Dépôt GitLab :
https://depot.dinf.usherbrooke.ca/dinf/cours/h21/ift780/troc3202/projet_session_rn

IFT 780 - Réseaux Neuronaux, projet de session

Apprentissage actif

Par Colin Troisemaine, Guillaume Bouchard et Éloïse Inacio

Résumé

Les réseaux de neurones sont des programmes polyvalents permettant d'apprendre une fonction. Ils sont notamment utilisés en classification pour attribuer à chaque donnée une catégorie. L'entraînement supervisé nécessite une base de données labellisée, mais celle-ci n'est pas toujours disponible. Nous pouvons alors avoir recours à l'apprentissage actif : le réseau décide quelle donnée lui serait le plus utile et demande quelle est sa classe. Trois critères de sélection ont été déterminés pour évaluer la pertinence de la donnée : *Random sampling, Uncertainty sampling* et *Diverse Mini Batch sampling.* Ils ont ensuite été implémentés sur deux types de réseaux neuronaux VGGNet16 et LeNet. Des expériences ont été menées sur les bases de données MNIST et CIFAR-10. Nous avons observé qu'il était difficile d'implémenter l'apprentissage actif avec des réseaux neuronaux convolutifs profonds. En effet, les résultats ne montrent pas d'améliorations notables par rapport à l'apprentissage passif.

Sommaire

1. Introduction

Un réseau de neurones artificiels est un programme qui s'inspire de la biologie pour apprendre une fonction.

Il est composé d'unités computationnelles dont le fonctionnement ressemble à celui d'un neurone biologique. L'unité reçoit des entrées par des connexions (synapses) qui ont chacune un certain poids. Ces poids sont les paramètres que le réseau apprend lors de l'entraînement. Le neurone artificiel somme ses entrées et leur applique une fonction dite d'activation. Le résultat est la sortie du neurone (Fig.1).

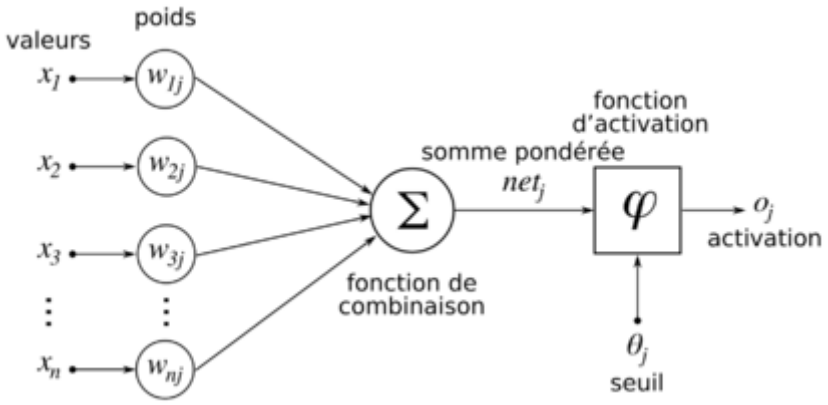


Figure 1 : Fonctionnement d'un neurone artificielle [9]

Les neurones sont disposés en couches (Fig. 2) qui peuvent avoir des rôles variés. Par exemple, une normalisation (*BatchNorm*) ou une convolution dont on apprend les filtres. La couche d'entrée (*input layer*) fait correspondre un élément de la donnée à un neurone. La couche de sortie (*output layer*) produit la prédiction du réseau comme la traduction d'un mot. Les couches intermédiaires sont dites cachées (*hidden layer*).

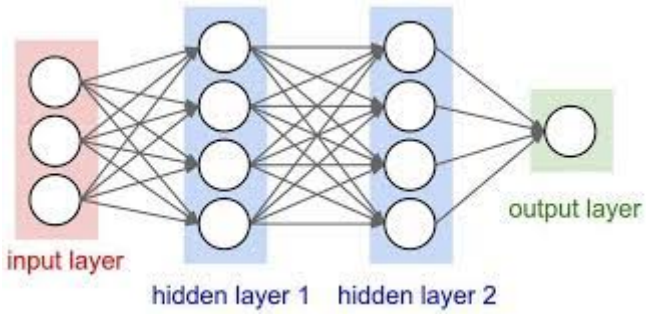


Figure 2 : Architecture générale d'un réseau de neurones artificiels [10]

Plus un réseau a de neurones et de couches de neurones, plus sa "capacité" augmente et plus la fonction qu'il apprend peut être complexe. La versatilité de ce type de programme lui permet de s'appliquer à de nombreux domaines comme des problèmes de classification, déterminer si une image représente un chat ou un chien, de régression, trouver le polynôme qui correspond le mieux aux données, etc.

Dans ce projet, nous nous intéresserons au problème de classification.

Pour déterminer la classe des données en entrée, il est parfois nécessaire de projeter ces données vers un espace où elles sont linéairement séparables, puis d'apprendre la fonction qui regroupe les données par classe. Pour ce faire, on peut utiliser des couches de neurones normales (produit scalaire) ou des couches convolutives (convolution).

Lors de l'apprentissage supervisé de la fonction, une donnée est propagée dans notre réseau puis la prédiction est comparée avec la classe réelle. Il est donc nécessaire que la donnée soit labellisée pour la phase d'apprentissage. L'erreur entre la prédiction et la classe réelle est calculée puis rétro propagée dans le réseau où les poids des synapses et des filtres de convolution sont actualisés : c'est la descente de gradient.

$$w_{n+1} = w_n - \eta \Delta_w L + régularisation$$

Où w est la matrice de poids à actualiser, η est le taux d'apprentissage et L est la fonction de perte.

Un taux d'apprentissage est ajouté pour contrôler le pas d'actualisation des paramètres et un terme de régularisation pour contrôler l'apprentissage.

De larges bases de données existent pour la classification mais toutes ne sont pas labellisées. Le procédé est long, coûteux et la classification peut être subjective ou encore nécessiter un expert. L'apprentissage actif est une des réponses à ce problème. Pour ne pas avoir à classer la base de donnée entière, on permet au réseau de choisir quelles données seraient les plus utiles à son apprentissage.

Ce rapport est organisé comme suit. D'abord, nous parlerons de la théorie de l'apprentissage actif et en particulier de comment faire choisir le réseau. Puis, nous implémenterons les méthodes sélectionnées dans deux types de réseaux de neurones : VGGNet16 et LeNet, que nous testerons sur deux bases de données : MNIST et CIFAR-10. Enfin, nous présenterons et commenterons nos résultats.

2. Notions principales

2.1. Apprentissage actif

Il est maintenant courant d'avoir accès à de larges bases de données sans pouvoir les labelliser. En effet, le procédé s'avère souvent coûteux car il demande du temps et un expert pour identifier manuellement les images.

Afin qu'un réseau de neurones puisse les classer, il faut d'abord l'entraîner. Nous avons vu que l'entraînement supervisé requiert que les classes d'un sous-ensemble de données soient connues pour l'entraînement. Lorsque ce n'est pas possible, l'apprentissage actif est alors envisageable.

L'apprentissage actif est une méthode d'apprentissage semi-supervisée : elle est supervisée car il est nécessaire qu'à la phase d'entraînement, le label de la donnée soit

connue pour qu'une fonction de perte soit calculée. Le réseau joue un rôle actif dans les données utilisées à l'entraînement puisqu'à chaque propagation, il détermine les données qui lui permettraient de s'améliorer le plus. Ces données sont ensuite classées manuellement par un *oracle* et ajoutées au sous-ensemble de données labellisées avant la prochaine propagation. Un oracle est un expert qui sera capable de correctement classer les données.

La première étape consiste à entraîner le réseau sur un petit sous-ensemble de données déjà classées appelé *seed*. Même si le modèle ne sera que peu performant pour ce qui est du taux de classification, il sera alors capable de déterminer quelles données seront les plus utiles pour son apprentissage. En d'autres termes, il lui faut une première idée d'où la séparation entre les classes se trouve. D'autres données seront ajoutées graduellement par apprentissage actif à la *seed*.

La seconde étape consiste à propager un lot de données non classées parmi lesquelles le réseau aura la liberté de choisir celles qui seront envoyées à l'oracle et ajoutées au sous-ensemble labellisé. Il faut alors utiliser un critère qui mesure quelles données permettent un apprentissage maximal du réseau. Nous explorerons plusieurs possibilités dans la section (section critère).

Le procédé est répété jusqu'à ce que la condition d'arrêt soit vérifiée (Fig.3), par exemple un nombre d'itération ou un taux de classification.

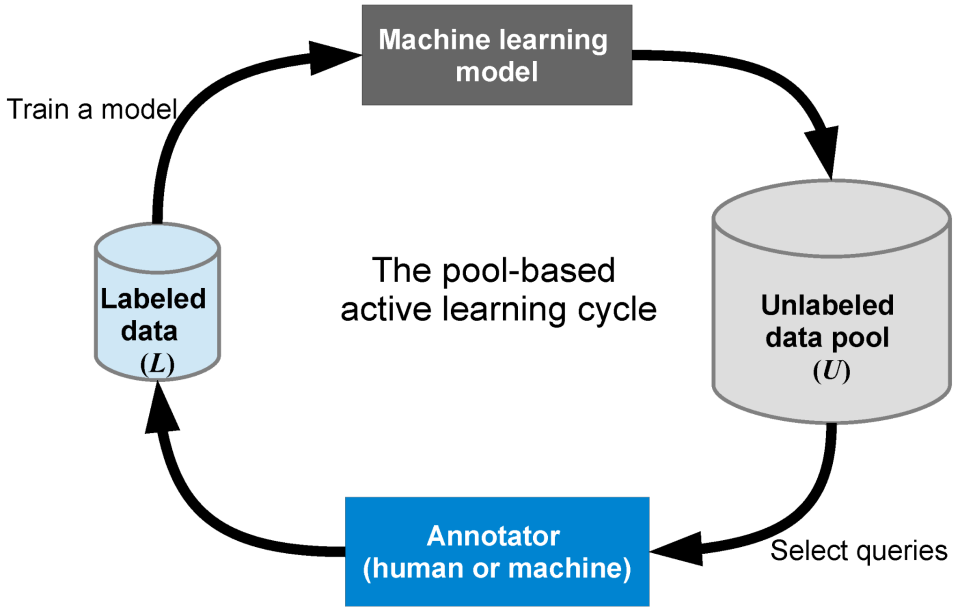


Figure 3 : Diagramme de fonctionnement de l'apprentissage actif [13]

Pour résumer, nous commençons par entraîner un modèle de classification à partir d'un petit ensemble de données labellisées, puis nous réalisons une prédiction sur un grand ensemble de données non labellisées, l'ensemble d'entraînement, parmi lesquelles le réseau choisit quelles données il souhaite ajouter à l'ensemble d'entraînement. Une fois ce choix fait, les données sont transmises à un oracle qui les classifie. Le réseau est ensuite

ré-entraîné sur la *seed* augmentée, puis sur l'ensemble d'entraînement réduit et fait un nouveau choix dans la base de données non labellisées.

Pour représenter ce que réalise l'apprentissage actif lors du choix des données, voici un exemple : nous disposons d'un ensemble formé d'un très grand nombre de points appartenant à deux classes différentes. Si les données sélectionnées pour augmenter la *seed* sont choisies aléatoirement, le résultat de la classification est peu probant (Fig. 4).

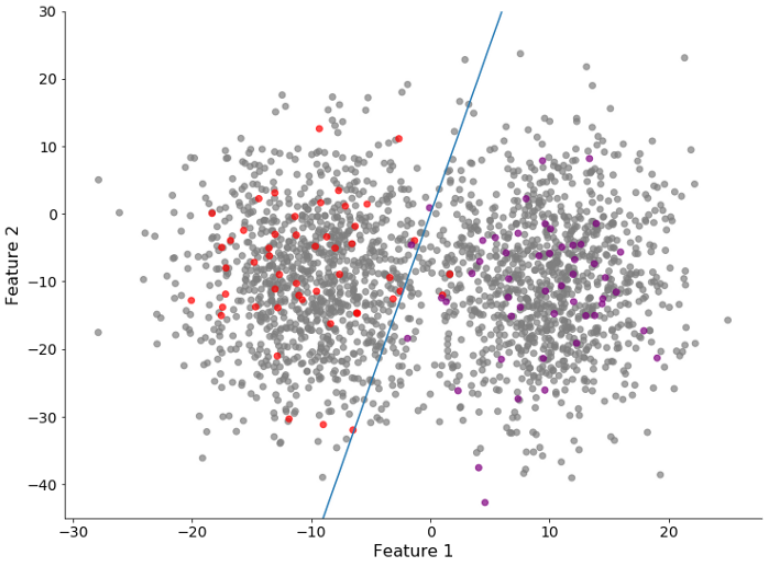


Figure 4 : Résultat de classification suite à un choix aléatoire des données composant la *seed* [7]

En revanche, si l'apprentissage actif est utilisé dans le choix des données à labelliser en se basant sur les critères de sélection choisis, une meilleure classification est permise (Fig. 5).

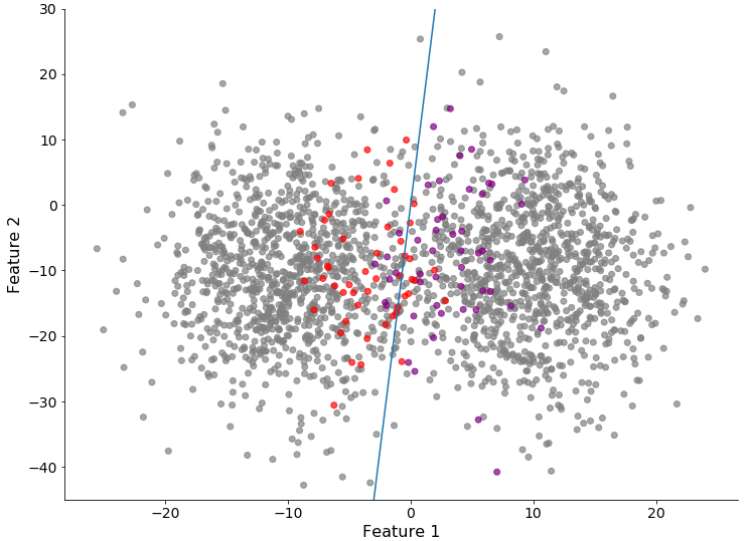


Figure 5 : Résultat de classification suite à l'implémentation de l'apprentissage actif [7]

Le principal inconvénient de l'apprentissage actif est que cette méthode sera plus lente à entraîner que l'apprentissage classique dit "passif" puisqu'on doit complètement entraîner notre modèle entre chaque choix de données. Cependant, le goulot d'étranglement de l'apprentissage machine est aujourd'hui l'accès à des bases de données de grande taille et de qualité. C'est donc ici que l'apprentissage actif intervient en orientant le choix des données pour sélectionner les plus utiles afin de réduire la taille des ensembles de données requis pour entraîner un modèle et obtenir les mêmes performances que par le biais de l'apprentissage classique.

2.2. Choix des bases de données

Les bases de données utilisées dans ce projet sont MNIST et CIFAR-10. Elles ont été sélectionnées pour la variété de données qu'elles représentent et leur facilité d'accès. En effet, elles font toutes les deux parties du package "*pytorch vision*".

De plus, elles représentent toutes les deux 10 classes, ce qui correspond à un niveau de difficulté adéquat pour une première implémentation de ce programme. Le classifieur des réseaux n'aura pas besoin d'être modifié lorsque la base de données est changée.

Enfin, elles représentent deux niveaux de difficulté : MNIST est composée d'images en niveau de gris alors que CIFAR-10 contient des images RGB.

2.2.1. MNIST

La base de données Mixed National Institute of Standard and Technology, abrégée MNIST, est une base de données composée de 70000 images de taille 28x28. Ces images proviennent d'une base de données antérieure : NIST.

Les données sont divisées en deux groupes : 60000 images d'entraînements et 10000 images de test. Les images représentent les 10 chiffres arabes manuscrits en niveau de gris.

Figure 6 : Exemple d'image de MNIST

Étant une base de données qui possède des images à un seul canal et relativement peu variées, MNIST est une base de données où un grand nombre de classifiers obtiennent de bons résultats facilement. De cette manière, nous pourrons observer l'effet de l'apprentissage actif sur un problème simple à résoudre.

2.2.2. CIFAR-10

La base de données Canadian Institute For Advanced Research 10, abrégée CIFAR-10, est une base de données composée de 60000 images 32x32x3.

Les données sont divisées en deux groupes : 5 lots de 10000 images d'entraînements et 1 lot 10000 images de test. Les lots d'entraînement sont formés tels que 5000 images de chaque classe en font partie. Dix classes sont représentées dans cette base de données : avion, voiture, bateau, camion oiseau, chat, cerf, chien, grenouille et cheval. Les classes sont exclusives, c'est-à-dire qu'aucune image ne représente deux classes à la fois.

Figure 7 : Exemple d'image de CIFAR-10

CIFAR-10 est une des bases de données les plus largement utilisées au sein du domaine de l'apprentissage machine, et plus précisément des méthodes de vision. Cette base de données est une référence en matière d'apprentissage qui nous permettra de visualiser l'influence de l'apprentissage actif sur un problème plus complexe que l'ensemble de données MNIST.

2.3. Choix des réseaux de neurones

2.3.1. VGGNet

Le VGGNet est un réseau de neurones convolutifs très profond qui contient entre 16 et 19 couches suivant les modèles. En 2014, l'équipe ayant développé ce réseau a obtenu la première place dans la catégorie localisation et la deuxième place dans la catégorie classification du *ImageNet Challenge*. Ce réseau est très documenté et réalise d'excellentes performances.

En effet, VGGNet possède une grande profondeur tout en gardant un temps de calcul raisonnable. Cela est dû à la taille de ses filtres convolutifs qui sont de taille 1x1 et 3x3. Il est possible de reproduire n'importe quel filtre avec un filtre 3x3 et un bon couple padding, stride. De plus, l'utilisation de filtres de petite taille permet de garder un nombre de paramètres raisonnable. Nous utilisons également des couches *batch-norm* et une couche *dropout* pour augmenter l'efficacité d'apprentissage de notre réseau.

Dans notre projet nous avons utilisé la structure à 16 couches, VGG16.

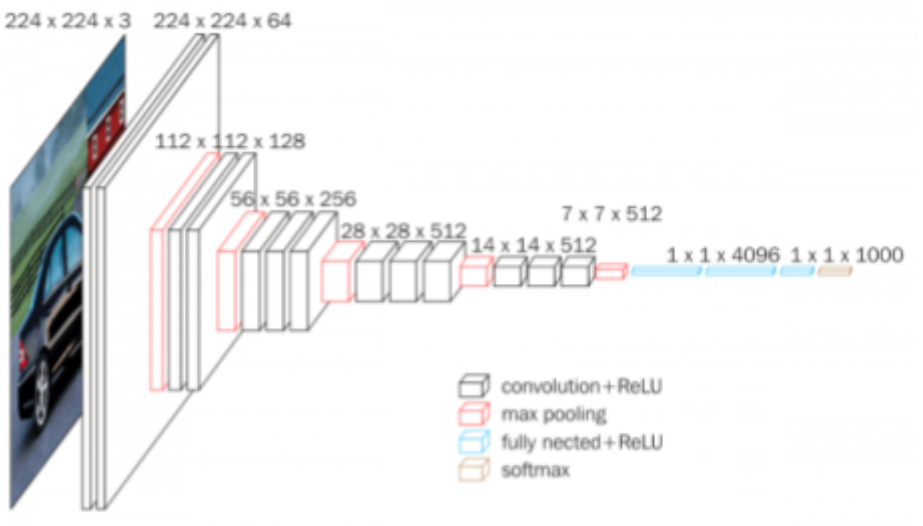


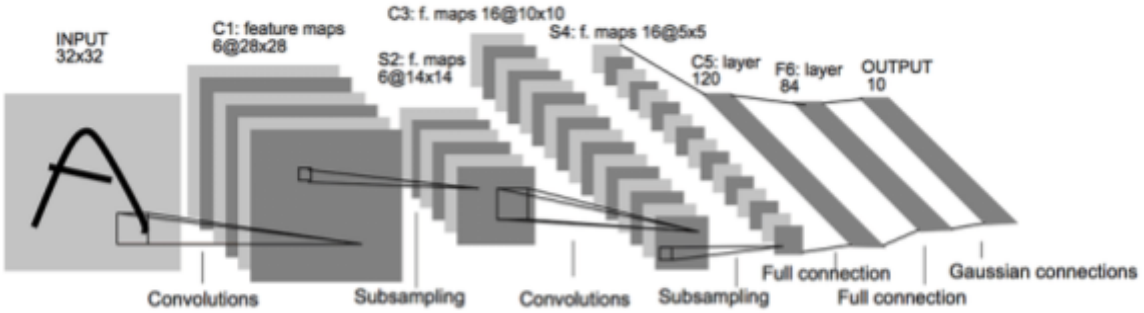
Figure 8 : Architecture du réseau VGG16 [14]

Nous avons choisi la version à 16 couches principalement car elle s'entraîne plus rapidement que sa version à 19 couches. De plus, la différence de résultats entre les deux structures, sur nos bases de données choisies, n'est pas assez importante par rapport à la différence de temps requis pour l'entraînement. Puisque l'on souhaite montrer l'impact de l'apprentissage actif sur un même réseau, l'efficacité de base du réseau n'est pas aussi pertinente que sa progression en fonction des différentes stratégies d'apprentissage actif choisies.

VGGNet présente donc un véritable intérêt pour étudier l'efficacité de l'apprentissage actif sur un modèle complexe.

2.3.2. LeNet

LeNet est un réseau de neurones à convolution créé par le français Yann LeCun en 1989. On trouve plusieurs représentations du LeNet sur internet mais nous avons choisi d'utiliser une structure qui s'approche de la structure originale, c'est-à-dire un réseau à 5 couches avec 2 convolutions (et 2 pooling) et 3 couches linéaires. Nous utilisons également la tangente hyperbolique comme fonction d'activation pour se rapprocher du réseau original qui n'utilisait pas Relu car celle-ci est apparue bien plus tard.



LeNet-5 structure [4]

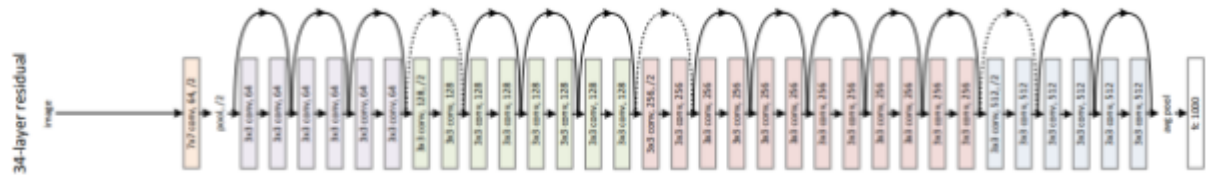
Figure 9 : Architecture de LeNet 5 [4]

Ce réseau est spécialisé dans l'analyse de caractères. On s'attend donc à ce qu'il présente de bonnes performances sur notre base de données MNIST. Néanmoins, il peinera certainement pour d'autres tâches d'analyse d'images.

Les raisons pour lesquelles LeNet a été sélectionné comme réseau de neurones pour ce projet sont sa simplicité d'implémentation et sa vitesse d'entraînement. On ajoute à cela son excellent fonctionnement sur la base de données MNIST et de très faibles performances sur CIFAR-10, il nous permet ainsi de couvrir de multiples cas. De cette manière, nous pourrons comparer l'influence de l'apprentissage actif sur un réseau complexe et un réseau plus simple.

2.3.3. ResNeXt

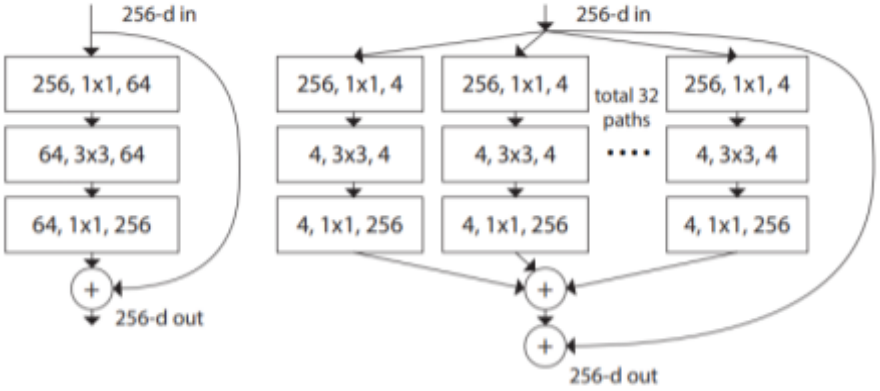
Le ResNeXt est un réseau de neurones convolutifs utilisé dans la classification d'images. Son architecture est tirée du réseau ResNet qui met en place le principe de bloc résiduel. Ce type de bloc permet d'augmenter l'efficacité de l'apprentissage des réseaux avec une très grande profondeur.



ResNet structure [5]

Figure 10 : Architecture de ResNet [15]

Contrairement au ResNet qui a une structure linéaire, le ResNeXt implémente des blocs résiduels qui sont parcourus en parallèle avant d'être fusionnés. Cette stratégie, tirée du réseau Inception, permet de créer une nouvelle dimension qui est la cardinalité. Elle correspond au nombre de divisions (chemins) dans un bloc. Cela permet de réduire le taux d'erreur du réseau. Dans l'exemple de la figure 11, la cardinalité est de 32.



Bloc résiduel de ResNet vs Bloc résiduel de ResNeXt [6]

Figure 11 : Composition des blocs résiduels de ResNet (à droite) et de ResNeXt (à gauche) [5]

Ce réseau est donc un réseau de neurones très complexe et très efficace. Il aurait donc permis de compléter nos deux autres modèles pour évaluer l'apprentissage actif sur des modèles récents. Néanmoins, au cours de l'implémentation de ce modèle, nous nous sommes rendu compte qu'il n'était pas convenable de l'utiliser avec le temps d'entraînement qu'il prenait. Nous avons donc arrêté de développer ce réseau pour nous concentrer sur VggNet et LeNet.

2.4. Choix des critères de sélection de l'apprentissage actif

Les critères de sélection de l'apprentissage actif sont choisis en deux étapes. D'abord, le scénario de demande est sélectionné, puis une stratégie liée à ce scénario. Il existe trois grands scénarios dans l'apprentissage actif.

- Le Membership Query Synthesis

Ce scénario est utilisé pour des modèles génératifs, ce qui n'est pas notre cas ici.

- Le Stream Based Selective sampling

Ce scénario est utilisé pour des modèles prédictifs avec des données qui arrivent au fil du temps. Les données sont évaluées une à une et sont sélectionnées par rapport à un critère de sélection. Ce scénario est très utilisé quand les données sont générées au cours de l'apprentissage du modèle.

- Le Pool Based sampling

Ce scénario est lui aussi utilisé pour des modèles prédictifs. Cependant, contrairement au *Stream Based Selective sampling*, le modèle a déjà accès à une grande base de données non labellisées. Nous allons donc évaluer des *batchs* de n données les unes envers les autres et sélectionner les k données les plus avantageuses selon notre stratégie. Ce scénario est très utilisé lorsqu'on entraîne un modèle sur une grande base de

données déjà générée et est donc la méthode préférée lors des études sur l'apprentissage actif. Pour cette raison, ce scénario sera implémenté dans le cadre de notre projet.

Quant aux stratégies, il existe un très large panel de méthodes pour réaliser le choix des données à labelliser par nos experts. Toutes permettent de mesurer l'information que l'on peut associer à nos données. Ainsi, les données non labellisées sont classées par ordre de priorité et les plus intéressantes pour l'apprentissage de notre modèle sont sélectionnées. Trois méthodes ont été considérées : *Random sampling*, *Uncertainty sampling* et *Diverse Mini Batch sampling*.

2.4.1. Random sampling

Même s'il ne s'agit pas d'un critère de sélection de l'apprentissage actif à proprement parler, le *Random sampling* fait ici partie des méthodes que nous avons été amenés à implémenter. Cette méthode, même lorsque nous nous en servons de la même manière que les autres stratégies, se comporte comme l'apprentissage passif, aussi dit classique. En effet, cette stratégie sélectionne au hasard les données à ajouter à notre ensemble d'entraînement.

Une fois nos courbes d'apprentissage actif avec la quantité de données de l'ensemble d'entraînement en abscisse dessinées, cette stratégie sera pour nous le point de repère pour déterminer le gain permis par les autres critères.

2.4.2. Uncertainty sampling

L'*Uncertainty sampling* est l'une des familles de stratégies de base de l'apprentissage actif et sans doute la plus utilisée.

Le principe est simple : le modèle fait une prédiction sur des données non labellisées et la confiance que le modèle a envers sa propre prédiction est calculée. Pour ce faire, nous considérons la sortie de la fonction d'activation softmax pour obtenir des probabilités pour chacune des classes possibles. Nous allons ensuite sélectionner les données qui ont obtenu le plus faible taux de confiance, le but étant de sélectionner des données qui sont les plus proches des frontières de décisions et qui sont donc difficiles à classer pour le modèle.

Pour évaluer ce taux de confiance, il existe différents critères que nous avons implémentés.

2.4.2.1. Least Confidence sampling

Ce critère prend directement en compte les scores des réseaux comme suit. Considérons ce tableau de prédictions de notre modèle sur les données x1 et x2 :

Données	Classe 1	Classe 2	Classe 3
x1	0.9	0.09	0.01
x2	0.3	0.5	0.2

Nous considérons les scores les plus élevés pour chaque donnée, c'est-à-dire celui de la classe prédite par le réseau. Ici, nous avons 0.9 pour x1 et 0.5 pour x2. Comme 0.5 < 0.9, nous pouvons en déduire que le réseau est moins sûr de sa prédiction pour x2 qu'il ne l'est pour x1.

Le réseau sélectionne donc la donnée x2 à envoyer à l'oracle pour être classé.

2.4.2.2. Margin sampling

Cette stratégie répond à un problème de la stratégie précédente. En effet, la stratégie *Least Confidence* ne prend en compte que le taux de confiance sur la classe de prédiction et non sur les autres classes. *Margin sampling* compense partiellement cette faiblesse en prenant comme valeur d'évaluation le taux de confiance de la classe prédit moins la valeur de la deuxième classe avec le taux de confiance le plus haut.

Reprenons notre exemple :

Données	Classe 1	Classe 2	Classe 3
x1	0.9	0.09	0.01
x2	0.3	0.5	0.2

Ici le score de x1 devient : $0.9 - 0.09 = 0.81$. Tandis que le score de x2 est de : $0.5 - 0.3 = 0.2$. C'est la variable avec le score le plus bas qui est sélectionné, ici x2.

2.4.2.3. Entropy sampling

Cette stratégie est une solution alternative au problème de la stratégie *Least Confidence sampling*. Néanmoins, contrairement à *Margin sampling* qui ne prend en compte que les deux prédictions des deux classes ayant les taux de confiance les plus hauts, la stratégie *Entropy sampling* prend en compte les prédictions sur toutes les classes.

Pour cela, elle utilise la mesure d'entropie qui est une mesure mathématique déterminant le taux d'information contenue dans une source d'information.

Nous utiliserons l'entropie de Shannon :

$$-\sum_{i=1}^{n} p(x_i) log_b p(x_i)$$

Cette mesure permet de savoir, grâce aux prédictions de toutes notre *batch* de données, quelles données contiennent le plus d'informations et doivent donc être envoyées à l'oracle.

Reprenons notre exemple :

Données	Classe 1	Classe 2	Classe 3
x1	0.9	0.09	0.01
x2	0.3	0.5	0.2

L'entropie de Shannon pour les données est 0.459 pour x1 et 1.485 pour x2. Ainsi, x2 possède plus d'information que x1 et c'est donc elle qui sera envoyée à l'oracle.

2.4.3. Diverse Mini Batch sampling

Diverse Mini Batch est une stratégie qui diverge des trois méthodes précédemment présentées puisqu'en plus de l'informativité mesurée, elle prend en compte le besoin de diversité dans son choix. En effet, les stratégies basées uniquement sur l'incertitude peuvent ne pas être les plus adaptées lorsqu'on utilise un scénario d'apprentissage actif où l'on choisit nos données à labelliser par *batch*. Si après un entraînement, le modèle est très mauvais pour prédire une certaine classe et relativement mauvais pour en prédire d'autres, les méthodes basées sur l'incertitude vont uniquement choisir des instances pour lesquelles le modèle est le plus mauvais, et l'ensemble de données à labelliser sera uniquement constitué de données d'une même classe, alors que le modèle aurait pu se contenter d'une plus petite partie pour apprendre à correctement les classer.

Pour pallier ce problème, la stratégie *diverse mini batch* utilise une méthode de partitionnement des données pour introduire une diversité des classes dans son choix de données. Pour cette raison, elle est plus adaptée lorsqu'on utilise l'apprentissage actif par *batch* plutôt que par *stream*. Son fonctionnement général est le suivant :

- $\beta * k$ individus sont sélectionnés à l'aide d'une stratégie basée sur l'incertitude. La plus courante est *margin sampling*, mais la meilleure dépendra du contexte.
- Les k individus les plus proches des centres des paquets générés par notre méthode de partitionnement des données sont sélectionnés pour être labellisés.

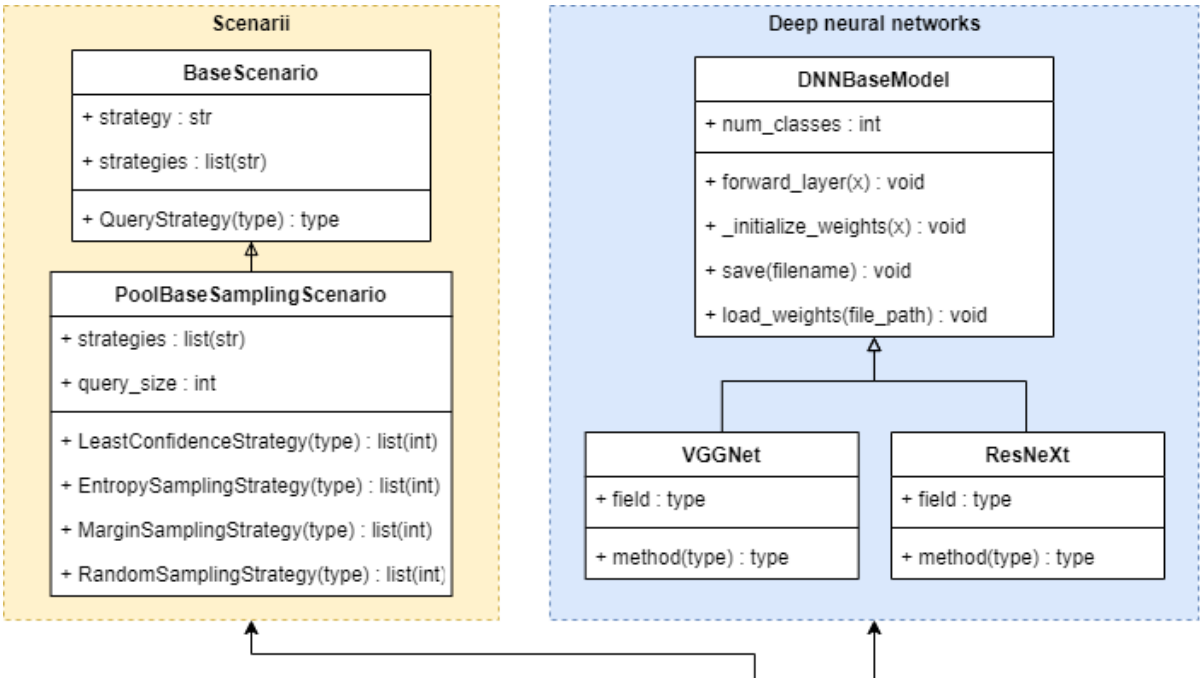
Le papier original de la méthode [15] expérimente avec des valeurs de β=10 et β=50, k est la taille de la *batch* de données à labelliser. Nous utiliserons ici une méthode de *K-Means* avec β=50 car nous disposons de grands ensembles de données.

3. Présentation du code

3.1. Structure du projet

Dans le but de créer un projet dont le code sera réutilisable dans le futur, nous avons porté une attention particulière à la structure du code, et notamment à sa modularité. Ainsi, même si nous n'avons pour l'instant implémenté qu'un seul *scénario* (*Pool Based sampling Scenario*), il hérite d'une classe "générale" intitulée *BaseScenario* de manière à ce que l'implémentation de scénarios supplémentaires soit la plus rapide possible. Dans la même optique, tous les réseaux que nous avons implémentés héritent de *DNNBaseModel*. Cela nous permet de définir les méthodes communes aux différents modèles (comme *train*, *save_weights*, *load_weights*, etc…) et de nous servir du modèle au sein du code de manière indépendante à son implémentation spécifique.

La classe *TrainTestManager* est certainement la classe la plus importante de notre projet. C'est elle qui contient toute la logique de l'apprentissage actif qu'elle applique par le biais de la méthode *active_learning_training()*. C'est une instance de cette classe qui est créée par le script *train.py* qui est notre point d'entrée (nous décrirons les paramètres d'appel par la suite). Afin de fonctionner correctement, elle a donc besoin de stocker la plupart des paramètres définis par l'utilisateur au moment de l'appel du script. Enfin, elle utilise la classe *DataManager* qui nous permet de manipuler les données dont nous avons besoin. On y retrouve notamment la méthode *add_indexes_to_seed_sampler* que l'on utilise pour ajouter à la *seed* les index des données sélectionnées par notre stratégie de *query*.



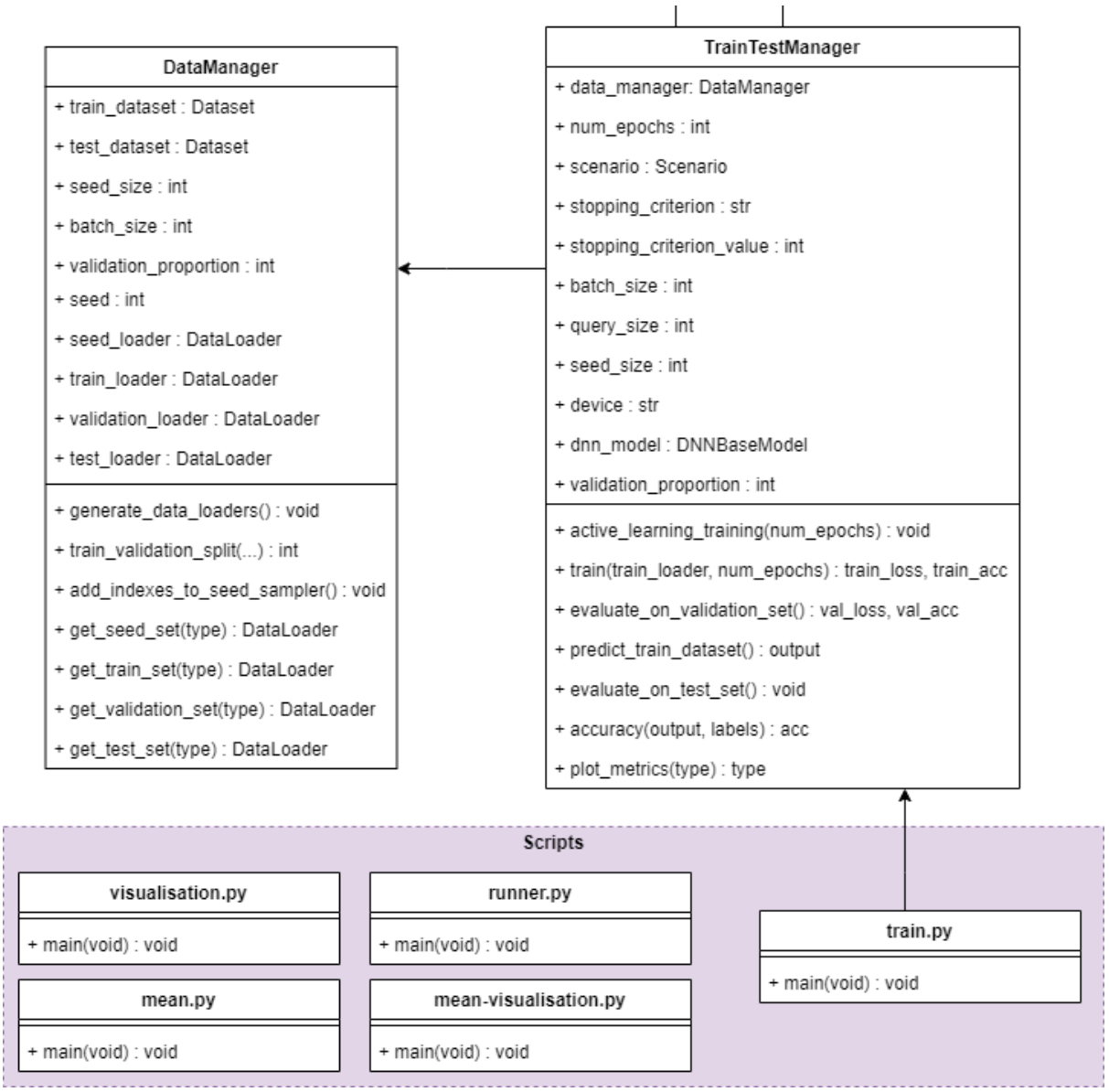


Figure 12 : Diagramme de classes de la structure du projet

Notre implémentation de l'apprentissage actif par mélanger l'ensemble d'entraînement et le séparer en 3 nouveaux ensembles : la *seed*, qui constitue l'ensemble d'entraînement de départ et qui s'agrandit au fur et à mesure des nouvelles *query*, les données d'entraînement, qui sont l'ensemble pour lequel le réseau fait des prédictions avant de choisir les données à ajouter à la *seed*, et enfin les données de validation qui servent à évaluer notre modèle après chaque nouvel entraînement sur la *seed*.
Les étapes de l'apprentissage actif implémenté sont :

1) Initialisation du modèle aléatoire et sauvegarde des poids.
2) Entraînement sur la *seed* et prédiction sur l'ensemble d'entraînement.
3) Envoi des scores au *PoolBaseSamplingScenario* qui va utiliser le critère de sélection que l'utilisateur a précisé pour créer la liste d'index des données sélectionnées de longueur *query_size*.
4) Agrandissement de la seed par le *DataManager* à qui on aura fourni la liste d'index.
5) Évaluation de la performance du modèle sur l'ensemble de validation.
6) Réinitialisation du modèle à partir des poids sauvegardés

7) Répétition des étapes 2 à 6 tant que l'on ne dépasse pas le *stopping_criterion_value*.
8) Sauvegarde des différentes métriques récoltées dans un fichier *json*.

3.2. Les différents scripts

Nous avons été amenés à créer 5 scripts différents pour les besoins du projet. Le premier, et certainement le plus important, est *train.py*. La commande relative à son usage est : `python train.py model dataset [parameters]`
Le détail des paramètres qu'il est possible de renseigner est donné ci-après :

- --model : le modèle utilisé (valeurs : vggnet / lenet).
- --dataset : la base de données utilisée (valeurs : CIFAR-10 / mnist).
- --scenario : le scénario d'apprentissage actif (nous n'avons implémenté que *poolbasedsampling* qui est donc l'unique choix).
- --strategy : le critère de sélection (valeurs : *least_confidence* / *entropy_sampling* / *margin_sampling* / *random_sampling*).
- --stopping_criterion : le critère d'arrêt de l'algorithme d'apprentissage (nous n'avons implémenté que *query_number* qui est l'unique choix)
- --stopping_criterion_value : la valeur d'arrêt relative au *stopping_criterion* utilisé (donc pour le critère *query_number* il s'agit de la quantité de données que le réseau peut ajouter au seed dataset).
- --seed_size : la taille de l'ensemble d'entraînement initial.
- --batch_szie : la taille des *batchs* utilisés lors de l'entraînement du modèle.
- --query_size : le nombre de données ajoutées à la *seed* à chaque itération de l'apprentissage actif.
- --optimizer : l'*optimizer* utilisé pour entraîner le modèle (valeurs : SGD ou Adam).
- --num_epochs : la quantité d'epochs pour l'entraînement à chaque itération de l'apprentissage actif.
- --validation : le pourcentage de l'ensemble d'entraînement utilisé pour la validation.
- --lr : le taux d'apprentissage.

Nous disposons ensuite du script *runner.py* qui nous a permis de lancer un grand nombre de fois le script *train.py* de manière séquentielle en précisant les paramètres à utiliser pour chacun des lancements de *train.py*.

Enfin, nous avons créé trois scripts pour la visualisation. Le premier est *visualisation.py* : il permet d'afficher 4 sous graphes différents (précision lors des phases d'entraînement et de validation, coût lors des phases d'entraînement et de validation) en superposant n'importe quelle quantité de courbes à partir des fichiers de résultats que l'on génère.

Le second script est *mean.py*. Il est nécessaire à l'utilisation de dernier script *mean-visualisation.py* puisqu'il calcule les métriques moyennes et leurs déviations standards de tous les fichiers de résultats d'un même dossier. Une fois cela fait, on peut afficher la courbe de la précision de validation moyenne en coloriant la déviation standard de chaque point par le biais du script *mean-visualisation.py*.

4. Analyse des résultats

4.1. Génération des résultats

La procédure mise en place est la suivante :

- Nous commençons avec une *seed* de taille spécifiée.
    - Le modèle est entraîné sur la *seed*.
    - La prédiction sur l'ensemble de données d'entraînement qui ne font pas partie de la *seed* est fournie à la *query strategy* qui indique quelles données labelliser et ajouter à la *seed*.
    - Les métriques suivantes sont récupérées : précision lors des phases d'entraînement et de validation, coût lors des phases d'entraînement et de validation.
- Les étapes précédentes sont réitérées jusqu'à ce que le crédit de requêtes ait été épuisé.
- Enfin, pour obtenir des résultats dans lesquels nous pouvons avoir confiance, l'expérience est répétée 20 fois et on élimine pour chacune des tailles de la *seed* (donc l'abscisse de nos métriques) le pire et le meilleur point.

Pour générer l'une des courbes présentées dans la section suivante, nous suivons ce procédé :

- Nous modifions le fichier *src/scripts/runner.py*, dont le rôle est de lancer à la suite le nombre de commandes dont nous avons besoin, afin qu'il exécute 20 fois notre algorithme d'apprentissage actif pour chacune des stratégies (ou critères de sélection).
- À la fin de chacune des exécutions d'un cycle complet d'apprentissage actif, un fichier de résultats est généré portant dans son nom le nom de la stratégie utilisée, l'ensemble de données, le modèle, la date et l'heure de création.
- Une fois nos 20 fichiers générés par stratégie, nous utilisons le script *mean.py* pour générer un fichier final de résultats qui comportera la moyenne de chacune des métriques de nos 20 fichiers et les déviations standard de chacun des points de ces métriques.
- Le script *mean-visualisation.py* crée ensuite la superposition des courbes moyennes pour toutes les stratégies.

Pour créer toutes les courbes dont nous avons besoin, nous avons exécuté 20 x 2 ensembles x 2 modèles x 5 stratégies = 400 fois un cycle d'apprentissage actif où le réseau s'entraîne sur 20 epochs sur chacun des 10 ensembles d'entraînement de tailles incrémentales, soit 80 000 epochs au minimum.

4.2. Présentation des résultats

Pour que nos résultats soient comparables les uns aux autres, nous avons fait attention à utiliser les mêmes paramètres pour chaque expérience :

- Scénario : *Pool based sampling.*
- 20 epochs par entraînement sur chaque nouvel ensemble de données (donc pour chaque point) afin de réduire la variance des résultats.
- 10% de données réservées à la validation.
- Taille de batch fixée à 10.

4.2.1. LeNet

Les courbes, présentées dans cette section, comparent la précision du modèle LeNet après avoir été entraîné pour 20 epochs en fonction des 4 différentes stratégies de sélection des données que nous avons implémentées et de *Random sampling* représentant l'apprentissage passif.

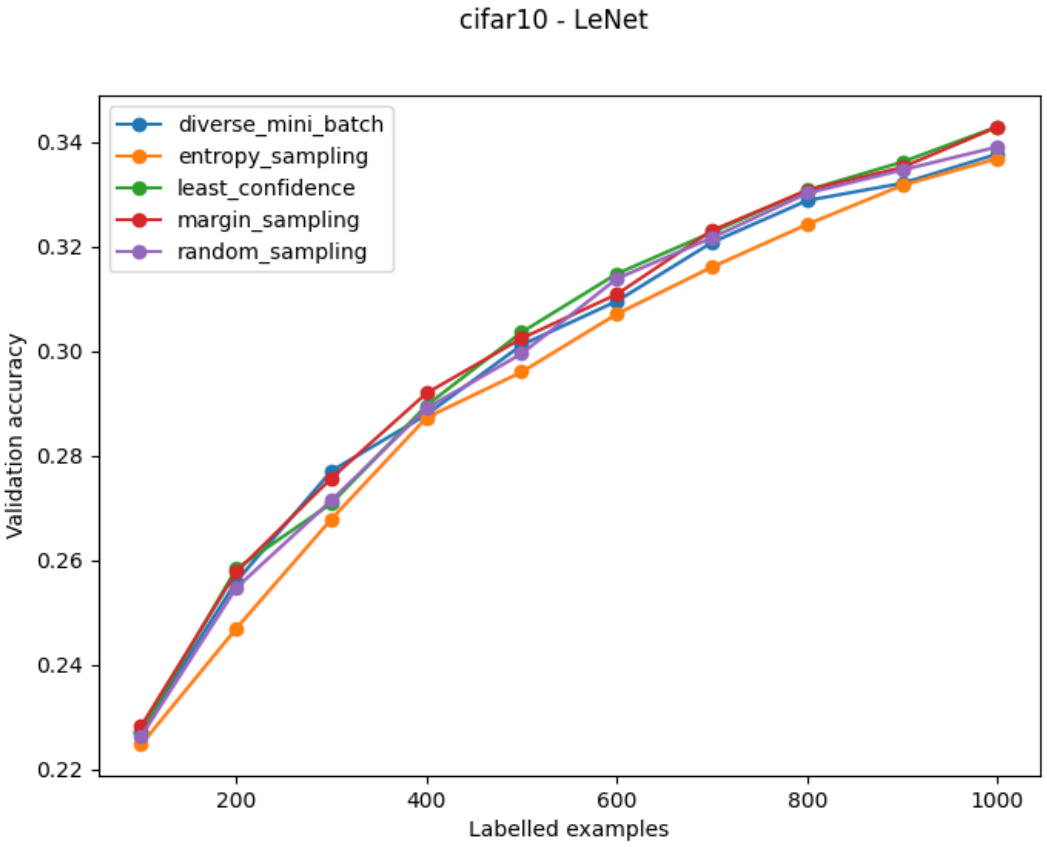


Figure 13 : Précision de validation en fonction de la taille de l'ensemble d'entraînement pour le modèle LeNet sur l'ensemble CIFAR-10

Lorsque nous comparons les différences de performances des stratégies de *query* sur CIFAR-10 avec le modèle LeNet, nous remarquons qu'il n'y a pas de différences majeures de performance entre les stratégies considérées. Cependant, *entropy sampling*

semble présenter des performances légèrement plus faibles quelle que soit la taille de l'ensemble d'entraînement.

Nous rappelons que LeNet a été créé en 1989 et qu'il ne possède que 5 couches de neurones à convolution. C'est donc un réseau simple qui n'a pas été conçu pour la classification d'images variées et en couleur que représente le défi de CIFAR-10. La complexité du problème est donc trop haute par rapport à la capacité du modèle, ce qui explique ses faibles performances.

Sur cet ensemble de données spécifiquement, le modèle ne montre pas de préférence suffisamment marquée pour une des classes de chaque instance considérée au moment de la prédiction lorsqu'il n'a pas encore appris à la classer. Il semble que *least confidence* et *margin sampling* seront deux mesures plus adaptées pour détecter ces instances que le modèle a du mal à classer.

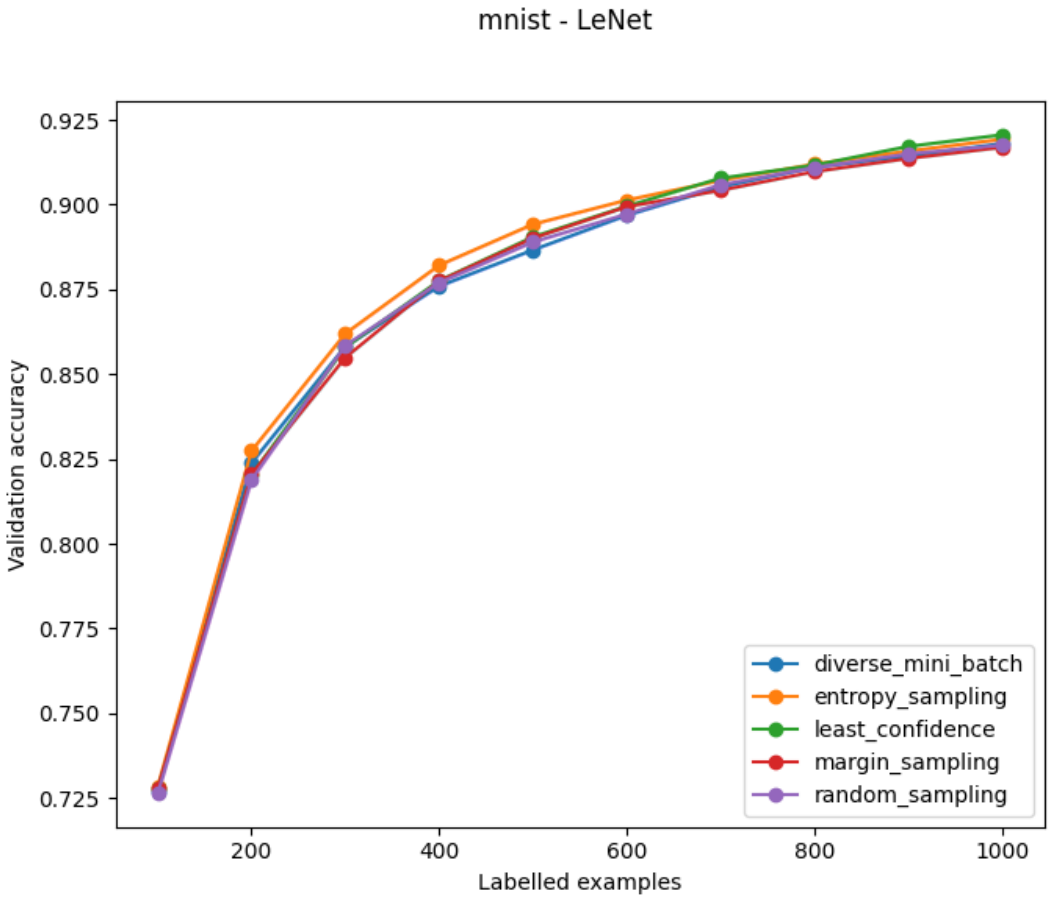


Figure 14 : Précision de validation en fonction de la taille de l'ensemble d'entraînement pour le modèle LeNet sur l'ensemble MNIST

LeNet est un réseau pour lequel nous nous attendions à obtenir de très bons résultats pour l'ensemble MNIST car la classification de caractères est le problème que ce réseau a été créé pour résoudre. Ainsi, en entraînant LeNet sur seulement 1 000 images sur un ensemble de 60 000 images, nous obtenons une impressionnante précision de validation de 92%.

Puisque le modèle performe si bien sur cet ensemble, il est capable de réaliser des prédictions précises et d'indiquer plus clairement lorsqu'il ne sait pas classer une instance.

Ainsi, l'entropie obtient ici un score légèrement supérieur aux autres métriques. Ce n'est pas un résultat dû au hasard puisqu'il a été obtenu sur une moyenne de 20 exécutions, et la variance est trop faible pour être discernable sur le graphique. Similairement au cas de CIFAR-10, le modèle donne souvent des probabilités égales pour tous les labels des données qu'il a du mal à classer, et il ne peut éliminer une classe que lorsqu'il est plus sûr de lui (ou bien éliminer plusieurs classes lors de sa prédiction) puisqu'il est très performant.

Concernant la stratégie de sélection *diverse mini batch* qui s'appuie sur l'utilisation d'une méthode de partitionnement des données en sélectionnant parmi les données les plus informatives celles qui sont les plus proches des centres des paquets générés par notre *K-Means*, nous nous attendions à ce qu'elle performe mieux que les stratégies basées sur l'incertitude car nous utilisons une configuration par *batch* de l'apprentissage actif. Ainsi, comme *diverse mini batch* est supposé prendre en compte la diversité, il devrait réaliser une sélection de données plus intéressante pour l'apprentissage que les stratégies basées sur l'incertitude.

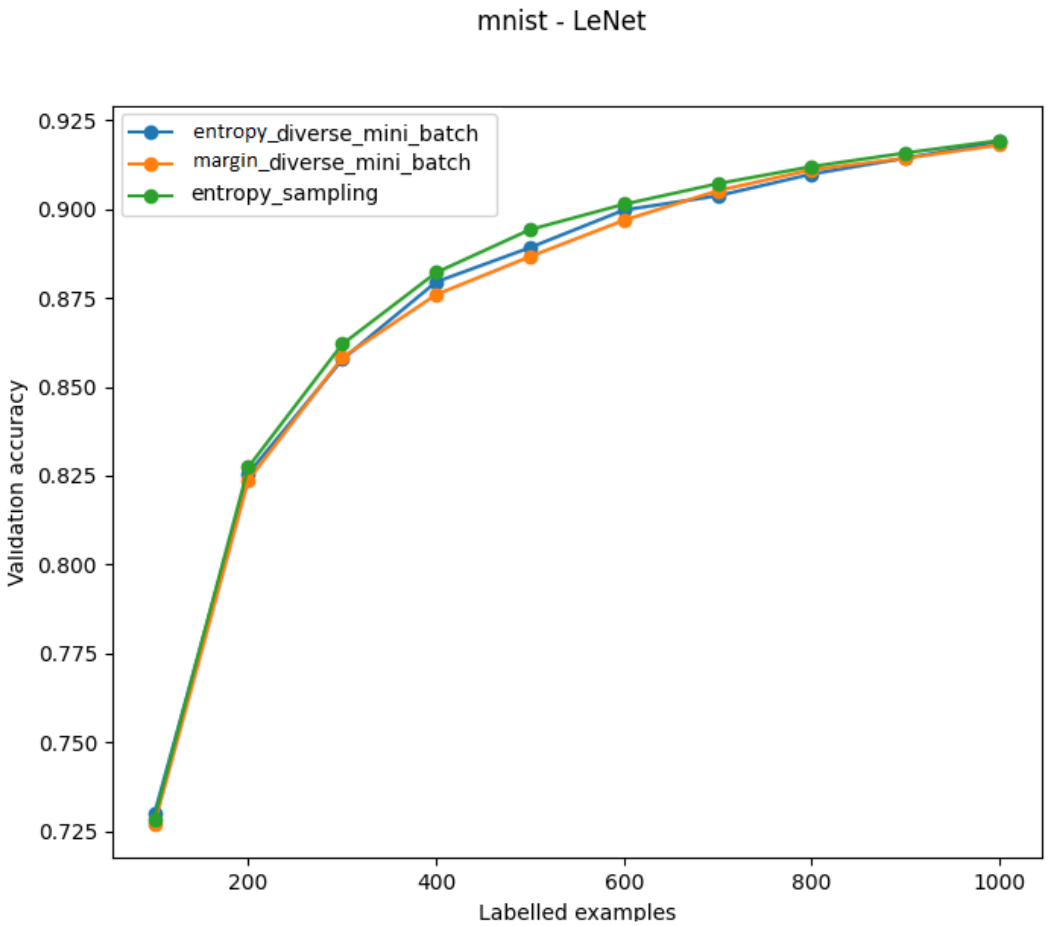


Figure 15 : Comparaison des critères de sélection pour *diverse mini batch*

La figure 15 compare la stratégie d'*entropy sampling* à *diverse mini batch* qui utilise comme stratégie de pré-sélection de ses données l'*entropy sampling* et le *margin sampling*. L'entropie présente ici des performances légèrement meilleures que *diverse mini batch* quelle que soit la stratégie de pré-sélection. Pour comprendre ce résultat, nous avons étudié les performances de la méthode de partitionnement des données utilisée au sein de *diverse*

mini batch. Il se trouve que *K-Means* obtient seulement 0.35 pour l'*adjusted rand score* qui nous permet de mesurer la similarité de la partition que *K-Means* a réalisé par rapport aux véritables labels associés aux données. Ainsi, *diverse mini batch* est handicapé par ce partitionnement des données de mauvaise qualité et ne pourra pas présenter de performances optimales. Il faudrait utiliser une méthode de partitionnement des données plus performante.

4.2.2. VGGNet

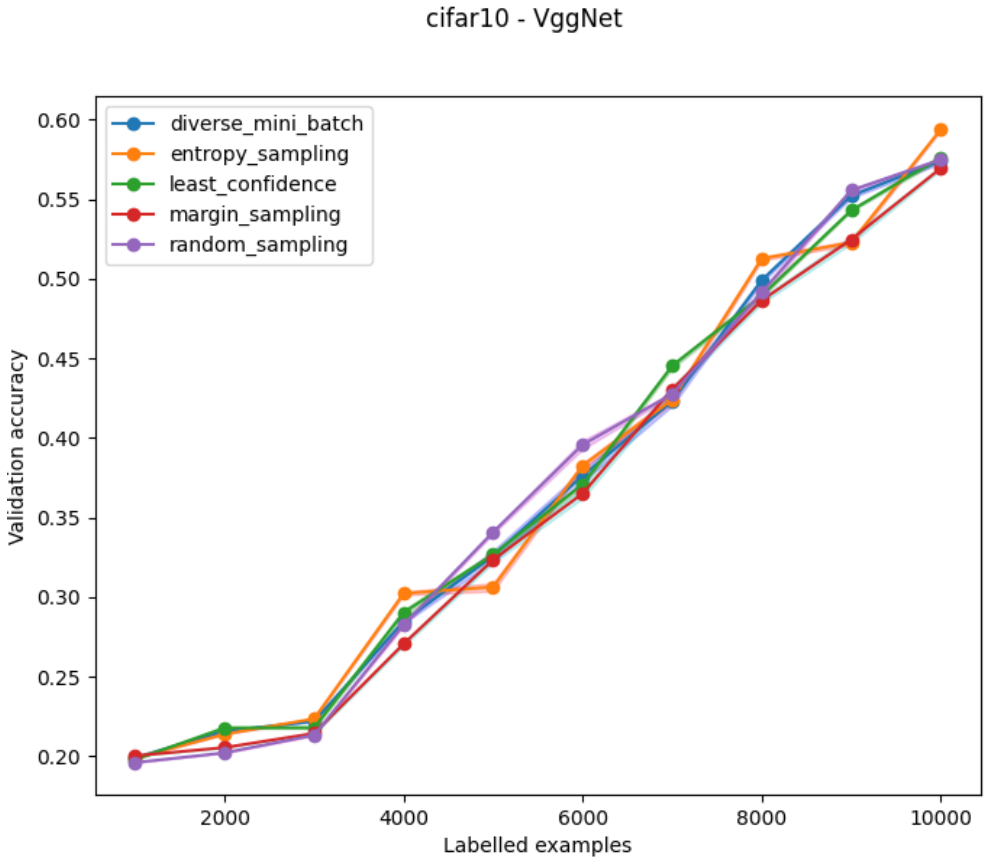


Figure 16 : Précision de validation en fonction de la taille de l'ensemble d'entraînement pour le modèle VGGNet sur l'ensemble CIFAR-10

CIFAR-10 représente un problème de classification complexe. Avec des images à 3 canaux et de natures variées, il est difficile pour un classifieur d'être performant à moins d'entraîner sur une base suffisamment grande et pour assez d'epochs. VGGNet est un modèle complexe puisqu'il comporte 16 couches de neurones à convolution., c'est pourquoi il est adapté pour résoudre le problème de classification de CIFAR-10. Ainsi, après avoir entraîné notre modèle sur 10 000 des 50 000 images totales, nous obtenons 60% de précision de validation. En comparaison, LeNet n'obtient que 42% de précision après avoir été entraîné sur un ensemble de même taille sur CIFAR-10.

Nous commençons à une taille de *seed* de 1 000 données et augmentons de 1 000 en 1 000 jusqu'à 10 000 car commencer à 100 et terminer à 1 000 comme nous l'avons fait avec LeNet ne montrait pas un apprentissage suffisant pour pouvoir tirer des conclusions.

Malgré cette performance relativement bonne, il n'est pas possible d'extraire de conclusion quant à quelle stratégie est meilleure que les autres. Toutes sont mêlées les unes aux autres, et si, par exemple, *random sampling* sort du lot pour 2 points, il n'est pas spécialement meilleur sur le reste de la courbe. Nous savons que les réseaux de neurones profonds ne sont pas très bons pour exprimer lorsqu'ils ne sont pas certains de leur prédiction ou non car ils tendent à être trop confiants en leur prédiction. Comme pour LeNet, les stratégies de *sampling* basées sur l'incertitude ne sont pas adaptées dans une configuration par *batch*. Ici les *batchs* sont de taille 1 000, ce qui pourrait expliquer pourquoi la performance des stratégies basées sur l'incertitude est parfois moins bonne que l'apprentissage passif (ici *random sampling)*. Puisque ces stratégies sélectionneraient plus d'instances d'une même classe qu'il ne suffit au VGGNet pour apprendre à la classer.

Suivant cette logique, on devrait s'attendre à ce que *diverse mini batch* soit meilleur que les stratégies basées sur l'incertitude. Mais pour la même raison que LeNet avec MNIST, notre partitionnement des données par *K-Means* n'est pas assez performant pour que *diverse mini batch* fonctionne comme prévu. Ici, le *adjusted rand score* est même de 0.040, ce qui indique que *K-Means* n'est pas efficace et que *diverse mini batch* revient presque à une sélection aléatoire basée sur la pré-sélection par *Margin sampling* car sa méthode de partitionnement des données n'a pas été en mesure de modéliser la distribution des classes au sein de l'espace des images. Les paquets ne sont peut être pas sphériques, ou peut être existe-il de nombreux paquets pour une même classe (par exemple les images de chevaux noir seraient dans un paquet différent des chevaux blancs). Mais une recherche d'hyperparamètres pour *K-Means* faisant varier le nombre de paquets n'a pas permis d'améliorer sa performance.

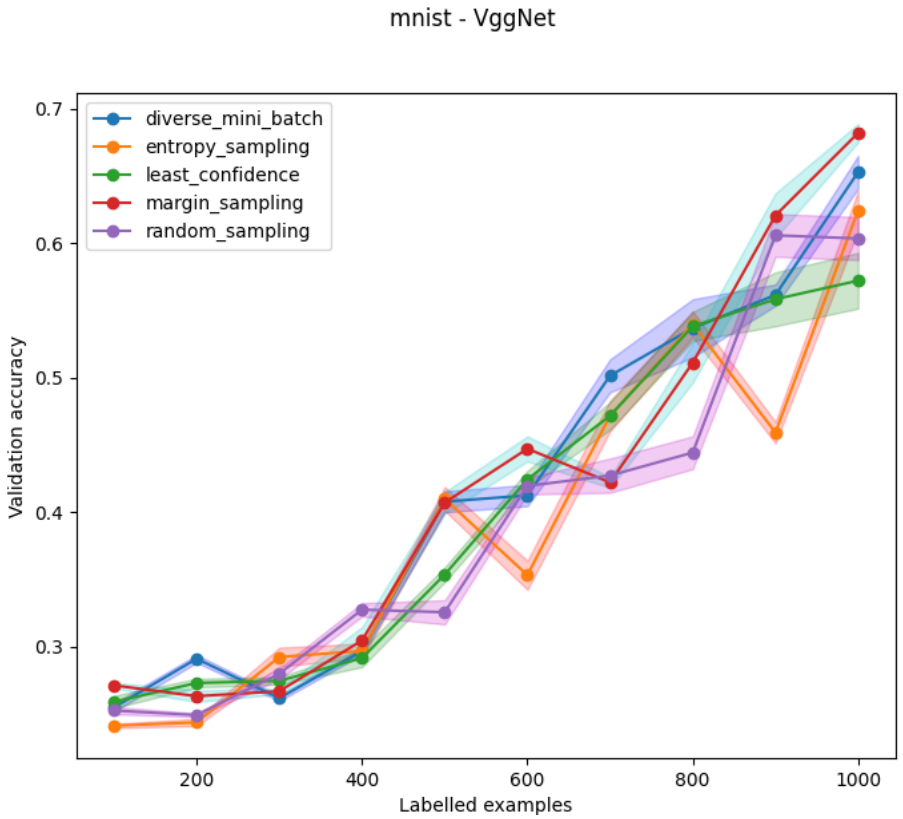


Figure 17 : Précision de validation en fonction de la taille de l'ensemble d'entraînement pour le modèle VGGNet sur l'ensemble MNIST

Sur la base de données MNIST, l'utilisation de VGGNet représente l'application d'un modèle complexe sur un problème simple. VGGNet possède un grand nombre de couches et donc de poids, il requiert ainsi un plus grand nombre de données et d'epoch pour que son entraînement se stabilise. Cela explique la variance relativement grande que nous observons sur la figure 17.

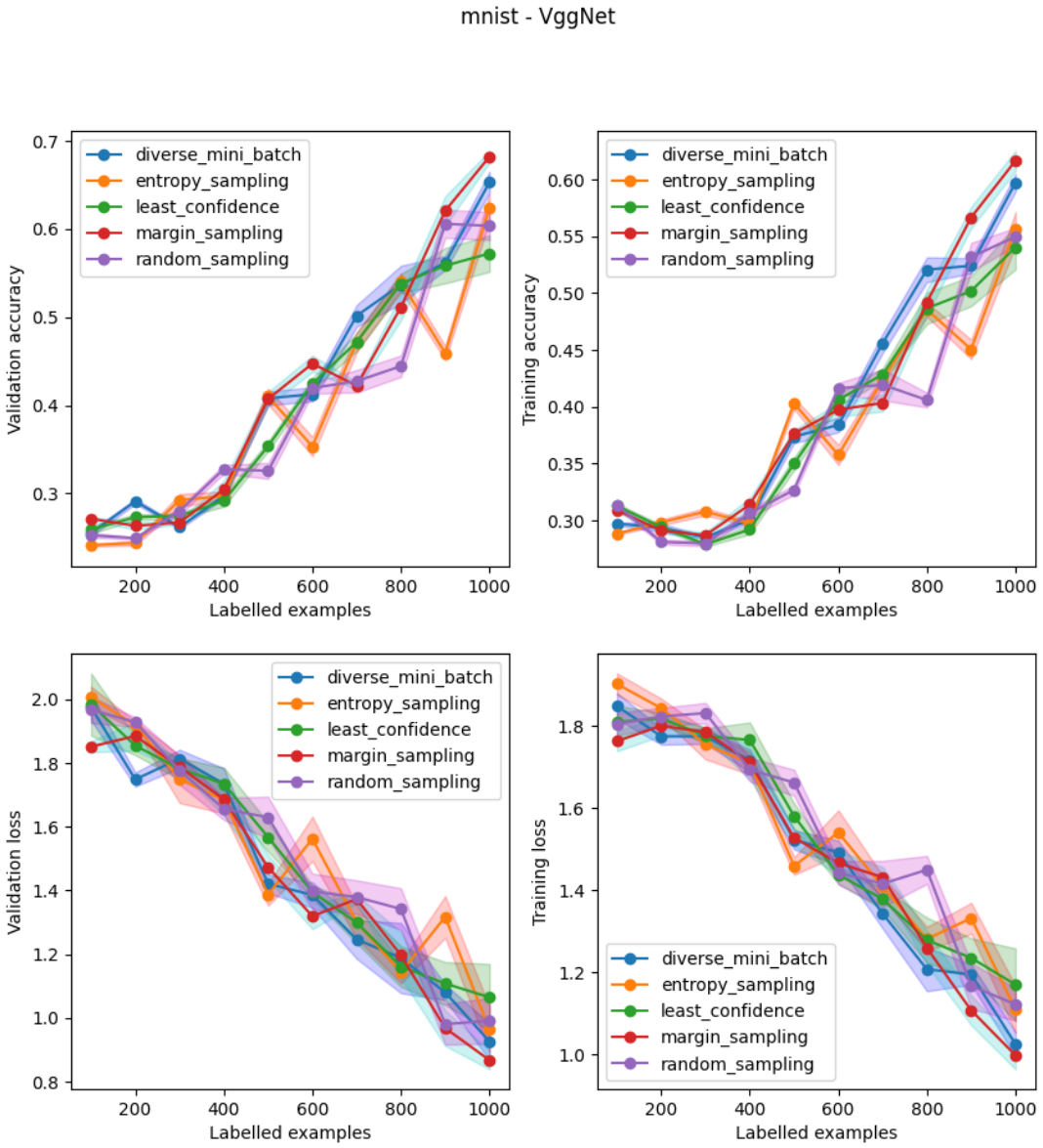


Figure 18 : Précision de validation (en haut à gauche), précision d'entraînement (en haut à droite), perte de validation (en bas à gauche) et perte d'entraînement (en bas à droite) en fonction de la taille de l'ensemble d'entraînement pour le modèle VGGNet sur MNIST

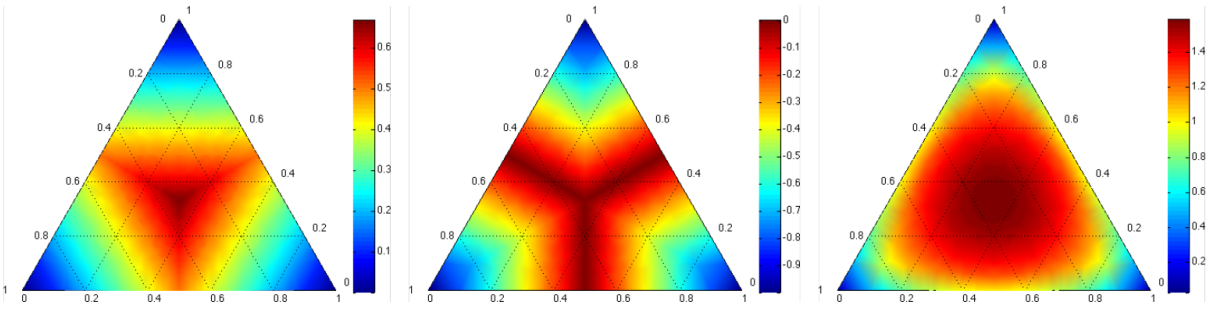
Pour nous convaincre de l'affirmation précédente, nous avons tracé la précision d'entraînement de VGGNet sur MNIST. Nous en concluons que le modèle a besoin d'une certaine quantité de données avant de voir sa performance s'améliorer. Cela alors que LeNet est immédiatement capable d'intégrer les données qui lui sont fournies pour améliorer sa prédiction. Nous observons que VGGNet améliore tout de même de manière très régulière ses résultats, presque linéairement. Peut-être serait-il capable d'obtenir des

performances similaires à LeNet avec plus de données d'entraînement. Néanmoins, sa précision varie beaucoup entre 100 et 1 000 données. Pour cette raison, il est difficile de tirer des conclusions concernant quelle stratégie est la meilleure pour cette configuration de modèle et d'ensemble de données.

4.3. Interprétation des résultats

Dans la section précédente, nous avons pu observer qu'il y avait peu de différence entre les performances de chacun des critères implémentés pour l'apprentissage actif. Une cause possible est le fait que les réseaux de neurones sont trop extrêmes dans leurs prédictions et ne peuvent donc pas dire pour quelles données ils sont moins certains que d'autres. De plus, il se peut que la *query size* utilisée ait été trop importante. Dans ce cas, les informations se chevauchent, ce qui affaiblit la différence entre le *random sampling* et les autres critères.

Enfin, le manque de résultat probant peut être dû à la nature même des critères de sélection. En théorie de l'information, l'entropie est une mesure de la quantité d'information nécessaire pour encoder une distribution. C'est pourquoi elle peut être considérée comme une mesure de l'incertitude et de l'impureté en apprentissage machine.



(a) least confident (b) margin (c) entropy

Figure 19 : Carte de chaleur illustrant la mesure d'informativité réalisée par les 3 stratégies d'incertitude pour un problème à 3 labels [11]. Les coins représentent là où un seul label a une probabilité très élevée, avec le bord opposé montrant la plage de probabilité pour les deux autres classes lorsque cette étiquette a une probabilité très faible

Comme l'illustre la figure 19, l'entropie diverge des autres mesures basées sur l'incertitude dans sa représentation de l'informativité. Par exemple, l'entropie ne considère pas les instances où seul un des labels est hautement improbable (le long des côtés extérieurs du triangle) comme étant des points intéressants, tandis que les deux autres méthodes considérées estiment que ces instances sont à favoriser si le modèle ne peut pas distinguer entre les deux labels restants. Aucune des trois méthodes n'est généralement meilleure qu'une autre, mais elles semblent mener à des résultats plus ou moins bons en fonction du problème considéré.

Néanmoins, nous pouvons conclure que *least confidence sampling* semble être la métrique qui permet d'obtenir les résultats les plus consistants parmi les deux modèles et les différents ensembles de données. Là où *random sampling* pouvait avoir par exemple des pics de précision vers le haut ou vers le bas, l*east confidence sampling* permet systématiquement d'obtenir une amélioration par rapport aux résultats précédents. Nous

observons ce comportement de manière très claire à la figure 18. Là où toutes les méthodes présentent des performances irrégulières, *least confidence sampling* est la seule méthode à présenter une courbe lisse et régulière.

Nous nous attendions à ce que la *diverse mini batch* produise de meilleurs résultats que les stratégies basées seulement sur le degré d'information contenu dans une certaines données. En effet, cette méthode permet de prendre en compte la diversité des classes des données que le réseau sélectionne pour l'oracle. Or, il semble que la méthode de partitionnement des données ne soit pas assez performante pour permettre à *diverse mini batch* d'être véritablement efficace. Il faudrait donc utiliser une méthode de partitionnement des données plus élaborée pour que la performance de *diverse mini batch* soit plus représentative de ce que cette stratégie est capable de réaliser.

Pour ce qui est de la différence de performance des réseaux suivant la base de données, nous l'expliquons d'abord par le fait que LeNet est un réseau convolutif peu profond qui a été créé dans le but de résoudre un problème simple de classification. Il sera donc bien plus performant sur la base de données MNIST que sur CIFAR-10. De plus, sa simplicité permet de plus facilement déduire son degré de confiance en sa prédiction. Les mesures d'informativité sont donc plus efficaces.

Nous avons le problème inverse avec VGGNet : étant un réseau convolutif très profond, il est particulièrement performant pour la base de données CIFAR-10 mais il semble converger erratiquement lorsqu'utilisé sur la base MNIST. En effet, comme cette seconde base de données représente un problème plus simple, le modèle a de la difficulté à converger directement vers un minimum. Une solution serait d'expérimenter avec un taux d'apprentissage plus faible ou une descente de gradient non-stochastique. Aussi, à cause de sa plus grande complexité, les résultats semblent plus aléatoires. Il est alors plus difficile de savoir si le réseau est certain de sa prédiction.

4.4. Utilisation de Calcul Canada

Afin de générer les courbes présentées, nous avons entraîné nos modèles de nombreuses heures. Pour accélérer leur génération, nous avons utilisé les serveurs de Calcul Canada. Grâce au script *launch_calcul_canada.py*, nous avons créé sur un nœud de calcul un environnement python et nous avons lancé le script *runner.py* qui a lui-même lancé à la suite autant de fois que nécessaire le script *train.py* avec les paramètres entrés. De cette manière, nous avons pu générer séquentiellement tous les résultats qu'il nous fallait sans utiliser plus d'un des quatre GPUs qui étaient à la disposition du groupe d'IFT780.

Nous notons également que, comme nous générions nos fichiers de résultats à la fin de chaque boucle complète d'apprentissage actif, nous n'avons pas perdu de résultats et nous pouvions interrompre l'exécution sans perdre ce qui avait déjà été produit.

5. Conclusion

Les résultats que nous avons pu produire montrent qu'il est difficile de mettre en place l'apprentissage actif avec succès lorsqu'on travaille avec des réseaux de neurones profonds. En effet, ces réseaux sont généralement trop optimistes dans leurs prédictions et n'indiquent pas clairement lorsqu'ils ne sont pas certains du label à associer à une instance. Pour cette raison, les stratégies de sélection des données pour l'apprentissage actif basées sur l'incertitude du modèle ne sont pas les plus adaptées. Cette affirmation est renforcée par le fait que ces stratégies sont plus performantes dans un contexte d'apprentissage actif d'image à image et non de sélection d'images par *batch*. Malheureusement, les réseaux de neurones profonds sont longs à entraîner, il n'est donc réalistiquement pas possible d'utiliser cette méthode.

Pour cette raison, des stratégies telles que *Diverse Mini Batch* ont été développées. Elles reposent sur l'intégration d'une notion de diversité au moment du choix des données à ajouter à notre ensemble de données d'entraînement. Cependant, elle requiert l'utilisation d'une méthode de partitionnement des données performante pour pouvoir présenter un réel intérêt.

Pour terminer, il est important de réaliser un choix éclairé lorsqu'il s'agit de sélectionner notre critère de sélection des données pour l'implémentation de l'apprentissage actif dans un réseau de neurones.

Références :

[1] https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96
[2] Very deep convolutional networks for large scale image recognition, K. Simonyan, A. Zisserman, ICLR 2015.
[3] Nash, Will & Drummond, Tom & Birbilis, Nick. (2018). A review of deep learning in the study of materials degradation.
[4] Mazieres, Antoine. (2016). Cartographie de l'apprentissage artificiel et de ses algorithmes.
[5] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun. (2015). Deep Residual Learning for Image Recognition.
[6] Saining Xie and Ross Girshick and Piotr Dollár and Zhuowen Tu and Kaiming He.(2017). Aggregated Residual Transformations for Deep Neural Networks.
[7] Ana Solaguren-Beascoa, PhD - Active Learning in Machine Learning, A brief introduction to the implementation of active learning
[8] Ajit Kale - Medium : #APaperADay Week1: A Meta-Learning Approach To One-Step Active Learning
[9] https://fr.wikipedia.org/
[10] lebigdata.fr
[11] Burr Settles - Active Learning Literature Survey (2010)
[12] Bo Du -Exploring Representativeness and Informativeness for Active Learning (2019)
[13] https://medium.com/@kaleajit27/apaperaday-week1-a-meta-learning-approach-to-one-step-active-learning-5ffea59099a2
[14] https://neurohive.io/en/popular-networks/vgg16/
[15] https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035