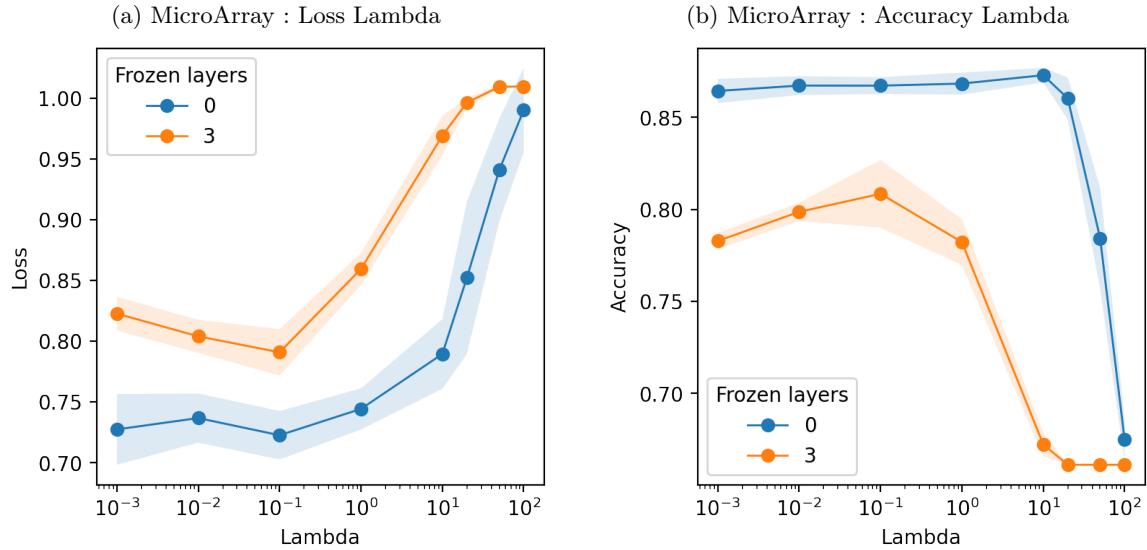


FIGURE 3.9 – Fine-tuning avec données non-étiquetées du modèle self-supervisé de prédiction de données masquées : Accuracy et Loss en fonction de lambda pour les données MicroArray.

**Lambda** Le paramètre suivant à tester est le lambda apparaissant dans le calcul de la fonction de coût. Ce paramètre est un ajout de ce modèle et doit servir à balancer la partie de l'erreur quadratique moyenne avec la partie de la cross-entropie. On observe donc la loss globale et l'accuracy pour différents lambdas variant entre 1e-3 et 1e2 (Figure 3.9). On voit donc que pour zéro couche gelée, le minimum de la loss se trouve en un lambda égal 0.1 alors que le maximum de l'accuracy se trouve lui en un lambda égal 10 et pour trois couches gelées, le meilleur lambda est 0.1. Nous allons donc tester dans la suite, trois valeurs de lambda : 0.1, 1 et 10.

### 3.3.2.2 Résultats

Maintenant que le modèle est construit, nous pouvons observer ces performances. Les tests effectués et les courbes de leurs résultats sont les mêmes que précédemment avec le fine-tuning n'utilisant que les données étiquetées. Nous retrouvons donc l'accuracy (Figure 3.10 et ??) et les différentes fonctions de coûts (Figure 3.11) comme métriques de ce modèle de fine-tuning, ainsi que le nombre d'époques et le temps moyen par époque (Figure 3.12).

Nous avons donc toujours les différentes tailles de masques, représentées par des couleurs (de bleu vers rouge par ordre croissant) pour les courbes en fonction de la taille des données d' entraînement, avec la courbe en noir étant la courbe de référence du modèle supervisé.

De plus, les deux nombres de couches gelées, 0 et 3, pour respectivement le ré- entraînement complet de l'encodeur et le gel total de l'encodeur, sont toujours tous deux testés.

**Accuracy et Cross-Entropie** Nous testons alors les performances du modèle de fine-tuning avec données non étiquetées en fonction de trois lambda différents : 0.1, 1 et 10 (Figure 3.10).

Lorsque l'on regarde pour le ré- entraînement de l'encodeur tout d'abord, on voit que les performances ne sont pas améliorées. L'accuracy n'est en aucun cas meilleure que la courbe de référence en noir, et on voit même qu'elle est moins bonne lorsque l'on s'intéresse aux plus grandes tailles de masques avec

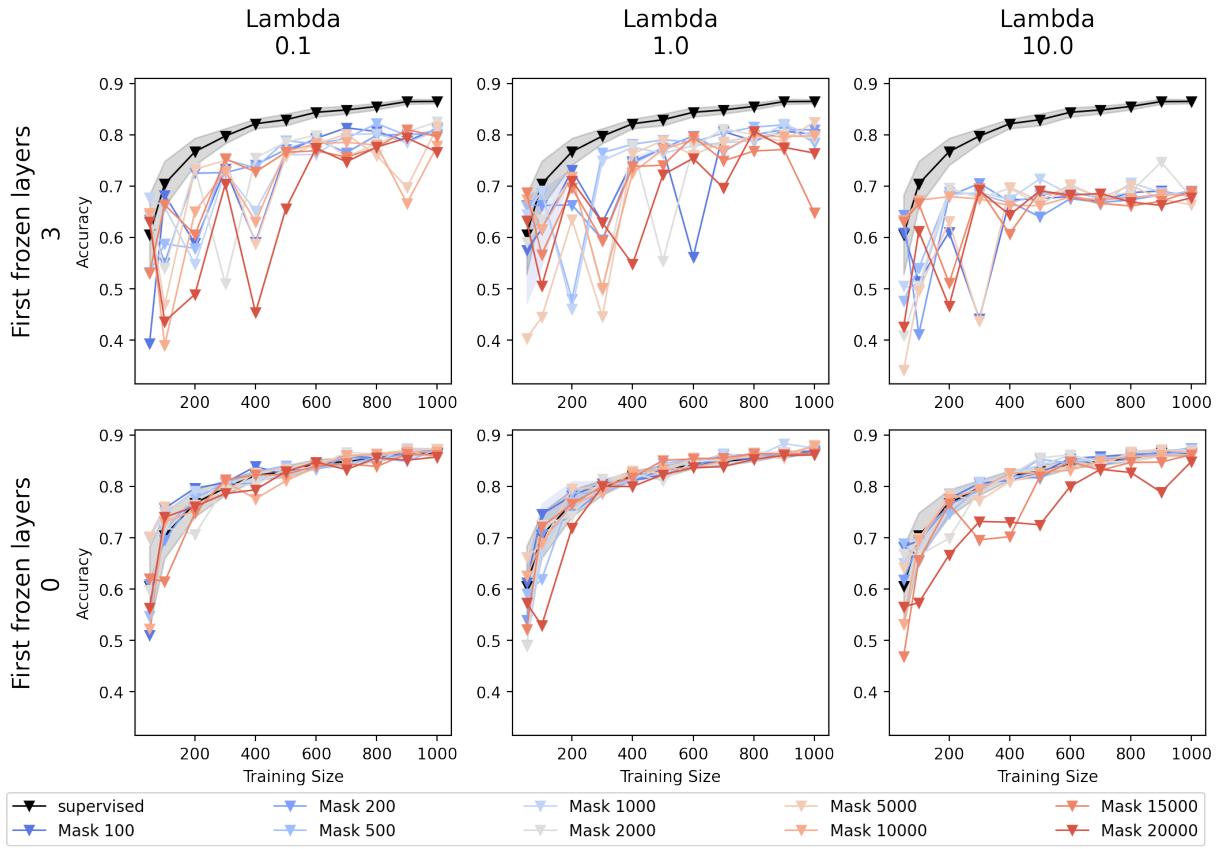


FIGURE 3.10 – Fine-tuning avec données non-étiquetées du modèle self-supervisé de prédiction de données masquées : Accuracy en fonction de la taille des données d’entraînement pour les données MicroArray

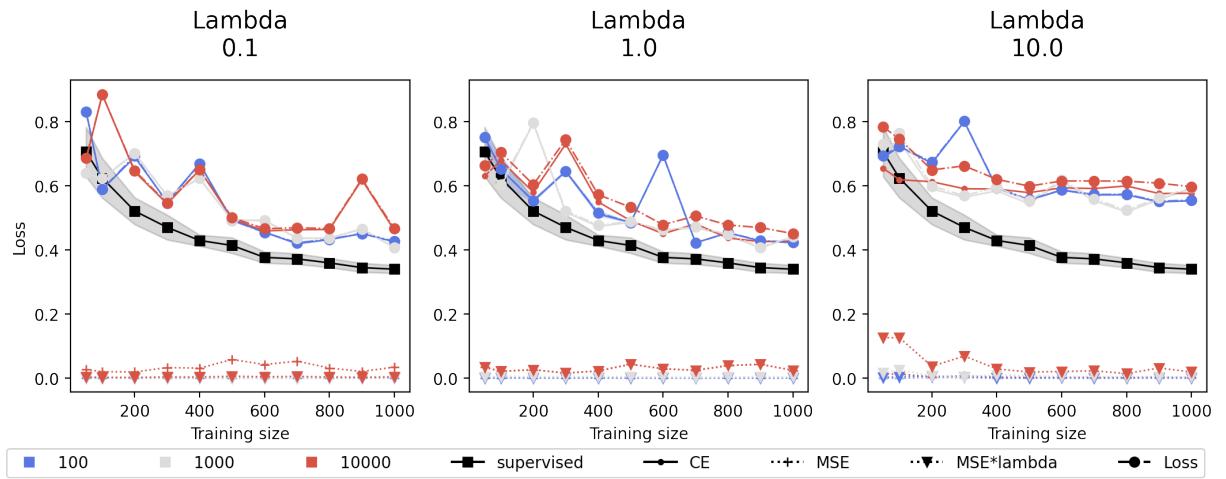


FIGURE 3.11 – Fine-tuning avec données non-étiquetées du modèle self-supervisé de prédiction de données masquées : Détail des fonctions de coût en fonction de la taille des données d’entraînement pour les données MicroArray

le lambda égal à 10. L’ajout de la MSE avec les données non étiquetées ne permet pas d’améliorer les résultats observés précédemment avec le fine-tuning simple.

Lorsque l’on s’intéresse aux résultats avec l’encodeur gelé, on remarque que le modèle n’arrive plus à prédire correctement et n’atteint pas l’accuracy de la courbe de référence, se plaçant largement en dessous. Pour le cas extrême de lambda égal à 10, on remarque que même pour des grandes tailles N, l’accuracy est très basse (<70%).

L’ajout de l’erreur quadratique moyenne dans la fonction de coût a donc l’air d’empêcher le modèle, ne

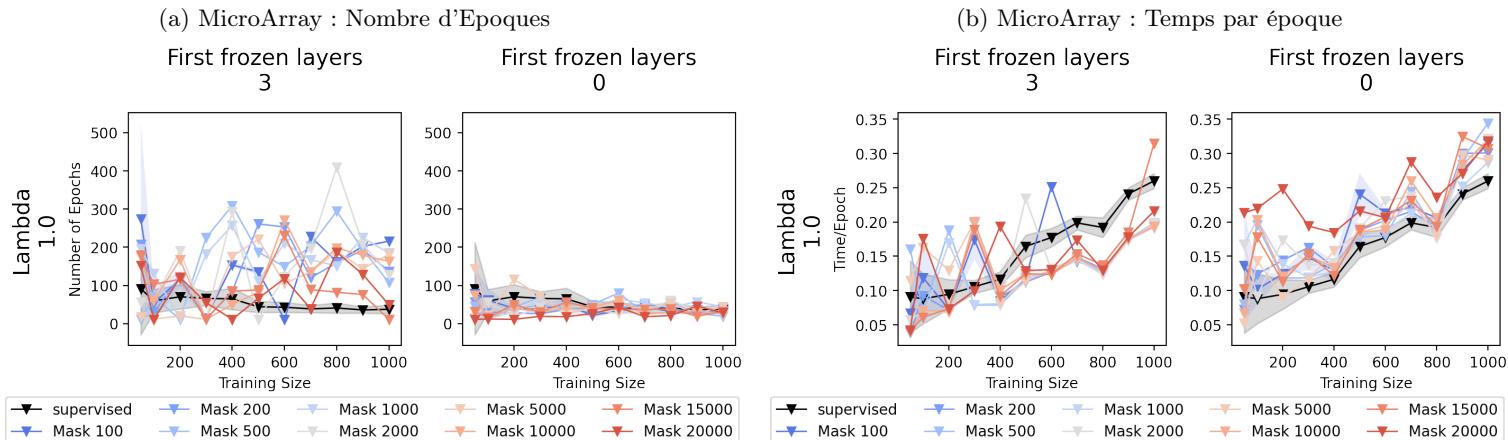


FIGURE 3.12 – Fine-tuning avec données non-étiquetées du modèle self-supervised de prédiction de données masquées : Nombre d'époques et temps par époque en fonction de la taille des données d'entraînement. Données Microarray

lui permettant plus d'apprendre à prédire correctement les phénotypes. Pour essayer <sup>de</sup> comprendre comment cet ajout modifie le modèle, voyons comment les différentes parties de la fonction de coût s'agencent.

**Fonction de coût** La fonction de coût utilisé dans ce modèle est constituée à la fois d'une cross-entropie avec les données étiquetées et d'une MSE avec les données non étiquetées. On peut donc regarder pour les trois lambda 0.1, 1 et 10, comment ces différentes parties de la loss s'agencent pour former la loss globale, et se comparent à la cross-entropie du modèle supervisé. Nous avons donc la figure 3.11 qui présente avec la cross-entropie du modèle supervisé en noir, les différentes parties formant la fonction de coût : cross-entropie, MSE, MSE\*lambda, loss globale.

Nous remarquons tout d'abord que la MSE, même multipliée par lambda, est largement inférieure à la cross-entropie alors que nous avons vu l'impact qu'elle a sur l'accuracy, métrique directement liée à la cross-entropie. Nous regardons ensuite le niveau de la cross-entropie, la partie concernant juste la prédiction du bon phénotype. On peut voir que celle-ci est bien au-dessus de la cross-entropie de référence, ce qui est cohérent avec l'accuracy observée. La loss globale est au niveau ou un peu supérieur à la cross-entropie en fonction du lambda et donc de l'ajout de la MSE.

La MSE, malgré sa faible valeur, empêche donc le modèle par la cross-entropie. Bien que les éléments pris en compte par la cross-entropie restent les mêmes que pour le modèle de fine-tuning précédent, la différence, c'est-à-dire l'ajout de la MSE, suffit à l'empêcher et le modèle n'apprend plus assez bien.

**Taille du masque** Nous avions remarqué des différences quand à la taille du masque dans le précédent modèle de fine-tuning avec seulement les données étiquetées. Pour ce modèle-ci, nous ne remarquons plus ces différences, et ce dans aucune des métriques. Elle n'a aucune influence sur ce modèle de fine-tuning.

**Nombre d'époques et temps moyen par époque** Concernant le nombre d'époques (Figure 3.12 (a)), on observe qu'il reste similaire à la courbe de référence lorsque le modèle est ré-appris dans sa globalité, et comme observé pour le fine-tuning simple, il augmente quand l'encodeur est gelé.

Le temps par époque (Figure 3.12 (b)) est bien supérieur à celui observé précédemment dans le modèle de fine-tuning avec seulement des données étiquetées. C'est normal au vu des opérations supplémentaires pour créer les données non étiquetées et calculer la fonction de coût. Ce temps moyen par époque est toujours réduit lorsqu'on gèle l'encodeur, le modèle n'ayant plus à mettre à jour ce dernier.

Ces observations sont logiques et ne montrent aucune potentielle amélioration des performances de prédiction.

**Conclusion** L'ajout de la prise en compte de données non étiquetées prédites par le modèle self-supervised de prédiction de données masquées, ne permet donc pas d'améliorer les performances du modèle supervisé. Cela empêche même les performances du fine-tuning simple avec seulement les données étiquetées, en prenant plus de temps à l'exécution.

## 3.4 Contrastive Learning

Après avoir pu tester un premier modèle de self-supervised learning avec une tâche prétexte de prédiction de données masquées, nous pouvons tester une autre technique de self-supervised learning : le contrastive learning. Avant de pouvoir comparer ce modèle avec le modèle supervisé lorsqu'on l'utilisera pour le fine-tuning, nous allons étudier plusieurs paramètres pour construire un bon modèle de contrastive learning.

### 3.4.1 Modèle de contrastive learning

#### 3.4.1.1 Implémentation

Les différents hyperparamètres et paramètres à choisir pour le modèle de contrastive learning sont le projecteur, le type de masque, la température et le calcul de la fonction de coût avec quelles paires négatives sont prises en compte et la température.

**Encodeur et Projecteur** Le modèle de contrastive learning est composé d'un encodeur et d'un projecteur. L'encodeur ne change pas et est toujours identique à celui utilisé dans le modèle supervisé. Pour le projecteur, la question s'est posé d'en mettre ou non. Des tests (Annexe C.1) ont été effectués sur la pertinence d'un projecteur d'une couche à 500 neurones et montrent que la présence d'un tel projecteur empire les résultats. On a donc choisi de simplement avoir un projecteur vide. La sortie de l'encodeur est donc aussi celle du projecteur ~~et le~~ qui est utilisé pour la contrastive loss.

**Optimiseur** L'optimiseur utilisé pour le contrastive learning, et son taux d'apprentissage, reste le même quel que soit le jeu de données, ou le type de masque utilisé. Celui qui performe le mieux est l'optimiseur Adam avec comme taux d'apprentissage 1e-5.

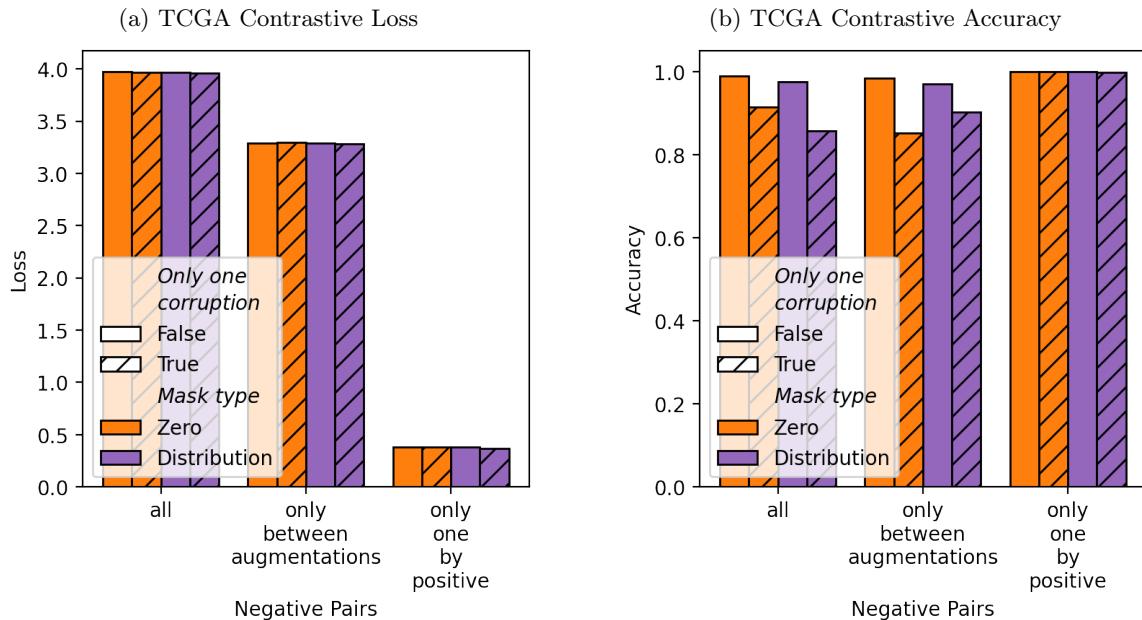


FIGURE 3.13 – Modèle contrastive learning : Loss (a) et Accuracy (b) pour différents calculs de fonction de coût, nombres et types de corruptions. Les résultats présentés ici ont été obtenus avec le jeu de données TCGA, mais les mêmes résultats sont obtenues avec le jeu de données MicroArray.

L'optimiseur choisi permet d'utiliser au mieux la fonction de coût, mais il reste encore à préciser le calcul de cette dernière.

**Paires Négatives** Le calcul de la fonction de coût se fait toujours en commençant par la similarité entre les paires et en les comparant aux valeurs objectives de 1 pour les paires positives et 0 pour les paires négatives. Nous pouvons influer ce calcul en choisissant quelles paires négatives sont prises en compte. Nous avons donc le choix entre les prendre toutes ('all'), ne prendre que celles entre projections venant

d'augmentations différentes ('only between augmentation') ou n'en prendre seulement qu'une par paire positive ('only one by positive') et ainsi équilibrer leur nombre.

Nous observons donc la valeur de la fonction de coût en fonction de ces différents ensembles de paires négatives (Figure 3.13 (a)). On remarque facilement qu'équilibrer le nombre de paires positives et négatives permet d'avoir le meilleur coût. La même conclusion peut être tirée en regardant la contrastive accuracy (Figure 3.13 (b)), mesurant l'habileté du réseau à trouver les bonnes paires positives.

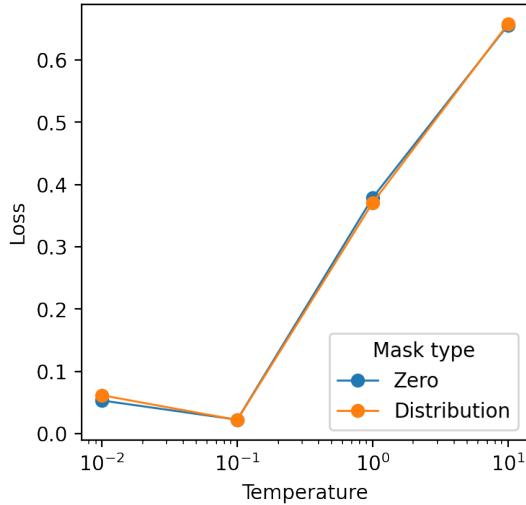


FIGURE 3.14 – Modèle contrastive learning : Loss en fonction de la température pour les deux types de masques. Les résultats présentés ici ont été obtenus avec le jeu de données TCGA, mais les mêmes résultats sont obtenus avec le jeu de données MicroArray.

**Température** Un deuxième paramètre important de la fonction de coût est la température. Elle permet d'ajuster la similarité des paires. La figure 3.14 présente le coût en fonction de la température pour les deux types de masques. On remarque donc que la température qui permet d'attendre le plus bas coût est 0.1, quel que soit le type de masque.

**Nombre de corruptions** Le nombre de corruptions est aussi à prendre en compte pour la structure du modèle. On peut en avoir deux, et ainsi comparer les projections de ces vues, ou n'en avoir qu'une, et ainsi comparer la projection de cette vue avec la projection de la donnée elle-même.

Nous pouvons voir, dans la figure 3.13, que le coût n'est pas modifié que l'on utilise une ou deux augmentations, et qu'on ne remarque pas non plus de différence d'accuracy lorsqu'on équilibre les paires positives et négatives. Par contre, les résultats d'accuracy observés pour les deux autres ensembles de paires négatives laissent à penser qu'utiliser deux corruptions serait mieux que d'en utiliser qu'une. C'est donc ce qui est choisi pour les tests suivants.

### 3.4.1.2 Résultats

Maintenant que le modèle est construit, nous pouvons regarder, de la même façon que le modèle self-supervisé précédent, l'évolution du coût en fonction de la taille du masque, ainsi que l'influence du type de masque.

**Loss by Mask size** A cause de sa croissance exponentielle, nous choisissons d'afficher la courbe de la contrastive loss sous un axe logarithmique (Figure 3.15). Nous remarquons que le coût et l'accuracy empirent avec l'augmentation de la taille du masque, tout en restant dans de très bonne valeur tant que le masque n'excède pas environ 90% du nombre de variables pour une donnée.

Cette figure montre aussi de meilleurs résultats en utilisant deux augmentations au lieu d'une, ce qui confirme les observations déjà faites auparavant.

**Type de masque** Au cours de toutes ces figures, nous avons pu comparer les deux types de masques disponibles. Les résultats obtenus avec ces deux masques sont très similaires, pour ne pas dire identiques. Ce n'est que lorsqu'on arrive aux extrêmes de taille de masque avec un taux de corruption à 90 et 100% que l'on aperçoit une différence, qui alors sûrement causée par la perte plus importante d'informations

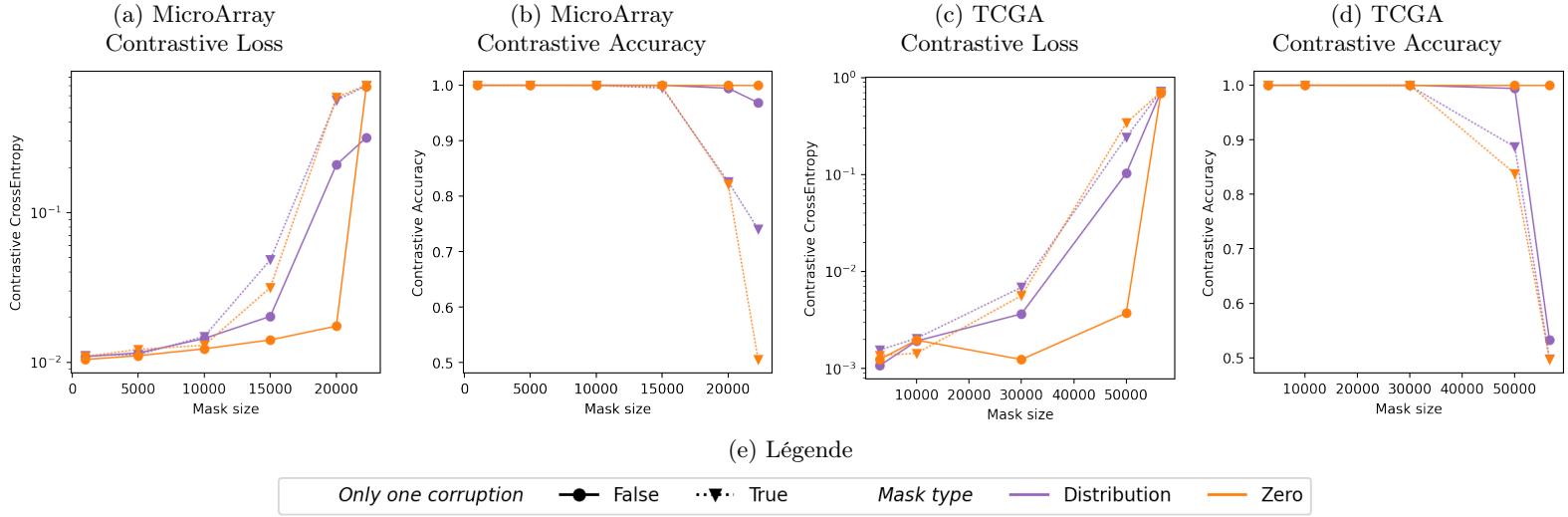


FIGURE 3.15 – Modèle contrastive learning : Loss et Accuracy en fonction de la taille de masque pour les données (a,b) MicroArray et (c,d) TCGA.

lorsque les variables sont remplacées par des zéro, une valeur ne contenant aucune information, que lorsque les variables sont remplacées par des valeurs déjà présentes dans le batch.

### 3.4.2 Fine-tuning

Nous utilisons maintenant les modèles entraînés avec les différentes tailles de masques et les deux types de masques pour effectuer le fine-tuning avec les données étiquetées. Le modèle repris est le même que le modèle supervisé, avec seulement l'encodeur, extrait du modèle de contrastive learning, qui est pré-entraîné.

Les paramètres d'apprentissage, tel que l'optimiseur et le taux d'apprentissage sont alors les mêmes que l'apprentissage en supervised learning. Nous avons SGD avec taux d'apprentissage à  $1e-3$  et momentum à 0.85 pour le jeu de données MicroArray et l'optimiseur Adam avec taux d'apprentissage à  $1e-4$  pour le jeu de données TCGA.

Les différentes métriques observées pour analyser les performances du fine-tuning du modèle contrastif restent les mêmes que celles précédemment vues, soient l'accuracy, la cross-entropie, l'aire sous la courbe, le nombre d'époques et le temps moyen par époques. Elles sont toutes affichées en fonction de la taille des données d'entraînement de la même façon que la courbe de référence du modèle supervisé à laquelle on les compare.

La courbe de référence est d'ailleurs toujours affichée en noir alors que les autres couleurs de courbes représentent la taille des différents masques testés, dans un dégradé du bleu vers le rouge dans l'ordre croissant de la taille de masque.

Le paramètre unique au fine-tuning, c'est-à-dire le nombre de couches gelées de l'encodeur, est toujours représenté. Nous avons donc à chaque fois, les résultats obtenus en ré-entraînant la totalité de l'encodeur (0 couches gelées), et les résultats obtenus en gelant la totalité de l'encodeur (3 couches gelées).

**Accuracy, Cross-Entropie et AUC** Nous allons nous intéresser tout d'abord aux trois principales métriques : l'accuracy, la cross-entropie et l'aire sous la courbe (AUC), et ce pour les deux jeux de données et les deux types de masque.

**MicroArray** Lorsqu'on s'intéresse aux données MicroArray (Figure 3.16), nous ne remarquons pas d'amélioration des performances de prédictions, au contraire. Les deux types de masque, avec le masque à zéro à gauche et le masque distribution, qui remplace par des valeurs du même batch, à droite, montrent les mêmes résultats. Seules les valeurs extrêmes de tailles de masque montrent une différence avec des performances moins bonnes pour le masque à zéro.

Quand on regarde les résultats pour le ré-entraînement complet de l'encodeur, nous ne remarquons aucune amélioration de l'accuracy, ni de la cross-entropie, ni de l'AUC. Le ré-entraînement complet du modèle fait que les résultats du modèle supervisé sont tout juste atteints.

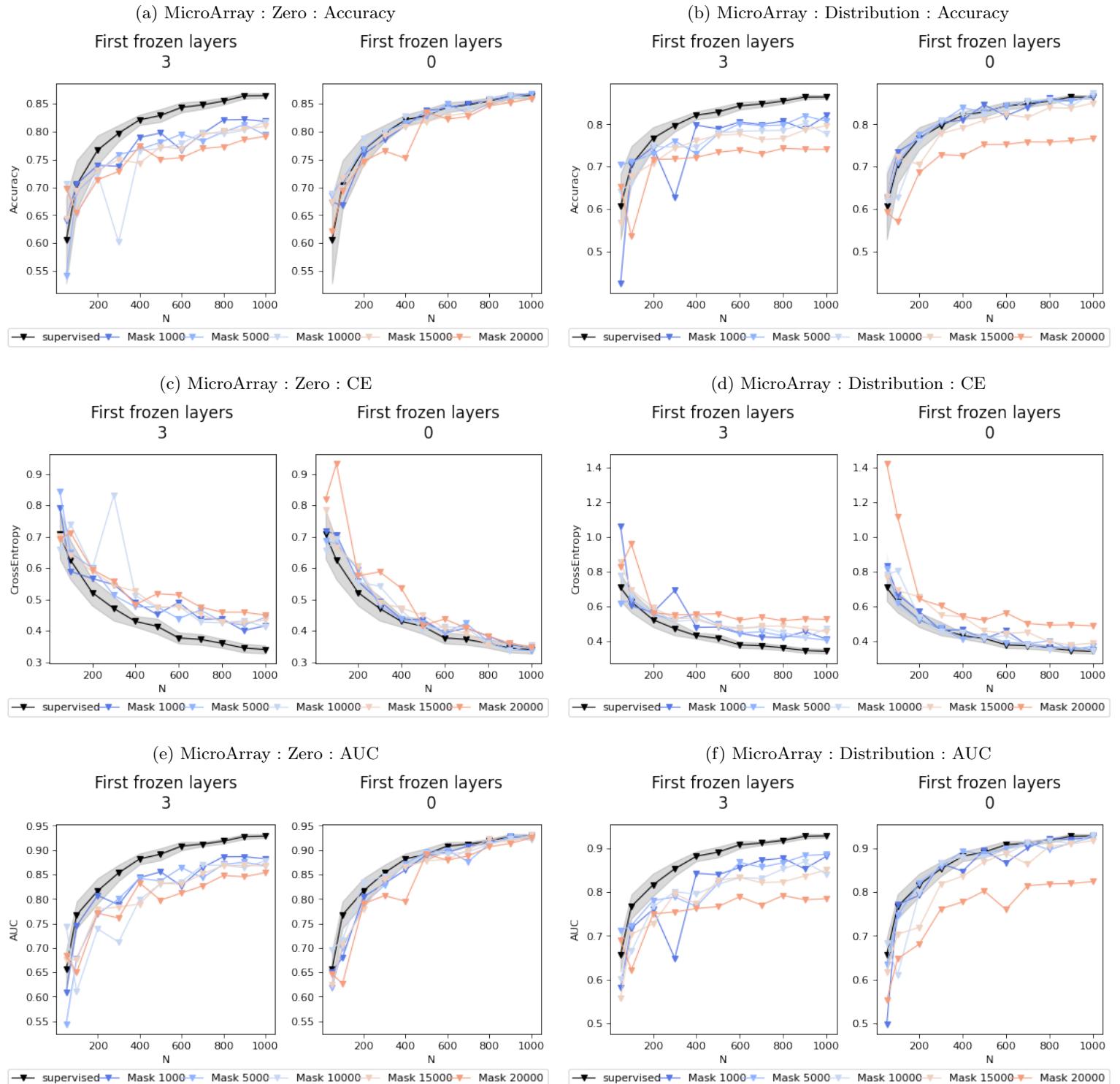


FIGURE 3.16 – Fine-tuning simple du modèle contrastive learning : Accuracy, cross-entropie et AUC en fonction de la taille des données d’ entraînement pour les données MicroArray.

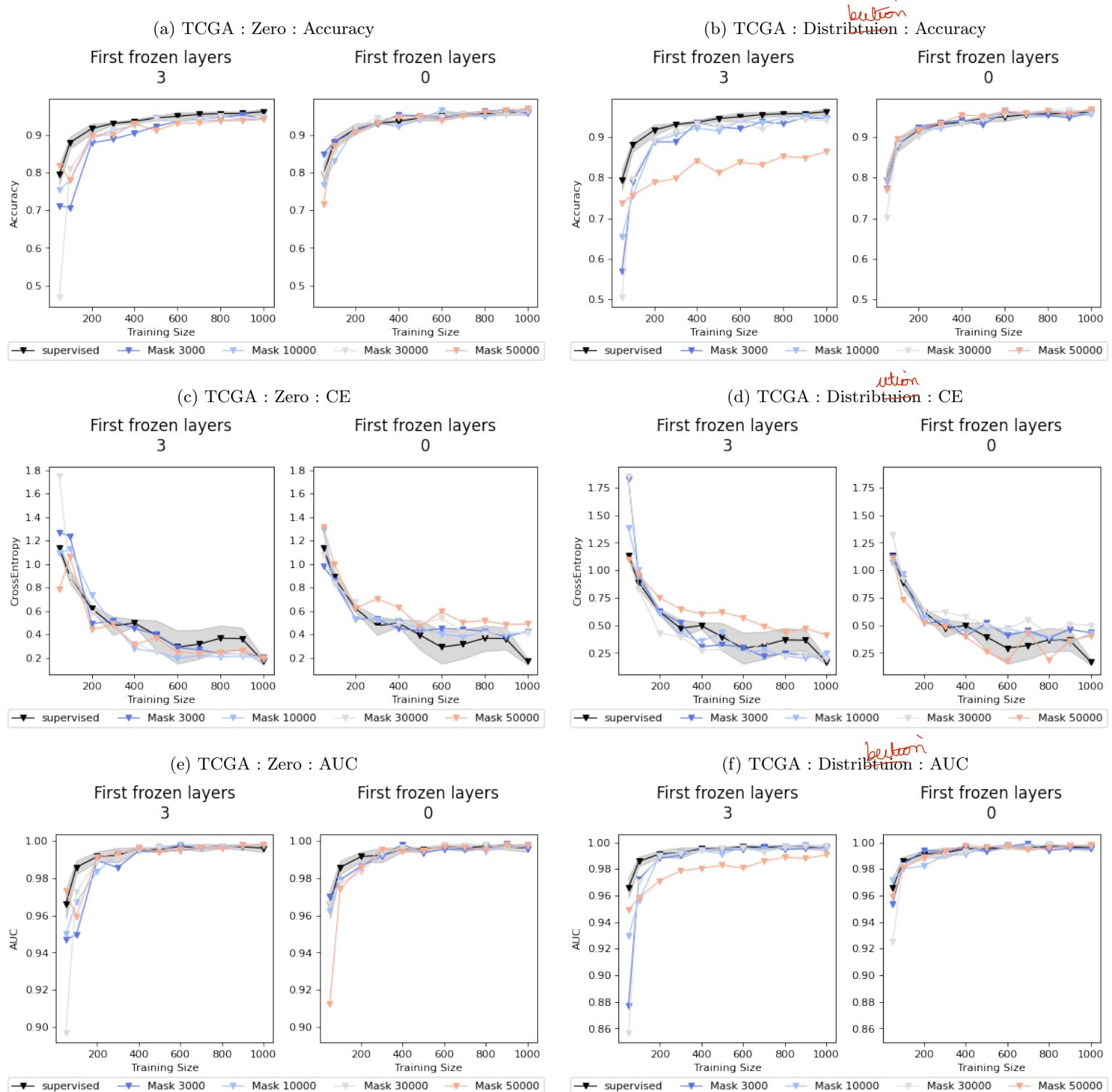


FIGURE 3.17 – Fine-tuning simple du modèle contrastive learning : Accuracy, cross-entropie et AUC en fonction de la taille des données d’entraînement pour les données TCGA.

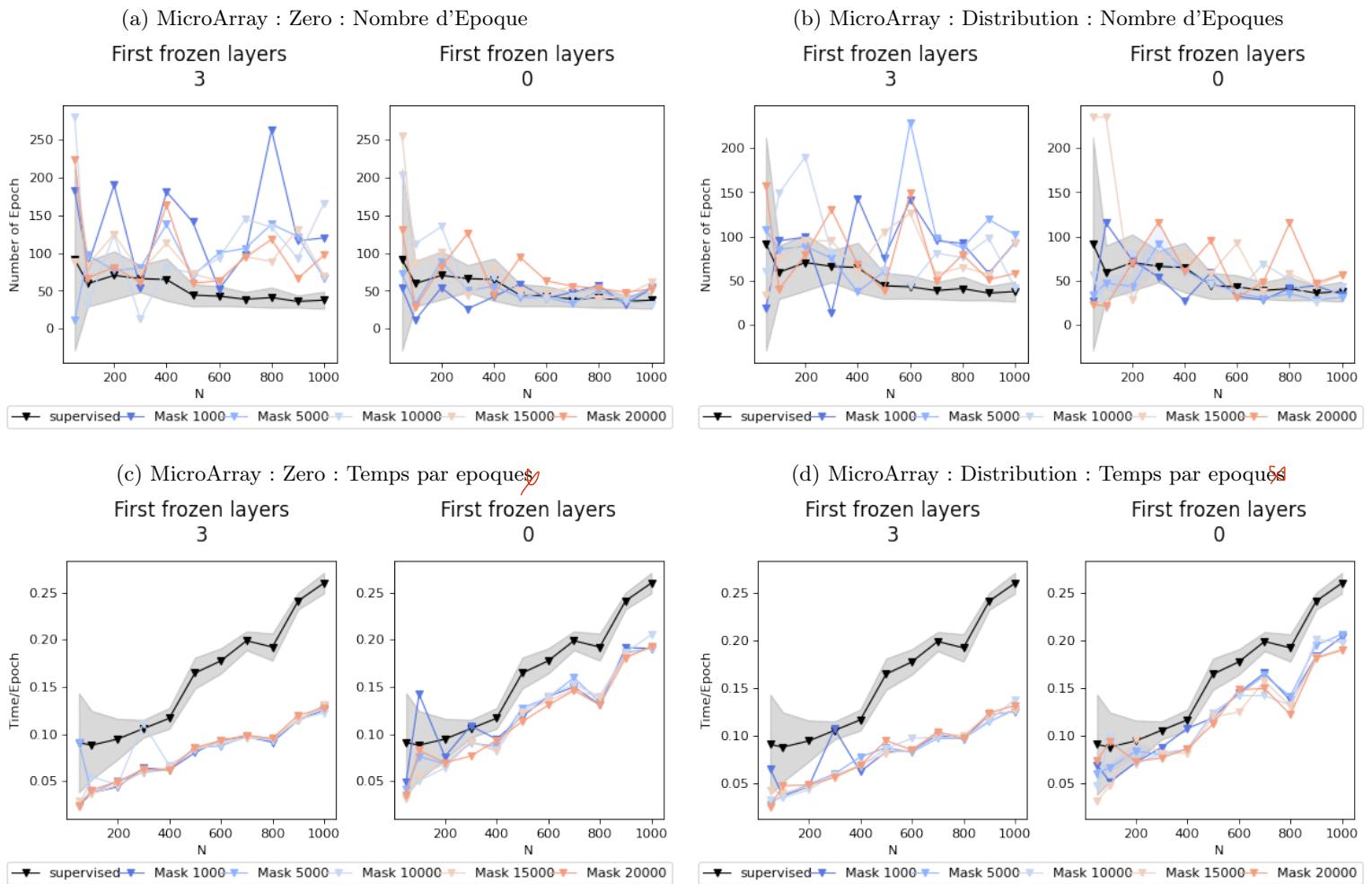


FIGURE 3.18 – Fine-tuning simple du modèle contrastive learning : Nombre d'époques et temps moyen par epochs en fonction de la taille des données d'entraînement pour les données MicroArray.

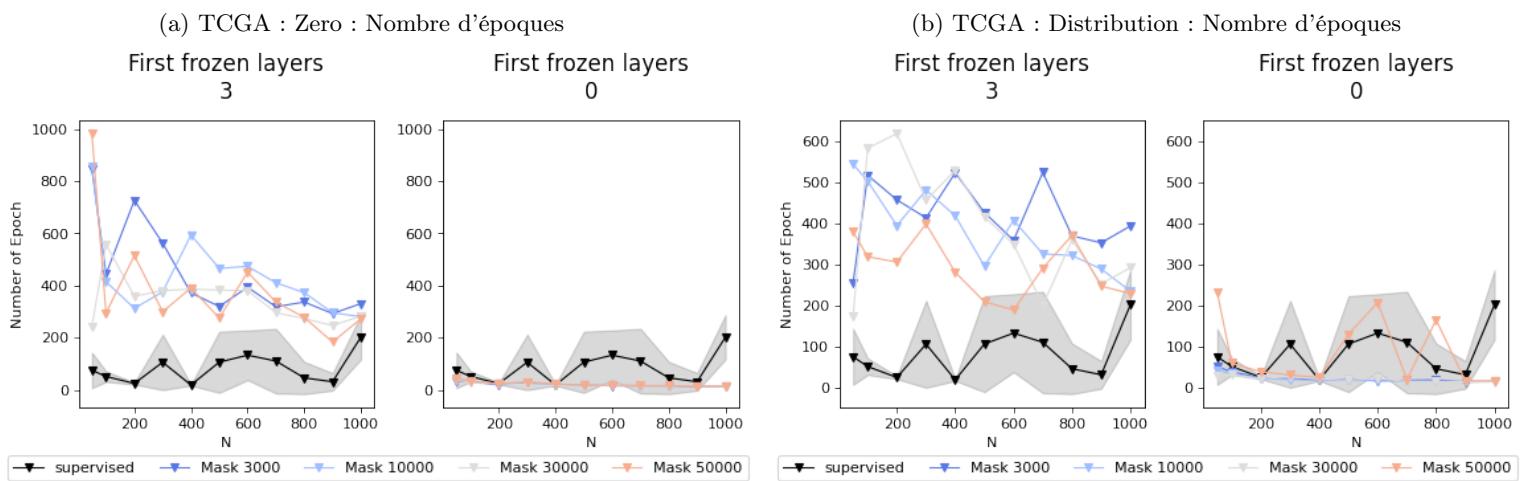


FIGURE 3.19 – Fine-tuning simple du modèle contrastive learning : nombre d'époques en fonction de la taille des données d'entraînement pour les données TCGA.

Quand on regarde les résultats obtenus avec le gel complet de l'encodeur, les courbes des différentes métriques n'atteignent même pas le niveau du modèle supervisé.

L'utilisation du contrastive learning n'améliore donc pas les performances, au contraire, elle les empire.

**TCGA** Lorsqu'on s'intéresse aux données TCGA (Figure 3.17), les performances observées sont légèrement meilleures. Nous remarquons toujours la perte de performances lorsque de grandes tailles de masques sont utilisées et l'absence de différence entre les deux types de masques.

Pour le ré-entraînement de l'encodeur, on observe que les courbes de l'accuracy, de l'AUC et de la cross-entropie sont, comme les données précédentes, au même niveau que la courbe du modèle supervisé. Les performances ne sont améliorées et le modèle est ré-appris pour atteindre les mêmes performances qu'en supervisé.

Dans le cas du gel complet de l'encodeur, on remarque que les trois métriques sont meilleures que celles observées avec les données MicroArray. Cependant, l'accuracy reste en-dessous de la courbe de référence, montrant clairement que les performances ne sont pas améliorées. Le modèle a l'air par contre plus confiant dans les bonnes prédictions, comme on le voit avec une légère amélioration de la cross-entropie et une aire sous la courbe au même niveau que la courbe de référence.

Toutefois, malgré ces améliorations, l'utilisation du contrastive learning n'améliore pas non plus les performances du modèle supervisé.

**Nombre d'époques** Pour confirmer ces conclusions, nous allons regarder l'évolution du nombre d'époque. Celui-ci est présenté en figure 3.18 (a,b) pour les données MicroArray et en figure 3.19 pour les données TCGA.

Pour les deux jeux de données, on remarque encore que le nombre d'époques nécessaire lorsqu'on gèle l'encodeur est supérieur à celui lors ré-entraînement total. Il est donc plus lent d'ajuster le modèle en ne jouant que sur le classificateur qu'en mettant à jour le modèle en entier. On observe aussi que ce nombre d'époques reste invariable par rapport à la taille du masque. Ceci ne change pas des observations des précédents fine-tuning.

Pour les données MicroArray, le nombre d'époques en ré-entraînant l'encodeur ne change pas par rapport à la courbe de référence du modèle supervisé, il est juste plus sujet à variation, ce qui vient sûrement de la qualité variable de l'encodeur extrait. Le pré-entraînement de l'encodeur n'a donc pas permis de se rapprocher de la représentation finale des données utilisées dans ce cas-là.

Pour les données TCGA, par contre, le nombre d'époques nécessaire lorsque l'on ré-entraîne entièrement le modèle est légèrement inférieur à celui du modèle supervisé de référence. Sans vraiment être sûre que cette baisse soit significative, on peut quand même dire que dans le cas des données TCGA le modèle de contrastive learning permet au moins de nous rapprocher de la bonne représentation des données que forme l'encodeur, à défaut d'en améliorer les performances.

**Temps moyen par époque** Le temps par époques est un indicatif de la qualité de nos modèles. Les courbes de temps moyen par époque en fonction de la taille des données d'entraînement n'étant pas différentes selon l'origine des données (TCGA ou MicroArray), seules celles du jeu de données MicroArray (Figure 3.18 (b)) sont présentées. Ces différentes courbes montrent que le temps par époque du fine-tuning ne change pas en fonction de la taille du masque, et est proportionnel à la taille des données d'entraînement. On remarque de plus que le temps du fine-tuning du ré-entraînement complet de l'encodeur est inférieur à celui du modèle supervisé, l'apprentissage y serait donc étonnamment plus rapide. Pour l'entraînement avec gel de l'encodeur, l'apprentissage prend un temps par époque beaucoup plus court que le modèle supervisé, ce qui est cohérent avec l'absence de mise à jour de l'encodeur dans ce cas-là.

Notre modèle a donc l'air d'apprendre correctement, ce qui est confirmé en regardant la position des paires de projections dans un espace bidimensionnel (Annexe C.2) où on voit que les paires positives sont rapprochées et éloignées des autres.

**Conclusion** Nous ne nous pouvons donc que remarquer l'absence d'amélioration des performances de prédiction de phénotypes avec l'utilisation du contrastive learning, qui a pourtant l'air de bien fonctionner. Cette technique, bien que populaire et performante dans de nombreux domaines comme l'image, n'a pas réussi à donner place à de meilleurs résultats, ni même vraiment à s'en rapprocher. La raison a cela vient possiblement des différentes augmentations choisies, pas assez pertinentes quand à nos données et tâche de prédiction en aval.

# Conclusion

Le but de ce stage était d'améliorer les performances de prédiction d'un phénotype, la présence ou le type de cancer, pour des jeux de données de petites tailles. Les données transcriptomiques, ou médicales en général, sont bien plus coûteuses à produire que d'autres données comme celle d'image. Les jeux de données sont donc beaucoup plus petits. Trouver un modèle permettant de bonnes performances malgré la faible quantité de données étiquetées est donc le problème à résoudre pour le progrès du deep learning dans le domaine de la santé.

J'ai pu créer un modèle supervisé permettant de montrer la base des performances à améliorer, performances que l'on voit nettement diminuer avec la diminution du nombre de données. Différents modèles de self-supervised learning ont alors pu être faits pour essayer d'améliorer ces performances.

Tout d'abord, on a créé le modèle self-supervised utilisant la tâche prétexte de prédiction de données artificiellement masquées. Deux sortes de fine-tuning ont ainsi pu être testées, et comparées avec les performances du modèle supervisé.

Le fine-tuning avec seulement les données étiquetées a montré que le modèle de prédiction des données masquées permettait de construire une aussi bonne représentation des données que le modèle supervisé. Ce modèle comprend donc bien les données et au moins une partie de leur concept sous-jacent permettant la tâche de prédiction. Cependant, cela n'a pas suffi pour améliorer significativement les performances de prédictions.

En deuxième tentative, nous avons testé une variante du fine-tuning précédent en y impliquant les données prédites par le modèle self-supervised, augmentant ainsi le nombre d'exemples à disposition prenant part à la fonction de coût. Cette tentative était un échec. Au lieu de pousser le modèle dans la bonne voie, plus proche de prédictions attendues comme on l'espérait, c'est l'inverse qui s'est produit avec des performances résultantes moins bonnes que celle du modèle de référence.

Ensuite, nous sommes passés à une autre technique de self-supervised learning : le contrastive learning. Représentant l'état de l'art du self-supervised learning dans le domaine de l'image, nous avons essayé de reproduire ses résultats sur nos données transcriptomiques. C'est ainsi que deux sortes d'augmentations de vues ont pu être testées : l'une masquait simplement les données à zéro, l'autre les remplaçait par d'autres valeurs existantes dans le batch. Ces deux masques, bien que bien différents, ont au final donné des performances identiques. De plus, ces performances sont à peine arrivées à la hauteur du supervised learning. Le contrastive learning n'a donc, lui non plus, pas réussi à dépasser les performances du modèle supervisé.

Chaque modèle a pu être testé avec deux jeux de données transcriptomiques ayant des origines différentes, l'un provenant de puce à ADN et l'autre de séquençage ARN. Les performances obtenues sur ces deux types de données sont relativement proches. Les données TCGA donnent de légère amélioration de performances comparées à celles MicroArray, mais ces améliorations ne sont pas assez fortes pour considérer que les performances dépassent celles obtenues en supervised learning.

En conclusion, nous n'avons donc pas réussi à améliorer les performances de prédictions obtenues avec le modèle supervisé. L'absence d'une grande variété de tâches prétextes en est peut-être la cause, la qualité de celles-ci ayant été prouvées plusieurs fois critiques dans d'autres articles relatant du domaine de l'image.

Pour améliorer ces résultats, il faudrait donc trouver d'autres tâches prétextes à réaliser sur nos données, ce qui donnerait peut-être de meilleurs résultats.

Bien que l'objectif de ce stage, qui était donc d'améliorer les performances de prédiction de phénotype, n'ait pas été rempli, ce stage m'a permis d'étendre mes connaissances sur le deep learning. J'ai pu, par l'intermédiaire de tous les modèles testés, apprendre de nouvelles techniques du deep learning et approfondir mes connaissances sur tout ce qui peut composer un réseau de neurones. Mes compétences techniques ont aussi pu être améliorées avec notamment l'apprentissage d'une bibliothèque Python pour le machine learning : Tensorflow.

# Bibliographie

- [1] Christina B. AZODI et al. “Transcriptome-Based Prediction of Complex Traits in Maize[OPEN]”. In : *The Plant Cell* 32.1 (oct. 2019), p. 139-151. ISSN : 1040-4651. DOI : 10.1105/tpc.19.00332. eprint : [https://academic.oup.com/plcell/article-pdf/32/1/139/36889375/plcell\\_v32\\_1\\_139.pdf](https://academic.oup.com/plcell/article-pdf/32/1/139/36889375/plcell_v32_1_139.pdf). URL : <https://doi.org/10.1105/tpc.19.00332>.
- [2] Dara BAHRI et al. “SCARF : Self-Supervised Contrastive Learning using Random Feature Corruption”. In : *CoRR* abs/2106.15147 (2021). arXiv : 2106.15147. URL : <https://arxiv.org/abs/2106.15147>.
- [3] Ting CHEN et al. “A Simple Framework for Contrastive Learning of Visual Representations”. In : *CoRR* abs/2002.05709 (2020). arXiv : 2002.05709. URL : <https://arxiv.org/abs/2002.05709>.
- [4] Ting CHEN et al. “Big Self-Supervised Models are Strong Semi-Supervised Learners”. In : *CoRR* abs/2006.10029 (2020). arXiv : 2006.10029. URL : <https://arxiv.org/abs/2006.10029>.
- [5] Yifei CHEN et al. “Gene expression inference with deep learning”. In : *Bioinformatics* 32.12 (fév. 2016), p. 1832-1839. ISSN : 1367-4803. DOI : 10.1093/bioinformatics/btw074. URL : <https://doi.org/10.1093/bioinformatics/btw074>.
- [6] Spyros GIDARIS, Praveer SINGH et Nikos KOMODAKIS. “Unsupervised Representation Learning by Predicting Image Rotations”. In : *International Conference on Learning Representations*. 2018. URL : <https://openreview.net/forum?id=S1v4N210->.
- [7] Ian GOODFELLOW, Yoshua BENGIO et Aaron COURVILLE. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [8] National Cancer INSTITUTE. *Genomic Data Commons Data Portal*. 2021. URL : <https://portal.gdc.cancer.gov/>.
- [9] Longlong JING et Yingli TIAN. “Self-supervised Visual Feature Learning with Deep Neural Networks : A Survey”. In : *CoRR* abs/1902.06162 (2019). arXiv : 1902.06162. URL : <http://arxiv.org/abs/1902.06162>.
- [10] Gustav LARSSON, Michael MAIRE et Gregory SHAKHNAROVICH. “Colorization as a Proxy Task for Visual Understanding”. In : *CoRR* abs/1703.04044 (2017). arXiv : 1703.04044. URL : <http://arxiv.org/abs/1703.04044>.
- [11] Yuanyuan LI et al. “A comprehensive genomic pan-cancer classification using The Cancer Genome Atlas gene expression data”. eng. In : *BMC genomics* 18.1 (juil. 2017). PMC5496318[pmcid], p. 508-508. ISSN : 1471-2164. DOI : 10.1186/s12864-017-3906-0. URL : <https://doi.org/10.1186/s12864-017-3906-0>.
- [12] Paul C. OKORO et al. “Transcriptome prediction performance across machine learning models and diverse ancestries”. In : *Human Genetics and Genomics Advances* 2.2 (2021), p. 100019. ISSN : 2666-2477. DOI : <https://doi.org/10.1016/j.xhgg.2020.100019>. URL : <https://www.sciencedirect.com/science/article/pii/S2666247720300191>.
- [13] Deepak PATHAK et al. “Context Encoders : Feature Learning by Inpainting”. In : *CoRR* abs/1604.07379 (2016). arXiv : 1604.07379. URL : <http://arxiv.org/abs/1604.07379>.
- [14] Miriam PILES et al. “Machine learning applied to transcriptomic data to identify genes associated with feed efficiency in pigs”. In : *Genetics Selection Evolution* 51.1 (mar. 2019), p. 10. ISSN : 1297-9686. DOI : 10.1186/s12711-019-0453-y. URL : <https://doi.org/10.1186/s12711-019-0453-y>.
- [15] Aaron M. SMITH et al. “Standard machine learning approaches outperform deep representation learning on phenotype prediction from transcriptomics data”. In : *BMC Bioinformatics* 21.1 (mar. 2020), p. 119. ISSN : 1471-2105. DOI : 10.1186/s12859-020-3427-8. URL : <https://doi.org/10.1186/s12859-020-3427-8>.
- [16] Aurora TORRENTE et al. “Identification of Cancer Related Genes Using a Comprehensive Map of Human Gene Expression”. In : *PLOS ONE* 11.6 (juin 2016), p. 1-20. DOI : 10.1371/journal.pone.0157484. URL : <https://doi.org/10.1371/journal.pone.0157484>.

- [17] Jinsung YOON et al. “VIME : Extending the Success of Self- and Semi-supervised Learning to Tabular Domain”. In : *Advances in Neural Information Processing Systems*. Sous la dir. de H. LAROCHELLE et al. T. 33. Curran Associates, Inc., 2020, p. 11033-11043. URL : <https://proceedings.neurips.cc/paper/2020/file/7d97667a3e056acab9aaf653807b4a03-Paper.pdf>.
- [18] Richard ZHANG, Phillip ISOLA et Alexei A. EFROS. “Colorful Image Colorization”. In : *CoRR* abs/1603.08511 (2016). arXiv : 1603.08511. URL : <http://arxiv.org/abs/1603.08511>.

# Annexes

<b>A Modèle Supervisé</b>	<b>iv</b>
A.1 Création du Modèle Supervisé . . . . .	iv
A.2 Métriques Supplémentaires du Modèle Supervisé . . . . .	v
<b>B Modèle de Prédiction de Données Masquées</b>	<b>viii</b>
B.1 Création du modèle . . . . .	viii
B.2 Fine-tuning avec données étiquetées . . . . .	ix
B.3 Fine-tuning avec données non étiquetées . . . . .	x
<b>C Modèle de Contrastive Learning</b>	<b>xiii</b>
C.1 Création du modèle . . . . .	xiii
C.2 Visualisation des paires . . . . .	xiv

# Modèle Supervisé

## A.1 Création du Modèle Supervisé

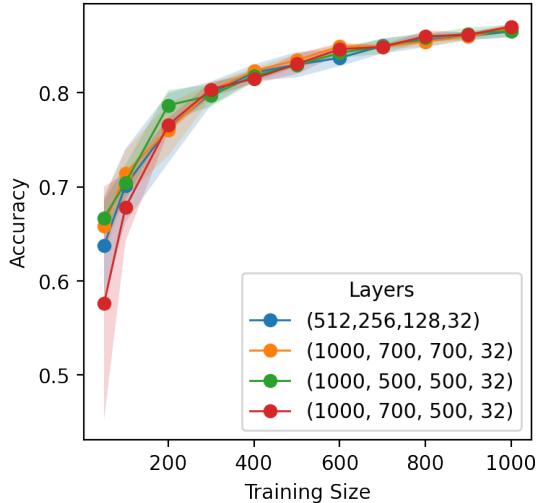


FIGURE A.1 – Accuracy en fonction de la taille de données d’entraînement pour les données MicroArray, selon différentes architectures d’encodeur.

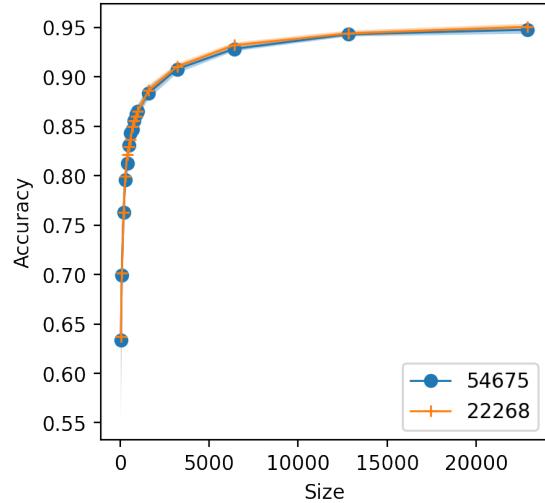


FIGURE A.2 – Accuracy en fonction de la taille de données d’entraînement pour les données MicroArray, selon le nombre de probes.

**Architecture** Plusieurs architectures d’encodeur ont été testées (Figure A.1). On observe la même évolution de l’accuracy en fonction de la taille d’entraînement quelque soit les différents nombres de neurones testés.

**Nombre de Probes** Dans l’optique d’utiliser des données supplémentaires, nous avons diminué le nombre de probes des données MicroArray. La figure A.2 montre les courbes d’accuracy en fonction de la taille d’entraînement. On observe exactement la même courbe pour les deux dimensions de données. Réduire ce nombre de probes n’impacte donc pas les performances.

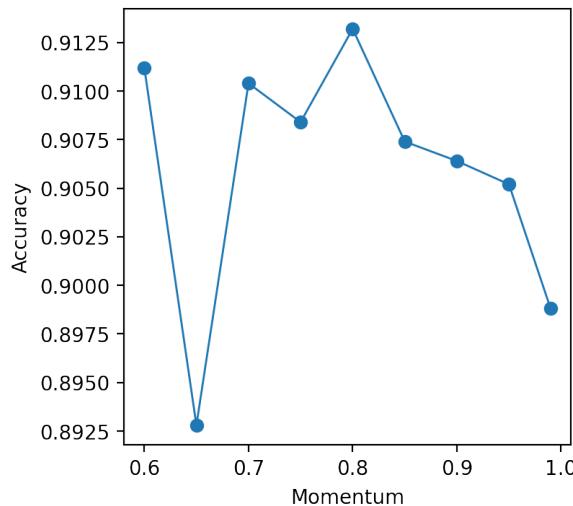


FIGURE A.3 – Accuracy en fonction du momentum utilisé pour la SGD pour les données MicroArray.

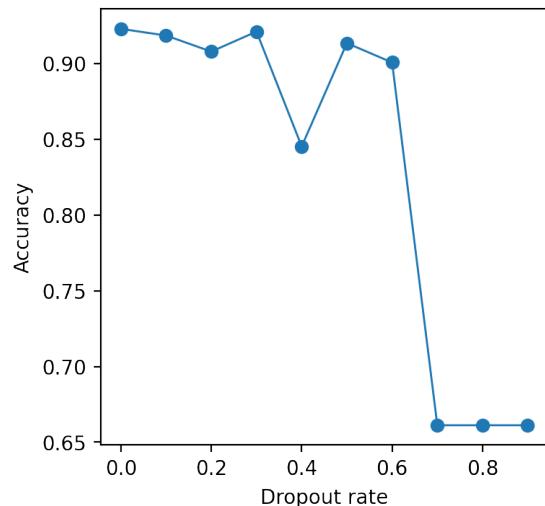


FIGURE A.4 – Accuracy en fonction du taux de dropout de l’encodeur pour les données MicroArray.

**Momentum** L’optimiseur utilisé pour les données MicroArray est SGD avec momentum. Il faut donc déterminer celui-ci (Figure A.3). L’accuracy étant très instable et n’ayant pas l’air de varier beaucoup en fonction du momentum, surtout lorsqu’on regarde entre 0.7 et 0.9, nous avons choisi la valeur 0.85.

**Dropout** Un taux de dropout est ajouté à chaque de l’encodeur. On en teste donc différentes valeurs (Figure A.4), pour finalement choisir la valeur 0.1.

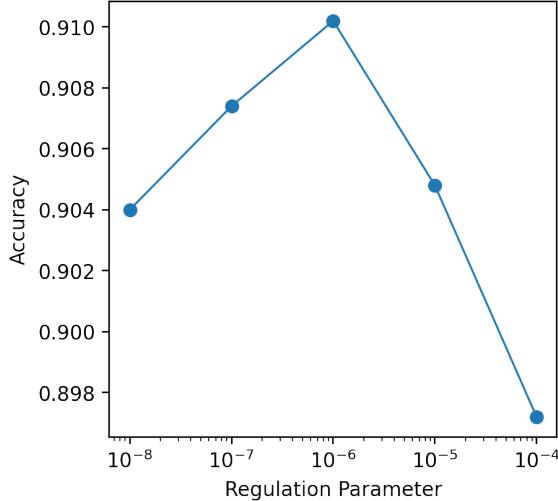


FIGURE A.5 – Accuracy en fonction du taux de régulation utilisé dans l’encodeur pour les données MicroArray. Régularisation L1 L2 au tel paramètre pour la première couche, divisé par 10 pour la deuxième, divisé par 100 pour la troisième.

Paramètres de régularisation par ordre de couche ( $10^{-p}$ )	Test Accuracy
6 7 8	0.9310
6 7	0.9436
<b>6 7 7</b>	<b>0.9472</b>
6 7 9	0.9354
6 6 7	0.9192
6 7 7 7	0.8946
6 7 8 9	0.9332
7,6 8,7 8,7	0.9251

TABLE A.1 – Accuracy pour différents paramètres de régularisation L1/L2. En gras, les paramètres retenus pour l’encodeur.

**Régularisation** Le dernier hyperparamètre testé pour le choix de l’encodeur est le taux de régularisation. Nous testons tout d’abord des taux de régularisation L1 L2, identique pour L1 et L2 et décroissant (divisé par 10) au fil des couches (Figure A.5). Les taux, par ordre de couche, qui maximise l’accuracy sont alors 1e-6, 1e-7, 1e-8. A partir de cette suite de taux, nous testons plusieurs autres combinaisons (Figure A.1). De ces tests, ressortent les taux suivants : 1e-6, 1e-7, 1e-7 qui maximisent l’accuracy et sont alors choisis pour l’encodeur.

## A.2 Métriques Supplémentaires du Modèle Supervisé

Seule l’accuracy a été présentée dans le rapport. Les autres métriques, c’est-à-dire l’aire sous la courbe, le score brier (pour les données MicroArray), la cross-entropie, le nombre d’époques et le temps par époque, sont présentées ici, en fonction de la taille des données d’entraînement, pour les données MicroArray (Figure A.7) et pour les données TCGA (Figure A.6).

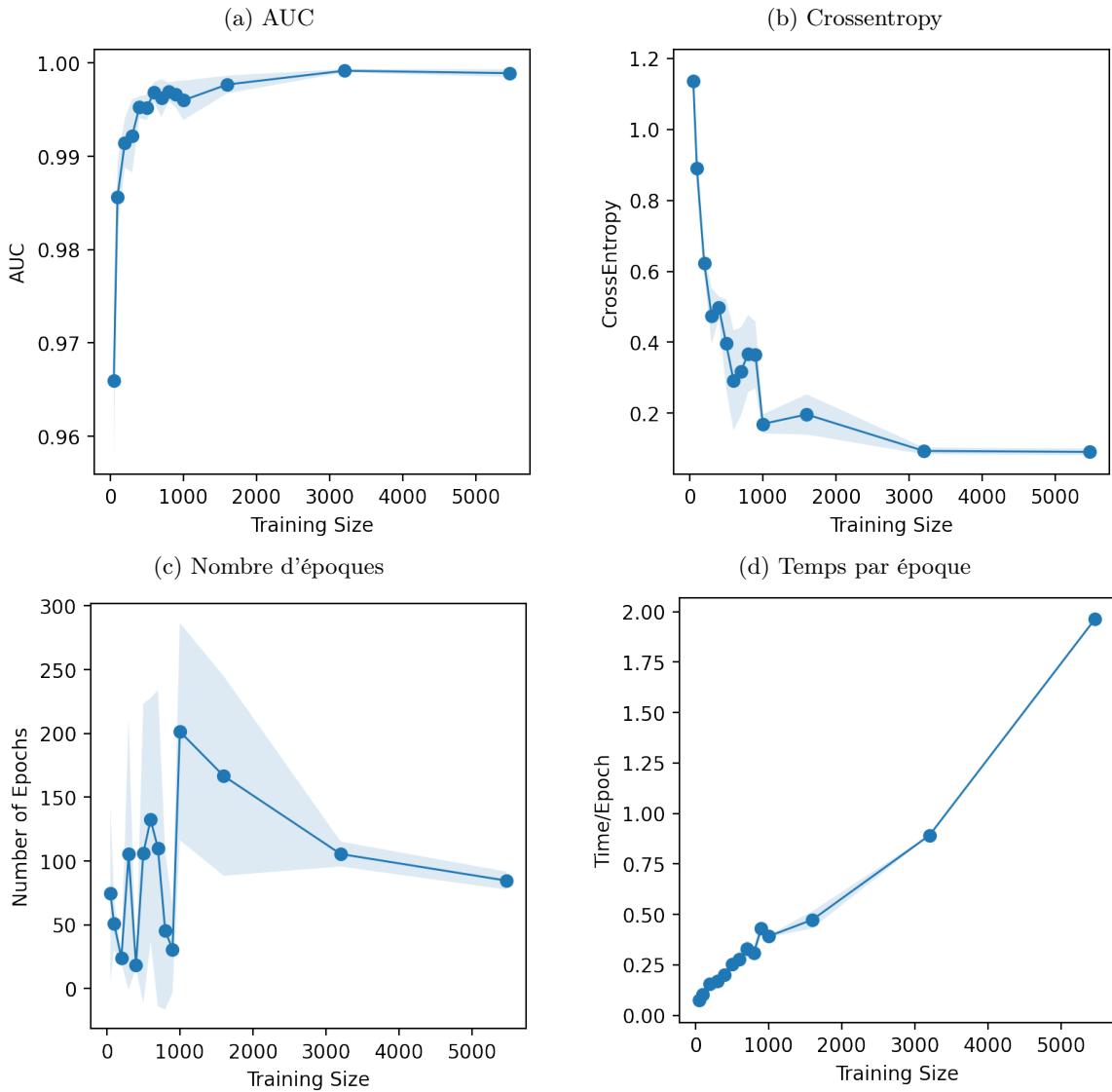


FIGURE A.6 – Autres métriques du modèle supervisé sur les données TCGA. La cross-entropie (b) et l'aire sous la courbe ROC (a) montre les mêmes résultats observés avec l'Accuracy (Figure 3.2 (b)). Le nombre d'époque (c) est très instable et ne montre pas de tendance en fonction de la taille des données d'entraînement. On observe le temps moyen par époque (d) augmenter avec la taille.

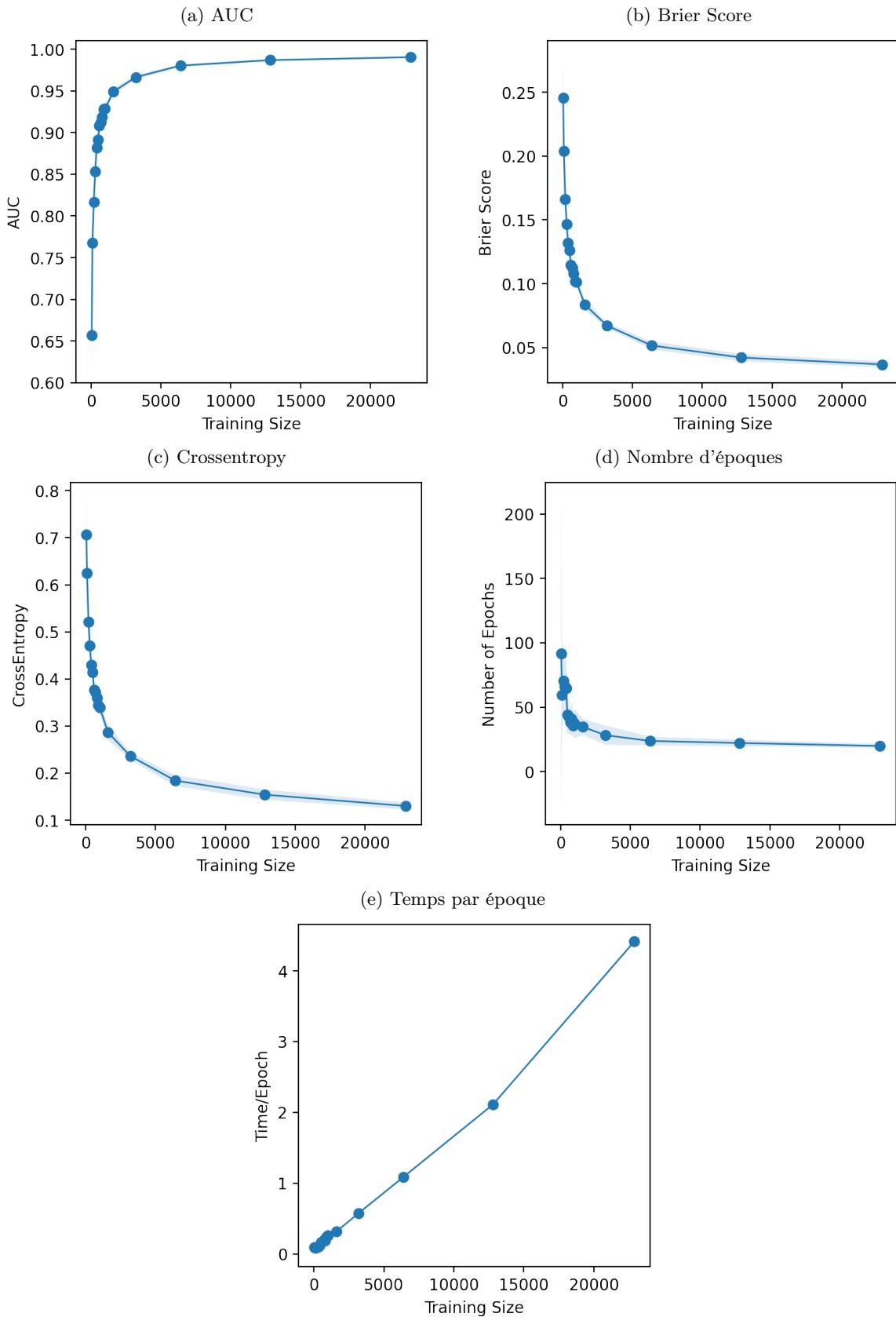


FIGURE A.7 – Autres métriques du modèle supervisé sur les données MicroArray. La crossentropy (c), le score Brier (b) et l'aire sous la courbe ROC (a) montre les mêmes résultats observés avec l'Accuracy (Figure 3.2 (a)). Le nombre d'époque (d) reste inchangé selon la taille des données d'entraînement, et on observe le temps moyen par époque (e) augmenter avec la taille.

# Modèle de Prédiction de Données Masquées

## B.1 Création du modèle

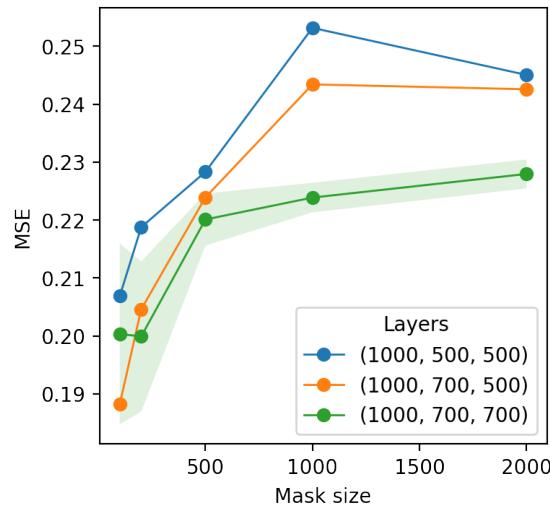


FIGURE B.1 – MSE en fonction de la taille de masque pour les données MicroArray, selon différentes architectures d'encodeur.

**Architecture de l'encodeur** Différentes architectures ont été testées pour l'encodeur (Figure B.1). Celle permettant la plus basse erreur quadratique moyenne est, par ordre de couches, celles avec les nombres de neurones 1000, 700, 700. C'est donc celle retenue pour l'encodeur. Une explication pour cette amélioration est la baisse de différence de taille entre la dernière couche de l'encodeur et la couche de décodeur, correspondant à la taille du masque.

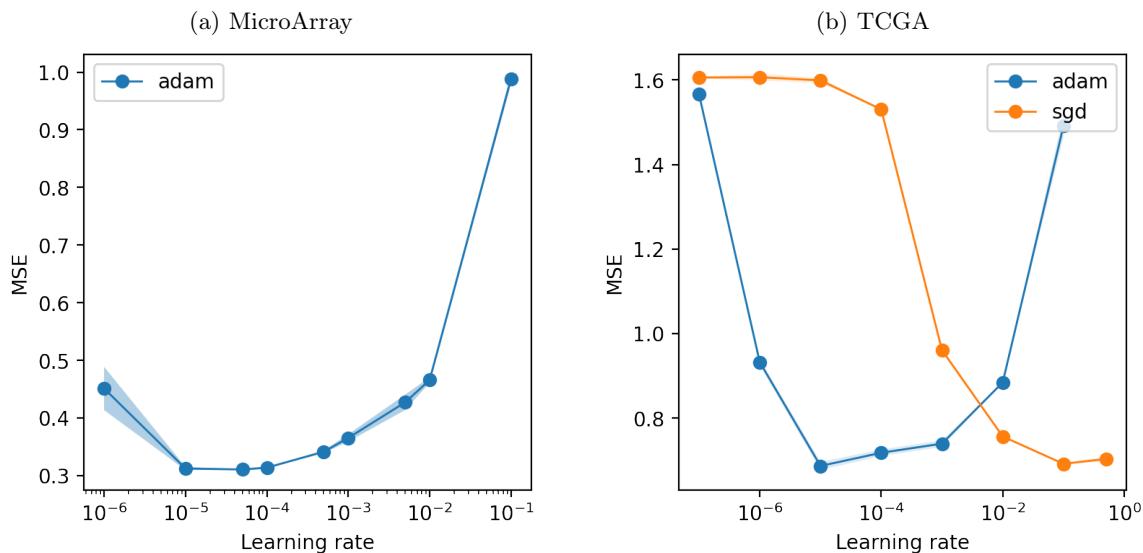


FIGURE B.2 – MSE en fonction du taux d'apprentissage, selon deux optimiseurs, pour les données (MicroArray) et les données (TCGA).

**Optimiseur et taux d'apprentissage** Comme chaque nouveau modèle, on teste ses performances selon l'optimiseur et le taux d'apprentissage (Figure B.2). On teste cela pour chacun des jeu de données. Les valeurs retenues, pour lesquelles la MSE est la plus basse, sont donc Adam à 5e-5 pour les données MicroArray, et Adam à 1e-5 pour les données TCGA.

## B.2 Fine-tuning avec données étiquetées

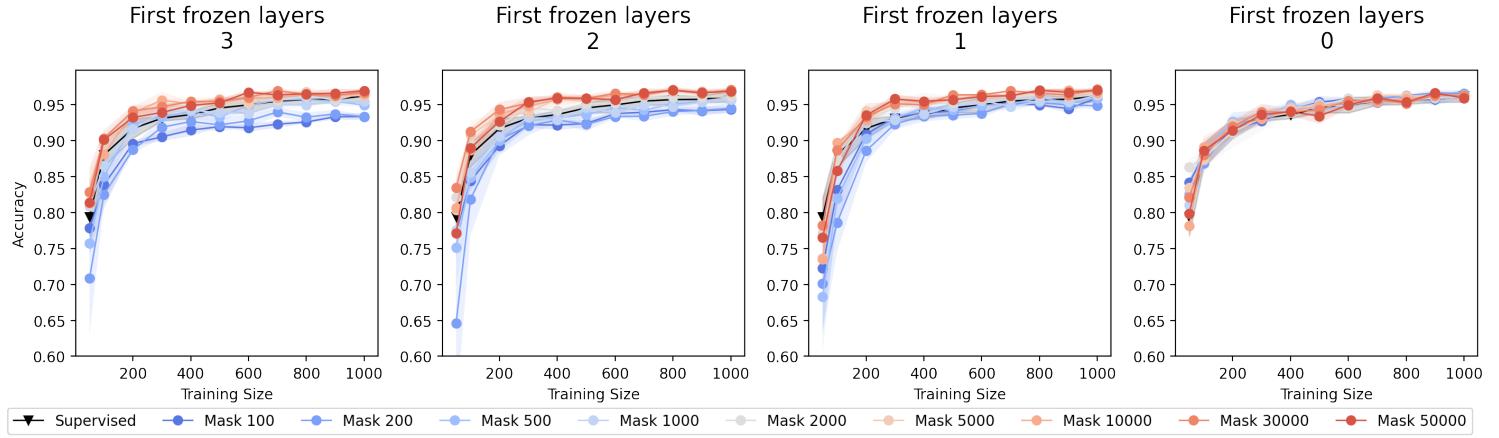


FIGURE B.3 – Accuracy en fonction de la taille des données d'entraînement, selon différents nombres de couches gelées pour les données TCGA.

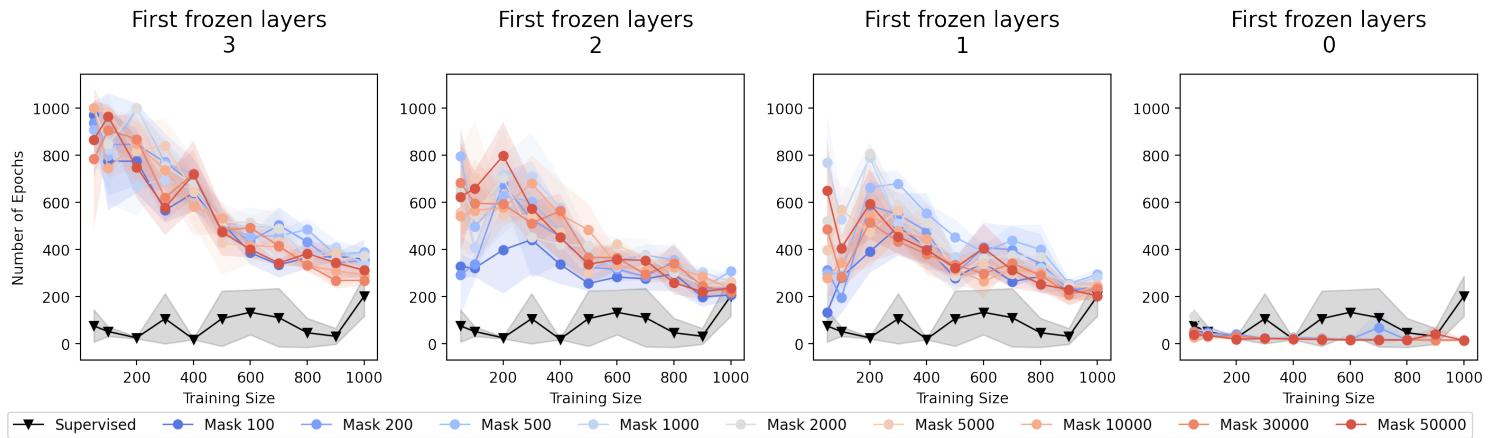


FIGURE B.4 – Nombre d'époques en fonction de la taille des données d'entraînement, selon différents nombres de couches gelées pour les données TCGA.

**Gel des couches** On peut observer les impacts qu'ont différents nombres de couches gelées, et non seulement tout, avec 3 couches, ou rien, avec 0 couche (Figures B.3, B.4 et B.5). On observe une évolution progressive entre ces différents gels, que ce soit pour l'accuracy, le nombre d'époques ou le temps par époque. Le gel de la première ou des deux premières couches représentent donc les résultats intermédiaires pouvant être obtenus entre le gel total ou nul. Le gel de la première couche montre le plus de changements. On peut supposer que la raison est sa taille, c'est en effet la plus grande. On remarque que le gel de la dernière couche cause aussi des baisses de performances. On peut donc supposer que cette couche contient des informations surtout utiles pour la tâche prétexte de prédiction de données masquées.

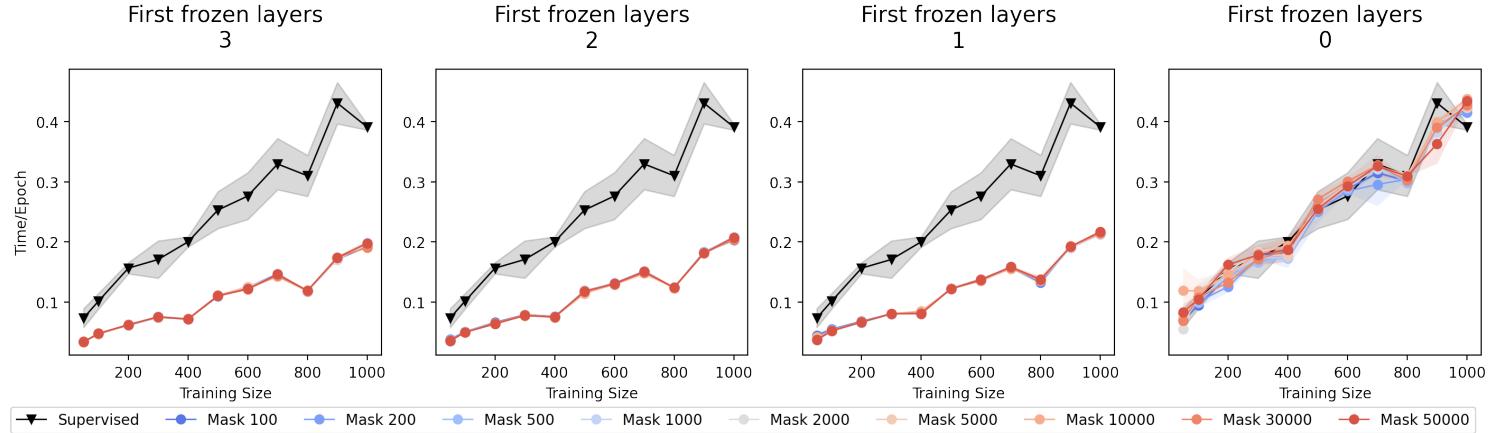


FIGURE B.5 – TCGA Temps par époque

### B.3 Fine-tuning avec données non étiquetées

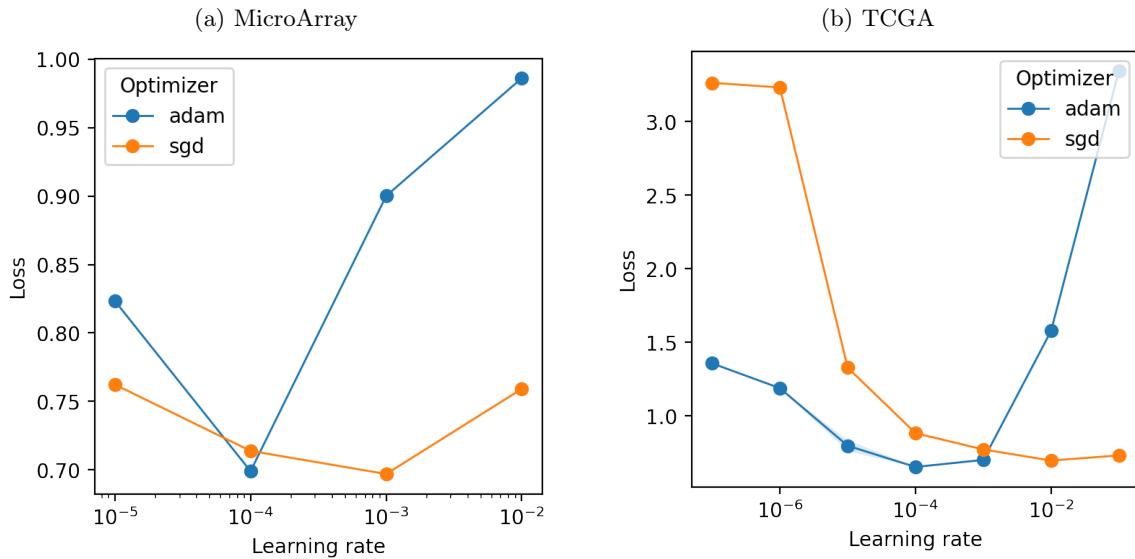


FIGURE B.6 – Fine-tuning avec données non-étiquetées du modèle self-supervisé de prédiction de données masquées : Coût en fonction du taux d'apprentissage et de l'optimiseur pour les données MicroArray (a) et TCGA (b).

**Optimiseur et taux d'apprentissage** Dû aux modifications apportées à ce modèle par rapport au précédent modèle de fine-tuning, afin d'intégrer des données non étiquetées, résultats de prédiction de modèle self-supervisé, nous devons tester si le meilleur optimiseur et taux d'apprentissage sont toujours les mêmes. Nous observons donc l'évolution du coût en fonction du taux d'apprentissage selon les deux optimiseurs Adam et SGD (Figure B.6). On obtient donc de meilleures performances pour les mêmes paramètres que le modèle supervisé, c'est-à-dire SGD avec momentum avec taux d'apprentissage à  $1e-3$  pour les données MicroArray et Adam avec taux d'apprentissage à  $1e-4$  pour les données TCGA.

**Lambda** Pour l'évolution du coût en fonction du lambda pour les données TCGA, présentée figure B.7, on observe la même évolution que pour les données MicroArray avec un encodeur gelé. Avec un encodeur non gelé, la courbe stagne. Les données de lambda critique, permettant peut-être d'améliorer les performances, sont alors 0.1, 1 ou 10, comme le jeu de données MicroArray.

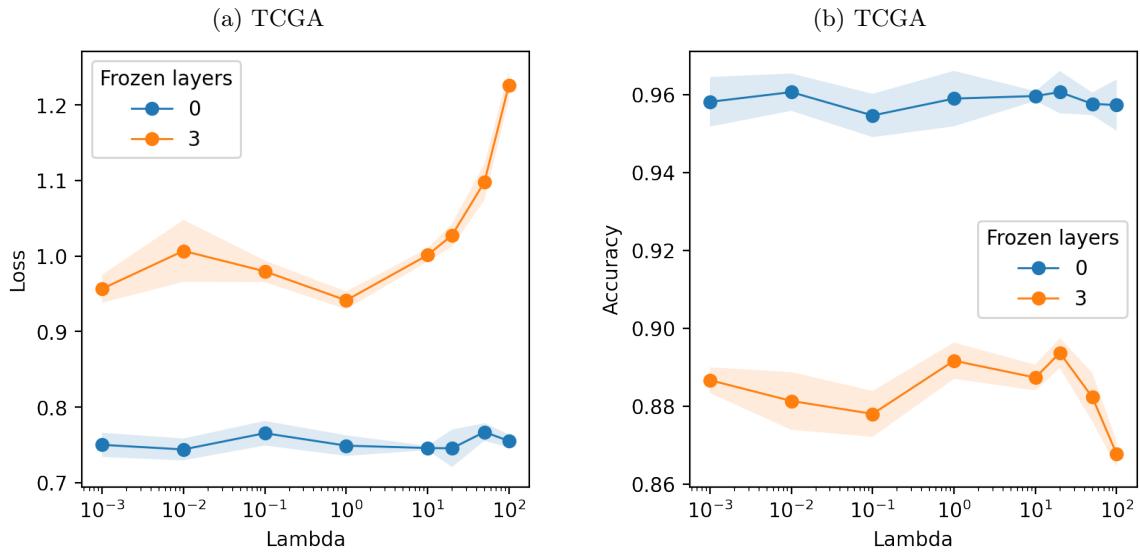


FIGURE B.7 – Fine-tuning avec données non-étiquetées du modèle self-supervised de prédiction de données masquées : Coût (a) et Accuracy (b) en fonction du paramètre lambda pour les données TCGA.

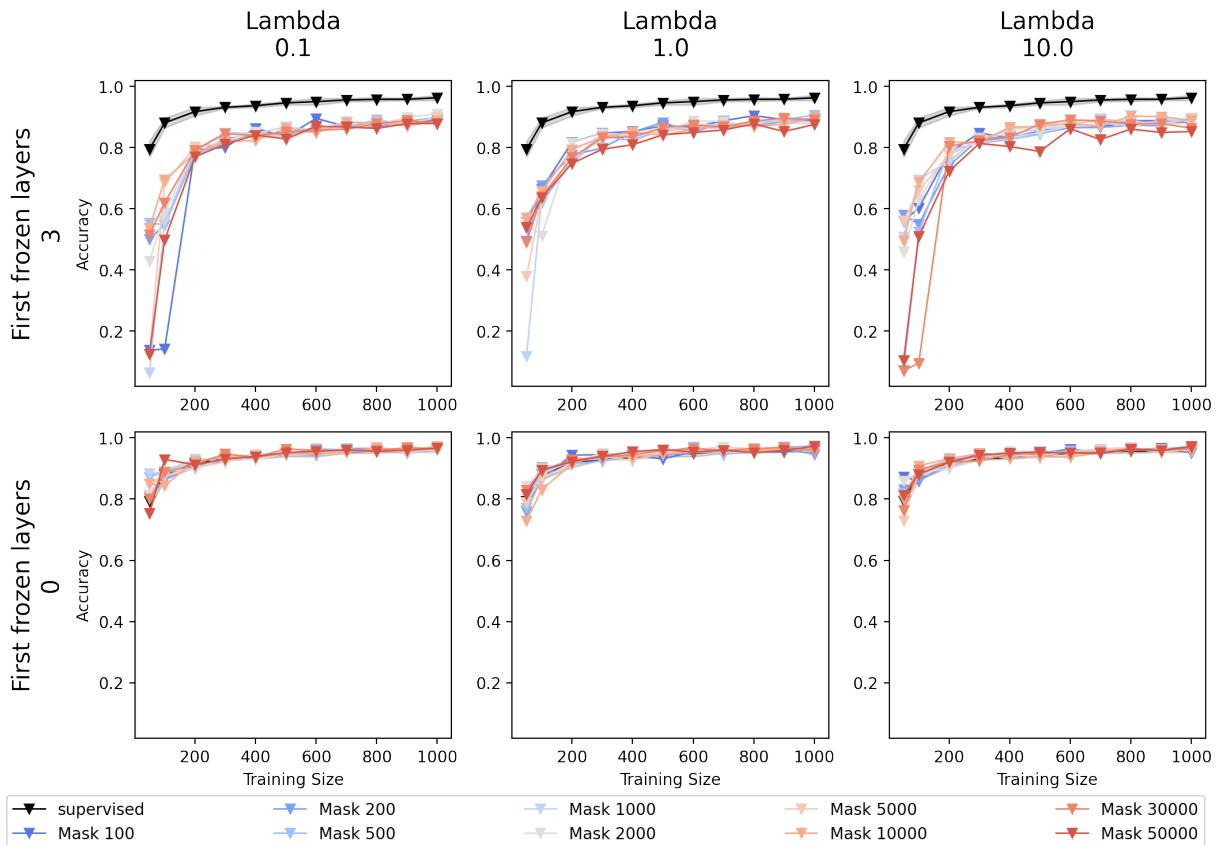


FIGURE B.8 – Fine-tuning avec données non-étiquetées du modèle self-supervised de prédiction de données masquées : Accuracy en fonction de la taille des données d’entraînement pour les données TCGA

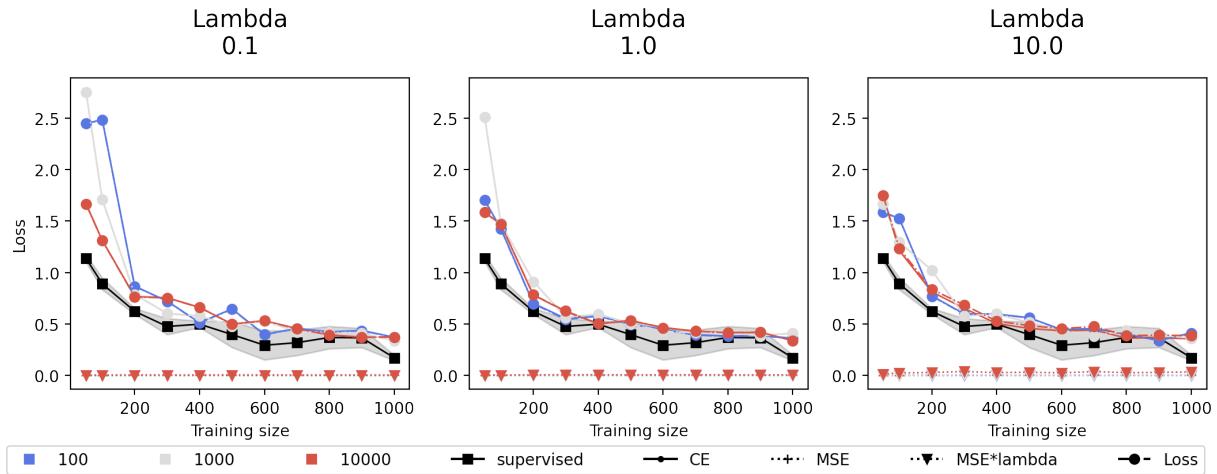


FIGURE B.9 – Fine-tuning avec données non-étiquetées du modèle self-supervisé de prédiction de données masquées : Détail des fonctions de coût en fonction de la taille des données d’entraînement pour les données TCGA

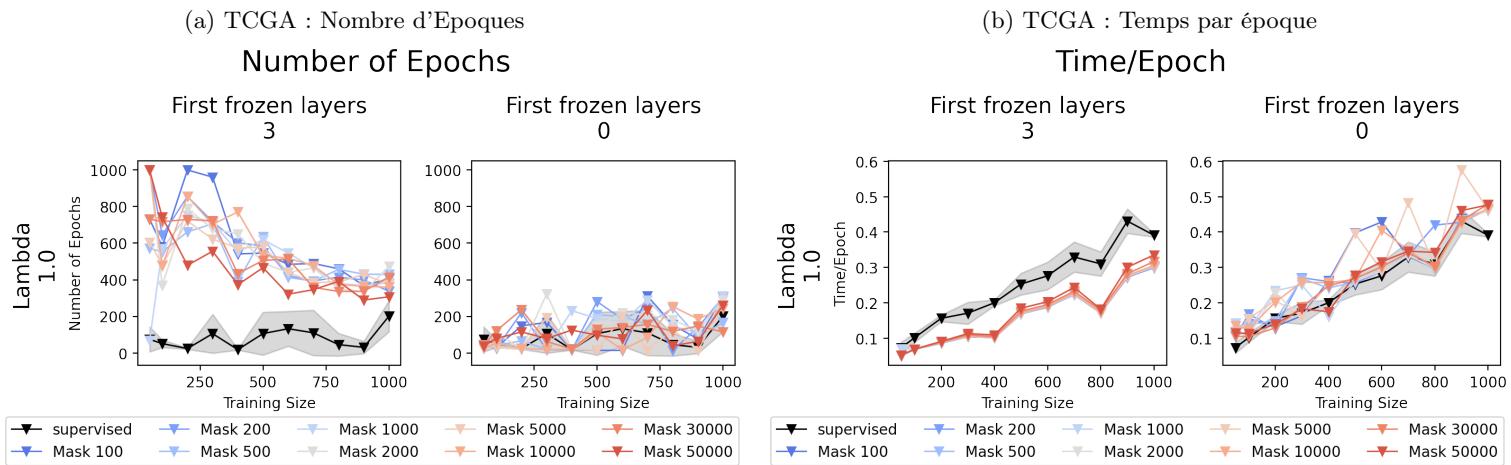


FIGURE B.10 – Fine-tuning avec données non-étiquetées du modèle self-supervisé de prédiction de données masquées : Nombre d’époques et temps moyen par époque en fonction de la taille des données d’entraînement pour les données TCGA

**TCGA** Les figures B.8, B.9 et B.10 présentent les différentes métriques de test du modèle de fine-tuning avec données non étiquetées, soient l’accuracy, les différents coûts, le nombre d’époques et le temps moyen par époque, pour les données TCGA. On y fait les mêmes observations qu’avec les données MicroArray.

# Modèle de Contrastive Learning

## C.1 Création du modèle

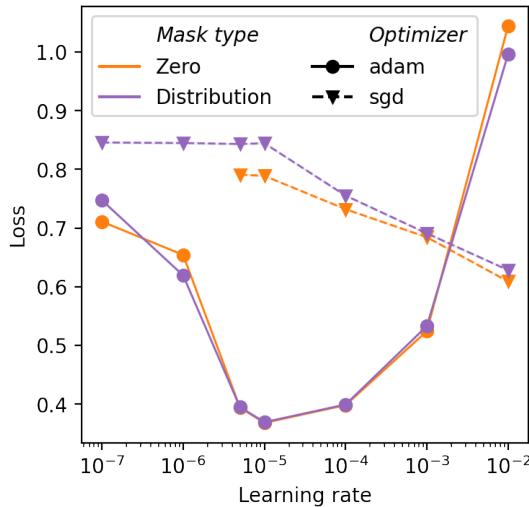


FIGURE C.1 – Coût en fonction du taux d'apprentissage et de l'optimiseur pour les données TCGA.

**Optimiseur et taux d'apprentissage** La figure C.1 présente l'évolution de la contrastive loss en fonction du taux d'apprentissage pour deux optimiseurs. On observe donc bien que les meilleures performances sont obtenues avec l'optimiseur Adam avec le taux d'apprentissage 1e-5.

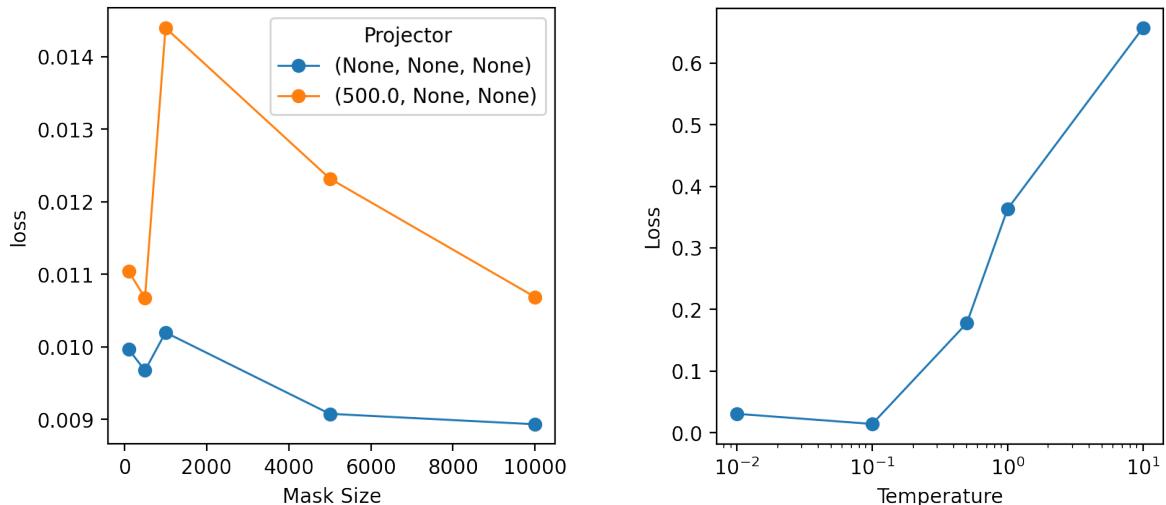


FIGURE C.2 – Coût en fonction de la taille de masque pour les données MicroArray, selon différentes architectures de projecteur.

FIGURE C.3 – Coût en fonction de la température pour les données MicroArray.

**Projecteur** La figure C.2 présente les courbes de la contrastive loss en fonction de la taille du masque pour deux projecteurs : celui ne contenant qu'une couche de 500 neurones et celui vide. On voit bien que les meilleurs résultats sont obtenus avec le projecteur vide d'où l'utilisation de celui-ci pour le modèle de contrastive learning.

**Température** La figure C.3 présente les courbes de la contrastive loss en fonction de la température pour les données MicroArray. On observe toujours la même température donnant les meilleures performances : 0.1.

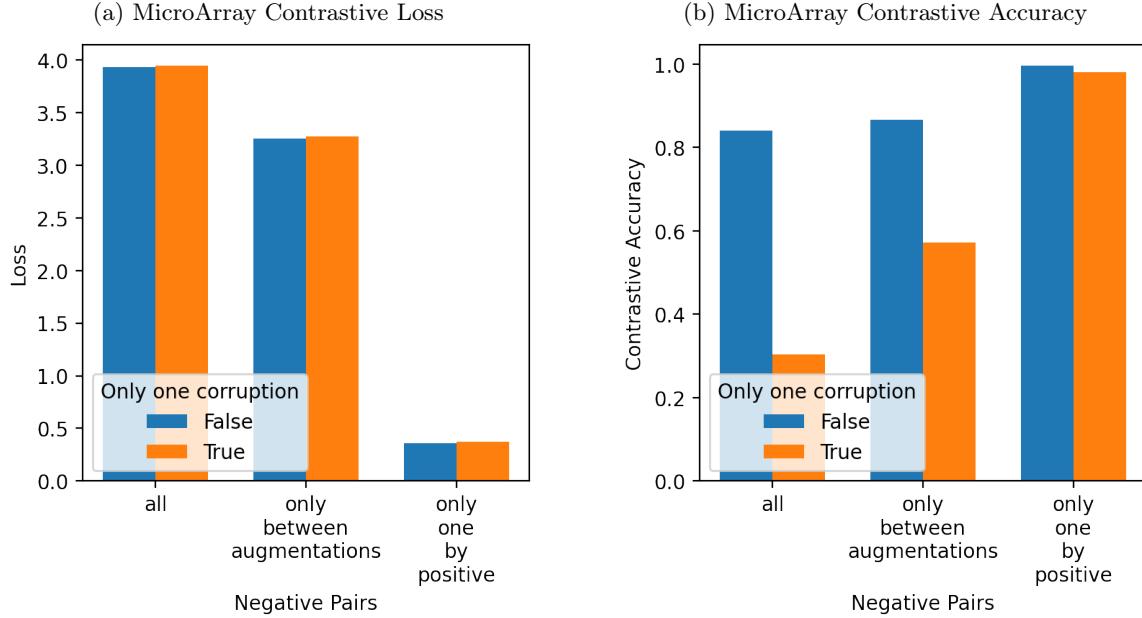


FIGURE C.4 – Modèle contrastive learning : Loss (a) et Accuracy (b) pour différents calculs de fonction de coût, et nombres de corruptions.

**Paires négatives** La différence de coût observée avec les données TCGA en fonction des différentes prises en compte des paires négatives, est retrouvée aussi avec les données MicroArray (Figure C.4). On observe de la même façon que les meilleures performances sont obtenues en ne prenant qu'une paire négative par paire positive et en utilisant deux corruptions.

## C.2 Visualisation des paires

**Visualisation des paires** En projetant les sorties du modèle de contrastive learning des paires de projection provenant des différentes augmentations dans un espace en deux dimensions, on peut observer si elles sont bien rapprochées et éloignées des autres, comme ce que l'on voudrait dans un bon modèle de contrastive learning. On observe cela pour les deux masques : zéro (Figure C.5) et distribution (Figure C.6). On observe donc bien que c'est plus facile de rapprocher les paires lorsque le masque est petit, mais qu'elles sont quand même très bien rapprochées jusqu'à ce que le masque soit égal à la dimension des données où alors la distinction entre les paires n'est plus possible et qu'elles sont juste toutes regroupées. Le contrastive learning est donc réussi, et les mauvaises performances observées au fine-tuning ne sont pas dues à une mauvaise performance du contrastive learning.

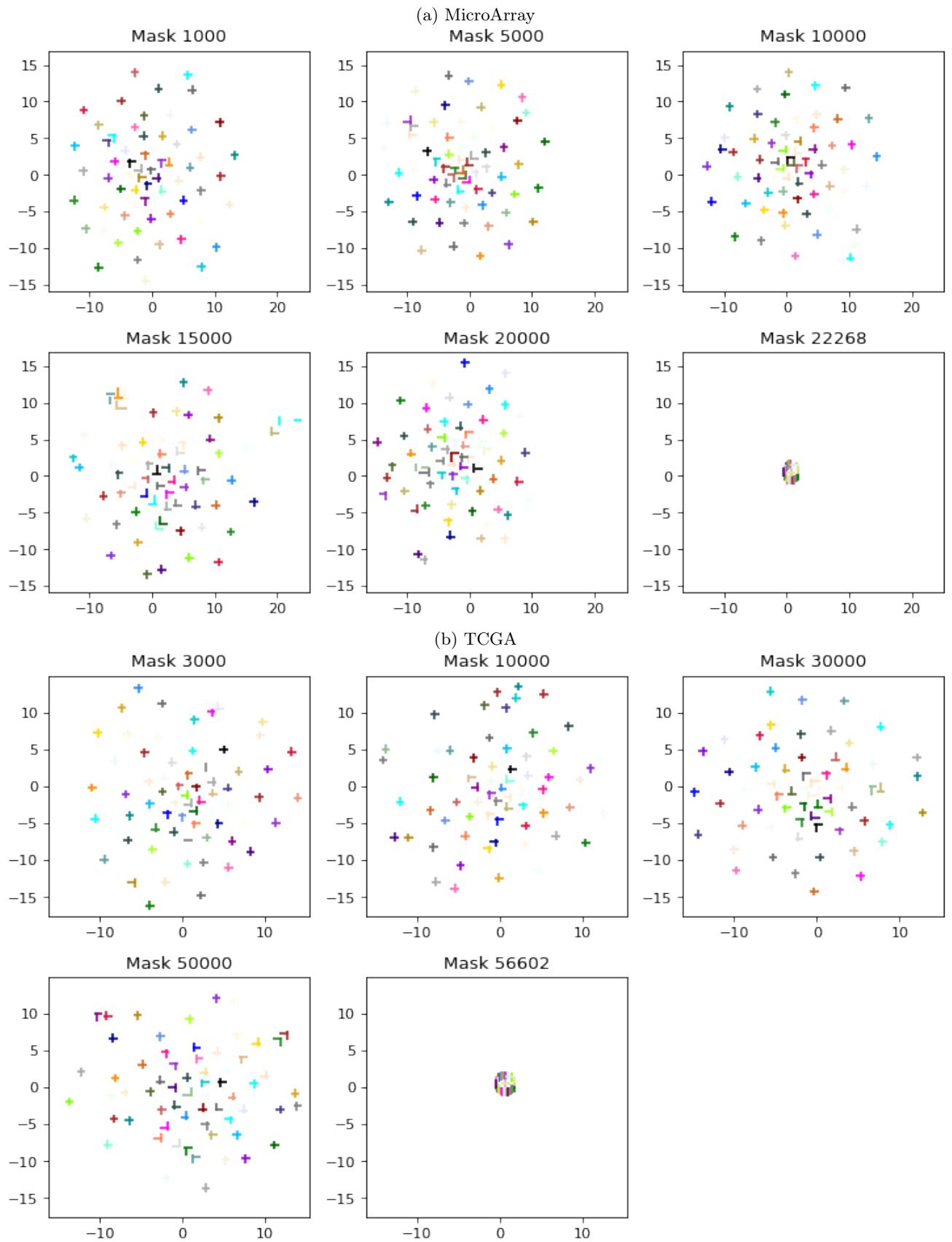


FIGURE C.5 – Visualisation de la position de 62 pairs dans un espace bidimensionnel pour les deux jeux de données et le masque Zero.

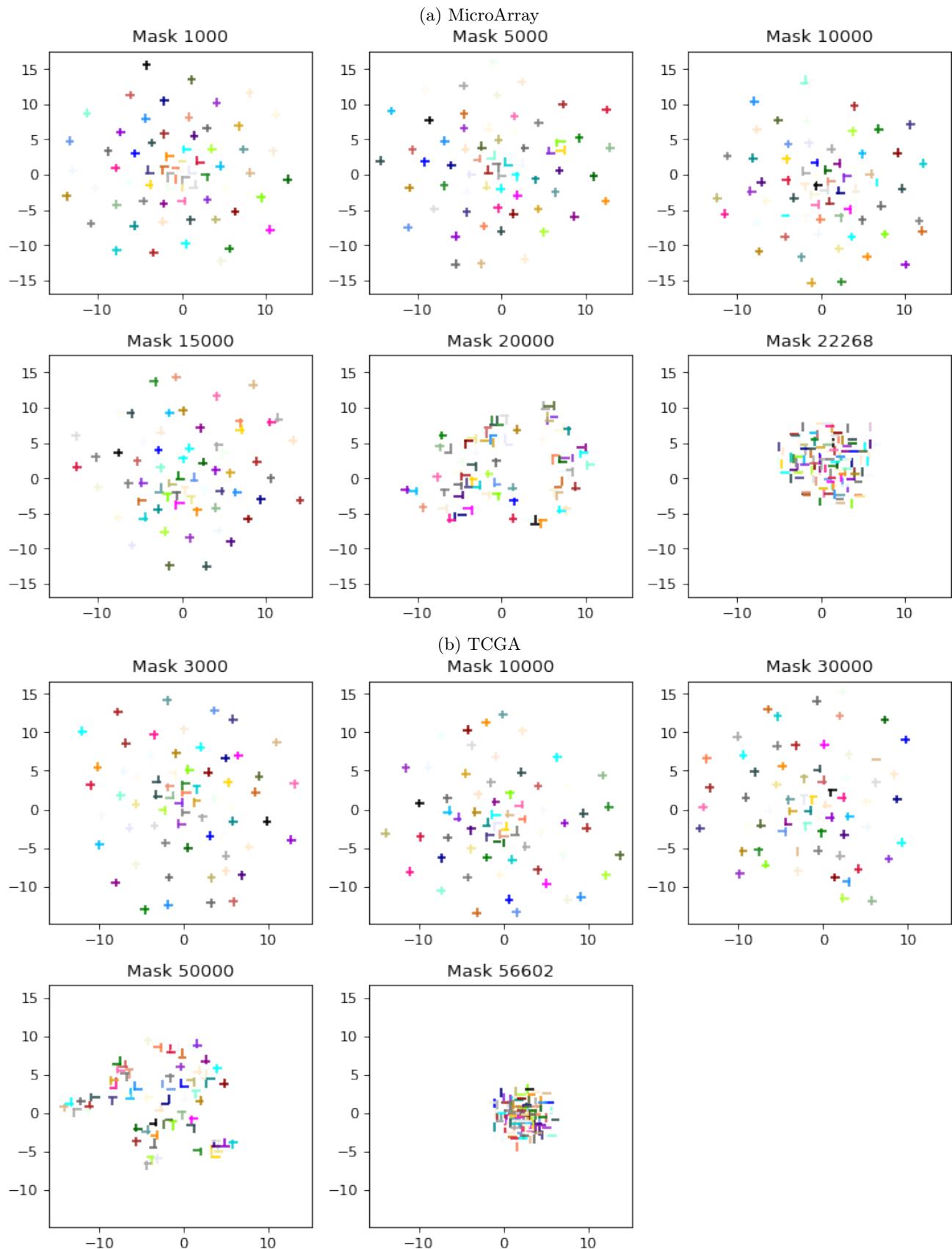


FIGURE C.6 – Visualisation de la position de 62 pairs dans un espace bidimensionnel pour les deux jeux de données et le masque Distribution.