Intérêt de l'intégration de la phylogénèse dans la génèse d'une entité neurogénétique numérique

Jean-Patrice Glafkides

UNIVERSITE PARIS VIII

THESE

Pour obtenir le grade de

DOCTEUR DE l'UNIVERSITÉ PARIS VIII

Discipline : INFORMATIQUE

Présentée et soutenue par

GLAFKIDES Jean-Patrice

Le 7/12/2023

INTERET DE L'INTEGRATION DE LA PHYLOGENESE DANS LA GENESE D'UNE ENTITE NEUROGENETIQUE NUMERIQUE

Directeur de thèse : **Herman AKDAG**

JURY :

Mme Hajer BAAZAOUI	Professeure des Universités Université de Cergy Pontoise	Rapporteur
M. Cyril DE RUNZ	MCF HDR Université de Tours	Rapporteur
Mme Lynda SEDDIKI	MCF Université Paris 8	Examinatrice
Mme Caroline CHOPINAUD	PhD, Intelligence Artificielle Hub France IA	Examinatrice
Mme Anissa MOKRAOUI	Professeure des universités Laboratoire de Traitement et Transport de l'Information (L2TI)	Examinatrice
M. Arab ALI CHERIF	Professeur des Universités Université Paris 8, France	Examinateur

REMERCIEMENTS

J’aimerai tout d’abord remercier l’ensemble des membres de mon jury pour avoir accepté d’évaluer cette thèse.

Mme Hajer BAAZAOUI, M. Cyril DE RUNZ pour avoir accepté d’en être rapporteur, ainsi que Mme Lynda SEDDIKI, Mme Anissa MOKRAOUI, Mme Caroline CHOPINAUD et M. Arab ALI CHERIF, pour avoir accepté d’examiner ce mémoire.

Je tiens à remercier mon directeur de thèse M. Herman AKDAG pour la confiance qu’il m’a accordée en acceptant de diriger mes travaux de recherche contre vents et marées.

J’aimerais également les remercier pour leur disponibilité, écoute, respect, encouragement et compréhension.

C’est à M. Gene I. SHER et au travers de lui, à l’entreprise Datavaloris que je témoigne ma gratitude la plus profonde pour les longues heures d’encadrement qu’il m’a consacré, sa patience, et ses conseils précieux.

J’adresse également mes remerciements aux chercheurs et personnels du LIASD puis du laboratoire PARAGRAPHE pour l’accueil et les conditions de travail privilégiées ainsi que pour les séminaires de recherche qui furent pour moi un moyen d’élargir mes connaissances scientifiques. Une pensée particulière pour Mme Rakia JAZIRI.

Ces remerciements ne peuvent se conclure sans exprimer ma gratitude aux membres de ma famille pour leur soutien quotidien, leurs encouragements récurrents, et leur croyance en moi et en mes capacités et surtout leur patience.

Enfin, mes remerciements les plus chaleureux et ma reconnaissance la plus sincère vont à mon épouse qui fut pour moi une raison de me battre et réussir. Merci pour ton soutien continu, pour tes encouragements et tes coups de gueule. Je te remercie aussi pour avoir accepté de sacrifier de ton quotidien pour m’accompagner dans la réalisation de mes rêves.

Je remercie Monsieur Khaldoun ZREIK, directeur du laboratoire de Recherche PARAGRAPHE.

Pour finir je remercie les personnes qui ont relu ce mémoire de thèse, à savoir Caroline WOLFF, Solange BELKHAYAT-FUCHS, Sarah PEETERS, Alain MELLER, Gabriel ROBERT, Cédric PEINTURIER, Jean-Christophe GAY, Miguel MEMBRADO...

RESUME DE THESE

Les réseaux de neurones artificiels sont entraînés en imitant la plasticité synaptique biologique, mais ces algorithmes ont des limites : une méthode de construction des modèles d'IA considérée comme trop empirique, des problèmes d'apprentissage liés à la propagation d'erreurs lorsque la topologie devient complexe ou encore l'affectation au départ de poids aléatoires. Pour tenter d'améliorer ces modes d'apprentissage, une nouvelle méthode appelée Phylogenetic Replay Learning (PRL) est présentée dans cette thèse. L'objectif est d'en évaluer l'efficacité d'apprentissage en utilisant un algorithme de planification génétique de l'évolution de la topologie lors de l'apprentissage.

Les expériences démontrent que l'approche phylogénétique, PRL, est capable de produire un modèle plus performant qu'avec la méthode classique dite DDL. Second point, l'entropie de Shannon des poids des modèles générés par la PRL montre que les couches contiennent une meilleure répartition statistique des informations que lorsqu'un modèle est entraîné avec DDL.

Les différentes étapes constituant la recherche :

1. Évaluation de la méthode "phylogenetic replay learning", PRL : en s'appuyant sur le chemin évolutif (préalablement enregistré) d'un modèle initial vers un modèle final appelé Champion, on réentraîne le modèle initial X fois en suivant le même chemin phylogénétique enregistré et on vérifie, pour les X clones du modèle Champion obtenus, la bonne cohérence des résultats et obtenir les métriques d'apprentissage.
2. Évaluation de la méthode d'apprentissage directe (classique), DDL : en partant du modèle Champion trouvé précédemment (réinitialisé avec des poids aléatoires), on le réentraîne X fois avec la méthode DDL. On obtient X nouvelles versions du modèle Champion dont on enregistre les résultats d'apprentissage.
3. Comparaison des résultats obtenus via les deux méthodes. S'il y a une différence, on identifie laquelle et on en recherche la ou les raison(s).
4. Tests de transférabilité : on valide les précédentes expériences en utilisant de nouveaux jeux de données. Le modèle initial est de nouveau entraîné et évalué avec un nouveau jeu de données en suivant le même chemin évolutif enregistré précédemment (méthode PRL). On lance le réapprentissage du modèle Champion en suivant la méthode DDL.
5. Tests de reproductibilité : on refait les 4 premières étapes en partant d'un nouveau modèle initial, d'un nouveau chemin phylogénétique et donc d'un nouveau Champion. Évaluation de l'efficacité de la PRL dans des contextes différents.

Mots clés : Neural Networks, Neuroevolution, Phylogenetic Replay Learning, Deep learning, Vanishing gradient.

THESIS SUMMARY

Interest of using phylogenesis in building digital neuro-genetic neural networks entity

Artificial neural networks are trained by imitating biological synaptic plasticity, but these algorithms have limitations: a topology construction method deemed too empirical, error propagation issues when the topology becomes complex, or the initial assignment of random weights. To improve these learning methods, a new method called Phylogenetic Replay Learning (PRL) has been developed and presented during this thesis. It takes inspiration from natural evolutionary mechanisms. The goal of this thesis is to evaluate the effectiveness of this new method using a genetic planning simulation algorithm for the evolution of a brain.

These experiments demonstrate that the phylogenetic approach, PRL, can produce a model more effective than with the DDL method. Secondly, the Shannon entropy of the weights of the models generated by PRL shows that the deeper layers statistically contain more information than when a model is trained traditionally (with the DDL).

The different steps constituting the research are:

1. Evaluation of the "phylogenetic replay learning" method, PRL: based on the evolutionary path (previously recorded) from an initial model to a final model called Champion, the initial model is retrained X times following the same recorded phylogenetic path, and the consistency of the learning results is verified.
2. Evaluation of the direct learning method (classical), DDL: starting from the previously found Champion model (reset with random weights), it is retrained X times using the DDL method. The learning results are recorded.
3. Comparison of the results obtained via the two methods. If there is a difference, it is identified and the reason(s) for it are sought.
4. Transferability tests: the previous experiments are validated using new datasets. The initial model is retrained and evaluated with a new dataset following the same previously recorded evolutionary path (PRL method). The obtained Champion model is relearned also following the DDL method. This transferability test is replicated several times on the two methods, each time with a different dataset.
5. Reproducibility tests: a new experiment is conducted starting from a new initial model, a new phylogenetic path, and thus a new Champion; the first 4 steps above are repeated. The effectiveness of the PRL is evaluated in different contexts.

Keywords: Neural Networks, Neuroevolution, Phylogenetic Replay Learning, Deep learning, Vanishing gradient.

Table des matières

LISTE DES ABREVIATIONS.

Acc : Accuracy, métrique de l'efficacité d'apprentissage d'un modèle[i] de réseau de neurones.

ADN : Acide désoxyribonucléique.

AI : Artificial Intelligence, voir IA

DL : Deep Learning.

DDL : Direct Deep Learning (voir définition dans le glossaire).

ECD : Extraction de Connaissances à partir des Données.

HOF : Hall of Fame (voir définition dans le glossaire).

IA : Intelligence Artificielle, aussi appelé AI en anglais pour Artificial Intelligence.

ML : Machine learning.

NN ou RNN : Réseau de neurones numériques.

PRL : Phylogenetic Replay Learning (voir définition dans le glossaire).

SGBD : système de gestion de base de données.

TF : TensorFlow, Framework (voir définition dans le glossaire) de réseau de neurones de Google.

Val_acc : Validation Accuracy, métrique de l'efficacité d'un modèle de réseau de neurones sur le jeu d'entrainement.

VGE : Vanishing Gradiant effect.

[i] Un modèle est dans le cadre de cette thèse, la représentation d'un réseau de neurones numérique avec des caractéristiques propres : organisation topologique des neurones numériques entre eux, poids, couches, input (format des données entrées), output (format de données sorties) … Il est parfois appelé "Agent" dans l'algorithme de neuroévolution.

INTRODUCTION

L'intelligence artificielle généraliste (AGI, pour *artificial general intelligence*) fait référence à une forme d'intelligence artificielle qui possède la capacité d'apprendre et de comprendre n'importe quelle tâche intellectuelle que peut réaliser un être humain. Contrairement aux intelligences artificielles spécialisées ou étroites (dites *narrow*), qui sont conçues pour effectuer des tâches spécifiques comme la reconnaissance d'image, la traduction automatique ou la recommandation de produits, une AGI serait capable de s'adapter à une grande variété de domaines et de tâches.

Les travaux en intelligence artificielle généraliste tentent donc de créer des machines « intelligentes » qui simulent les cerveaux biologiques avec un haut niveau d'abstraction. L'ambition à terme des chercheurs en IA généralistes est qu'elles atteignent un niveau d'abstraction tel, qu'elles puissent être utilisées dans toutes tâches dont la complexité requiert normalement l'exercice de l'intellect humain, voire qu'elles soient douées de conscience et deviennent même globalement plus « intelligentes[ii] » que l'homme. Elles pourraient comprendre, s'auto-améliorer et pourquoi pas se reproduire… sous le contrôle humain si possible !

Nul ne sait à ce jour si cet objectif est atteignable, mais ce qui est certain c'est que de nombreuses innovations restent encore à apparaître, notamment celle s'inspirant des mécanismes biologiques qui sont la base de l'intelligence humaine. Cette thèse contribue à ce mouvement.

La représentation Figure 1 des types d'intelligence artificielle suit généralement le schéma sous forme du diagramme présenté ci-après :

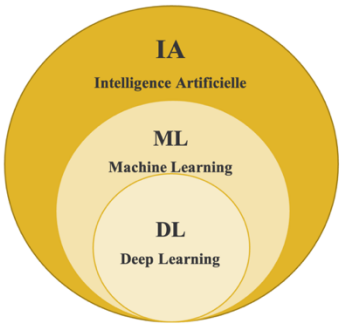


Figure 1- Diagramme de Venn Pour l'IA, le ML & le DL

Le champ de l'intelligence artificielle regroupe de nombreuses technologies imaginées puis mises en œuvre depuis la fin des années 50 pour essayer des résoudre des problèmes demandant de faire preuve des qualités propres aux humains. Cet ensemble nommé IA s'est attaché par exemple à

[ii] Larousse : Aptitude d'un être humain à s'adapter à une situation, à choisir des moyens d'action en fonction des circonstances, Ensemble des fonctions mentales ayant pour objet la connaissance conceptuelle et rationnelle.

trouver des représentations des connaissances qui permettent une plus grande liberté d'évolution pour des programmes les exploitant, introduisant ainsi une façon moins impérative et procédurale de concevoir un programme informatique.

Au sein de cet ensemble, les IA basées sur le *machine learning* (ML) ont été développées. Elles posent un abandon complet d'une approche impérative de programmation et y substituent un apprentissage préalable produisant une machine pertinente sur de nouveaux cas. Là également, de nombreuses technologies ont été produites.

Au sein du ML, une classe d'IA est développé en associant des couches de neurones artificielles denses, c'est le *Deep Learning* (DL) sur lequel cette thèse se focalise.

La plupart des IA de type DL reposent sur des modélisations mathématiques qui permettent de concevoir des architectures de réseaux de neurones artificiels. On parle aujourd'hui de « jumeaux numériques » qui imitent - ou plutôt simulent - le fonctionnement des neurones biologiques. Les algorithmes évoluent rapidement depuis ces vingt dernières années grâce, entre autres, à l'augmentation des puissances de calcul. Les processeurs modernes décuplent les capacités de traitement et donc accroissent l'efficacité des ordinateurs de façon inédite et exponentielle. Cette efficacité permet enfin aux réseaux de neurones numériques (RNN) d'atteindre des capacités traitement des données complexes utiles aux besoins humains. Néanmoins de nouvelles limites sont atteintes.

En 1996, par expérience personnelle en banque, les *traders* des salles de marché s'appuyaient encore sur de l'analyse géométrique et statistique de courbes des cours de Bourse alors que la capacité des réseaux de neurones aurait déjà pu les aider dans l'exécution de leurs tâches quotidiennes. A l'époque, le milieu professionnel n'était pas exposé aux apports potentiels du ML dans leur secteur d'activité. Moins d'un an plus tard, en mai 1997, le super calculateur Deep Blue d'IBM battait le champion du monde d'échecs en titre, Gary Kasparov [1]. Mais malgré cette preuve éclatante de l'efficacité du ML, l'approche algorithmique classique restait encore la base de travail de la plupart des entreprises.

En 1998, se déroulent de nouvelles tentatives de sensibilisation à l'IA/ML. Des sociétés évoluant dans le secteur de l'assurance de voitures effectuent des tests utilisant des solutions basées sur les réseaux de neurones pour améliorer le calcul de risques incendie, accidents et risques divers (IARD). Au lieu des 5 critères habituels utilisés par les professionnels de l'assurance pour calculer les risques, cette solution pouvait tenir compte de plus de 20 critères. Cependant, dans les faits, les entreprises n'étaient toujours pas prêtes à se passer d'une formule mathématique mise en œuvre dans un algorithme classique, ni à adopter un système basé sur du ML proposant une solution plus fine mais difficilement explicable.

A l'époque, rares étaient les experts à défendre auprès des entreprises une approche basée sur le ML pour modéliser des problèmes complexes. *Alpha go* de Google va changer la donne en octobre 2015. Ce programme permet une prise de conscience sur l'intérêt de ces technologies d'IA, tant au sein du grand public qu'au cœur du milieu de la recherche. Avec ce programme qui simule un joueur de *go*, Google a su démontrer la puissance de cette nouvelle approche [2] pour résoudre une problématique de systèmes complexes. L'IA/ML, plus précisément l'IA/DL, entrait enfin dans une phase un peu plus opérationnelle.

Au fil des années, la science de l'IA/ML n'a cessé de progresser sans pour autant trouver une façon de fabriquer, de manière mathématique, la structure des modèles[iii] à base de réseaux de neurones numériques (aussi appelés « entités neuronales »). Il est alors difficile pour l'IA de sortir du carcan de la recherche empirique dans certains cas.

L'IA/DL se définit comme une topologie[iv] de réseaux de neurones numériques comportant plusieurs couches (voir définition dans le glossaire). C'est l'architecture classique des modèles de réseau de neurones utilisée actuellement.

Pour construire ces modèles, il existe deux principales méthodes. La première consiste à fabriquer manuellement la topologie (ou à copier un modèle existant que l'expert essaye d'adapter) parfois aidé d'outillages de type AutoML pour tester automatiquement certains paramètres. Cette première approche la plus usitée aujourd'hui. La seconde est une construction automatisée de la topologie basée sur des algorithmes génétiques souvent nommée « neuroévolution »[v] [3] moins fréquente.

La plus utilisée des méthodes consiste à utiliser un modèle de réseaux de neurones avec une topologie unique fixe, déterminée par son concepteur. À partir d'un jeu de données représentant la problématique à résoudre, il fait apprendre à ce modèle un comportement en utilisant un algorithme d'apprentissage choisi. S'il est satisfait du résultat, il conserve le modèle. Sinon l'expert recommence avec un nouveau modèle. Cette approche serait parfaite s'il était possible

---

[iii] Un « modèle » est, dans le cadre de cette thèse, la représentation d'un réseau de neurones numérique avec des caractéristiques propres : organisation topologique des neurones numériques entre eux, poids, couches, *input* (format des données entrées), *output* (format de données sorties) … Il est parfois appelé "agent".

[iv] La topologie du réseau de neurones d'un modèle représente son architecture (sous forme de graphe). C'est-à-dire le nombre de neurones, la manière dont ils sont reliés, leur méthode de fonctionnement… Dans cette thèse, les poids des liens entre les neurones du réseau ne sont pas traités comme faisant partie de la topologie impactée par les mutations.

[v] Le processus neuroévolutif est une mise en œuvre d'algorithmes génétiques pour générer de meilleurs modèles d'intelligence artificielle selon les objectifs à atteindre (par exemple, reconnaissance de forme, génération de langage …) en simulant un processus d'évolution « naturelle ». La neuroévolution n'est utilisée dans cette thèse que pour définir le chemin évolutif (phylogénétique) au moment de la phase 0 de la méthodologie de la thèse.

de construire la topologie du modèle de manière formelle en lien avec le problème à résoudre ; ce qui n'est aujourd'hui pas le cas. De nos jours, la construction du modèle est empirique et s'appuie souvent sur un échange de schémas de modèles ou de bonnes pratiques entre experts. C'est une approche manuelle « artisanale » qui nécessite du temps et de l'énergie engendrant des coûts cachés (par exemple, le nombre d'expériences préalables qui ont été nécessaires dans le temps pour obtenir un résultat considéré comme acceptable pour une problématique donnée).

La neuroévolution utilise, quant à elle, des algorithmes évolutifs pour créer les modèles de réseaux de neurones. Cette approche combine des concepts de l'apprentissage automatique (voir définition dans le glossaire) et de l'évolution artificielle pour faire évoluer les topologies et les paramètres des réseaux de neurones.

Avec le concept de la neuroévolution, les réseaux de neurones sont considérés comme des individus dans une population. Les algorithmes génétiques sont utilisés pour générer de nouvelles variantes de réseaux de neurones à partir des individus existants en appliquant des opérations inspirées de la biologie, telles que la mutation et la recombinaison. La sélection « naturelle » est utilisée pour favoriser les réseaux qui auraient les meilleures performances pour une tâche donnée.

La neuroévolution est particulièrement adaptée aux problèmes pour lesquels les architectures optimales des réseaux de neurones ne sont pas encore clairement définies ou lorsque la recherche d'une solution optimale est rendue difficile du fait d'un espace des possibles trop vaste. La neuroévolution a ainsi été appliquée avec succès dans des domaines tels que l'apprentissage par renforcement, la robotique et la production de contenus.

C'est un mécanisme itératif qui inclut la recherche de la meilleure topologie dans la sélection des modèles considérés comme les plus efficaces pour un jeu de données choisi. Même si l'on obtient de bons résultats, la neuroévolution est coûteuse en termes de ressources du fait d'une recherche nécessaire dans l'espace des possibles. Cependant, à chaque modification majeure du jeu de données (soit de la problématique donnée), la recherche neuroévolutive de la meilleure topologie doit être relancée sur le nouveau jeu de données pour tout réapprendre dans ce nouveau contexte. De ce point vue, l'approche n'apporte pas de changement notable par rapport au cas de la construction manuelle du modèle de DL en général

En effet, la topologie d'un modèle obtenu par neuroévolution est fixe pour un jeu de données donné et refaire le processus neuroévolutif (sur un jeu de données différent ou non) générera un nouveau modèle adapté mais différent du précédent.

Aucune des méthodes de construction examinée ne permet de prendre en compte une topologie variable tout au long de l'apprentissage du modèle. A l'inverse, pour les animaux avec système

nerveux central supérieur, la topologie est variable tout au long de l'apprentissage. Cette différence fondamentale constitue l'objet central de ma thèse.

A ce jour, les méthodes utilisées par les experts en IA pour créer de nouveaux modèles ne considèrent pas le changement topologique programmé lors de l'apprentissage d'une entité neuronale numérique. En d'autres termes, la topologie (hors poids) du réseau neuronal artificiel, après l'apprentissage du modèle, reste la même qu'avant l'apprentissage (algorithmes de *deep learning*). Lorsque ce n'est pas le cas, l'apprentissage nécessite une nouvelle recherche topologique coûteuse dans l'espace des possibles (algorithmes neuroévolutif). Pourtant la nature utilise bien le changement programmé de la topologie des cerveaux des espèces intelligentes lors de leur évolution. Est-ce que, dans le contexte de l'IA, la prise en compte de cette évolution programmée de la topologie pourrait permettre de créer des modèles de réseaux de neurones numériques plus efficaces, ou ayant des capacités d'abstraction plus élevées ?

Cette réflexion et les travaux exposés dans cette thèse ont démarré dans les années 2000. L'aboutissement en est une nouvelle approche et méthode, qui est exposée, évaluée et validée au sein de ce manuscrit. *In fine*, cette méthode permet une modification programmée de la topologie des réseaux lors de la réutilisation d'un modèle de réseaux de neurones et permet un apprentissage plus efficient comparé aux autres méthodes.

A l'origine, le sujet de la thèse est parti d'une intuition : existe-t-il une corrélation dans la nature entre l'évolution des organes des espèces et l'évolution des organes lors de l'embryogénèse (formation de l'embryon lors de la gestation), toutes espèces confondues ? Le cerveau de tout être vivant n'échappe pas à cette règle avec, une programmation génétique innée et également épigénétique avec une part d'adaptabilité acquise de l'espèce, les deux influant sur la topologie, en constante modification tout au long de la vie.

Il ne faut pas perdre de vue cette part d'adaptabilité de l'espèce elle-même, durant l'évolution, notamment celle liée au constant apprentissage de la vie et dont la conséquence est un apprentissage progressif des plus efficaces.

De nombreux enjeux entrent en ligne de compte dans la mise au point d'une IA pour trouver une solution qui soit non seulement optimisée mais également potentiellement industrialisable. Une mise au point qui reste très proche de la façon dont la nature évolue dans le monde du vivant. Le point focal de cette thèse sera de se concentrer tout particulièrement sur l'impact d'une programmation génétique de l'évolution topologique lors de l'apprentissage du modèle de réseaux de neurones numériques.

Le propos est donc d'établir l'importance et l'influence de l'évolution topologique programmée du réseau de neurones numériques.

L'approche développée dans cette thèse se distingue des travaux habituels de recherche en IA/DL sur les réseaux de neurones numériques par le fait que l'objectif ici n'est pas de trouver de nouvelles approches purement mathématiques et algorithmiques mais de "jouer" par biomimétisme sur l'évolution topologique afin de mieux maitriser progressivement la complexité du réseau de neurones.

Le sujet de cette thèse est d'évaluer l'importance et l'utilité d'une programmation génétique de l'évolution de la topologie d'un réseau de neurones dans l'apprentissage d'un modèle dédié à la reconnaissance d'images.

Il s'agit donc d'intégrer la variation topologique du réseau de neurones au cours de la croissance de ce dernier et donc de planifier cette évolution topologique dans l'algorithme en se basant sur un arbre phylogénétique (également appelé arbre généalogique dans le domaine du vivant) connu. Dans notre protocole, il est déterminé et enregistré lors d'une étape 0 (voir définition dans le glossaire) et il représente ainsi le code génétique à associer au modèle initial. Chaque étape de la construction du chemin phylogénétique porte des mutations (changement topologique - voir définition dans le glossaire) que l'on applique sur la topologie des modèles successifs lors de l'apprentissage. Au niveau de la problématique considérée, on planifie ainsi génétiquement un développement du modèle pour atteindre un modèle champion (représentant un état adulte - voir

définition dans le glossaire), obtenu en partant d'un modèle initial (représentant un état embryonnaire - voir définition dans le glossaire) vers des niveaux d'abstraction élevés.

Cette recherche est importante car il est nécessaire de trouver des pistes qui permettront en IA/DL d'obtenir des topologies qui atteignent des niveaux d'abstraction plus élevés, ce que les modèles existants ne permettent pas encore aujourd'hui. En d'autres termes, il est difficile pour les algorithmes IA actuels d'appréhender des problèmes de plus en plus complexes.

Si l'on se réfère au monde biologique, plus les fonctions cognitives des cerveaux sont développées, plus la phase d'apprentissage est longue. Elle est, de surcroît, associée à une modification structurelle programmée génétiquement dans notre ADN lors de la croissance. Ainsi, par exemple, la topologie d'un cerveau d'adolescent a subi une évolution programmée et est différent de la topologie du cerveau d'un bébé.

Mais cela ne se passe pas ainsi dans l'univers numérique. Les modèles de réseaux de neurones que l'on produit aujourd'hui, que ce soit avec des technologies de type *Tensorflow* de Google ou autres concurrents, ont de plus en plus de mal à apprendre dès lors que leurs topologies dépassent un certain nombre de seuils (en termes : de nombre de neurones, de taille, de complexité, de couches, de connexions...). Par ailleurs, le développement de ces modèles ne tient compte que d'une phase d'apprentissage purement algorithmique correspondant à de l'acquis dans le monde biologique. En y greffant une évolution programmée "génétiquement" de la topologie durant l'apprentissage, pourrait-on enfin le faciliter et à terme, peut-être obtenir des modèles plus complexes ? La difficulté d'apprentissage de ces nouveaux modèles est en effet un des freins actuels majeurs à l'obtention de modèles à haute capacité d'abstraction.

La société Datavaloris, créée en 2015, est spécialiste en algorithmes génétiques appliqués aux réseaux de neurones numériques. Elle a soutenu cette thèse en permettant d'utiliser leurs algorithmes comme base de travail. Leur expert en algorithmie génétique a permis d'obtenir un algorithme génétique adapté au projet permettant d'enregistrer les étapes évolutives pour pouvoir ainsi les réutiliser.

La présente recherche a pour objet l'"Intérêt de l'Intégration de la phylogénèse dans la genèse d'une entité neurogénétique numérique". Cette thèse sera traitée en huit chapitres, tous essentiels pour faire le tour du sujet proposé.

A commencer par une introduction. Ce premier chapitre reprendra le contexte dans lequel cette recherche s'est réalisée. Après un aperçu de l'état de l'art du domaine des réseaux neuronaux, il sera mis en exergue les principales problématiques qui touchent ce secteur, ce qui permettra de mieux saisir les enjeux d'une telle recherche notamment dès lors qu'il s'agit de réseaux neuronaux à haut degré d'abstraction.

Une fois le contexte posé et l'originalité de l'approche expliquée, le deuxième chapitre s'intitule *Domaine de la Recherche.* Il présentera plus en profondeur sur les principes fondamentaux des réseaux neuronaux actuels ainsi que sur les concepts qui seront manipulés durant cette thèse.

Le cœur du sujet de cette thèse est abordé dans le troisième chapitre *Problème et Méthode* abordera Nous y posons en détail la problématique et les différentes étapes de la méthodologie mise en application. Ce chapitre vise à mieux comprendre les expérimentations décrites dans la suite du texte..

Un quatrième chapitre relate toutes les descriptions et analyses des sept expériences réalisées durant cette recherche. Ces sept expériences ont pour objectif d'évaluer la pertinence de la méthode phylogénétique au monde de l'intelligence artificielle. Il précède le chapitre, *Discussions* qui comme son titre l'indique, permettra de discuter sur cette recherche et d'ouvrir le sujet vers de nouvelles recherches potentielles.

La conclusion intégrera les perspectives que ce manuscrit ouvre.

Les trois derniers chapitres permettent au lecteur de disposer de l'ensemble des outils d'appréhension du thème avec les connaissances nécessaires. Ils consistent en Bibliographie, Liste des tableaux et Figures et Annexes.

1. REVUE DE LITTÉRATURE

Dans ce chapitre il sera présenté le cadre théorique des objets manipulés pour atteindre l'objectif puis deux exemples dans le cadre expérimental qui sont des approches complémentaires dans l'objectif de l'apprentissage des modèles complexes. Aucune publication n'a été trouvée concernant la programmation de l'évolution topologique d'un RNN lors de l'apprentissage du RNN outre celle effectuée dans le cadre de cette thèse [41].

1.1 CADRE THEORIQUE

1.1.a RESEAUX DE NEURONES – DESCRIPTION ET LIMITES

Le cerveau humain composé de 100 milliards de neurones est capable de tâches complexes allant jusqu'à la conscience de soi. La recherche sur le cerveau humain est de plusieurs ordres, essayer de comprendre son fonctionnement d'un côté, et essayer de le reproduire d'un autre.

Notre sujet ici n'est pas tant de le comprendre que d'essayer de le reproduire ou plus humblement de simuler certaines de ses capacités.

En simplifiant il est communément admis qu'un neurone biologique est composé de trois zones, le soma ou corps cellulaire qui s'active ou non, l'axone qui propage le flux d'activation aux autres neurones au travers des synapses et des dendrites qui assurent le transfert de l'activation.

C'est la somme des influx nerveux reçus qui initie ou non l'activation du neurone puis de la propagation (potentiel d'action)

Le réseau de neurones biologique (Figure 2) est un très grand graphe de traitement parallèle.

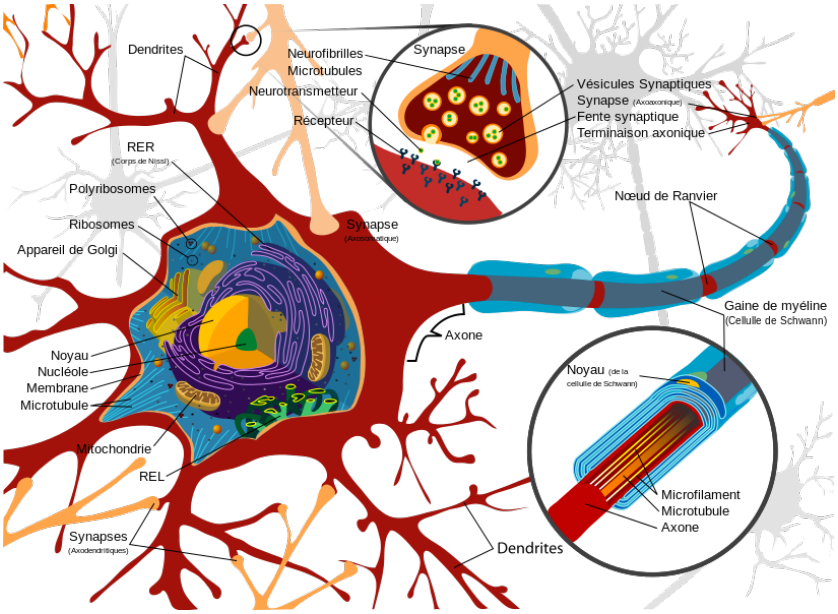


Figure 2 - le neurone biologique (source WikiPédia)

Ses neurones sont hautement interconnectés. Les flux nerveux qui le traversent, représentent sa capacité d'interaction, lié d'un côté aux senseurs qui l'alimentent, et de l'autre aux actuateurs qui interagissent avec l'environnement.

Le neurone virtuel (Figure 3) est, quant à lui, une abstraction du fonctionnement de ce neurone biologique débarrassé des contraintes biologiques (mécanismes d'alimentation, réplication...).

De fait, de nombreux modèles virtuels s'appuient sur les équations de Hodgkin-Huxley et ne cherchent pas à les compléter, mais au contraire à les simplifier, tout en conservant le maximum de pertinence biologique [5].

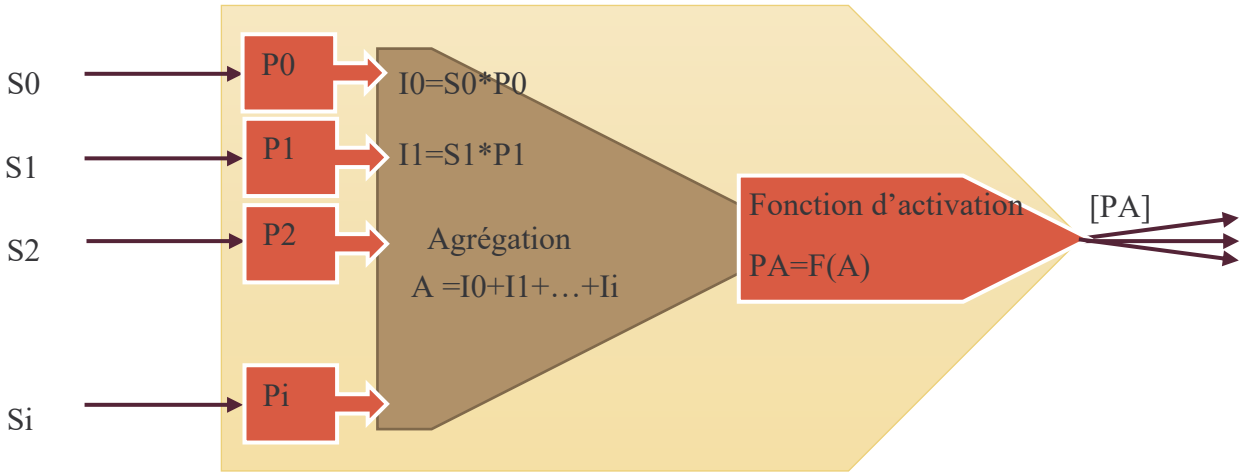


Figure 3 - Formalisation d'un neurone numérique

- Les dendrites/synapses sont donc remplacées par des connexions pondérées (P0, Pi). Les pondérations peuvent être négatives pour simuler les mécanismes d'inhibition.
- La fonction d'activation remplace le seuil de potentiel d'action.

Même s'il est difficile de déterminer lequel des deux modèles est le plus flexible, la différence majeure entre le neurone artificiel et le neurone biologique est l'encodage des flux.

- Le neurone biologique utilise un encodage par fréquence. C'est la fréquence de réception des signaux qui déclenche ou non un potentiel d'action. Cette approche est plus lente car il faut attendre un certain nombre de signaux entrant. Mais elle est moins sujette aux erreurs qu'avec une machine biologique analogique qui en côtoie en permanence.
- Le neurone numérique utilise un codage par amplitude. C'est l'amplitude/valeur du signal qui déclenche l'activation du neurone. Cette approche est plus rapide car un signal contient toutes les informations nécessaires. Elle est plus adaptée au monde numérique car elle nécessite plus de précision qu'une machine biologique ne peut fournir.

Dans tous les cas les deux sont des approximateurs universels (Turing) [6].

Il existe plusieurs topologies de réseaux de neurones. Pour n'en citer que quelques-unes,

- Feed Forward
- Réseaux dit Récurrents (ou à mémoire)
- Réseaux dits à apprentissage supervisés
    - Rétropropagation de gradiants (le plus courant) [7]
    - À convolution
- Réseaux dits à apprentissage non supervisés
    - Hebbian
    - Neuromodulation
    - Apprentissage Compétitif
    - Kohonen ou Self Organizing Map
    - Hopfield
- Et bien d'autres.

Le réseau de neurones est constitué d'une couche d'entrée connectée aux senseurs, d'une couche de sortie connectée aux actuateurs et éventuellement de couches dites cachées.

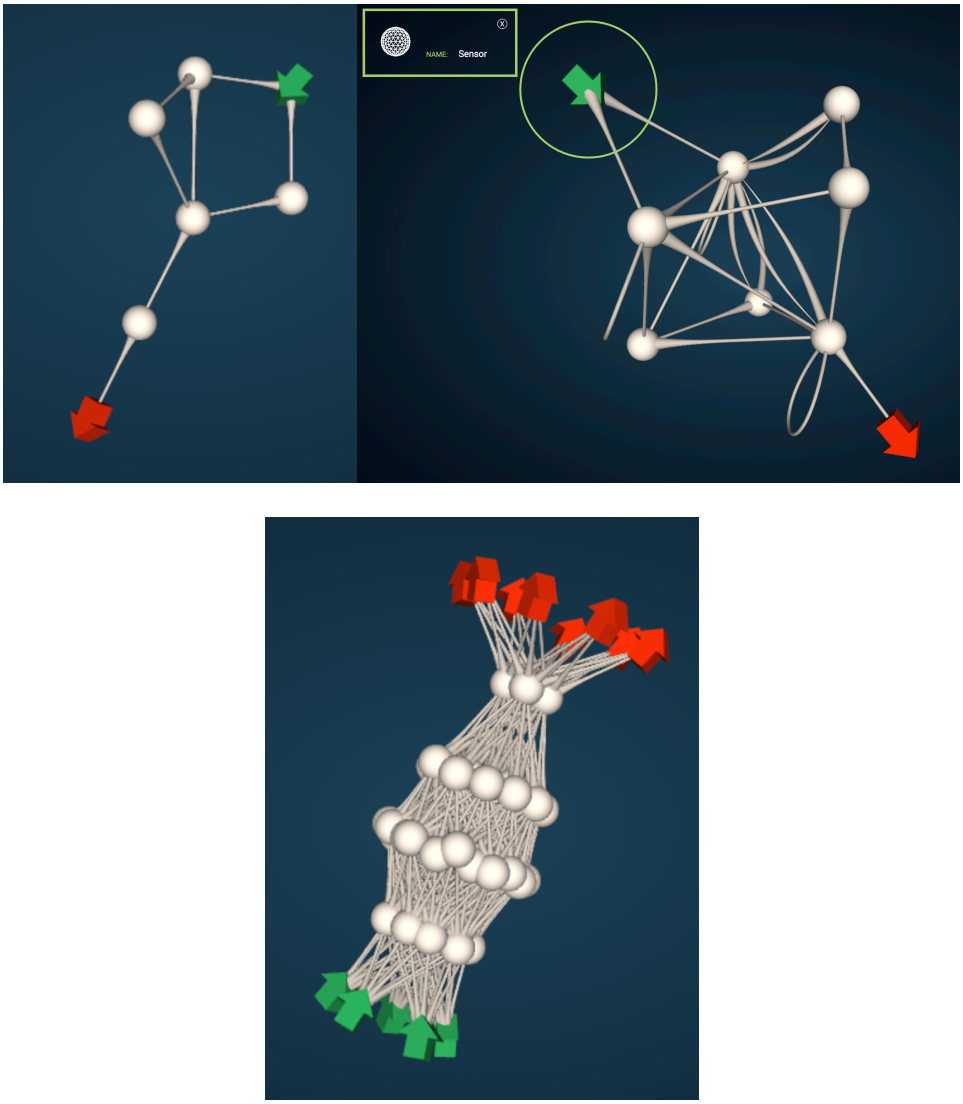


Figure 4 - Exemples de réseaux de neurones numériques (représentation 3D par Datavaloris)

La densité (nombre de neurones d'une couche) est à la discrétion du concepteur du réseau, en particulier pour les couches cachées.

Dans tous les cas, ces topologies (Figure 4) nécessitent un paramétrage initial structurant.

La densité et la profondeur des RNN créent une limitation lors de l'apprentissage appelé le *vanishing-gradient effect* (VGE).

L'algorithme d'optimisation de réseau de neurones (NN) le plus courant est basé sur l'utilisation de la descente de gradient (ou rétropropagation de gradient).

Cela implique d'abord de calculer l'erreur de prédiction faite par le modèle, puis de rétropropager cette erreur sous forme d'un gradient.

Ce gradient d'erreur est propagé vers l'arrière à travers le réseau depuis la couche de sortie jusqu'à la couche d'entrée, mettant à jour les poids pour ainsi minimiser la différence (l'erreur) entre la sortie du modèle réelle et la sortie attendue.

Les NN intègrent désormais de nombreuses couches. L'ajout de couches dites profondes augmente sa capacité, ce qui le rend capable d'approximer des fonctions plus complexes entre l'entrée et la sortie lorsqu'un grand ensemble de données d'apprentissage est fourni.

Un des problèmes avec les réseaux d'entraînement à plusieurs couches (par exemple, les réseaux de neurones profonds) est que le gradient diminue considérablement à mesure qu'il se propage vers l'arrière du réseau (vers les couches d'entrée). L'erreur devient si faible au moment où elle atteint les couches proches de l'entrée du modèle qu'elle n'a que peu d'effet. Ainsi, ce problème est connu sous différents noms : problème de la disparition du gradient ou problème de l'évanescence du gradient, ou encore problème de la dissipation du gradient nommé en anglais, le *vanishing gradient effect* [20], [21], [22], [23].

Plusieurs approches peuvent être utilisées pour réduire le VGE bien qu'aucune ne soit parfaite. Elles peuvent cependant être combinées avec l'approche PRL :

- Les fonctions d'activation, comme relu par exemple [24].
- Couches d'initialisation normalisées [25]-[26] et couches de normalisation intermédiaires [27] : elles permettent aux réseaux composés de dizaines de couches de commencer à apprendre/converger avec descente de gradient stochastique (SGD) avec rétropropagation [28].
- Des architectures spécifiques comme les réseaux de neurones résiduels (ResNet) qui tentent de diminuer l'effet de ce problème en utilisant des liens pass-through [29].
- La régularisation des réseaux de neurones profonds par le bruit : on injecte du bruit lors de la procédure d'apprentissage, ajoutant ou multipliant du bruit au sein des unités cachées des NN [30].
- La méthode d'apprentissage en cascade profonde propose une solution pour atténuer le VGE [31] en formant des réseaux profonds en cascade ou de manière ascendante couche par couche. Il réduit le VGE, mais ne s'est pas avéré meilleur que le DDL (voir définition dans le glossaire).

1.1.b ÉVOLUTION VERS LES ALGORITHMES GENETIQUES ET AU-DELA…

Le neurone, et plus globalement le cerveau, sont issus d'une évolution biologique de plusieurs milliards d'années depuis l'apparition de la vie sur terre (3,77 à 4,28 milliards d'années).

La nature a effectué des milliards de milliards de mutations afin d'aboutir à notre cerveau. Notre système nerveux central est la preuve de l'efficacité du mécanisme évolutif et de l'approche par réseau de neurones (cela ne remet pas en cause tout autre moyen existant).

Là encore il s'agit de copier la nature dans ce qu'elle a fait de mieux (à notre connaissance) et de s'en inspirer pour bénéficier des mêmes capacités évolutives.

Pour qu'un processus d'évolution se mette en place, il est nécessaire d'avoir une population d'organismes, d'avoir un environnement dans lequel elle évolue et des critères de sélection/reproduction.

Dans la nature l'évolution est basée sur un critère simple de survie adaptative et compétitive. Elle a abouti à la création de l'ADN, de l'ARN… et donc de tout un arsenal d'outils permettant la reproduction et l'adaptation des espèces.

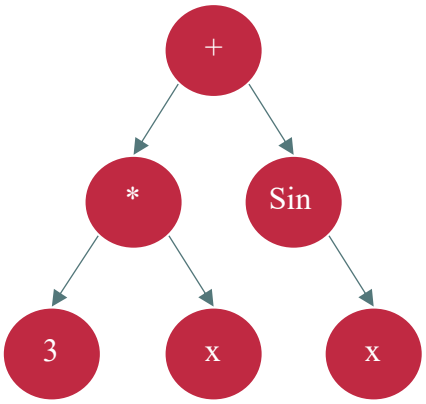
Dans le monde numérique il faut également construire un code génétique définissant le génotype d'un individu et par extension le phénotype de l'individu qui est l'expression de ses gènes.

Un algorithme génétique utilise ensuite des règles de mutations et des critères de sélection.

Une méthode couramment employée est une représentation en arbre [8] permettant facilement d'effectuer des permutations. La feuille de l'arbre représente des données et les nœuds des opérations et les racines sont les sorties.

Voici un exemple simple :

Ceci est le génotype de l'individu :



Cela donne le phénotype suivant : (3*x)+sin (x)

Figure 5 - exemple d'un génotype en arbre

On voit bien dans cet exemple (Figure 5) que la mutation peut être l'ajout d'une branche, la permutation d'un opérateur… et dans une reproduction « sexuée » la branche d'un individu peut être associée à celle d'un autre pour former un « enfant ».

Ici, il y a un lien direct entre le génotype et le phénotype, Ce lien peut être indirect à l'instar de notre ADN et de notre représentation physique. Le lien direct est plus lisible pour le chercheur mais limité dans les possibilités de l'expression du génotype (codes non exprimés tels que le code lié aux évolutions).

Il existe un autre mode de représentation qui est un affichage de type graphe.

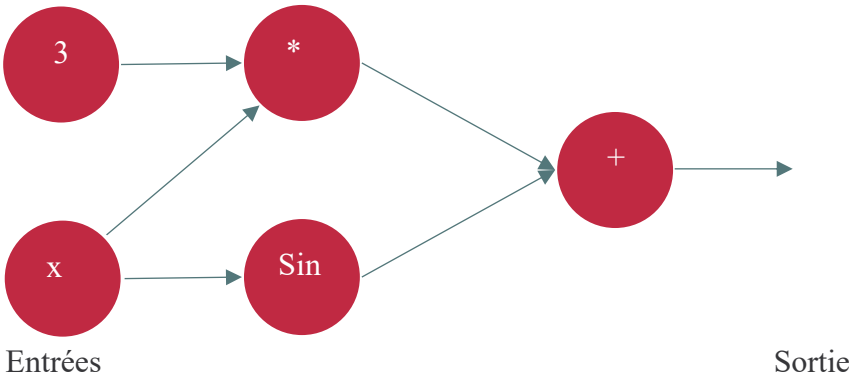


Figure 6 - exemple d'un génotype en graphe

Cette représentation (Figure 6) est plus proche de la représentation d'un réseau de neurones.

Un critère de sélection de l'individu peut être le nombre minimum d'opérations pour atteindre un même résultat.

Aujourd'hui il existe de multiples algorithmes d'évolution :

- Les algorithmes Génétiques classiques (GA) qui existent depuis 1954 et popularisés par John Holland dans son livre [9].
- Les algorithmes de programmation génétique (GP), une spécialisation des GA.
- Les stratégies évolutives dont la stratégie d'évolution elle-même évolue.
- Les programmations évolutives pour les machines à états finis (FSM).
- Les algorithmes Mémétiques (MA) ou algorithmes Hybrides (HA) où il est défini des mutations dites globales et des mutations locales.

Quel que soit le nom qu'on leur donne, leur fonctionnement reste le même pour tirer profit d'un mécanisme de sélection.

A la fin de 2017, bon nombre d'articles ont été publiés par les équipes de recherches d'Open AI et de Uber [10], [11], [12]. Une unité de recherche d'Open AI [13], a montré qu'un simple algorithme évolutif peut optimiser un réseau de neurones profond aussi bien voire mieux que les algorithmes classiques basés sur une descente de gradient (comme par exemple Adam) [3], [14], [15], avec des résultats surprenants dans certains cas [16] ...

Et tout cela parce que l'algorithme évolutif est une méthodologie basée sur des groupes de populations et qu'il est donc plus facile de paralléliser les tâches pour obtenir des résultats en heures plutôt qu'en semaines (ce que corrobore les articles publiés en référence). Ces résultats s'appliquent aussi bien en classification qu'en apprentissage renforcé.

L'objectif ultime reste la recherche de l'architecture de modèles optimum [17].

1.1.c METHODES D'APPRENTISSAGE ACTUELLES

Aujourd'hui, le mode de construction d'une topologie de réseaux de neurones se fait toujours par tâtonnement. Et les résultats probants obtenus sont généralement par la suite analysés et optimisés par des mathématiciens. Beaucoup de topologies connues (CNN, RNN…) et utilisées sont ainsi nées de façon empirique.

En l'absence de règles écrites pour la création de ces modèles, la tendance actuelle des recherches est également à l'optimisation de ce que l'on nomme les hyperparamètres. L'hyperparamètre est le nombre de couches, l'agencement de ces couches, les critères d'apprentissage... c'est à dire tous les paramètres utiles à la création d'un modèle IA. La discipline représentant la fabrication des modèles est appelée aujourd'hui ModelOps (abréviation de Model Operations) qui est un sous-ensemble du ML Ops (Machine Learning Operations). Les outils utilisés pour la fabrication des modèles sont répertoriés dans la catégorie AutoML (Automation Machine Learning). Ils sont

censés agir sur les hyperparamètres. L'objectif avec cette optimisation des hyperparamètres est d'essayer de tendre vers le modèle le plus efficace possible.

Pour rappel, Google avait, pour sa part, baptisé sa propre tentative d'automatiser les hyperparamètres AutoML, nom repris aujourd'hui pour cataloguer tous les outils de cette catégorie. De nombreux outils d'automatisation de Machine Learning du marché entrent dans cette classe (Auto-Keras, SMAC, Auto-Sklearn…).

La façon de procéder pour obtenir une topologie optimale avec ces méthodes, consiste au départ par une planification des tests à réaliser par l'expérimentateur. Ce dernier donne alors les options voulues pour les hyperparamètres à tester : par exemple, travailler sur 5 à 10 couches, chaque couche pouvant avoir de 10 à 10000 neurones par pas de 1000 ; donner ici également les différentes règles que devront suivre les poids initiaux. Ensuite l'outil d'AutoML expérimente (de manière combinatoire) automatiquement les différentes règles d'apprentissage et de topologies en faisant varier les paramètres choisis par l'expert. Une fois les expérimentations effectuées, les résultats sont comparés et l'essai avec la performance optimum, est conservé. Si aucun résultat n'est considéré comme acceptable (contraintes de production, biais comportementaux…), l'expert recommence alors depuis le début avec une nouvelle planification de tests.

En résumé, aujourd'hui, pour produire un modèle IA (pour rappel composé d'une topologie/structure et de poids associés à chaque connexion entre les neurones la composant), il n'existe que deux options : soit reprendre un modèle existant conçu par un expert et essayer de l'adapter à son propre besoin, soit tenter de construire un modèle ex-nihilo sans qu'il n'existe de règles de construction connue (juste des intuitions) en espérant que cette construction, donnera naissance au bout du compte au modèle idéal (avec ou sans optimisation des hyperparamètres).

C'est grâce à une autre possibilité, en l'occurrence le concept de neuroévolution appliquée sur des frameworks standards (voir définition dans le glossaire), que les experts peuvent concevoir de nouveaux modèles IA sans être obligés de connaître au préalable des règles de construction. La neuroévolution pousse ces derniers à se focaliser sur l'objectif à atteindre plutôt que sur la façon de structurer le modèle pour y parvenir.

Pour les trois options précitées, travailler de manière empirique, jouer sur les hyperparamètres, ou utiliser la neuroévolution, tous les modèles ainsi obtenus restent cependant des modèles statiques, en d'autres termes où la topologie n'évolue pas au cours de l'apprentissage.

Et dans tous ces cas, cette topologie fixe empêche les modèles, quand ils sont complexes, d'apprendre ou de réapprendre de manière optimale. C'est précisément à ce niveau que l'apprentissage phylogénétique pourrait apporter une solution.

1.1.d PHYLOGENETIQUE ET EMBRYOGENESE

La phylogenèse qui se définit par l'étude des relations de parenté entre êtres vivants. Elle permet de représenter l'évolution des espèces souvent sous forme de carte ou d'arbre phylogénétique.

Dans le numérique, il s'agit de conserver l'arbre phylogénétique des entités au sein du code génétique comme vu au point précédent.

L'embryogénèse s'occupe de l'apparition de caractéristiques d'un individu pendant sa phase embryonnaire. Ce qui nous intéresse dans cette thèse c'est l'organe cerveau.

La comparaison entre la phylogénèse et l'embryogénèse montre un lien entre l'évolution des espèces et la croissance d'un individu en ne considérant qu'un caractère donné [18].

La première personne à l'avoir perçu et surtout exprimé au sein d'une théorie, (décriée par certains du fait, entre autres, de la falsification de certains dessins relatifs à la vision globale de ce lien), est Ernst Haeckel.

Le secteur de la biologie évolutive du développement, cherche à formaliser ce que Haeckel avait perçu intuitivement. L'analyse des gènes de différentes espèces et l'expression de ces derniers dans les processus de développement d'un individu, permet de comprendre les processus d'activation de certains caractères, et également d'évaluer les impacts de la pression environnementale sur la sélection naturelle.

On retrouve donc le lien entre la topologie des cerveaux et l'évolution des espèces dans l'œuvre contestée de Paul Mac Lean :

Paul Mac Lean est un médecin neurochirurgien et neurobiologiste américain (1913-2007). Il a introduit la théorie du cerveau triunique selon laquelle trois phases du cerveau sont apparues au cours de l'évolution de l'espèce humaine :

1. Le cerveau reptilien (ou cerveau primitif),
2. Le cerveau limbique apparu avec les premiers mammifères,
3. Le cerveau néocortex (ou cerveau « humain »).

Mais cela n'empêche pas la partie paléontologique et évolutive de la théorie d'être justifiée : le cerveau humain est le résultat de périodes de céphalisation successives [19] (Figure 7).

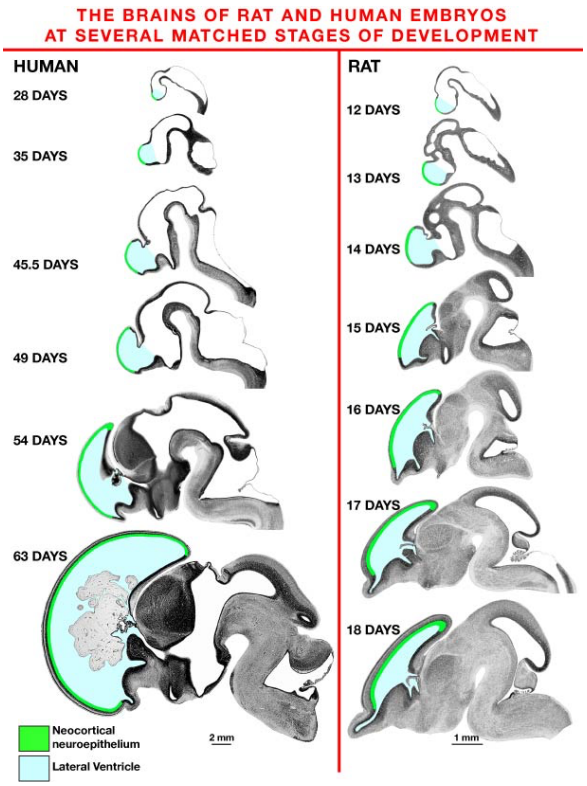


Figure 7 - Comparaison de l'embryogénèse du cerveau de 2 espèces phylogénétiquement différentes

.

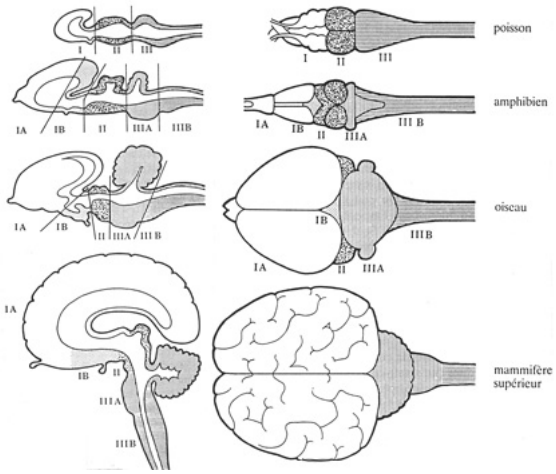


Figure 8 - Comparaison topologique de cerveaux de 4 espèces distinctes

La région IIIA (Figure 8) dans le schéma ci-dessus, montre, par exemple, l'apparition du cervelet grâce auquel les animaux supérieurs réalisent des gestes sophistiqués comme la marche. Chez l'oiseau, cette structure est tout particulièrement développée.

1.1.e RASSEMBLEMENT DES ORIENTATIONS.

Il convient de noter que l'objectif de cette thèse n'est pas de discuter ou de comparer une recherche de modèle spécifique ou une méthode neuro-évolutive comme EANT1/2 [32], CoSYnE [3], DXNN ou NEAT par rapport à d'autres méthodes déjà abordées [33], [34], [35]. Elle a pour but d'explorer l'utilisation de l'approche phylogénétique, plus précisément l'apprentissage par rétro-propagation de gradient avec des mutations préplanifiées, et donc l'entraînement des couches, l'une après l'autre, au fur et à mesure que le réseau neuronal profond se construit.

Malgré toutes les applications existantes des modèles multicouches profonds, il reste encore difficile de construire des modèles qui généralisent ou s'adaptent efficacement à certains types de problèmes et à des données complexes.

D'autres chercheurs travaillent sur de nouveaux types de topologies de modèles.

La topologie qui est testée dans [40] comprend un système algorithmique de neuromodulation qui pilote le fonctionnement du réseau de neurones pour en augmenter la capacité d'adaptation.

Les animaux sont excellents pour, en fonction de l'environnement, interagir avec un monde extérieur riche, imprévisible et en constante évolution, une propriété qui manque actuellement aux systèmes artificiels intelligents. Cette propriété d'adaptation repose sur la neuromodulation cellulaire, le mécanisme biologique qui contrôle de manière dynamique les propriétés intrinsèques des neurones ainsi que leur réponse aux stimuli extérieurs et ce, de manière adaptée au contexte.

Cette nouvelle approche est prometteuse mais elle n'enlève en rien l'intérêt de l'approche PRL. Elle serait même complémentaire à cette dernière.

1.2 CADRE EXPERIMENTAL

Voici deux exemples tirés de la littérature en lien avec l'objectif de maitriser la difficulté d'apprentissage des modèles de réseaux de neurones profond impactés par le VGE.

1.2.a DEEP CASCADE LEARNING (2018)

Dans le papier [31], les auteurs décrivent une méthode de construction de réseau de neurones dits en cascade.

Les couches sont ajoutées les unes après les autres avec utilisation, à chaque itération, d'une couche d'« output » pour assurer l'apprentissage de la couche ajoutée (Figure 9).

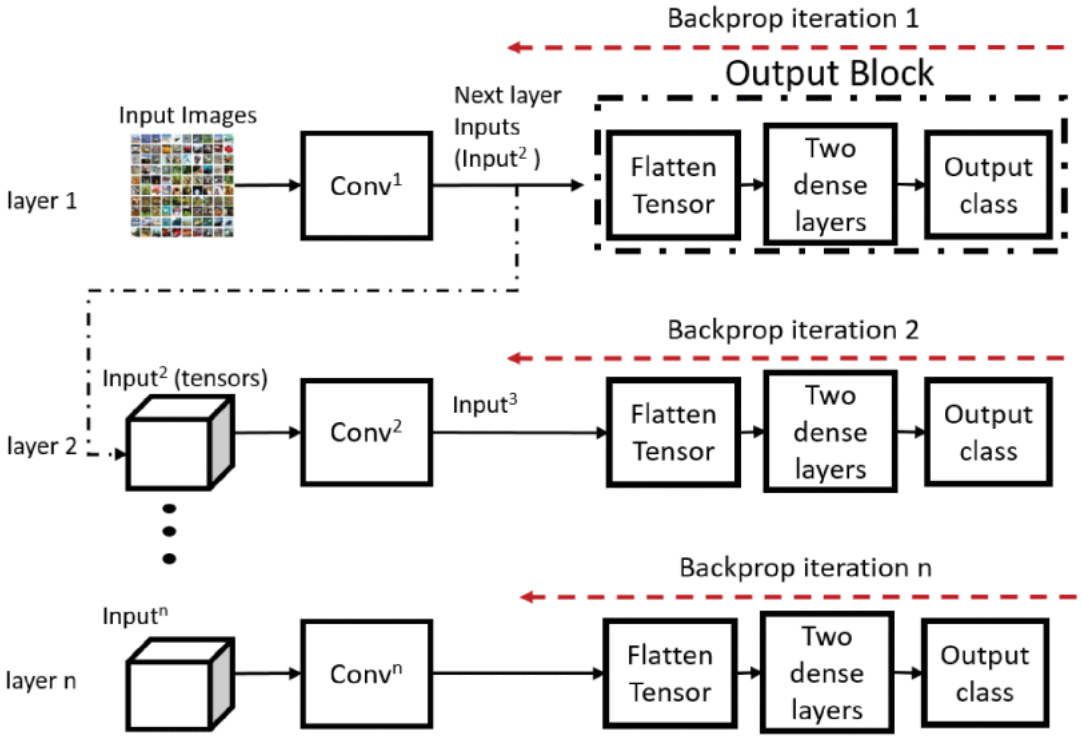


Figure 9 - Deep Cascade Learning :

Training proceeds layer by layer; at each stage using convolutional layer outputs as inputs to train the next layer. The features are flattened before feeding them to the classification stage. In contrast with the cascade correlation algorithm, the output block is discarded at the end of the iteration (see Algorithm 1), and typically, it contains a set of fully connected layers with nonlinearities and dropout.

Une telle formation de réseaux profonds en cascade contourne directement le problème bien connu du « vanishing gradient effect » en garantissant que la sortie est toujours adjacente à la couche en cours d'entrainement.

Ils démontrent ainsi que les couches dites hautes (les premières) alors qu'elles sont mal entrainées avec la méthode classique (DDL), le sont correctement avec cette méthode.

Cette approche est intéressante car elle indique que l'approche phylogénétique sera efficace.

Leur approche bien que séduisante induit cependant un biais important, elle limite la topologie du réseau de neurones et impose un mode de construction linéaire. Il est donc difficile d'envisager cette méthode avec des topologies comportant de multiples branches (ce qui est de plus en plus le cas dans les structurations récentes telles que les topologies de type « Transformers »).

De plus les résultats (Figure 10) ne démontrent pas une meilleure efficacité que celle des méthodes d'entrainement classiques.

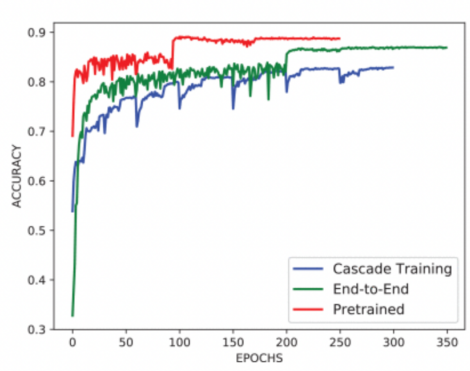


Figure 10 - DeepCascade learning result comparison

La méthode d'apprentissage DeepCascade propose une solution pour atténuer le VGE [31] en formant des réseaux profonds en cascade ou de manière ascendante couche par couche. Il réduit le VGE, mais ne s'est pas avéré meilleur que le DDL en termes d'efficacité.

1.2.b RESEAUX RESIDUELS (2016)

Le papier [29] décrit une méthode de construction de réseaux de neurones qui a émergé il y a déjà quelques années : Le ResNet.

Ce papier montre que plus le réseau de neurones contient de couches (donc profond), plus il a du mal à apprendre. Les auteurs ne discutent pas du calcul de la densité informationnelle de chaque couche mais se penchent sur le comportement de réseaux de profondeur différentes.

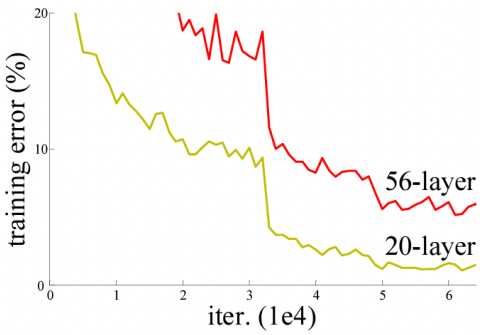


Figure 11. Impact de la profondeur du réseau sur l'apprentissage

Il y est constaté que les erreurs sont plus importantes au sein des réseaux plus profonds (Figure 11).

Une solution potentielle y est proposée : la modification de la structure de ces réseaux linéaires en utilisant des blocs résiduels. Les blocs résiduels consistent à simplement rajouter un lien entre des couches distantes.

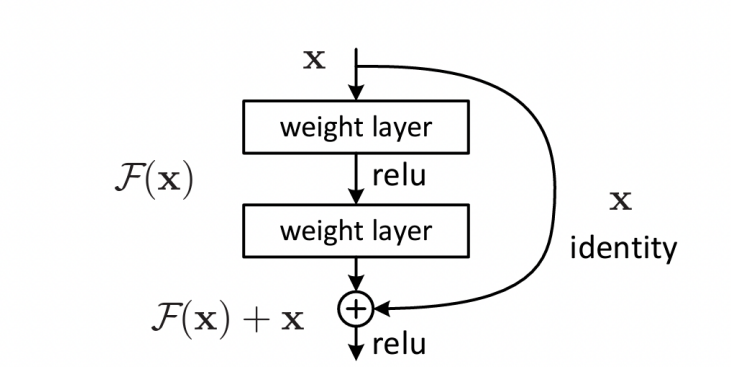


Figure 12. Bloc de construction d'un ResNet

L'article démontre que ce « Bypass » (Figure 12) permet au réseau profond de mieux apprendre.

Table 1. ResNet résultat de l'erreur

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Plus le réseau est profond plus les blocs résiduels sont efficaces, limitant le VGE (Table 1).

Cette approche bien qu'intéressante impose, une fois de plus, une topologie au réseau et là, seule une catégorie de réseaux de neurones peut convenir.

1.3 SYNTHESE DES LIMITES DE L'ETAT DE L'ART

La recherche bibliographique sur l'utilisation de la phylogénétique montre qu'il n'existe pas à ce jour de recherche sur l'application de la phylogénétique aux réseaux de neurones numériques, ni de planification de mutation topologique sur un réseau de neurones numérique lors de l'apprentissage.

On trouve en revanche beaucoup d'articles sur l'utilisation de l'IA dans la classification phylogénétique des espèces mais qui ne sont bien évidement pas le sujet de cette thèse et seront donc ici ignorés.

Les études connexes à cette thèse concernent le « Vanishing Gradiant Effect ». Certaines méthodes décrites semblent s'apparenter à ce qui est étudié dans cette thèse mais elles sont pourtant bien différentes.

Ces recherches bibliographiques ont été utiles pour rationaliser et comprendre en quoi la méthode phylogénétique est innovante et sur quoi, elle peut être efficace.

2. PROBLEMES ET MÉTHODOLOGIE PROPOSÉE

Dans ce chapitre seront abordés les problématiques rencontrées actuellement et la méthodologie mise en œuvre pour répondre à la question de recherche.

2.1 PROBLEMATIQUES ACTUELLES

Les enjeux majeurs de l'intelligence artificielle (IA) contemporaine sont réunis ici pour mieux comprendre le contexte et les défis liés à la création de réseaux neuronaux toujours plus performants et aptes à résoudre des problèmes de plus en plus complexes.

La revue de littérature révèle que tous les modèles d'IA actuels, quelle que soit leur méthode d'élaboration, sont statiques, c'est-à-dire que leur structure ne se modifie pas pendant l'apprentissage. Cette rigidité limite leur capacité d'apprentissage, en particulier pour les modèles de grande taille et complexes, en partie à cause de la problématique de l'évanescence du gradient d'erreur (le VGE). L'apprentissage phylogénétique pourrait offrir une solution à ce problème.

L'un des défis cruciaux de l'IA est de déterminer la structure la plus optimale possible en fonction de l'objectif poursuivi. Il s'agit de trouver un équilibre entre divers facteurs tels que la rapidité de calcul, le coût en termes de temps et de ressources humaines, la taille du modèle et sa consommation énergétique pour résoudre un problème spécifique.

Au départ, l'efficacité était le seul critère de succès pour résoudre un problème. Cependant, ce point de vue est devenu problématique quand la capacité d'abstraction est devenue un critère de performance essentiel pour l'IA et que la consommation de ressources, qui ne sont pas illimitées, a été prise en compte.

Avec l'évolution des besoins, les modèles d'IA sont devenus plus grands, rendant l'optimisation essentielle. Les frameworks actuels imposent un traitement matriciel, ce qui conduit à des optimisations mathématiques qui, dans la plupart des cas, limitent les capacités d'apprentissage des modèles, comme en témoigne le problème du VGE.

Pour remédier à ces limitations, la pratique courante consiste à modifier des modèles existants, comme Bert et les versions de GPT, qui sont surdimensionnés et pré-entraînés, pour les spécialiser dans un domaine d'application donné. Toutefois, malgré leur efficacité apparente, l'utilisation de ces modèles est limitée en raison de leur coût énergétique, principalement dû à leur taille, une conclusion qui avait déjà été établie dans les travaux de Bender et al, [4].

Un autre défi est la dépendance des résultats aux poids initiaux du modèle. Plus un modèle est grand, plus il y a de poids à déterminer au départ, ce qui rend difficile l'obtention d'un apprentissage efficace, surtout en ce qui concerne la détermination des poids initiaux nécessaires à l'apprentissage. Seules de grandes entreprises sont capables de relever ce défi aujourd'hui.

La taille n'est pas le seul problème. À mesure que les modèles deviennent plus complexes, avec des topologies comportant de nombreuses couches, leur capacité d'apprentissage diminue en raison du VGE. Il existe certes des techniques pour atténuer ce phénomène, comme l'utilisation de fonctions d'activation spécifiques ou la structuration des couches pour connecter directement les couches initiales et finales.

Un autre obstacle est l'utilisation de structures de couche capables de résoudre des problèmes plus complexes, comme la conception de boucles de mémoire. Les couches LSTM, par exemple, ne répondent que partiellement à ce besoin et augmentent la complexité des modèles.

Ces solutions techniques et mathématiques ne résolvent pas fondamentalement les difficultés d'apprentissage inhérentes à ces structures complexes. La difficulté est exacerbée par l'absence de règles formelles pour construire des architectures d'IA, constituant un obstacle majeur à la création de modèles complexes.

Contrairement au domaine numérique, le monde biologique présente un exemple de réseau neuronal complexe et efficace : le cerveau humain. Malgré sa complexité, le cerveau ne semble pas souffrir des mêmes problèmes que les IA en termes d'évolution de taille ou de complexité.

2.2 QUESTION DE DEPART ET OBJECTIF

Pour résoudre les problèmes posés par les réseaux neuronaux complexes, pourquoi ne pas utiliser un mécanisme d'apprentissage copiant une fois de plus la nature ?

La nature a programmé l'évolution topologique du cerveau biologique dans l'ADN des espèces facilitant, entre autres, l'apprentissage. La question est donc de savoir si copier ce mécanisme naturel aurait un sens ou une utilité en l'intégrant sur les cerveaux numériques ? Si l'on pouvait coder l'historique de l'évolution topologique qui nous aura permis d'obtenir le meilleur des modèles dans un contexte donné et l'intégrer dans l'apprentissage d'un autre modèle en gardant ou pas, le même contexte ?

Cette approche baptisée Phylogénétique Replay Learning, permettrait alors de réduire l'importance des paramètres initiaux sur le résultat final de l'apprentissage ou encore d'augmenter la capacité d'apprentissage des modèles en réduisant le vanishing gradiant effect.

L'objectif est de montrer que l'apprentissage itératif et planifié d'un réseau de neurones numérique issu de mutations successives est plus performant que l'apprentissage du même réseau de neurones (final ou pris dans sa version optimale) gardant sa topologie durant l'apprentissage.

2.3 METHODOLOGIE

Elle consiste en

- La comparaison de la méthode d'apprentissage classique (dite DDL) avec la méthode Phylogenetic Replay Learning (dite PRL - voir définition dans le glossaire) conçue pour résoudre le problème d'apprentissage des modèles complexes.
- De réitérer plusieurs fois l'action pour que la comparaison soit statistiquement représentative.
- De mettre en place des métriques telles l'efficacité et l'entropie des modèles obtenus, qui seront utilisées pour effectuer la comparaison.

Cette méthodologie se décline opérationnellement en 3 phases plus une phase de prérequis dite Phase 0 indispensable pour obtenir l'historique des mutations topologiques lors de l'évolution d'un modèle initial vers un modèle Champion.

Phase 0 :

En effet, au lieu d'utiliser uniquement un processus d'apprentissage direct sur une topologie fixe (méthode dite DDL) comme le font les experts actuellement, la phase 0 permettra d'enrichir le processus avec des modifications planifiées à appliquer sur la topologie du modèle initial en procédant par étape lors de l'apprentissage de ce modèle initial pour obtenir un modèle Champion (méthode PRL).

La phase 0 correspond donc à la définition du chemin phylogénétique du modèle d'expérimentation. Pour créer ce chemin, on applique un algorithme génétique de neuroévolution sur le modèle initial choisi. On passe d'une étape à une autre, en ne mémorisant que les modifications de chaque modèle intermédiaire, M_n. Pendant l'élaboration du modèle final champion M_c, un chemin phylogénétique est donc enregistré. A la fin, de l'étape 0 on obtient bien un modèle Champion M_c auquel est rattaché un chemin évolutif qui est le chemin phylogénétique (sous la forme d'un code génétique numérique stocké au sein d'un fichier) ainsi que sa topologie de départ M_0.

Il est à noter que l'efficacité de la neuroévolution (ici utilisée en phase 0) n'est pas étudiée dans cette thèse, seule l'efficacité de l'approche PRL y est évaluée.

Phase 1 :

Les modèles Champions obtenus résultant des expérimentations avec la méthode DDL et celle PRL sont comparés.

Phase 2 :

Capacité de transférabilité de la Méthode PRL

La phase 2 soit la transférabilité consiste à l'issue de la phase 0, à partir du modèle « Champion » M_c et de son chemin phylogénétique, de rejouer DDL et PRL dans un champ d'application IA différent (soit avec un jeu de données différent que celui utilisé en phase 1) afin d'évaluer la capacité de transposition du mécanisme d'apprentissage phylogénétique tout en continuant à le comparer avec ce que l'on aurait obtenu avec la méthode DDL dans ce nouveau champ d'application.

Reproductibilité :

Capacité de reproductibilité de la Méthode PRL

Pour tester la reproductibilité, les phases de 0 à 2 seront recommencées mais cette fois en partant d'un nouveau modèle initial M'_0 doté également d'un nouveau jeu de données.

Pour les phases 1, 2 et la reproductibilité, le framework d'apprentissage Keras de Google sera utilisé pour effectuer des expérimentations en environnement contrôlé. L'apprentissage des modèles est effectué par rétropropagation de gradients. Un mécanisme d'arrêt d'apprentissage automatique est mis en place pour qu'il s'arrête automatiquement si le modèle n'apprend plus après plusieurs epoch (voir définition dans le glossaire) successifs.

Comme il n'y a pas d'algorithme de neuroévolution au sein de ce framework, la solution Raise de la société DataValoris, est utilisée en complément pour fournir ce service. A noter que cet algorithme ne sera utilisé que dans la phase 0 de la méthodologie.

Précisons également que l'algorithme phylogénétique (PRL) a été spécifiquement développé pour la thèse pour venir en complément de la solution Raise afin de procéder à l'enregistrement des chemins phylogénétiques (mutations successives) et également de pouvoir les rejouer (comme un lecteur/enregistreur).

Enfin le champ d'application sur lequel cette approche sera testée est dédié à la reconnaissance d'images. Il s'agira alors de comparer les résultats des modèles possédant une topologie fixe et ayant appris directement (méthode DDL) avec ceux possédant une topologie variable et ayant appris de manière évolutive (méthode PRL).

Le socle logiciel utilisé pour les expériences récupérera les métriques permettant de comparer les méthodes. Les critères de comparaison comprendront le score d'apprentissage sur les données de validation, l'entropie de Shannon, le nombre d'epochs nécessaire à l'apprentissage et également les écarts type issus des différents processus d'apprentissage des réseaux de neurones.

Des jeux de données publics ont été utilisés.

Nomenclature utilisée pour nommer les modèles :

$M^{t\,m}{}_{e}$ où ^{t} est le numéro de l'expérience, ^{m} la méthode concernée et $_{e}$ l'étape considérée.

$M^{1.2d}{}_{c}$ représente le Modèle champion ($_{c}$) de l'expérience 1.2 ($^{1.2}$) avec la méthode DDL (^{d}).

$M^{1}{}_{0}$ représente le Modèle initial ou étape 0 ($_{0}$) de l'expérience 1 (^{1}) de la phase 0 car aucune méthode n'est spécifiée.

2.4 RATIONALISATION DES HYPOTHESES

Comparer l'apprentissage itératif et planifié d'un réseau de neurones issu de mutations successives à l'apprentissage du même réseau de neurones (pris dans sa version optimale) en gardant fixe sa topologie durant l'apprentissage.

Les jeux de données utilisés seront publics, ne seront pas augmentés et les mêmes pour toutes les expérimentations.

La méthode d'apprentissage utilisée pour obtenir les résultats sera la même (rétropropagation de gradiants).

La méthode de calcul d'efficacité (score) calculée sera la même et basée sur la val_acc (*validation accuracy*) fournie par le framework Keras.

Les métriques des modèles intermédiaires seront relevées à chaque étape.

Chaque champion sera associé à son arbre phylogénétique avec les mutations que son modèle initial a subies.

La comparaison entre expérimentation est faite sur une même base de corpus, de hardware.

L'apprentissage direct (DDL) sera plutôt maximisé pour que les résultats soient le moins discutables et éviter ainsi de favoriser la méthode phylogénétique (PRL).

Le nombre de tests pour chaque expérimentation principale sera supérieur ou égale à 50 pour que les résultats soient statistiquement représentatifs (certains résultats secondaires non parlant en termes de résultats sont utilisés juste pour l'orientation, 10 tests au moins).

2.5 PROTOCOLE EXPERIMENTAL

Pour chaque expérimentation avec un jeu de données choisi, le protocole employé consiste si le chemin phylogénétique n'existe pas de le produire (Phase 0), de comparer les métriques de la méthode PRL et DDL (Phase 1) puis de comparer les métriques des deux méthodes sur un autre jeu de données (Phase 2).

2.5.a PHASE 0 : CONSTRUCTION DU MODELE CHAMPION ET DU CHEMIN PHYLOGENETIQUE

- Construction automatique d'un modèle « champion » M_c basé sur la neuroévolution pour l'évolution et le DL pour l'apprentissage sur le corpus de données choisi.
- Stockage de son arbre phylogénétique
- Stockage des informations statistiques des modèles ainsi générés
- Définition de la base de comparaison.

Lors de la construction du chemin phylogénétique, le processus neuroévolutif effectue une sélection (voir définition dans le glossaire) basée sur le score généré par l'algorithme d'apprentissage. Le score utilisé comme fitness est dans notre cas la précision du test du modèle M_n. Le système est réglé de telle sorte que la vitesse d'apprentissage diminue lorsque le score ne s'améliore pas pendant 3 évaluations consécutives (Patience - voir définition dans le glossaire). Pour chaque cycle (voir définition dans le glossaire) 10 Mutants (voir définition dans le glossaire) sont formés, puis leurs scores sont comparés aux Modèles M du Hall of Fame (voir définition dans le glossaire), HOF. Si un score d'un modèle M_{n+1} enfant/mutant au sein de la génération (voir définition dans le glossaire) actuelle est supérieur à celui d'un modèle M_n au sein du HOF qui a les mêmes caractéristiques topologiques, le modèle mutant remplace le modèle au sein du HOF. Si le modèle mutant a un score le plus élevé et a une topologie non présente dans le HOF, le modèle avec le score le plus faible dans le HOF est supprimé et le nouveau modèle est ajouté à sa place.

PRL nécessite l'existence d'un chemin phylogénétique du modèle qui doit être formé. La première phase (phase 0) vise à construire le modèle Champion M_c tout en enregistrant son parcours phylogénétique (mutations qui ont été appliquées séquentiellement pour le générer). La neuroévolution est utilisée pour accomplir cela (Figure 13).

La neuroévolution génère un chemin phylogénétique (Figure 14) de la topologie du modèle le plus performant appelé « champion » M_c. Dans cette figure théorique, le champion $M_c = M_3$ a 3 ancêtres. La figure montre également quelles mutations topologiques ont été appliquées pour passer d'un modèle à l'autre. L'architecture neuronale est construite par le processus évolutif (qui peut être des CNN, des couches denses, une structure de type Resnet, etc.).

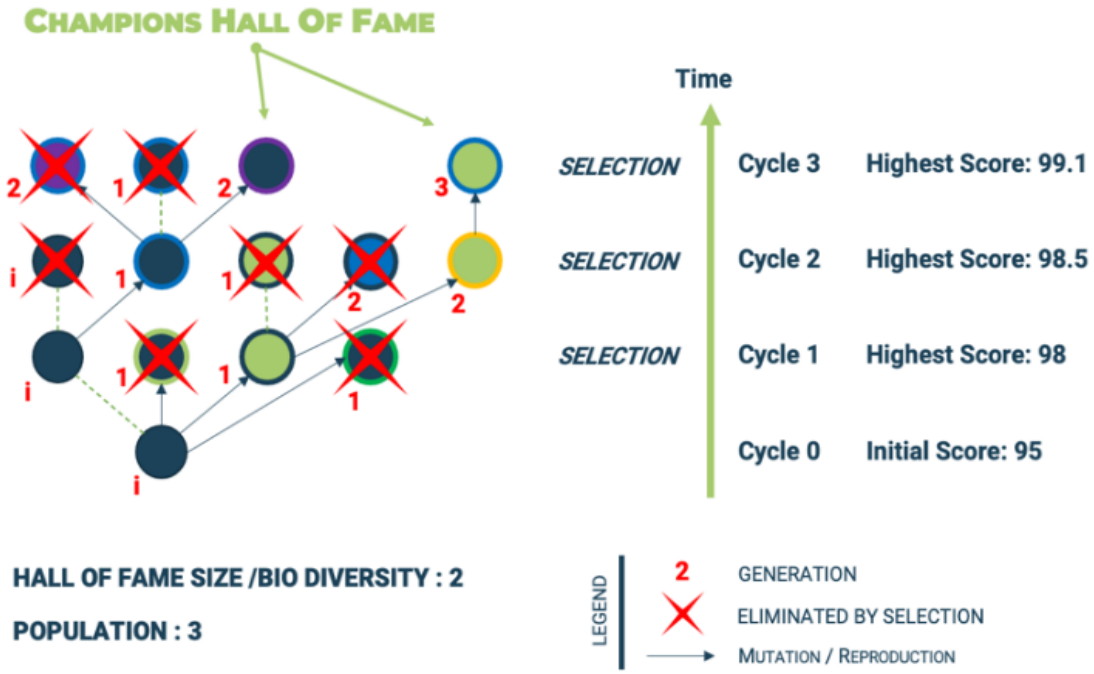


Figure 13. Exemple du processus de sélection (NeuroEvolution) [41]

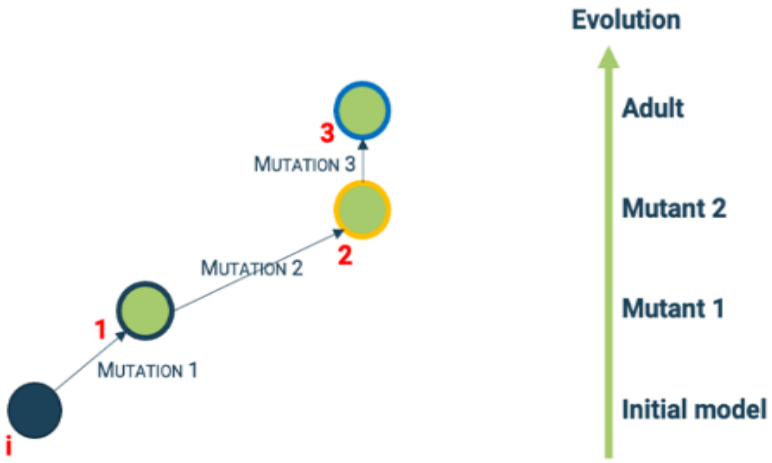


Figure 14. Chemin Phylogénétique du champion.[41]

Une autre alternative possible serait, pour pallier le manque d'historique des modèles que l'on trouve aujourd'hui dans la littérature, une approche de construction « artificielle » du chemin évolutif. Elle pourrait peut-être être utilisée [31], et tout modèle profond serait reconstruit couche après couche en utilisant soit une couche de sortie temporaire créée spécifiquement (de la longueur de couche de sortie qui sera ajoutée ensuite) jusqu'à la dernière couche [37] soit en ré-attachant ponctuellement la couche de sortie après chaque couche itérativement ajoutée afin de pouvoir effectuer les apprentissages à chaque étape de reconstruction.

Ces approches de reconstruction du chemin évolutif sont limitées à des topologies simples et principalement séquentielles. De telles limitations sont levées dès qu'il s'agit de construire un chemin basé sur la neuroévolution ou si l'historique des modifications est consigné par les experts ayant constitué le modèle final.

2.5.b PHASE 1 : COMPARAISON DES TESTS D'EFFICACITE DDL ET PRL

A ce stade, le modèle champion et un modèle initial avec le chemin phylogénétique sont définis. Les tests peuvent démarrer. Ils consistent en :

- Apprentissages multiples, afin d'avoir une vision ensembliste sur les résultats, initié sur des poids initiaux stochastiques du modèle champion M_c sans utilisation des mécanismes évolutifs. C'est un apprentissage direct (DDL) du modèle topologique adulte M_c comme les data-scientists le font traditionnellement dans leur framework.
- Apprentissages multiples basés sur des poids initiaux stochastiques avec utilisation de l'arbre phylogénétique du champion M_c. Chaque test est donc une succession d'apprentissage par rétropropagation de gradients et de mutations programmées par l'arbre phylogénétique depuis le modèle topologique initial M_0 jusqu'à la topologie du

modèle final dit champion M_c avec transferts de poids. Cet ensemble sera appelé « Apprentissage Phylogénétique ou PRL ».

Apprentissage avec la Méthode PRL :

Le chemin phylogénétique qui mène au modèle champion M_c est défini, il peut être rejoué à volonté depuis le modèle initial M_0 jusqu'au modèle champion (Figure 15) M_c.

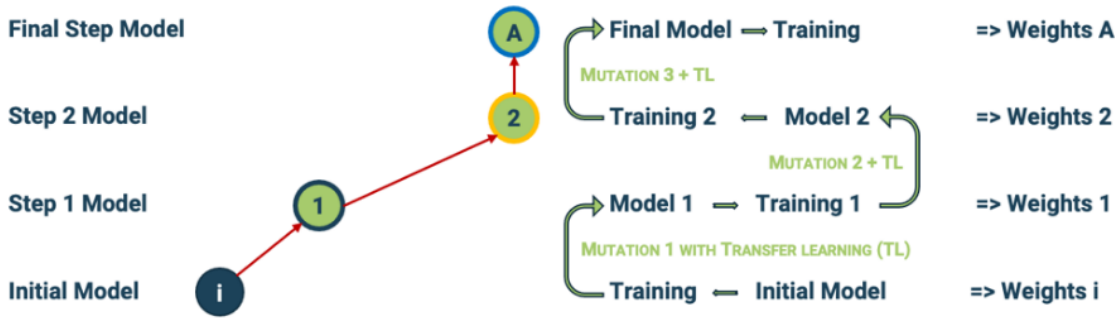


Figure 15. The phylogenetic learning path from initial model to the final one.[41]

Lors de la relecture du chemin phylogénétique, il faut :

1. Initialiser le modèle initial avec un nouvel ensemble de poids synaptiques aléatoires
2. Générer les poids aléatoires pour les couches impactées par les mutations.

Cela engendre ensuite le modèle final M'_c avec la même topologie que le modèle champion M_c, mais avec son propre ensemble de paramètres. C'est un moyen d'évaluer statistiquement pendant le test l'impact des poids aléatoires initiaux et de l'apprentissage en général.

2.5.c Phase 2 : Comparaison des tests de transferabilite ou de transposition

Il s'agit de faire des tests d'apprentissages PRL du champion M_c existant en le transposant sur un autre jeu de données de données et d'en comparer les résultats avec les tests d'apprentissage direct DDL du champion M_c sur ce même nouveau jeu de données. Ce jeu de données respecte les mêmes caractéristiques que l'original (taille de l'input, nombre de classe de classification) ayant servi a produire le champion mais avec des classes d'objets différentes.

Cette phase générera de nouveaux modèles finaux champion M''_c avec la même structure topologique du modèle M_c

2.5.d TESTER LA REPRODUCTIBILITE

Il s'agit dans ce cas de vérifier que la méthode PRL est efficace versus DDL si on la réutilise en rejouant les phases 0, 1 et 2 en utilisant sur un autre champ d'application avec un autre jeu de données et un autre modèle.

Ici sont utilisés de nouveaux modèles initiaux et champions, respectivement N_0 et N_c.

Des tests de transférabilité seront aussi effectués sur ces nouveaux modèles pour obtenir des $N"_c$.

2.6 TECHNIQUES, BIAIS, ECHANTILLON, TYPES D'ANALYSES

2.6.a Environnement de travail

Voici les Framework d'apprentissage qui seront utilisés pour toutes les expérimentations de cette thèse :

Type A : Keras au-dessus de plaidML (pour le support de GPU AMD) sur Python 3.7.

Intel Core i9 8 cœurs
64 Go 2667 MHz DDR4
AMD Radeon Pro 5500M VRAM (totale) : 8 Go, Unités de calcul : 22, Performances de calcul en demi-précision maximales : 9.26 TFLOPs

Type B : Keras au-dessus de TensorFlow (pour le support de GPU NVIDIA au Genci ou Cloud IBM).

Framework Neuroévolutif : Neural Network Topology Optimizer v3 de la plateforme Raise de DataValoris

Un serveur avec des cartes Nvidia Tesla v100 GPU.

Soit sur un accès aux ressources HPC/AI de l'IDRIS sous allocation 2021- AD011012674 accordé par le GENCI. Soit sur serveur Cloud IBM

Pour les tests, même si deux types de hardwares ont été utilisés selon les expérimentions, les mêmes environnements sont utilisés pour les comparaisons d'un même modèle M_c, M'_c et $M"_c$ sur une expérimentation donnée.

2.6.b ÉCHANTILLON ET JEUX DE DONNEES

Les trois mêmes datasets seront utilisés pour les besoins des tests principaux.

Deux autres datasets ont été utilisés pour valider le résultat sur d'autres champs de recherche en test de reproductibilité.

- *JEU DE DONNEES MNIST*

Corpus de 50,000 32x32 images de chiffres en niveau de gris d'entrainement, labellisées en 10 catégories (0,1,2,3,4,5,6,7,8,9), et 10,000 images de test.

```
keras.datasets.mnist.load_data()
```

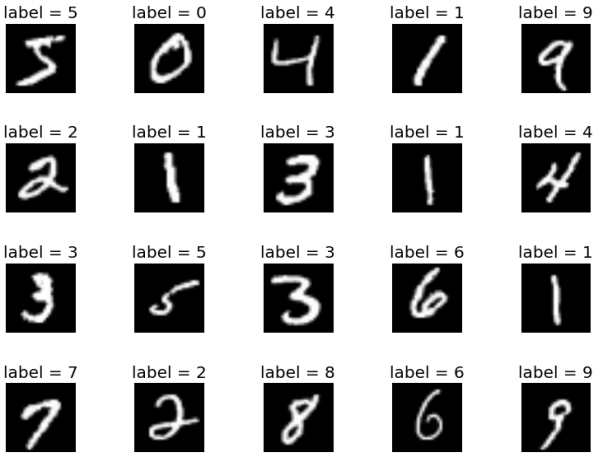


Figure 16 représentation du dataset MNIST

- *JEU DE DONNEES MNIST FASHION*

Les caractéristiques de ce jeu de données sont un volume de 50,000 images en 32x32 pixels en niveau de gris, labellisées en 10 catégories de vêtements (*T-shirt, Pantalon, Pullover, Robe, Manteau, Sandal, Shirt, Sneaker, bag, boots*) pour l'entrainement, et de 10,000 images de test.

```
keras.datasets.fashion_mnist.load_data()
```

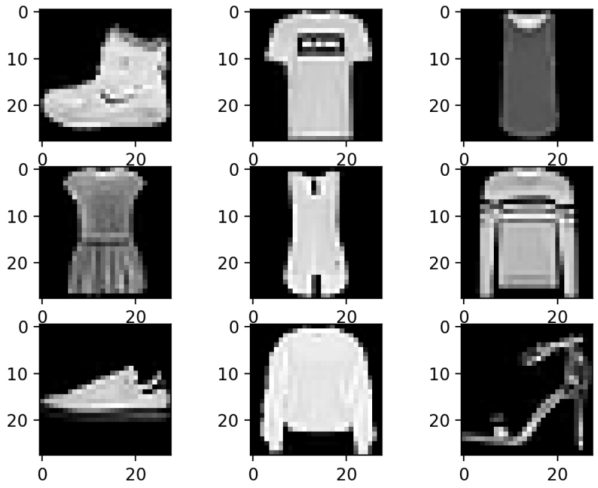


Figure 17 représentation du dataset MNIST Fashion

- *JEU DE DONNEES CIFAR 10 « GREY » BASE SUR LE CIFAR10 MODIFIE*

Corpus initial de 50,000 images en couleur 32x32 pixel d'entrainement, labellisées en 10 catégories, et 10,000 images de test.

```
keras.datasets.cifar10.load_data()
```

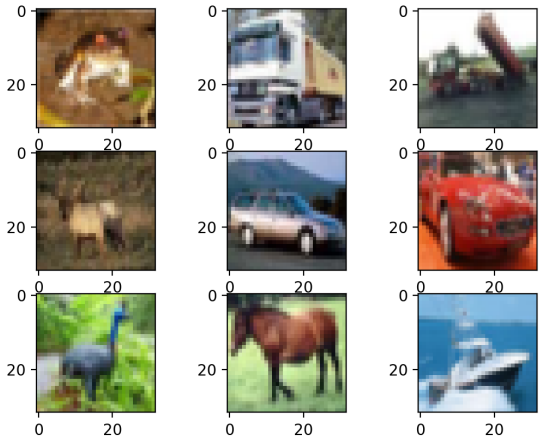


Figure 18 représentation du dataset CIFAR10

Le traitement effectué sur ce jeu de données pour passer de la couleur aux Niveaux de gris :

```
train_data_downscaled = np.zeros((50000, 28, 28))
test_data_downscaled = np.zeros((10000, 28, 28))
for i, img in enumerate(X_train):
        small_img = cv2.resize(img, dsize=(28, 28), interpolation=cv2.INTER_CUBIC)
        train_data_downscaled[i] = cv2.cvtColor(small_img, cv2.COLOR_BGR2GRAY)
X_train=train_data_downscaled
for i, img in enumerate(X_test):
           small_img = cv2.resize(img, dsize=(28, 28), interpolation=cv2.INTER_CUBIC)
           test_data_downscaled[i] = cv2.cvtColor(small_img, cv2.COLOR_BGR2GRAY)
X_test=test_data_downscaled
# END OF CIFAR10 RESHAPE
```

L'objectif de cette correction est double.

1/ de complexifier la tâche d'apprentissage du réseau de neurones. En effet la couleur est un élément important dans la reconnaissance de ces objets.

2/ de faciliter le transfert phylogénétique en conservant les input et output entre les 3 jeux de données.

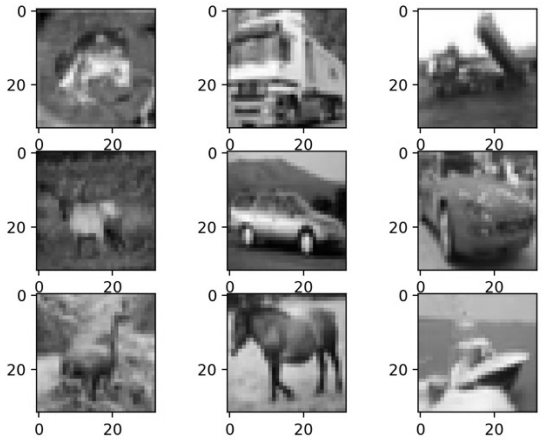


Figure 19 représentation du dataset CIFAR10 désaturé

- *JEU DE DONNEES TINY IMAGENET*

Tiny ImageNet est un sous-ensemble du jeu de données du célèbre ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

L'ensemble de données contient 100 000 images de 200 classes (500 images pour chaque classe) avec une résolution de 64 × 64 pixels en couleur. Chaque classe contient 400 images d'entraînement, 50 images de validation et 50 images de test

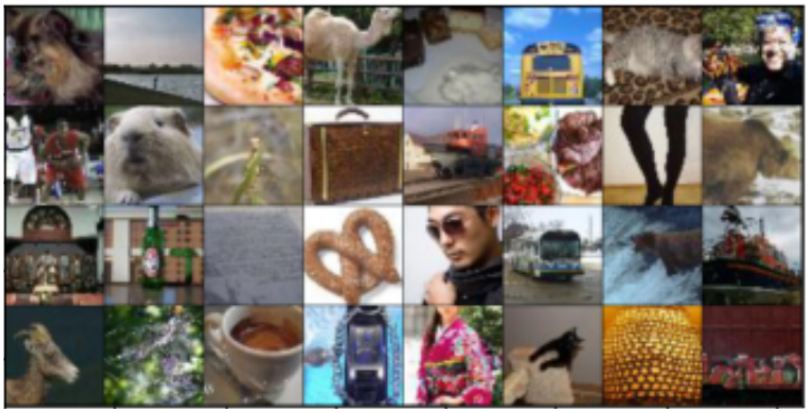


Figure 20 représentation du dataset Tiny ImageNet

- *SIMULATEUR GYM (CARTPOLE)*

Un CartPole est un terrain de jeu simulé simple fourni par OpenAI pour entraîner et tester des algorithmes d'apprentissage par renforcement.

Le projet CartPole est de faire apprendre un modèle de réseau de neurones sur des informations issues d'un simulateur par renforcement. Ici le simulateur GYM a été utilisé dans le contexte du CartPole où le but est de faire tenir le bâton le plus longtemps malgré les mouvements aléatoires du chariot.

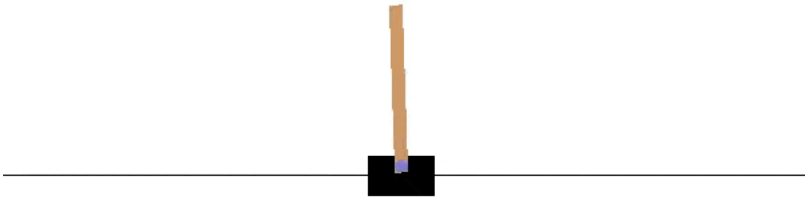


Figure 21. Cartpole

- *TRAITEMENTS EFFECTUES SUR LES JEUX DE DONNEES MNIST, FASHION MNIST ET CIFAR :*

Aucune augmentation de données n'est utilisée et la répartition Train/Test est celle fournie en standard du site source (Keras).

Normalisation des données

```
# transformation de l'output en 10 colones chacune représentant une réponse 1000000000 correspond à la première catégorie, 0100000000 à la deuxième et ainsi de suite…
Y_train = np_utils.to_categorical(Y_train)
Y_test = np_utils.to_categorical(Y_test)

# mise ne forme technique du tableau d'input (images)
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')

# normalization des pixels des images d'une Valeur 0-255 en nombre compris entre 0 et 1
X_train = X_train / 255
X_test = X_test / 255
```

2.6.c BASE DU CODE D'APPRENTISSAGE KERAS ET NEUROEVOLUTION

Voici le code de base utilisé pour les apprentissages entre les évolutions. Sauf exceptions qui seront précisées, seule la patience du early_stop, entre 3 et 9, et le nombre d'epoch, entre 20 et 60, de l'entrainement, évolueront.

```
# compilation du modèle utilisé
Model.compile(loss=Loss, optimizer=Optimizer, metrics=[Config['metrics']])

#definition des condition d'arrèt de l'apprentissage pour éviter le sur-apprentissage
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=3,
verbose=0, mode='auto', baseline=None, restore_best_weights=True)

# definition du controle de la Vitesse d'apprentissage lors de la retropropagation de gradiant pour affiner l'apprentissage en fin de cycle
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1,
min_lr=0.001, verbose=1)

#log du temps d'apprentissage
time_callback = client.TimeHistory()
# lancement de l'apprentissage en précisant les données d'apprentissage et de validation
FitResults = Model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=20,
batch_size=Config['batch_size'], callbacks=[early_stop, reduce_lr, time_callback],
verbose=2)
```

Pour la partie neuroévolutive la technologie de la société Datavaloris a été utilisée. Ils ont accepté, pour des raisons de recherche, que leurs algorithmes soient utilisés gracieusement sans dévoiler leurs secrets industriels. Dans le cadre de cette thèse, il n'est pas nécessaire d'en savoir plus que

ce qui sera décrit dans ce document. Tout algorithme génétique appliqué au graphe de couches des modèles Keras aboutissant à une amélioration des modèles est utilisable.

La technologie utilisée est un algorithme génétique qui fait muter la topologie des modèles « Keras » dans notre cas, puis en évalue l'efficacité et sélectionne les modèles mutants les plus performants pour aboutir à un modèle champion.

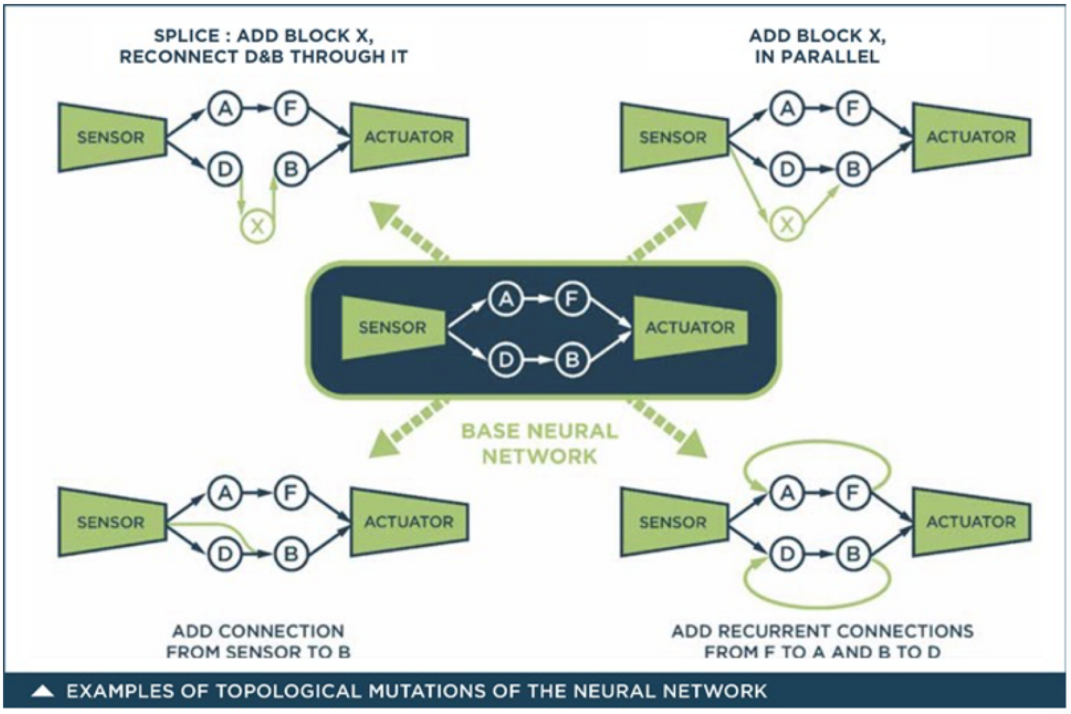


Figure 22 Exemple de mutations [41]

Les développements qui ont été effectuées pour cette thèse et qui ont été appliquées à l'algorithme génétique sont l'enregistrement des séquences de mutations (Figure 22) pour générer un chemin phylogénétique (Figure 23) et surtout pouvoir le rejouer à volonté à partir du modèle mère initial.

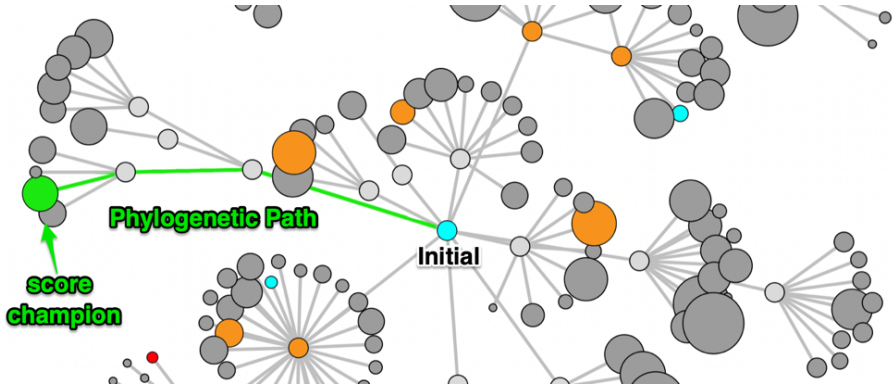


Figure 23 représentation d'un chemin phylogénétique

Attention ! En algorithmie génétique classique, la génération est ici appelée cycle. Cela étant précisé, un modèle M_3 avec 98,23% de score du cycle 8 et de génération 3, a donc 3 ancêtres dans son arbre phylogénétique et a été obtenu au cycle 8 (il aurait tout autant pu apparaître en cycle 3 au minimum ou 12…).

La neuroévolution ne sert que pour la Phase 0 de construction de l'arbre phylogénétique (manquant dans les modèles que l'on trouve sur internet).

Elle n'a pas d'importance en soit pour la thèse mais elle est détaillée un peu ici pour permettre aux lecteurs intéresser de comprendre son fonctionnement.

Le code génétique d'un modèle décrit de manière neutre la structure en graphe de la topologie du réseau de neurones du modèle considéré. Il inclut aussi l'historique de mutation depuis le modèle initial. Il est stocké au format XLM JSON voir Figure 24


```
1676005168789646592.genotype
{
  "Actuators": [
    {
      "Age": 0,
      "Coordinate": [
        2,
        0.5
      ],
      "FromCoordinate": [
        1,
        0.5
      ],
      "Id": "",
      "ModelName": "",
      "ModelPath": "",
      "Parameters": [],
      "Signal_Shape": [
        10
      ],
      "Type": "Signal_Channel",
      "Version": 1
    }
  ],
  "Constraints": [],
  "Dataset": "client_defined",
  "Generation": 9,
  "GenotypeSummary": [
    8,
    15
  ],
  "Id": "1676005168789646592",
  "InvalidMutations": [],
  "LayerDepth": 8,
  "MutationHistory": [ ... ],
  "ParentId": "1675802650869300992",
  "Sensors": [ ... ],
  "TimeCreated": "Fri Feb 10 04:59:29 2023",
  "Topology": [
    [ ... ],
    [ ... ],
    [ ... ],
    [ ... ],
    [ ... ],
    [ ... ],
    [ ... ],
    [ ... ]
  ],
  "TopologyHistory": [ ... ],
  "TotNodes": 15,
  "TotParameters": 12129390,
  "TrnFitness": 1.0,
  "TstFitness": 0.9868729710578918,
  "ValFitness": 0.9868729710578918,
  "Valid_NodeTypes": [],
  "custom_functions": {}
}
```


Figure 24 - Ex de génotype d'un modèle

Ce génotype est transformé en un phénotype qui est pour les besoins de la thèse un modèle au format Keras/TensorFlow.

Règles utilisées par le moteur évolutif :

```
    "PopSize": 5,
Taille de la population produite a chaque étape.

    "FitnessGoal": 1.0,
Objectif du moteur évolutif qui s'arrêtera lorsqu'atteint ici un score de 100%

    "LearnRate": 0.1,
Paramètre permettant de définir la puissance des mutations appliquées

    "SelectionType": "UnScaled",
Paramètre permettant de limiter ou non la production de mutant de taille inférieur au parant

    "SelectionParameter": 0.001,
Critère permettant de définir la tolérance de remplacement d'un champion par un mutant de
performance supérieur ou équivalente

    "HOFSize": 10,
Taille du pool de champion

    "NeuralEffeciency": 0.5,
Paramètre permettant de définir la puissance des mutations appliquées

    "ProblemDomain": "Classification",
    "LayerSearchConstraint": "PureLayers",
    "MutationSearchConstraint": "Unconstrained",
Paramètres definissant les contraintes de mutations

    "FreezeWeights": false,
Paramètre indiquant si un mutant voit ses poids hérités du parent figés lors de
l'apprentissage. Peut améliorer la vitesse d'apprentissage local des modèles.
```

2.6.d METRIQUES D'EVALUATION

La métrique utilisée pour la comparaison des modèles est la précision du jeu de test (Test Accuracy). L'arrêt précoce (early stop) a été appliqué, basé sur le score suivi et qui n'a pas été utilisé pour l'entraînement. La précision est utilisée comme métrique.

Pour mieux comprendre la différence de densité informationnelle des modèles, l'entropie de Shannon de leurs poids [36] (1), (2) après entraînement est calculée :

$$P_i = \frac{i}{\sum_{i=1}^{n}(i)} \qquad (1)$$

H entropy :

$$H = -\sum_{i=1}^{n} P_i \ln(P_i) \qquad (2)$$

2.6.e BIAIS METHODOLOGIQUES

Pour limiter des biais méthodologiques et faciliter la reproductibilité des expérimentations, il a été choisi d'utiliser des corpus de données publics et non augmentés. C'est un choix permettant de faire des comparaisons objectives.

Le propos de la thèse n'est pas de prouver l'efficacité de l'augmentation de données, sujet traité par ailleurs dans d'autres publications. Ces techniques complémentaires pourront être appliquées ultérieurement et dans les phases d'industrialisation.

La réduction de dimension du CIFAR10 induira un biais d'apprentissage car certaines images seront beaucoup plus difficiles à reconnaître et donc à apprendre. L'avantage de cette approche est que cette complexité d'apprentissage simule un système difficile à apprendre, comme il aurait été possible avec des jeux de donnés beaucoup plus conséquents mais que la puissance de calcul à disposition pour cette thèse n'aurait pas permis d'expérimenter.

Pour les comparaisons des résultats des tests, les hyperparamètres d'apprentissage seront les mêmes sauf pour le DDL où ils seront renforcés (epoch et patience doublée). Ainsi en cas de résultats en faveur du PRL, les résultats de la méthode DDL maximisés diminueront les contestations.

La notion de complexité d'un modèle M_c est en soit un sujet. Ce qui peut sembler complexe a un expert ne l'est peut-être pas dans la réalité. La règle utilisée (et discutable) est qu'un modèle est d'autant plus complexe qu'il est profond et dotés d'embranchements.

3. ANALYSE DES RÉSULTATS

Dans ce chapitre, les résultats des expérimentations qui ont été effectuées en suivant le protocole défini plus avant, y sont détaillés et analysés.

La première expérience sert de base à l'analyse et donne les premiers résultats sur la pertinence des méthodes.

Les expériences suivantes permettent de tester la reproductibilité des résultats obtenus en première expérimentation dans des contextes différents. Que ce soit de données mais aussi de méthodes d'apprentissage.

3.1 EXPERIENCE 1.

L'expérience 1, utilise le jeu de données MNIST

3.1.a EXP 1 - PHASE 0 : CONSTRUCTION DU CHEMIN PHYLOGENETIQUE

La Table 2 montre le modèle simple M^1_0 utilisé comme modèle de départ. Il comprend 7850 paramètres et 1 couche cachée dans une architecture séquentielle.

Table 2. Modèle initial 1, M^1_0.

Layer Number	Type	Output	Params
1	InputLayer	N, 28, 28, 1	0
2	Flatten	N, 784	0
3	Dense	N, 10	7850

Création du Champion 1 : M^1_c et de son chemin phylogénétique :

L'expérience a été configurée comme suit :

- Lors de l'utilisation de la méthode neuroévolutive, une population de départ de 20 modèles minimalistes aléatoires est générée.
- 5 mutants sont générés à chaque cycle (par mutation) à partir des meilleurs modèles du HOF (avec une taille maximale du HOF de 10), où la probabilité d'utiliser un modèle comme parent de la progéniture mutante étant proportionnelle à son score (précision) par rapport aux autres modèles du HOF.

- Cette expérience a utilisé le jeu de données MNIST.
- Le moteur évolutif a appliqué 1 à 2 mutations (choisies au hasard) pour créer un modèle de descendants mutants à partir du parent choisi.

Les paramètres d'apprentissage en profondeur utilisés étaient les suivants : 20 epochs avec une patience de 3 (nombre d'epoch sans amélioration avant arrêt de l'apprentissage), basée sur la métrique de val_loss.

La patience est un paramètre d'apprentissage qui définit le nombre de tentatives d'apprentissage infructueux avant d'arrêter le processus d'apprentissage.

Dans la liste des champions générés à l'aide de la méthode neuroévolutive (phase 0), le meilleur M^1_c est choisi, comme indiqué dans le Table 3.

Table 3. Information du Champion 1 M^1_c choisi (MNIST).

Score	Cycle	Génération	Paramètres	Noeud	Couches
0.9944	96	19	409158	25	13

Le champion M^1_c (= M^1_{19}) choisi a 409158 paramètres (poids, biais…) répartis entre 25 nœuds (éléments de la structure du réseau de neurone qui peuvent être soit des nœud techniques soit des noeuds de type couche de neurones) de 13 couches profondes de réseaux de neurones. Il a été généré au 96ème cycle et est de la génération 19 (il a 19 ancêtres et a nécessité 19 étapes de mutations). Son score de 99,44 % est proche de celui de l'état de l'art sur le corpus MNIST non augmenté. Les mutations enregistrées à chaque étape qui conduisent à la topologie championne finale sont affichées dans la Table 4. À chaque étape évolutive, 1 à 2 mutations ont été appliquées. Le nombre de mutations appliquées à chaque étape est limité à un maximum de 2 afin de générer un modèle complexe avec de petits changements entre chaque étape, ce qui permet à PRL de travailler sur des parties plus petites lors de chaque mutation.

La Table 4 présente une base de comparaison ; il montre les scores PRL du champion M^1_c à chaque étape de son parcours évolutif (ici 19 générations). Ces scores ont été utilisés comme critères de sélection pour l'entrée HOF des descendants au cours du processus évolutif. Ce premier résultat montre que le modèle a augmenté de taille. Il s'agit d'un comportement classique d'un algorithme génétique si aucune restriction de taille n'est favorisée lors de la génération et de la mutation du modèle (les mutations de réduction de taille sont moins efficaces au début de l'évolution). On voit aussi que l'entropie de Shannon décroît de génération en génération, passant de 8,98 à 8,90 (modèle initial exclu, i.e.12,51).

Table 4. Score pour chaque étape évolutive du champion M^1_c.

Etape	Taille	SCORE	SHANNON	Etape	Taille	SCORE	SHANNON
0	7850	8.79	12.5151	10	264970	99.3	8.92824
1	94906	97.51	8.98112	11	269130	99.27	8.92447
2	27082	98.43	8.97188	12	300874	99.28	8.92426
3	58538	98.96	8.98559	13	300874	99.33	8.91894
4	90346	99.03	8.98102	14	304970	99.34	8.91918
5	90282	99.03	8.96616	15	304970	99.34	8.91798
6	183818	99.23	8.95914	16	304970	99.35	8.91156
7	183818	99.19	8.9567	17	304970	99.36	8.90396
8	258570	99.24	8.94914	18	405486	99.41	8.90111
9	264330	99.29	8.9393	19	409158	99.44	8.90037

Cette réduction d'entropie peut être interprétée comme l'augmentation de l'organisation et de la quantité d'informations utiles stockées par les poids du modèle. L'ensemble de poids presque aléatoire pour M^1_0 passe à un ensemble de poids qui stocke des informations utiles avec une distribution plus organisée pour M^1_{19}.

3.1.b EXP 1 - PHASE 1 : TESTS D'EFFICACITÉ : DDL *VS* PRL

Au cours de la phase 1, les résultats des deux différentes approches d'apprentissage pour évaluer l'impact de l'utilisation de PRL par rapport au DDL sont rassemblés.

Ces tests vont permettre de constater l'impact de l'application de la phylogénèse à l'équivalent de l'embryogénèse pour la construction d'une entité neurogénétique numérique.

Il s'agit donc de rejouer l'évolution d'un modèle initial « primitif » M^1_0 en appliquant les mutations qui ont été enregistrées dans le chemin évolutif lors de la construction d'un modèle « final » champion M^1_c de réseau de neurones.

Plus le modèle est complexe et profond, plus l'effet de la méthode PRL devrait être efficace contrant, entre autres, l'effet grandissant du gradient évanescent (VGE).

Les résultats bruts sont décrits puis suivis de l'analyse statistique de ces derniers pour en tirer des conclusions partielles.

- *DDL DU CHAMPION 1 $M^{1D}{}_{C}$*

Pour évaluer la capacité d'apprentissage du modèle généré par la phase 0, le Champion 1, $M^{1}{}_{c}$, 50 exécutions de la méthode d'apprentissage standard DDL sont appliquées directement à la topologie de ce modèle $M^{1}{}_{c}$.

Les poids initiaux de chaque expérience ont été initialisés aléatoirement. Ce nombre d'exécutions permet de calculer un écart-type statistiquement pertinent. En théorie, les résultats d'apprentissage DDL du modèle $M^{1d}{}_{c}$ pourrait être les mêmes (voir meilleurs) que le champion 1 original $M^{1}{}_{c}$ car la seule différence se trouve dans les poids du réseau de neurones, mais la probabilité que ces 409158 paramètres aléatoires soient précurseurs d'un optimum post apprentissage reste très faible et proche de 0.

Pour effectuer ces expériences et maximiser la probabilité d'atteindre un bon minimum local, 60 epochs par cycle ont été utilisés, avec une patience fixée à 6 (soit le double de ce qui sera utilisé pour le PRL). Au cours de nos expériences, un maximum de 53 epochs ont été utilisés avant l'arrêt précoce (quand le modèle n'apprend plus). En moyenne, 45 epochs sur 60 ont été exploités avant le déclenchement de l'arrêt précoce.

Pour rappel, au cours de la phase 0 de la méthode PRL, le champion 1, $M^{1}{}_{c,}$ a atteint une précision de 99,44 %. Son entropie de Shannon était de 8,90037.

Le meilleur score/précision obtenu lors de l'expérimentation en utilisant la DDL pour arriver au modèle champion $M^{1d}{}_{c}$ est de 99,05 %, avec une différence statistiquement significative (Table 5).

Le VGE est soupçonné d'être à l'origine de ce mauvais résultat. De plus, il est visible que l'entropie de Shannon du modèle $M^{1d}{}_{c}$ le plus performant formé à l'aide de l'approche DDL (9,1227) est également supérieure à l'entropie du modèle champion 1 $M^{1}{}_{c}$ produit lors de la phase 0 (8,9003).

Table 5. Résultats du DDL sur le champion 1 : $M^{1d}{}_{c}$ (MNIST).

$M^{1d}{}_{c}$	SCORE	SHANNON
BEST	99.05	9.1227
MEAN	98.93	9.1615
Standard Deviation	0.067	0.0168

Les résultats obtenus lors de l'utilisation de DDL sur la topologie du modèle champion sont également moins bons que le résultat obtenu lors de la phase 0 de cette expérience.

- *PHYLOGENETIC REPLAY LEARNING DU CHAMPION 1* $M^{1P}{}_{C}$

A partir du modèle initial $M^1{}_0$, les mutations sont appliquées en fonction du chemin phylogénétique du modèle champion 1, $M^1{}_c$. Les poids sont générés de manière aléatoire pour les nouvelles couches mutées ainsi que pour le modèle $M^1{}_0$ de départ. L'expérience PRL a été reproduite 50 fois pour recueillir des données statistiquement représentatives sur lesquelles baser les résultats.

Les poids des couches non mutées n'étaient pas réinitialisés entre les mutations (ce qui peut être considéré comme un apprentissage par transfert partiel). La Table 6 montre les résultats des 50 expériences PRL.

Le meilleur score atteint de $M^{1p}{}_c$, est de 99,40% avec une moyenne de 99,26%. Ce score est très proche de celui du modèle champion 1 original $M^1{}_c$ qui atteignait 99,44%. Il y a donc une cohérence substantielle.

Table 6. Résultats du PRL sur le champion 1 : $M^{1p}{}_c$ *(MNIST).*

Step	Mean Score	Std. Deviation	Best Score	Shannon	Step	Mean Score	Std. Deviation	Best Score	Shannon
0	92.14	0.0771	92.33	12.5163	10	99.16	0.0647	99.32	8.8497
1	97.68	0.2711	98.11	8.9256	11	99.17	0.0700	99.35	8.8454
2	98.35	0.0925	98.58	8.9015	12	99.18	0.0563	99.31	8.8437
3	98.88	0.0841	99.02	8.9079	13	99.19	0.0641	99.34	8.8415
4	99.01	0.0676	99.16	8.8960	14	99.19	0.0626	99.34	8.8396
5	99.05	0.0634	99.13	8.8802	15	99.19	0.0618	99.31	8.8373
6	99.12	0.0553	99.23	8.8749	16	99.24	0.0606	99.36	8.8262
7	99.11	0.0541	99.22	8.8702	17	99.24	0.0521	99.37	8.8208
8	99.14	0.0515	99.27	8.8628	18	99.24	0.0542	99.35	8.8185
9	99.14	0.0660	99.26	8.8547	19	99.26	0.0628	99.40	8.8147

L'entropie de Shannon du modèle $M^{1p}{}_c$ le plus performant formé à l'aide de l'approche PRL (8,8147) est inférieure à l'entropie du modèle champion 1 $M^1{}_c$ produit lors de la phase 0 de la méthode (8,9003) ce qui en soit est intéressant même si la différence n'est pas significative.

- *ANALYSE DES RESULTATS*

La Table 7 montre les informations statistiques des expérimentations DDL and PRL.

Table 7. Statistiques de l'expérience 1

	DDL	PRL		
Mean Score	98.93%	99.26%	POOLED VARIANCE	4.2E-07
VARIANCE	4.5E-07	3.9E-07	T STAT	-25.13668
OBSERVATIONS	50	50		

- Les scores sont inférieurs à ceux produits par le champion $M^1{}_c$ (qui a suivi le chemin optimal en phase 0). Cela est probablement dû aux poids générés aléatoirement à chaque étape. Mais, il est également visible que l'écart type des expériences est faible, il y a donc une cohérence de performance en ce qui concerne les résultats produits par PRL.
- Le score produit par PRL est meilleur que celui produit par DDL. Avec un maximum moyen de 99,26% contre 98,93% de DDL, la différence est statistiquement significative ($p < 0,001$ - Table 7. Statistiques de l'expérience 1) et la distribution est bien séparée Figure 25. Même constat, en comparant les deux maximums de 99,40 % (PRL $M^{1p}{}_c$) et 99,05 % (DDL $M^{1d}{}_c$). Donner plus de temps à DDL pour s'entraîner (60 epochs) n'améliore pas ses performances (l'arrêt précoce se produit presque toujours avant le nombre maximum d'epochs).
- L'écart type de PRL est inférieur (meilleur) à celui du DDL (Table 7). C'est un faisceau d'indices qui semble confirmer que PRL est une approche plus robuste et plus résistante à l'initialisation aléatoire des poids.
- Au cours de l'expérience PRL, la valeur de Shannon a constamment diminué à chaque étape (Table 6) du processus. Cela peut être vu comme une organisation/densité informationnelle croissante du modèle tandis que la complexité du modèle augmente à chaque étape.
- L'entropie de Shannon du modèle basé sur PRL est inférieure (meilleure) que celle du modèle basé sur DDL ; 8,81 contre 9,16.

Les deux derniers résultats renforcent l'hypothèse que le PRL atténue le VGE. Bien que d'autres tests devraient être menés pour conforter l'analyse de cette méthode phylogénétique, ces résultats semblent d'ores et déjà annoncer une voie prometteuse. La Table 8 montre que lors de l'utilisation

de DDL, l'entropie de Shannon des dernières couches du modèle est inférieure à celles du modèle formé par PRL (valeurs en gras pour l'entropie la plus faible dans la Table 8). L'entropie calculée pour chaque couche du champion 1 est affichée à des fins de comparaison.

Table 8. Comparaison des entropies de Shannon par couche du NN.

LAYER #	TYPE	DDL M^{1d}_c	PRL M^{1p}_c	Ref. M^{1}_c
2	CONV2D	9.162	8.815	
3	SEPCNV2D	8.372	**8.234**	8.235
4	CONV2D	15.159	**15.138**	15.142
5	DENSE	14.776	**14.727**	14.715
6	DENSE	**11.683**	11.687	11.691
7	CONV2D	14.155	**14.081**	14.054
8	CONV2D	14.186	**14.077**	14.068
9	DENSE	12.104	**12.101**	12.095
10	DENSE	**11.684**	11.688	11.689
11	DENSE	**17.405**	17.512	17.517
12	DENSE	**11.689**	11.711	11.713
13	DENSE	**12.466**	12.588	12.582

Cela laisse entendre que l'entrainement standard (DDL) affecte principalement les dernières couches du modèle en raison du VGE. Le modèle DDL stocke ses informations dans ces couches de manière plus dense, tandis qu'avec la méthode PRL, l'ajustement du poids et le stockage des informations sont répartis plus uniformément entre les couches hautes et basses du modèle final. De plus les Shannon de chaque couche du modèles PRL sont plus proches de celles du modèle Champion issu de la phase 0 utilisé comme référence de métrique (en **bleu** dans la table 8). L'entropie totale de Shannon est plus faible dans PRL que dans DDL.

Table 9. Comparaison DDL vs. PRL pour chaque étape évolutive.

STEP	DDL Max. score	PRL Mean score	STEP	DDL Max. score	PRL Mean score
0	92.140	92.144	10	98.760	99.161
1	98.160	97.679	11	98.870	99.173
2	98.130	98.347	12	98.870	99.179
3	98.470	98.879	13	98.760	99.193
4	98.430	99.007	14	98.860	99.186
5	98.420	99.048	15	98.750	99.192
6	98.560	99.115	16	98.860	99.239
7	98.780	99.105	17	98.860	99.239
8	98.770	99.135	18	98.820	99.243
9	98.940	99.138	19	98.790	99.258

La Table 9 montre que si, à chaque étape, le même modèle est entrainé (en réinitialisant d'abord ses poids) à l'aide de DDL, il atteint à la fois une précision finale inférieure (baisse d'efficacité) et, sur la base de son score d'entropie de Shannon, stockerait moins d'informations. Les différences de performances entre les modèles entraînés DDL et PRL augmentent à mesure qu'ils deviennent plus complexes et profonds.

La Figure 25 montre une représentation graphique des résultats de la Table 9 où l'écart entre les 2 courbes est flagrant et également que chaque expérience suit une distribution gaussienne.

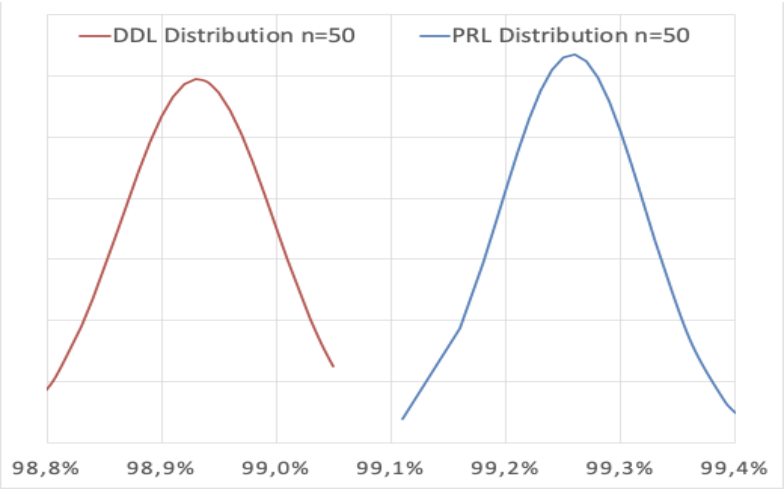


Figure 25. Distribution des scores DDL et PRL pour l'expérimentation 1 sur corpus MNIST

La Figure 26. Graphique de résultat de l'expérimentation 1 montre les 3 résultats obtenus sur le score et l'entropie de Shannon :

- Score d'évolution et Shannon récupérés lors de la phase 0 de la création du champion 1 M^1_c. En bleu dans le graphique.
- Score DDL moyen et Shannon du modèle produit à chaque étape, utilisé en apprentissage direct DDL, jusqu'au modèle final M^{1d}_c. En rouge dans le graphique.
- Score PRL moyen et croissance du modèle de Shannon du champion moyen M^{1p}_c par étape de mutation. En vert dans le graphique.

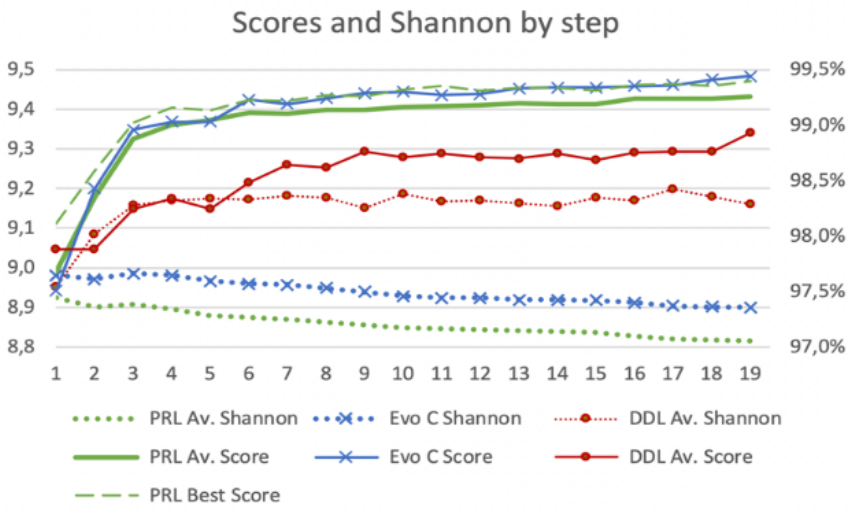


Figure 26. Graphique de résultat de l'expérimentation 1 par étape (MNIST).

En ce qui concerne la représentation des résultats, les lignes pleines représentent le score (échelle à droite, le meilleur situé le plus en haut), les lignes en pointillé représentent l'entropie de Shannon (échelle à gauche, le meilleur situé le plus en bas) et à titre de comparaison, le meilleur score PRL est représenté par la ligne en trait vert. Le score moyen de Shannon est, à chaque étape, supérieur (moins ordonné et donc moins porteur d'information) pour le modèle sous DDL en rouge pointillé que pour celui obtenu avec la méthode PRL en vert pointillé.

D'autres tests devront être menés pour conclure si ce comportement pourrait s'appliquer également à d'autres modèles complexes (voir discussion sur la notion de complexité 4.2.d).

3.1.c EXP 1 - PHASE 2 : TESTS DE TRANSFERABILITE

Dans cette sous-section, il s'agit de répondre aux deux questions suivantes :

1. Peut-on utiliser PRL pour reprendre le modèle M^1 des expériences précédentes sur de nouveaux ensembles de données du même champ d'application ?

et

2. Le résultat de ce transfert avec PRL reste-t-il plus efficace qu'avec DDL ?

Afin de répondre à ces questions, PRL a de nouveau été appliqué au modèle initial $M^1{}_0$ avec le chemin phylogénétique du champion 1 $M^1{}_c$, mais, cette fois, en l'entraînant sur un corpus de données différent. Le but de cette expérience est d'évaluer si un modèle avec son chemin évolutif enregistré à partir d'un ensemble de données A peut être appliqué sur un autre ensemble de données B structuré de la même façon (même nombre de catégorie d'objets – 10- et même taille d'images 28*28 pixels)

- *EXPERIENCE 1.1 : CHAMPION 1 MNIST => JEU DE DONNEES MNIST FASHION*

Les deux méthodes d'apprentissage sont réappliquées au corpus de données de Fashion MNIST. Cet ensemble de données a le même format d'entrée et de sortie que le MNIST standard. Dans cet ensemble de données, la classification se fait sur divers objets de mode (robes, chaussures, etc.) plutôt que sur des chiffres dessinés. Cet ensemble de données s'avère plus complexe que le MNIST standard à discriminer.

DDL Score moyen du M1.1dc sur jeu de données FashMNIST
Score : 90.44% Shannon : 9.0238

PRL Score moyen du $M^{1.1p}{}_c$ sur jeu de données FashMNIST
Score : **91.98%** Shannon : **8.4813**

Cette expérience montre que la méthode PRL peut être réappliquée à un modèle existant, et le faire réapprendre sur un ensemble de données connexes, mais différent. PRL fournit un meilleur résultat que DDL. À titre de comparaison, le score du modèle à l'état de l'art appliqué au Fashion MNIST est de 91,4% sans augmentation des données [38]. Le score PRL de 91,98% de l'expérience est donc un résultat concurrentiel qui surpasse l'état de l'art. L'une des implications potentielles de ce résultat de transférabilité est que PRL pourrait permettre de réutiliser plus facilement des topologies de modèles existantes sur de nouveaux champs d'application (mais qui restent tout de même liés à celui de départ). La transférabilité à de nouveaux champs d'application

pour lesquels ces architectures n'ont pas été conçues à l'origine devient possible tout en conservant des scores très élevés, ce que DDL ne permet pas facilement.

Table 10. Comparaison des Shannon par couche pour FASHION MNIST.

LAYER	TYPE	DDL	PRL
2	CONV2D	9.0238	**8.4813**
3	SEPCNV2D	8.307	**8.0439**
4	CONV2D	15.1492	**15.1338**
5	DENSE	14.7804	**14.7331**
6	DENSE	11.6941	**11.6841**
7	CONV2D	14.1506	**13.9888**
8	CONV2D	14.1568	**13.9923**
9	DENSE	12.1115	**12.1017**
10	DENSE	11.6922	**11.6905**
11	DENSE	**17.3703**	17.4783
12	DENSE	**11.6984**	11.71
13	DENSE	**12.3777**	12.5757

La Table 10 montre à nouveau que PRL est plus en mesure de réduire le VGE. Les premières couches ont de meilleures valeurs d'entropie de Shannon lorsqu'un modèle est formé via PRL, alors que les dernières couches ont de meilleures valeurs d'entropie lorsque c'est DDL qui est utilisé pour former le modèle.

- *EXPERIENCE 1.2 : CHAMPION 1 MNIST => JEU DE DONNEES CIFAR10 GRAY*

Dans cette expérience, le modèle Champion 1 M^1_c est entrainé sur le corpus de données CIFAR10 converti en niveaux de gris (C10G). Pour cette expérience, il a été converti à la résolution 28 * 28 * 1, en niveau de gris, afin de pouvoir utiliser à nouveau le même modèle et tester sa transférabilité. Cet ensemble de données rétrogradé est beaucoup plus difficile pour l'entrainement que pour le MNIST.

DDL Score moyen du $M^{1.2d}_c$ - jeu de données C10G:
Score: 54.40% Shannon: 9.1690

PRL Score moyen du $M^{1.2p}_c$ - jeu de données C10G
Score: **65.01%** Shannon: **8.9345**

Ces résultats montrent à nouveau la capacité du PRL à transférer un modèle existant sur un ensemble de données nouveau, mais lié (avec la même résolution d'image et le même nombre de catégorie d'image). Dans nos expériences, PRL produit systématiquement de meilleurs résultats que DDL, à la fois en termes de précision et de densité d'informations (valeurs d'entropie de Shannon).

3.2 EXPERIENCE 2 (REPRODUCTIBILITE).

Cette expérience numéro 2 reprend les mêmes phases que l'expérience 1. L'expérience 1 avait pour but de valider l'intérêt ou non de l'approche phylogénétique vis-à-vis d'un apprentissage classique DDL. Cette deuxième expérience permettra d'évaluer la reproductibilité de la méthode.

Le contexte de l'expérience 2 a été changé dans le but de complexifier le système, pour simuler un projet plus compliqué. Les mutations ont été ralenties pour obtenir un modèle champion plus complexe, c'est-à-dire plus profond, avec des branches et regroupements. De plus le modèle initial M^2_0 bien que plus profond (plus de couches) est plus petit en nombre de paramètres.

Le jeu de données utiliser est toujours MNIST.

L'environnement utilise un autre framework - PlaidML - et un autre support matériel - AMD - pour évaluer PRL sur un autre support.

3.2.a EXP 2 - PHASE 0 : CONSTRUCTION DU CHEMIN DU CHAMPION 2

Model Initial 2 : M^2_0

La Table 11 montre le modèle simple utilisé comme modèle initial. Il comprend 4 900 paramètres et 3 couches cachées dans une architecture séquentielle.

Table 11. Modèle Initial 2.

Layer Number	Type	Output	Params
1	InputLayer	N, 28, 28, 1	0
	Conv2D	N, 27, 27, 6	30
	MaxPooling2D	N, 9, 9, 6	0
2	Flatten	N, 486	0
3	Dense	N, 10	4870

- *CHAMPION 2 : CONSTRUCTION*

Cette phase 0 consiste en la construction d'un nouveau modèle Champion 2 : M^2_c.

Les règles évolutives utilisées pour l'obtention de l'arbre phylogénétique s'appuient sur des cycles de 5 mutants issus d'un vivier de parents des 10 plus performants. Le moteur évolutif a été volontairement ralenti (divisé par 4 grâce à des mutations moins impactantes sur la topologie) pour obtenir plus d'étapes et simuler un champ d'application plus complexe à apprendre que juste le MNIST, qui est vite assimilé par le mécanisme évolutif.

Les règles d'apprentissage sont celles décrites précédemment à savoir des apprentissages de 20 epochs avec une patience avant arrêt sur le Val_loss de 3.

La génération reste le nombre de parents depuis l'origine du test.

La Figure 27 Courbe de construction automatisé des champions successifs par neuroevolution, montre les courbes de progression lors de l'apparition des champions pendant de la phase 0 de neuroévolution.

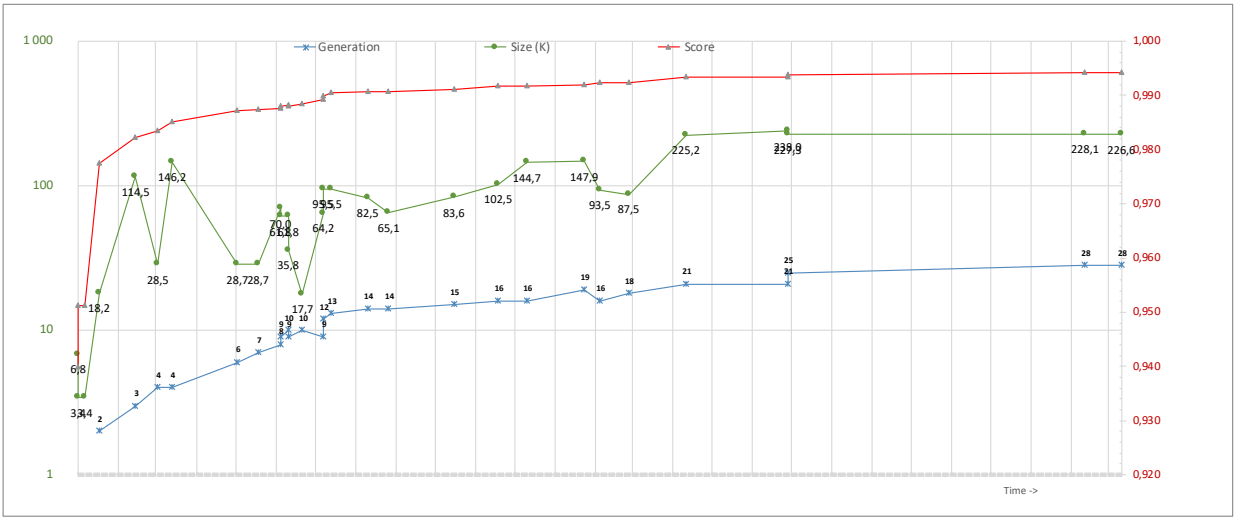


Figure 27 Courbe de construction automatisé des champions successifs par neuroevolution

La Table 12 quant à elle montre sous forme d'un tableau la liste des champions successifs avec l'évolution du score et de la taille dans le temps.

Table 12. Liste des Champions MNIST au cours du temps obtenus en phase 0

Score	cycle	Generation	Params	Noeuds	Couches
0,93990	-	-	6 790	5	4
0,95130	-	-	3 410	5	4
0,95130	1	-	3 410	5	4
0,97750	3	2	18 160	7	5
0,98230	8	3	114 454	10	5
0,98350	11	4	28 468	8	5
0,98510	13	4	146 184	10	6
0,98710	22	6	28 650	10	6
0,98740	25	7	28 650	10	6
0,98760	28	8	70 010	10	6
0,98800	28	9	61 816	15	6
0,98810	29	9	35 832	16	6

0,98820	29	10	61 816	15	6
0,98840	31	10	17 744	14	7
0,98930	34	9	64 184	17	7
0,98990	34	12	95 454	19	7
0,99040	35	13	95 454	19	7
0,99070	40	14	82 496	20	7
0,99080	43	14	65 110	21	6
0,99110	52	15	83 616	22	7
0,99170	58	16	102 498	24	7
0,99180	62	16	144 670	23	7
0,99200	70	19	147 940	28	10
0,99230	72	16	93 516	22	7
0,99240	76	18	87 508	28	9
0,99330	84	21	225 244	31	11
0,99340	98	21	238 010	31	11
0,99390	98	25	227 344	35	13
0,99420	139	28	228 140	40	15
0,99430	144	28	226 592	39	14

De cette liste de champions successifs, le dernier (le meilleur) est retenu pour la suite de l'expérience :

Score	Cycle	Génération	Params	Noeux	Couches
0,99430	144	28	226 592	39	14

Le Champion 2, M^2_c (=M^2_{28}) a donc 226 592 paramètres répartis en 14 couches.

Il a été généré lors du cycle 144 et est de génération 28 (et il a donc 28 ancêtres).

Son score est de 99,43% sur un corpus de données standard non augmenté. Même si le propos de cette thèse n'est pas de démontrer l'efficacité de l'utilisation de la neuroévolution pour la construction de modèles de réseaux de neurones, les résultats de cet algorithme sont intéressants.

- *ARBRE PHYLOGENETIQUE DU CHAMPION 2*

Les topologies des différentes étapes évolutives du Champion sont suffisantes pour reproduire l'expérience.

VOIR EN ANNEXE les topologies des modèles du champion 2, M^2_c

3.2.b EXP 2 PHASE 1 : TESTS D'EFFICACITE

Afin de pouvoir comparer les résultats, pour commencer, il sera défini une base de comparaison. Les résultats du champion issu de l'évolution initiale sont déjà acquis.

La prochaine action sera donc d'évaluer l'efficacité du modèle Champion en apprentissage direct

- *APPRENTISSAGE DIRECT DDL DU CHAMPION 2 MNIST*

Pour évaluer l'apprentissage direct du champion $M^2_{c,}$ on lance donc un apprentissage direct du modèle champion pour obtenir M^{2d}_c.

Comme les poids initiaux seront générés de manière aléatoire, il conviendra de faire de multiples apprentissages, pour évaluer la moyenne et l'écart-type de l'efficacité d'un apprentissage direct.

Pour rappel, en théorie, rien ne devrait empêcher l'obtention d'un modèle champion ayant appris de manière optimum par DDL, c'est à dire avec une efficacité égale ou supérieure à celle du Champion 2 généré par neuroévolution. Les 2 éléments qui pourraient être un frein à cela sont les poids initiaux et l'effet d'évanescence du gradient (VGE) rétropropagé vers les couches hautes.

Même si 226 592 paramètres semblent peu, la combinatoire est grande et donc la probabilité de tomber sur le résultat optimum est très faible.

Pour l'apprentissage, le paramétrage choisi pour en tirer le maximum est un nombre d'epochs de 60 et une patience de 6. Néanmoins 32 epochs au maximum ont été utilisés avant l'arrêt précoce pour cette expérience (soit moins que pour l'expérience 1). Cet arrêt précoce montre qu'il y a probablement une corrélation entre la facilité d'apprentissage et la simplicité du modèle.

Table 13. Echantillon de résultats de DDL Champion 2 M^{2d}_{c}

	val_acc	epochs
Train 1	98,83%	30
Train 2	98,85%	24
Train 3	98,77%	23
Train 4	98,91%	30
Train 5	98,76%	21
Train 6	99,06%	32
Train 7	98,89%	29
Train 8	98,91%	22
Train 9	98,99%	28
Train 10	98,91%	25
Train 11	98,85%	24
Train 12	98,89%	25
Train 13	98,76%	22
Train 14	98,84%	28
Train 15	99,05%	29
Train 16	99,08%	26

4 tentatives d'apprentissage direct sont en échec de convergence. C'est probablement dû à un bug Keras/PlaidML car ce problème ne s'est pas présenté pour l'expérience 1 du champion 1.

La base de comparaison est donc :

Ecartype :	0,10%
Moyenne :	98,90%
Max :	99,08%

Pour rappel, 99,43% est l'efficacité atteinte par le champion 2 M^2_c à sa construction en phase 0.

- *APPRENTISSAGE PHYLOGENETIQUE PRL DU CHAMPION 2 MNIST*

Ce test consiste à évaluer l'efficacité de la programmation génétique pour générer un nouveau modèle sur base de champion 2 : M^{2p}_0.

A partir du modèle initial 1 les mutations seront appliquées séquentiellement selon la séquence phylogénétique du Champion 2 source.

Les poids initiaux seront aussi aléatoires et donc pour les mêmes raisons que pour l'apprentissage direct, plusieurs tests seront à effectuer. Les poids non mutés sont transmis de génération en génération à chaque étape en utilisant le service de transfert learning du framework.

Voici les résultats des scores sur 10 apprentissages phylogénétiques :

Table 14. Résultats PRL champion 2 M^{2p}_c (MNIST)

Etapes	Ecartype	Score Moyen	Original Score	Paramètres
0	0,72%	94,31%	94,60%	4 900
1	0,59%	95,81%	95,96%	8 160
2	0,48%	96,48%	95,35%	29 260
3	0,26%	97,78%	97,72%	29 324
4	0,12%	98,28%	98,35%	28 468
5	0,13%	98,54%	98,34%	28 594
6	0,09%	98,73%	98,71%	28 650

7	0,11%	98,50%	98,40%	8 070
8	0,11%	98,56%	98,60%	10 540
9	0,20%	98,51%	98,73%	15 580
10	0,11%	98,73%	98,84%	17 744
11	0,22%	98,67%	98,84%	94 554
12	0,11%	98,91%	98,99%	95 454
13	0,11%	98,98%	99,04%	95 454
14	0,06%	99,01%	99,07%	82 496
15	0,05%	99,11%	99,14%	83 096
16	0,07%	99,11%	99,18%	144 670
17	0,05%	99,15%	99,09%	143 972
18	0,06%	99,19%	99,15%	147 124
19	0,05%	99,17%	99,20%	147 940
20	0,06%	99,18%	99,24%	225 250
21	0,06%	99,18%	99,31%	225 276
22	0,04%	99,14%	99,31%	228 170
23	0,06%	99,14%	99,24%	227 374
24	0,07%	99,14%	99,32%	227 350
25	0,07%	99,18%	99,33%	226 904
26	0,08%	99,13%	99,37%	227 316
27	0,06%	99,18%	99,35%	227 532
28	0,06%	99,19%	99,43%	226 592

Le maxima atteint sur tous les apprentissages phylogénétiques est de 99,31 % avec une moyenne des maxima de chaque test à 99,24%.

Le temps pour l'apprentissage phylogénétique est d'environ 1H45

- Un premier constat est que l'efficacité de l'apprentissage phylogénétique M^{2p}_c ou DDL M^{2d}_c est moins bonne que celle du champion source M^2_c.
- Un second constat est que l'efficacité de l'apprentissage PRL est meilleure que celle de l'apprentissage DDL. La différence de la moyenne des maxima de 99,24% pour PRL contre 98,90% pour l'apprentissage DDL, qui bien qu'elle soit faible reste significative. Même le maximum 99,31% contre 99,08% reste pertinent. Cela confirme les résultats de l'expérience 1.
- Un troisième constat, et peut-être le plus important à noter, est que l'écart-type de l'apprentissage PRL est meilleur que celui de l'apprentissage DDL : soit 0,06% pour le premier contre 0,1% pour l'apprentissage DDL. Constat qui rend l'apprentissage PRL plus robuste face aux conditions de poids initiaux.
- Un quatrième et dernier constat est que l'apprentissage phylogénétique est plus coûteux en temps de calcul dans cette expérience que l'apprentissage direct. Aucune optimisation PRL n'est appliquée à ce stade.

3.2.c EXP 2 PHASE 2 : TESTS DE TRANSFERABILITE

Les tests de transférabilité ont été faits dans le contexte de l'expérimentation 2 sur deux autres jeux de données.

- *EXP 2.1 : CHAMPION 2 MNIST => JEU DE DONNEES MNIST FASHION*

Il s'agit comme pour l'expérience 1.1 d'appliquer le PRL du champion 2, M^2_c, sur le corpus Fashion MNIST et de voir si les résultats sont confirmés.

DDL Score moyen du $M^{2.1d}_c$ - jeu de données FashM :
Score : 90.71%

PRL Score moyen du $M^{2.1p}_c$ - jeu de données FashM :
Score : **92.62%**

- *EXP 2.2 : CHAMPION 2 MNIST => JEU DE DONNEES CIFAR 10 GRIS*

Il s'agit comme pour l'expérience 1.1 d'appliquer le PRL du champion 2, M^2_c, sur le corpus Cifar10G et de voir si les résultats sont confirmés.

DDL Score moyen du $M^{2.2d}_c$ - jeu de données C10G :
Score : 62.19%

PRL Score moyen du $M^{2.2p}_c$ - jeu de données C10G :
Score : **67.83%**

PRL obtient encore de meilleurs résultats que DDL. A noter que les résultats du modèle, pourtant pas prévu pour ces corpus, semblent être acceptables en termes de score et conformes à l'expérience 1.1 et 1.2

3.3 EXPERIENCE 3 (REPRODUCTIBILITE).

Pour cette expérience 3 qui reprend les mêmes phase 0, 1 et 2 que l'expérience 1 et 2, les tests sont réalisés sur un corpus différent et également plus gros afin de confirmer l'hypothèse que la méthode PRL reste plus efficace que celle DDL.

Dans cette cinquième expérience, l'ensemble de données TinyImageNet avec 200 classes et un nombre relativement faible d'échantillons d'apprentissage (dotés d'une plus grande difficulté d'apprentissage) ont été utilisés. 50 tests ont été exécutés en utilisant DDL et PRL.

Champion 3 M^3_c généré (TinyImageNet):
19,771,676 parametres, 65 noeuds, 30 couches, 44 générations
Score: 42.87% Shannon: 9.3795

DDL Score moyen du M^{3d}_c - jeu de données TIN :
Score : 38.37% Shannon : 9.3833

PRL Score moyen du M^{3p}_c - jeu de données TIN :
Score : **41.79%** Shannon : **9.3611**

L'apprentissage PRL produit à nouveau un meilleur résultat que celui DDL.

Étant donné que notre objectif est de comparer des "pommes aux pommes" et de trouver les performances relatives d'une méthode par rapport à une autre, PRL et DDL ont été appliqués à l'ensemble de données TinyImageNet d'origine. Il convient de repréciser qu'aucune augmentation de données (ajout d'images transformées au jeu d'apprentissage ou encore de pré-traitement supplémentaire) n'a été appliquée à l'ensemble de données. L'augmentation des données étant une

approche courante avec cet ensemble de données en raison du peu d'échantillons qu'il contient pour chaque classe.

3.4 EXPERIENCE 4 (REPRODUCTIBILITÉ).

Lorsque l'on fait référence à la complexité du modèle, il est supposé que le modèle possède beaucoup de branches, qu'il est profond et non séquentiel. Plus le modèle est complexe (en termes de topologie), plus il serait bénéfique de l'entrainer en utilisant PRL. Ainsi, afin d'explorer davantage ces hypothèses, une autre expérience a été menée où les règles de sélection naturelles de la phase 0 ont été changées.

Lors de cette expérience, une règle a été notamment ajoutée au processus de sélection pour privilégier les modèles qui apprennent plus rapidement. La vitesse d'entraînement du modèle est utilisée pour pondérer le score. Le score du modèle est alors une combinaison de l'efficacité et de la rapidité d'exécution. Avec cette approche, un modèle avec la même précision qu'un autre, mais avec une vitesse d'apprentissage plus courte (aka Time Epoch), est sélectionné pour entrer dans le HOF. Cette pression sélective a favorisé, en phase 0 de construction du chemin évolutif, les champions qui sont rapides à entrainer, donc probablement moins sensibles à l'effet de VGE.

Champion 4 M^4_c généré (MNIST) :
Score : 99.40% Shannon : 8.4799

DDL Score moyen du M^{4d}_c - Jeu de données MNIST :
Score : **99.37%** Shannon : 8.4150

PRL Score moyen du M^{4p}_c - Jeu de données MNIST :
Score : 99.33% Shannon : **8.3535**

Dans cette expérience, la valeur de Shannon est toujours plus faible lors de l'utilisation de PRL par rapport à DDL. Néanmoins la différence entre les résultats de cette expérience est tout de même moins significative, par exemple le score qui est quasi identique entre PRL et DDL, et reste également proche de celui du modèle champion M^4_c lui-même.

La notion de complexité d'un modèle, qui permet de faire l'estimation de l'efficacité de PRL, est donc non seulement liée à la complexité topologique (paramètres totaux, les nœuds totaux et les liaisons de nœuds), mais également à la rapidité d'apprentissage (du temps nécessaire pour apprendre) du modèle. Plus il est lent et difficile pour le modèle d'apprendre sur l'ensemble des données d'apprentissage, plus sa structure doit être complexe et plus la méthode PRL sera alors bénéfique sur son entrainement.

3.5 EXPERIENCE 5 (REPRODUCTIBILITE CIFAR10G)

Il s'agit de rejouer les phases 0, 1 et 2 en utilisant un autre corpus de données comme base de construction de l'arbre phylogénétique. Ici le CIFAR10G.

3.5.a EXP 5 - PHASE 0 ET 1 : CONSTRUCTION DU CHAMPION 5 SUR CIFAR10G

Un Champion 5 a été construit pour l'expérience 5 basée sur le corpus de données initial CIFAR10 Grey.

Le champion 5 M^5_c a été utilisé pour refaire les tests d'apprentissage PRL et DDL. Le modèle est particulièrement profond et complexe.

Voici les résultats pour un nombre de test n=10 :

Champion 5 M^5_c généré (CIFAR10G) :
1,018,526 parametres, 42 nodes, 18 layers, 27 générations
Score: **75.37%**

DDL Score moyen du M^{5d}_c - Jeu de données C10G : |
Score : 61.93% EcartType : 1.05%

PRL Score moyen du M^{5p}_c - Jeu de données C10G :
Score : **74.65%** EcartType : **0.49%**

Il est à noter que le score obtenu par un modèle PRL est de 75,30% soit équivalent au score du champion 4 généré par neuroévolution.

Cette expérience confirme les résultats déjà obtenus où l'écart entre PRL et DDL est significatif avec un écart-type beaucoup plus faible pour PRL.

Le score PRL obtenu dans l'expérience 1 était de 65,1% sur CIFAR10G $M^{1.1p}_c$ à partir du modèle Champion 1 M^1_c MNIST qui est proche du DDL mais bien en deçà du PRL.

3.5.b EXP 5 - PHASE 2 : TESTS DE TRANSFERT DE JEU DE DONNEES

- *EXP 5.1 - CHAMPION 5 C10G => JEU DE DONNEES MNIST FASHION*

DDL Score moyen du $M^{5.1d}_c$ – Jeu de données FashM :
Score : 91.29%

PRL Score moyen du $M^{5.1p}_c$ – Jeu de données FashM :
Score : **92.75%**

En utilisant le M^5_c sur le corpus Fashion MNIST, le score moyen obtenu est **92,75% en PRL** et 91,29% en DDL.

Là encore PRL est meilleur que DDL.

Le résultat est également meilleur que pour l'expérience 1 : **91.98% exp1 FMNIST**

- *EXP 5.2 - CHAMPION 5 C10G => JEU DE DONNEES MNIST*

DDL Score moyen du $M^{5.2d}_c$ – Jeu de données MNIST :
Score : 99.22%

PRL Score moyen du $M^{5.2p}_c$ – Jeu de données MNIST
Score : **99.50%**

En utilisant le M^5_c sur le corpus MNIST, le score moyen obtenu est 99,50% et 99,22% en DDL

Le score PRL est dans ce cas impressionnant car c'est le meilleur obtenu de toutes les expériences, y compris sur les champions MNIST 1 à 99.44%

3.5.c EXP 5 : ANALYSE DES IMPACTS DES POIDS INITIAUX

Pour évaluer l'impact des poids initiaux, bien que l'on observe dans l'approche PRL que les poids initiaux concernent moins de paramètres que pour le DDL et ce, avec une prise en compte progressive des nouveaux poids induits par les mutations successives, une autre expérience a été effectuée :

> Pour l'expérience 5 sur PRL, on repart avec le même modèle initial et le même chemin phylogénétique mais pour chaque test, le modèle initial reçoit des poids synaptiques initiaux générés de manière aléatoire. On constate lors de l'expérience que les performances des modèles de champion évolutifs étaient tous très similaires et plus

cohérents qu’en DDL avec l'initialisation de manière aléatoire d'un modèle complet (la Table 17 montre un écart type des 10 tests PRL assez faible).

La Figure 25 montre aussi que non seulement l’apprentissage PRL est plus efficace mais aussi que les poids initiaux interviennent moins dans le résultat. On ne peut tout de même pas statuer définitivement quant à l’impact réel de la méthode sur la réduction de l’aléa des poids initiaux.

3.6 EXPERIENCE 6 (REPRODUCTIBILITE CARTPOLE)

L'expérience 6 est une expérience pour sortir du champ d'application lié à la reconnaissance d'image. Cette expérimentation est un préalable à d'autres tests pour lesquels PRL pourrait aussi s'appliquer. Il n'y a pas de raison apparente pour que tout champ d'application s'appuyant sur des modèles de Réseaux de neurones ne puisse bénéficier de l'approche phylogénétique.

Ici nous sommes sur un apprentissage par renforcement avec des récompenses.

Le but du modèle dans cette expérience est de conserver le plus longtemps possible une barre à la verticale posée sur un chariot ayant des mouvements aléatoires droite / gauche (problématique en 2D). Dans l'expérience, le but opérationnel est de garder la barre verticale, au maximum sur 1000 mouvements du chariot (100% = 1000 mouvements sans tomber).

Le chariot est contrôlé par deux actions possibles +1 ou –1, chacune permettant de pointer sur un déplacement à gauche ou à droite, aléatoirement.

La récompense +1 est donnée à chaque pas de temps si la perche reste droite. Le but est d'empêcher le poteau de tomber (maximiser la récompense totale) comme dans la Figure 21.

L'épisode se termine lorsque le poteau est à plus de 15 degrés de la verticale (considéré comme étant tombé) ou que le chariot se déplace à plus de 2,4 unités du centre ou si le nombre de mouvements maximum autorisé est atteint.

Après obtention d'un modèle champion qui a atteint 1000 mouvements sans tomber au moins une fois :

DDL Scores
Score : 3% Maximum : 13%

PRL Scores
Score : **52 %** Maximum : **100%**

Chaque résultat de N est obtenu avec 10 tentatives.

N=17 soit 170 tentatives en tout.

PRL est dans ce contexte largement meilleur que DDL. Le max du DDL n'atteint pas la moyenne du PRL.

4. SYNTHESE ET DISCUSSION

4.1 SYNTHESE ET REPONSE A LA QUESTION DE RECHERCHE

4.1.a SYNTHESE

A l'issue des 7 expériences, l'approche phylogénétique a montré un intérêt dans tous les cas vis-à-vis de l'approche traditionnelle d'apprentissage par DDL. Pour l'expérience 4, le résultat est similaire mais montre quand même une plus grande cohérence de l'information entre les couches.

Table 15. Synthèse des scores des expériences par corpus

	Initial	Initial	Initial	Transfert	Transfert	Transfert	Transfert	Initial	Transfert	Transfert	Initial	Initial
Corpus	M	M	M	M	FM	FM	FM	C10G	C10G	C10G	TIN	CartPole
Modèle du champion #	1	2	4	5	1	2	5	5	1	2	3	6
Exp.	1	2	4	5.2	1.1	2.1	5.1	5	1.2	2.2	3	6
DDL %moy	98.93	98.90	**99.37**	99.22	90.44	90.71	91.29	61.93	54.40	62.19	38.37	3
PRL %moy	**99.26**	**99.18**	99.33	**99.50**	**91.98**	**92.62**	**92.75**	**74.65**	**65.01**	**67.83**	**41.79**	**52**

Les résultats sont clairement en faveur de l'approche phylogénétique par rapport à la méthode classique d'apprentissage.

Il est à noter que le modèle champion généré lors de l'expérience 5 gagne avec l'apprentissage PRL sur toutes les autres expériences que ce soit pour les résultats sur le corpus initial de l'expérience 5 avec CIFAR 10G mais aussi pour les autres corpus de données utilisés en mode test de transfert.

Est-ce que l'apprentissage PRL sur des corpus complexes est bénéfique ? Oui, néanmoins à ce stade il est impossible d'en tirer une conclusion globale sur l'efficacité systématique de la méthode PRL, car la phase 0 de neuroévolution a peut-être suivi un chemin évolutif favorable à PRL et donc une obtention d'un champion plus performant. Ce n'est donc pas un gage de reproductibilité.

Une recommandation aux experts ou producteurs de modèles de deep learning, serait de ne pas seulement publier leurs modèles finaux mais d'y associer systématiquement toutes les modifications (ordonnées) apportées à ce modèle.

4.1.b REPONSE A LA QUESTION DE RECHERCHE

Rappel de la question de recherche :

Pour résoudre les problèmes posés par les réseaux neuronaux complexes liés à leurs limitations d'apprentissage, pourquoi ne pas utiliser un mécanisme d'apprentissage copiant une fois de plus la nature ? Baptisé Phylogénétique Replay Learning (PRL), il permettrait alors de réduire l'importance de l'effet du gradient évanescent, de réduire l'impact des paramètres initiaux aléatoires sur le résultat final de l'apprentissage, permettant ainsi d'augmenter la capacité d'apprentissage des modèles.

Concernant l'impact du VGE, les résultats des expérimentations confirment, par l'efficacité des modèles PRL, ainsi que par la répartition homogène de l'information sur l'ensemble des couches des modèles montrée par l'entropie de Shannon, la réduction voire la maîtrise de l'impact de l'effet du gradient évanescent lors de l'apprentissage.

Concernant l'impact des poids initiaux, DDL a des résultats inférieurs à PRL mais encore plus vis-à-vis du champion issu de la phase 0. Le VGE est fautif mais les poids initiaux sont aussi à la manœuvre. Il est difficile d'évaluer l'impact de la méthode PRL sur l'absorption des impacts des poids initiaux. Néanmoins l'expérience 7 ainsi que les résultats des autres expériences montrent que le PRL réduit l'aléa induit par de mauvais poids initiaux.

4.1.c UNE MISE EN PERSPECTIVE DE LA RECHERCHE

Sur la base des expériences et des résultats, PRL a systématiquement surpassé le DDL, principalement en atténuant le problème VGE. L'approche phylogénétique PRL agit, comme le montrent les valeurs d'entropie de Shannon calculées pour chaque couche, grâce à un agencement des poids plus homogène dans le modèle final. Ces valeurs d'entropie sont plus faibles dans des couches plus profondes dans les modèles formés par PRL que pour ceux formés par DDL. De surcroit, autre avantage, l'approche phylogénétique PRL est plus résiliente à l'initialisation de poids aléatoires par rapport à celle DDL.

Les entités biologiques qui utilisent toutes le PRL ouvrent donc un chemin de recherche pour les entités neurogénétiques numériques pour lesquelles cette approche semble avoir une efficacité non négligeable dans l'apprentissage de ces réseaux de neurones.

Les expériences sur la transférabilité montrent également que la méthode PRL est efficace pour réutiliser les modèles sur des ensembles de données connexes. Des applications en robotique ou dans l'industrie où les environnements de fonctionnement de ces IA comportent des variations importantes deviennent accessibles. En effet un robot conçu en laboratoire mais qui, au final, doit intervenir dans un environnement où l'intervention humaine est coûteuse (sur la lune, dans une centrale nucléaire, dans des failles sous-marines…) bénéficiera de la méthode PRL car il évoluera et apprendra mieux dans l'environnement cible.

4.1.d LES LIMITES DE LA RECHERCHE

Un certain nombre de points ont été évoqués quant aux limites de cette recherche. Bien que les expérimentations faites soient statistiquement solides, la limitation en ressources de calcul n'a pas permis de travailler sur des modèles beaucoup plus complexes et profonds. L'adage « qui peut le plus peut le moins » pourrait s'appliquer "intuitivement" sur les résultats obtenus en le transformant en un « si l'apprentissage est amélioré sur des modèles simples, cela sera encore plus impactant sur des modèles de nature complexe ». Il restera cependant nécessaire d'effectuer des tests de validation et d'approfondissement sur de gros modèles.

L'efficacité de l'utilisation de PRL sur des méthodes d'apprentissage à base de rétropropagation de gradients est validée, en revanche l'efficacité de PRL sur d'autres méthodes doit encore être étudiée. Déjà l'expérience 7 qui utilise un apprentissage par renforcement semble être prometteuse mais ne confirme cependant rien en ce sens.

4.2 DISCUSSION SUR LES LIMITES

A partir de certains des points soulevés au chapitre précédent, quelques pistes et perspectives suivent pour permettre d'aller plus loin sur ces points.

4.2.a TEMPS D'APPRENTISSAGE

Les algorithmes de neuroévolution sont réputés pour avoir besoin de plus de temps pour obtenir des résultats mais dans les faits, ils en nécessitent beaucoup moins si l'on prend en compte le temps passé par les experts pour concevoir manuellement des topologies spécifiques par coups d'essais-erreurs. Un point méritant d'être précisé bien qu'il ne soit pas le sujet de cette thèse.

Si on se focalise sur le coût de PRL, seul le temps d'apprentissage (aussi appelé coût d'apprentissage) a de l'importance. Analyse :

Dans l'ensemble, le processus PRL nécessite plus d'epochs (19 étapes * 20 epochs au plus dans l'expérience 1), mais les durées des epochs sont plus courtes au cours des premières étapes (de 3 à 15 fois moins longues que les suivantes). La méthode classique DDL a requis au plus 60 epochs lors de nos expérimentations. Il n'est pas rare que les experts fassent 200 ou 300 epochs en forçant les apprentissages pour des gains discutables.

Il existe des techniques complémentaires qui font baisser ces temps d'apprentissage. Des méthodes telles que le freeze de couches (les poids sont figés et ne sont pas modifiés pour les couches non mutées) ou encore l'optimisation du nombre d'epochs …

Concernant l'optimisation du nombre d'epochs lors du processus PRL, des tests préliminaires de PRL avec 4 epochs pendant la première moitié des étapes, suivis d'une augmentation du nombre d'epochs jusqu'à la dernière étape, ont été réalisés. Et les résultats sont toujours restés meilleurs par rapport aux tests avec DDL, tout en diminuant le temps d'apprentissage nécessaire.

Tableau des durées d'apprentissage de la première expérience :

x = durée en seconde d'un epoch.

DDL:	60 epoch*x	= 60.0x
PRL:	(20 epoch*19 step)*(x/3)	=126.6x
PRL avec # d'epochs croissant:	(4 epoch*10 step+12 epoch*9 step)*(x/3)	= 49.3x

PRL est plus long que DDL, mais avec une simple optimisation, on descend en dessous du temps DDL pour un résultat correct.

Nomenclature utilisée : e60p6 veut dire 60 epochs et une patience de 6.

Table 16. Scores par étape PRL avec différentes règles d'apprentissage Expérience 2

Étape	e60p6	e5p3	e(Gen+5)p3	Apprentissage direct pour chaque génération e60p6
0	96,53%	88,25%	90,26%	96,37%
1	97,53%	92,13%	91,90%	96,09%
2	97,62%	92,31%	92,74%	97,38%
3	97,84%	93,35%	96,28%	97,33%
4	98,55%	96,21%	97,80%	98,46%
5	98,62%	97,44%	97,89%	98,47%
6	98,92%	96,44%	97,33%	98,59%
7	98,92%	95,71%	98,04%	98,75%
8	98,83%	64,12%	97,99%	98,63%

9	98,89%	95,48%	98,49%	98,69%
10	98,82%	97,22%	98,71%	98,80%
11	98,81%	69,37%	98,73%	98,87%
12	98,93%	94,90%	99,05%	98,71%
13	98,95%	98,61%	99,15%	98,86%
14	99,17%	92,79%	99,08%	98,92%
15	99,11%	98,33%	99,10%	98,71%
16	99,15%	95,04%	99,06%	98,96%
17	99,22%	96,52%	99,11%	98,96%
18	99,19%	98,46%	99,14%	98,84%
19	99,11%	96,86%	99,12%	98,98%
20	99,17%	98,05%	99,12%	98,93%
21	99,24%	98,70%	99,22%	98,97%
22	99,17%	98,47%	99,24%	98,91%
23	99,21%	98,45%	99,15%	98,90%
24	99,28%	98,36%	99,13%	99,02%
25	99,27%	98,75%	99,21%	98,86%
26	99,24%	98,32%	99,20%	98,97%
27	99,31%	98,79%	99,19%	98,92%
28	99,19%	98,72%	99,17%	98,83%

Table 16, qui reprend les résultats de l'expérience 2, à part en bridant violement à 5 epochs par étape, le nombre d'epochs impacte peu sur le résultat final, le maximum de 99,24% est atteint également.

L'efficacité par cette piste d'amélioration de l'apprentissage phylogénétique est montrée par le test avec nombre d'epochs progressif. On obtient des résultats équivalents tout en réduisant le nombre d'epochs total à faire.

La piste du regroupement d'étapes permettrait aussi de réduire le temps d'apprentissage en réduisant tout simplement le nombre d'étapes.

D'autres expériences pourraient être menées pour optimiser le temps d'apprentissage.

4.2.b REPARTITION DE L'INFORMATION

L'utilisation de l'entropie de Shannon pour évaluer la répartition de l'information au sein des couches des modèles, est une piste intéressante. Les résultats restent néanmoins difficiles à interpréter. Dire que plus petit implique plus d'information est un raccourci. Cela montre une plus grande organisation des informations s'éloignant du hasard pur mais jusqu'à quel point ? La

comparaison entre couche montre d'ailleurs que la comparaison ne peut se faire que dans la globalité ou pour les mêmes types de couches. On a des couches avec une entropie de 8 et d'autres de 12 dans le même modèle.

Dans tous les cas cette mesure montre que la méthode PRL obtient un effet plus grand sur les couches proches des senseurs/entrées que l'approche d'apprentissage direct.

Cette entropie pourrait peut-être être utilisée lors de la phase 1 pour faire une pression sélective sur les modèles générés.

D'autres méthodes d'évaluation de l'efficacité pourraient être imaginées. La loi de Benford [39] non vérifiée mais utilisée pour détecter des fraudes bancaires par exemple, pourrait aussi être un indicateur des modèles obtenus pour savoir s'ils suivent la loi (répartition naturelle) ou non (répartition « artificielle »).

4.2.c OPTIMISATION DES ETAPES

Une question a émergé lors des différentes expérimentations : certaines étapes obtiennent des régressions de score par rapport aux précédentes. La Table 17, étape 2, 9 et dans une moindre mesure étape 18 montrent ce fait.

De surcroît certaines étapes ont des gains faibles (11, 12…).

De même les 3 premières étapes montrent que DDL est plus efficace. Le regroupement de ces 3 étapes seraient peut-être intéressantes à évaluer.

Un autre point à noter est le delta de score entre PRL et DDL qui bien que croissant est, pour certaines étapes, plus important. L'analyse des couches et le delta associé permettraient aussi de voir les types de mutations ou de structurations qui impactent les modèles et d'en déduire des pistes d'amélioration.

Certaines étapes pourraient donc être fusionnées et probablement d'autres dédoublées. Ceci est une intuition qui nécessiterait de plus amples recherches pour avancer sur les points soulevés.

Table 17. Détail des scores de l'expérimentation 5 par étape sur 10 tests

15788334420587607O4				C10G 28,28,1		e20p3 : 20 epoch patience 3 val_los				Stochastic initial weight			max :	75.30%	direct	
Mutant List			e20p3	Val Acc									ave :	74.65%	max	
Parameters	Generation	Ecartype	Original Fit	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9	Test10	Average	*e60p6*	delta Max
69 842	0	0.49%	57.05%	57.05%	57.14%	56.78%	57.32%	56.78%	57.78%	57.67%	57.82%	56.93%	56.34%	57.16%	*59.16%*	*-1.34%*
69 474	1	0.46%	63.83%	62.77%	63.91%	63.78%	63.40%	63.07%	63.34%	62.46%	62.91%	63.26%	62.73%	63.16%	*63.54%*	*0.37%*
30 754	2	0.69%	58.66%	59.36%	59.71%	58.94%	57.84%	58.76%	59.10%	59.73%	59.44%	58.24%	58.02%	58.91%	*60.71%*	*-0.98%*
30 744	3	0.64%	61.60%	61.03%	62.59%	61.42%	61.75%	61.77%	61.35%	62.88%	61.50%	61.38%	60.83%	61.65%	*60.81%*	*2.07%*
30 744	4	0.48%	62.01%	61.90%	62.28%	62.19%	61.87%	62.25%	62.93%	63.08%	62.71%	61.62%	62.64%	62.35%	*60.77%*	*2.31%*
55 880	5	0.77%	65.14%	64.53%	66.19%	65.98%	65.27%	65.80%	66.25%	67.12%	66.00%	64.97%	65.01%	65.71%	***64.33%***	*2.79%*
67 208	6	0.91%	66.39%	66.74%	67.53%	67.17%	66.10%	66.98%	67.69%	68.85%	66.91%	66.29%	65.64%	66.99%	*62.74%*	*6.11%*
87 688	7	0.57%	67.74%	67.52%	69.11%	68.72%	67.86%	68.25%	67.90%	68.26%	68.07%	67.51%	67.25%	68.04%	*62.62%*	*6.49%*
67 208	8	0.63%	68.53%	67.60%	69.48%	68.77%	67.84%	68.33%	68.40%	68.18%	68.70%	67.72%	67.41%	68.24%	*63.74%*	*5.74%*
51 848	9	0.40%	67.57%	66.70%	66.68%	66.84%	66.92%	66.76%	66.52%	67.60%	67.60%	66.78%	66.50%	66.89%	*61.78%*	*5.82%*
67 218	10	0.69%	68.92%	67.33%	69.09%	68.01%	69.05%	69.16%	68.19%	67.64%	68.45%	67.66%	67.62%	68.22%	*62.74%*	*6.42%*
102 002	11	0.45%	70.46%	69.66%	70.68%	69.70%	70.32%	69.51%	70.29%	69.92%	70.05%	69.09%	69.89%	69.91%	*61.64%*	*9.04%*
115 954	12	0.61%	70.75%	69.48%	70.88%	70.96%	70.49%	71.26%	70.84%	70.10%	71.41%	70.08%	70.15%	70.56%	*62.50%*	*8.91%*
183 586	13	0.48%	71.26%	71.03%	71.03%	71.79%	70.83%	71.10%	70.83%	70.75%	71.92%	70.35%	70.75%	71.04%	*61.69%*	*10.23%*
219 426	14	0.41%	71.31%	71.27%	71.26%	71.90%	70.87%	70.31%	70.98%	71.21%	71.16%	70.87%	70.85%	71.07%	*62.67%*	*9.23%*
482 562	15	0.46%	72.86%	71.95%	72.72%	71.74%	72.07%	72.36%	72.49%	72.15%	72.33%	71.04%	72.17%	72.10%	*62.21%*	*10.51%*
484 098	16	0.48%	72.93%	72.78%	72.32%	72.71%	73.21%	72.64%	71.93%	72.98%	72.53%	71.70%	72.02%	72.48%	*62.88%*	*10.33%*
484 098	17	0.47%	73.01%	72.34%	72.96%	73.28%	73.37%	72.60%	72.27%	73.45%	72.98%	72.46%	72.25%	72.80%	*63.81%*	*9.64%*
500 482	18	0.57%	72.34%	72.42%	71.63%	72.18%	71.58%	72.62%	71.98%	72.55%	72.15%	71.31%	70.89%	71.93%	*62.73%*	*9.89%*
776 574	19	0.65%	73.05%	72.76%	73.60%	72.99%	72.13%	73.33%	73.48%	73.46%	72.83%	72.69%	71.54%	72.88%	*62.23%*	*11.37%*
1 001 758	20	0.43%	73.20%	73.35%	73.76%	73.64%	73.25%	73.61%	73.72%	73.62%	73.88%	73.21%	72.39%	73.44%	*60.92%*	*12.96%*
1 003 454	21	0.35%	73.73%	73.74%	73.55%	73.09%	73.45%	73.83%	74.06%	73.68%	73.56%	73.13%	72.99%	73.51%	*59.93%*	***14.13%***
1 071 070	22	0.43%	73.97%	73.49%	74.36%	73.83%	73.75%	73.10%	73.59%	74.23%	73.41%	73.12%	73.31%	73.62%	*61.05%*	*13.31%*
906 494	23	0.56%	74.16%	73.65%	74.61%	73.51%	73.85%	73.21%	74.19%	74.60%	73.29%	73.02%	73.61%	73.75%	*62.12%*	*12.49%*
906 494	24	0.50%	74.66%	73.58%	74.65%	73.80%	**74.47%**	74.29%	74.81%	74.60%	74.04%	73.78%	73.36%	74.14%	*61.88%*	*12.93%*
909 182	25	0.42%	74.80%	74.20%	**75.30%**	73.77%	74.30%	74.27%	73.86%	73.99%	74.20%	74.04%	**74.12%**	74.20%	*63.43%*	*11.87%*
909 390	26	0.42%	75.00%	**74.22%**	74.92%	74.25%	74.13%	**74.45%**	74.64%	74.84%	**74.80%**	73.59%	74.05%	74.39%	*62.15%*	*12.77%*
1 018 526	27	0.49%	**75.37%**	73.67%	74.38%	**75.06%**	74.24%	74.34%	**75.10%**	**74.96%**	74.49%	**74.05%**	73.89%	**74.42%**	*63.31%*	*11.79%*

Cette table montre aussi que les apprentissages directs sont d'autant moins bons que le nombre d'étapes augmente ce qui confirme une nouvelle piste à potentiellement suivre : démontrer que plus le problème à traiter sera complexe, plus l'intérêt de l'apprentissage phylogénétique sera important.

4.2.d COMPLEXITE DU MODELE

Comme vu plus haut, la méthode de construction du modèle permet aussi d'obtenir des modèles dont le DDL est proche du PRL, voire du champion.

Un modèle dont la pression sélective a privilégié, lors de la phase 0 de neuroévolution, la rapidité (synonyme de facilité d'apprentissage ?) d'exécution semble moins bénéficier de la méthode PRL. La complexité topologique du modèle ne semble pas à elle seule être un critère suffisant pour que PRL soit efficace. Les modèles utilisés dans cette thèse ne sont donc pas assez complexes pour contrebalancer une topologie ayant une vitesse d'exécution optimisée.

Il conviendrait de poursuivre des recherches, avec plus de puissance de calcul, par exemple, pour l'obtention de modèles très complexes de type GPT3 en appliquant une pression sélective sur la rapidité d'exécution. L'objectif est ainsi de pouvoir mieux définir les périmètres d'efficacité de PRL.

La vitesse d'exécution d'un modèle semble être corrélé avec la facilité d'apprentissage et donc lié à sa plus faible sensibilité à l'évanescence du gradient. Le VGE reste néanmoins un problème.

4.2.e RETROPROPAGATION DE GRADIENT ?

L'approche phylogénétique est efficace sur des modèles utilisant comme méthode d'apprentissage la rétropropagation de gradient (méthode la plus utilisée). Est-ce que cela serait aussi efficace avec une autre optimisation de poids comme, par exemple, l'apprentissage mémétique qui est moins affecté de par conception, par l'évanescence du gradient ?

L'expérience 6 utilisant un apprentissage par renforcement semble néanmoins bénéficier de la méthode PRL.

Dans ce cas la réduction de l'impact des poids aléatoires peut raisonnablement être invoquée. Cela reste toutefois à confirmer.

4.2.f METHODE DE TRAVAIL DES EXPERTS IA

La méthode PRL nécessite de connaître le chemin phylogénétique, l'utilisation de la neuroévolution est certes intéressante mais ne permet pas d'utiliser des modèles existants.

L'efficacité de la PRL démontre que les experts devraient noter les modifications qu'ils appliquent, au fur et à mesure, de la construction de leurs modèles. Si, en trouvant le modèle d'un confrère sur internet, il y avait également son modèle initial accompagné de la liste des modifications apportées, il serait alors possible d'appliquer le PRL.

CONCLUSION ET PERSPECTIVES

En conclusion, l'intuition initiale que la transposition, depuis le monde biologique, de la programmation génétique de l'évolution de la topologie du système nerveux central au cours de la croissance de l'individu, au monde numérique, semble avoir des bénéfices intéressants en termes d'apprentissage.

Cette thèse montre que le modèle d'IA/ML apprend mieux que l'état de l'art et que la méthode phylogénétique non seulement, ne remet pas en question les méthodes d'apprentissage existantes, mais en sont complémentaires. Il y a quelques applications concrètes qui peuvent découler de cette approche comme des systèmes dont le modèle d'IA devra être déployé et opéré dans des environnements très disparates voir inconnus. Des environnements où l'adaptation rapide devient fondamentale et où un processus de recherche évolutive des possibles serait trop lent et trop coûteux et les méthodes d'apprentissage traditionnelles pas assez adaptables. Exemple en robotique dans des environnements variés et extrêmes, ou encore pour des chaudières industrielles devant aussi bien s'adapter à un magasin, qu'à une usine…

Les résultats obtenus sont donc prometteurs et ouvrent une voie intéressante pour la recherche en IA/ML d'autant que l'obésité des modèles produits actuellement montrent qu'on atteint des limites dans les apprentissages bruts tant en termes de taille qu'en termes d'efficacité d'apprentissage.

Cette thèse a pour ambition d'ouvrir la porte à des recherches supplémentaires sur l'utilisation de la méthode PRL où un modèle avec son chemin phylogénétique peut être efficacement appliqué sur un nouvel ensemble de données ou encore une version mise à jour du même ensemble de données. Des recherches supplémentaires seraient nécessaires dans ce domaine.

L'utilisation de la méthode phylogénétique, où l'évolution programmée du modèle est combinée avec l'entrainement, produira des systèmes plus performants, par rapport aux systèmes où le modèle est entrainé directement (DDL).

Il est important pour les experts que les modifications effectuées soient stockées et historisées pour permettre de reproduire l'arbre phylogénétique et ainsi pouvoir l'utiliser pour bénéficier du réel potentiel de la méthode PRL.

Cette méthode pourrait être particulièrement efficace dans la formation de modèles très profonds et très complexes. De futurs travaux pourraient donc se concentrer sur l'expansion et l'exploration de cette méthode.

L’ouverture pour de nouvelles recherches est essentielle, car il reste un grand nombre de questions à résoudre :

a) Coûts de la méthode PRL ?
b) Comment évaluer la répartition d’information des couches ?
c) Doit-on regrouper des mutations sur une même étape pour en réduire le nombre ?
d) À partir de quel stade de maturité du modèle M_0, doit-on appliquer l’apprentissage phylogénétique (quel modèle initial choisir) ?
e) A quelle limite d’apprentissage (nombre d’epochs), doit-on s’arrêter pour chaque étape de mutation ?
f) A partir de quel seuil de complexité du modèle obtenu, la méthode PRL peut-elle être considérée comme pertinente ?
g) Est-ce que le *freeze* des couches non mutées accélère le processus PRL ?
h) Est-ce que la PRL s’applique à des modèles n’utilisant pas la rétropropagation de gradient ?
i) Quelle variation du champ d’application, la transférabilité supporte-t-elle ?

Etc.

Les expériences ont permis de répondre à certaines des questions sur l’intérêt de l’approche phylogénétique sur des modèles de DL. En revanche, elles soulèvent de nouvelles interrogations mais cette fois, sur son périmètre d’utilisation.

Un point d’étude potentielle : le coût d’apprentissage. C’est un élément important aujourd’hui car lié à la consommation énergétique des modèles. Au niveau de l’apprentissage, l’approche PRL semble équivalente ou légèrement plus coûteuse que la méthode DDL classique (voir le point sur le Temps d’apprentissage).

Toujours lié au coût, l’obtention du chemin phylogénétique peut être coûteux selon la méthode utilisée. Il a été choisi ici la neuroévolution pour le faire, ce qui n’est pas forcément plus coûteux que des techniques d’AutoML classiques (sans processus évolutif) utilisées par les experts dans les approches DDL traditionnelles.

Un second point assez important pour mériter réflexion est l’évaluation de la répartition de l’information au sein du modèle de deep learning. L’entropie de Shannon a été utilisée ici pour l’évaluer. D’autres méthodes seraient peut-être plus appropriées.

Le quatrième point à examiner de plus près est la pertinence de mutations successives. En effet le système de neuroévolution a généré des mutations aléatoires qui ont été conservées car cela donnait un avantage compétitif. Si l'on examine les gains de score ramenés par étape de mutation, il est probable que l’on puisse regrouper certaines étapes de mutations.

Un autre point à approfondir : déterminer à partir de quelle mutation, il pourrait être utile de lancer la méthode PRL. Ici, on a fait le choix de démarrer avec le modèle initial mais comme le montre la Table 17, on voit que l'avancée ne se fait pas forcément dès la première mutation mais à partir seulement de la 4ème voire de la 5ème mutation.

GLOSSAIRE

Apprentissage ou entrainement : Algorithme dont l'action est de faire « apprendre » un modèle (ajuster le graphe de neurones le composant) de façon à atteindre un objectif déterminé, généralement un comportement souhaité (par exemple, reconnaître une catégorie d'images). Généralement, l'apprentissage démarre avec un jeu de données lu en boucle (voir epoch) par un algorithme dit d'apprentissage, ce dernier s'appuyant sur un framework de réseaux de neurones.

Couche : Une couche représente une brique de construction d'un modèle. Elles sont disponibles au sein du catalogue du framework de réseau de neurones utilisé. Par exemple couche de convolution, couche dense, couche LSTM… chacune apportant au modèle une caractéristique mathématique, en général, spécifique. La topologie d'un modèle donné est constituée d'un ensemble de couches liées.

Cycle : Le numéro du cycle (en neuroévolution) d'un modèle représente le moment d'apparition de ce modèle précis. Un modèle de cycle 5 est apparu après 5 itérations du processus de neuroévolution. A ne pas confondre avec la génération. Un modèle qui est apparu par exemple dans le cycle 8 peut être seulement de génération 3 (M_3).

Direct Deep Learning (DDL) : est le nom donné dans le cadre de cette thèse à la méthode d'apprentissage standard d'un modèle, généralement utilisée par les experts en datascience. L'expert récupère un modèle existant ou en fabrique un, et lui fait apprendre un comportement, à partir d'un jeu de données (par exemple, reconnaître une image…) et d'un algorithme d'apprentissage adapté. Cette méthode d'apprentissage implique que le modèle utilisé possède une topologie (structuration des couches) fixe dont seuls les poids (représentation numérique de l'importance d'un lien entre deux neurones) sont modifiés lors de l'apprentissage. L'historique de construction du modèle (comment il a été conçu, pourquoi sa topologie a cette forme et comment on est arrivé à cette architecture de réseaux de neurones) n'est généralement pas connu, ni exploité.

Epoch : représentation technique en datascience utilisée lors de l'apprentissage d'un modèle. Il s'agit de l'action de faire lire complètement un jeu de données d'apprentissage à un modèle pour qu'il apprenne le comportement attendu. Lorsque l'on parle de 5 epochs, le jeu de données a été lu 5 fois pour apprendre (similaire à lire 5 fois un texte pour l'apprendre par cœur).

Etape : concept utilisé dans le cadre de la méthode PRL. C'est le passage d'un modèle M_n (existant à l'étape n) avec une topologie donnée, à un nouveau modèle $M_{n+1,}$ avec une topologie différente à l'étape n+1. Le modèle à l'étape n+1 pouvant comporter une à plusieurs mutations par rapport au modèle à l'étape n. L'étape correspond ainsi à l'état topologique d'un modèle

pendant son apprentissage PRL. Ce terme “étape” sera systématiquement utilisé dès qu’il s’agira de décrire le chemin évolutif programmé dans la méthode PRL. En effet, lors d’un processus de type PRL, le mécanisme correspond plus à une planification d’une série d’étapes codées génétiquement pour arriver à un modèle final Champion en partant d’un modèle initial donné. Dans un contexte de neuroévolution, le terme employé entre deux modèles successifs, Mn et Mn+1, est “génération”.

Framework de réseaux de neurones : Il s’agit d’une plateforme logicielle dont le rôle est de fournir des services pour définir la topologie d’un modèle de réseaux de neurones et de le construire. Elle permet également de l’utiliser : entraînement dans un contexte donné avec un jeu de données et enfin l’exploitation des modèles créés. Il en existe plusieurs, commerciaux ou open source. Dans le cadre de cette thèse, le choix s’est porté sur TensorFlow de Google, le plus utilisé à l’époque.

Génération : dans cette thèse, il sera considéré qu’en neuroévolution, la génération n du modèle considéré M_n représente le nombre d’ancêtres depuis le modèle initial M_0. Un modèle de génération 1 (M_1) descend directement du modèle initial M_0, alors qu’un modèle de génération 5 appelé M_5 descend également de M_0 mais en passant par 5 évolutions successives ($M_5 = M_{(4)+1} = M_{(((((0)+1)+1)+1)+1)+1}$*). Une évolution consistant à une ou plusieurs mutations. En neuroévolution il est normal que 2 modèles de génération 5 n’aient pas la même topologie car il est très peu probable qu’ils aient suivi les mêmes évolutions. Il est à préciser que la notion du n pour le modèle considéré M_n lorsque que l’on utilise la méthode PRL (chemin évolutif préenregistré) représente le nombre d’étapes nécessaires pour atteindre la topologie du modèle considéré M_n. Dans ce cadre, contrairement au contexte de neuroévolution, il n’est pas possible d’avoir 2 modèles avec une topologie différente lorsque l’on est à une étape n donnée car on utilise un même chemin évolutif unique programmé (phylogénétique) pour faire évoluer le modèle depuis M_0 jusque M_n.

* Suite arithmétique de raison 1 partant de M_0, $M_i = M_{(i-1)+1}$ pour i= 1 à n, i représentant une génération.

Hall of Fame : HOF en abrégé, est la liste des dix modèles de réseaux de neurones les plus performants que l’algorithme de neuroévolution a généré pendant la phase 0 (recherche du modèle Champion). Cette liste est utilisée par l’algorithme de neuroévolution pour construire les modèles de génération suivante et également pour comparer leur efficacité par rapport à ceux contenus dans la liste. Si l’un des modèles de la nouvelle génération est considéré par l’algorithme comme plus performant selon les critères de sélection mis en place, il vient en remplacer un moindre de cette liste.

Modèle IA : est dans le cadre de cette thèse, la représentation d'un réseau de neurones numérique avec des caractéristiques propres : organisation topologique des neurones numériques entre eux, poids, couches, input (format des données entrées), output (format de données sorties) … Il est parfois appelé "Agent" dans l'algorithme de neuroévolution.

Modèle Champion : est un modèle M_c représentant le meilleur modèle (optimum) que la neuroévolution a produit lors de la phase 0 pour résoudre un problème donné. Ce modèle champion sert d'étalon topologique dans les expériences de DDL et PRL

Modèle Initial : est le modèle M_0 utilisé comme point de départ pour la recherche des meilleurs modèles en neuroévolution. En général, c'est un modèle que l'on pourrait qualifier de "primitif" dans le sens où il possède peu de neurones et couches.

Mutant : est un modèle M_n ayant subi une ou des mutations qui le différencient du modèle parent M_{n-1} dont il est issu.

Mutations : à chaque génération du processus évolutif ou étape de la méthode PRL, une ou plusieurs mutations sont appliquées à la topologie d'un modèle parent M_n afin de créer un modèle enfant/mutant M_{n+1}. Une mutation topologique peut ajouter une couche au modèle, modifier les paramètres d'une couche existante, supprimer une couche, cloner une couche existante, ajouter, supprimer ou modifier un lien entre deux couches ou encore transformer une couche contre un autre type de couche…

Neuroévolution : le processus neuroévolutif est une mise en œuvre d'algorithmes génétiques pour générer de meilleurs modèles d'intelligence artificielle selon les objectifs à atteindre (par exemple, reconnaissance de forme, génération de langage…) en simulant un processus d'évolution « naturelle ». La neuroévolution n'est utilisée dans cette thèse que pour définir le chemin évolutif (phylogénétique) au moment de la phase 0 de la méthodologie de la thèse.

NN ou RNN : Réseau de neurones. Neural Network en anglais. Voir modèle IA.

Patience : c'est un paramètre du framework qui permet lors de la phase d'apprentissage de déterminer a quel moment on arrête l'apprentissage. Cela permet d'éviter que le modèle apprenne trop par cœur.

Phylogenetic Replay Learning (PRL) : c'est le nom choisi pour définir la méthode créée et évaluée lors de cette thèse. C'est une méthode d'apprentissage qui imite l'évolution structurelle d'un cerveau biologique au cours de sa croissance. Un cerveau biologique possède une topologie génétiquement programmée pour évoluer au cours du temps. C'est une programmation qui semble faciliter l'apprentissage du cerveau. L'approche PRL copie ce mécanisme de programmation en appliquant un chemin évolutif planifié de la topologie à un modèle initial M_0 dans le but d'obtenir un modèle Champion M_c.

Processus de sélection : est le mécanisme grâce auquel l'algorithme de neuroévolution sélectionne les meilleures entités M_c en fonction de leur score (fonction fitness) et les stocke dans la liste "*Hall of Fame*" (HOF).

Rétropropagation de Gradients : algorithme d'apprentissage d'un modèle de réseau de neurones numériques profond (dit de *deep learning*). Après avoir calculé le résultat de la lecture d'une donnée par le modèle, pour se rapprocher du résultat attendu, il s'agit de propager une éventuelle correction des poids du modèle (Gradient d'erreur) couche par couche en remontant le modèle depuis son output vers l'input (Rétropropagation).

Topologie : La topologie du réseau de neurones d'un modèle représente son architecture (sous forme de graphe). C'est-à-dire le nombre de neurones, la manière dont ils sont reliés, leur méthode de fonctionnement… Dans cette thèse, les poids des liens entre les neurones du réseau ne sont pas traités comme faisant partie de la topologie impactée par les mutations. Ces poids ne sont traités que lors de l'apprentissage du modèle par le framework de réseaux de neurones.

Transfert learning : service du framework de réseaux de neurones qui consiste dans la reprise des poids issus de l'apprentissage d'un modèle pour l'appliquer à un autre modèle. Dans le cadre de cette thèse, il est utilisé pour transférer l'apprentissage du modèle parent M_n et l'appliquer au modèle enfant M_{n+1}. Il est à préciser que les éléments (couches, liens…) du modèle M_n ayant subi des mutations doivent, quant à eux, être initialisés avec des poids aléatoires dans le modèle M_{n+1} car ils n'existaient pas dans le modèle parent.

BIBLIOGRAPHIE

[1] Hsu, Feng-hsiung (2002). "Behind Deep Blue: Building the Computer that Defeated the World Chess Champion". Princeton University Press. ISBN 0-691-09065-3.

[2] D. Silver, David et al.(2016), "Mastering the Game of Go with Deep Neural Networks and Tree Search," Nature, vol. 529, pp. 484-489, DOI: 10.1038/nature16961.

[3] F. Gomez, J. Schmidhuber and R. Miikkulainen (2008), "Accelerated Neural Evolution through Cooperatively Coevolved Synapses," Journal of Machine Learning Research, vol. 9, pp. 937-965, DOI: 10.1145/1390681.1390712.

[4] Bender et al (2021), “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?” In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT '21). Association for Computing Machinery, New York, NY, USA, 610–623. https://doi.org/10.1145/3442188.3445922

[5] B. Cessac, T. Viéville, C. Leininger (2007), https://interstices.info/le-cerveau-est-il-un-bon-modele-de-reseau-de-neurones/

[6] Hornik K, Stinchcombe M, White H (1989) Multilayer Feedforward Networks are Universal Approximators. Neural Networks 2, 359-366.

[7] Haykin S (1999) Neural Networks: A Comprehensive Foundation J. Griffin, ed. (Prentice Hall).

[8] Koza JR (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection. (MIT Press), ISBN 0262111705.

[9] Holland JH (1975) Adaptation in Natural and Artificial Systems J. H. Holland, ed. (University of Michigan Press).

[10] X. Zhang, J. Clune and K. Stanley (2017), "On the Relationship between the OpenAI Evolution Strategy and Stochastic Gradient Descent", arXiv: 1712.06564, DOI: 10.48550/arXiv.1712.06564.

[11] J. Lehman, J. Chen, J. Clune and K. Stanley (2018), "ES Is More Than Just a Traditional Finite-difference Approximator," Proc. of the Genetic and Evolutionary Computation Conference (GECCO '18), pp. 450- 457, DOI: 10.1145/3205455.3205474.

[12] E. Conti, Edoardo et al.(2017), "Improving Exploration in Evolution Strategies for Deep Reinforcement Learning *via* a Population of Novelty-seeking Agents," Proc. of the 32nd Int. Conf. on Neural Information Processing Systems (NIPS'18), pp. 5032–5043.

[13] F. Such et al.(2017), "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning" arXiv, DOI: 10.48550/arXiv.1712.06567.

[14] R. De Nardi, J. Togelius, O. Holland and S. Lucas (2006), "Evolution of Neural Networks for Helicopter Control: Why Modularity Matters," Proc. of the IEEE Int. Conf. on Evolutionary Computation, pp. 1799-1806, DOI: 10.1109/CEC.2006.1688525, Vancouver, Canada.

[15] V. Heidrich-Meisner, C. Igel, B. Hoeffding and Bernstein (2009), "Races for Selecting Policies in Evolutionary Direct Policy Search," Proc. of the 26th Annual Int. Conf. on Machine Learning (ICML '09), vol. 51, DOI: 10.1145/1553374.1553426.

[16] J. Lehman et al. (2020), "The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities," Massachusetts Institute of Technology, Artificial Life, vol. 26, no. 2, pp. 274–306.

[17] P. Vikhar (2016), "Evolutionary Algorithms: A Critical Review and Its Future Prospects," Proc. of the IEEE Int. Conf. on Global Trends in Signal Process., Inf. Comp. and Comm. pp. 261-265, Jalgaon, India.

[18] P. Mengal (1993), Histoire du concept de récapitulation : ontogenèse et phylogenèse en biologie et sciences humaines, (Masson)

[19] L. Krubitzer, L Lefebvre (2010),
https://lecerveau.mcgill.ca/flash/i/i_05/i_05_cr/i_05_cr_her/i_05_cr_her.html,
https://lecerveau.mcgill.ca/flash/a/a_05/a_05_cr/a_05_cr_her/a_05_cr_her.html

[20] S. Hochreiter (1991), Untersuchungen zu dynamischen neuronalen Netzen, Diploma Thesis, Josef Hochreiter Institut fur Informatik, Technische Universitat Munchen, Germany.

[21] F. Informatik, Y. Bengio, P. Frasconi and J. Schmidhuber Jfirgen (2003), "Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies," Chapter of Book: A Field Guide to Dynamical Recurrent Neural Networks, pp. 237 – 243, DOI: 10.1109/9780470544037.ch14, IEEE Press.

[22] Y. Bengio, P. Simard and P. Frasconi (1994), "Learning Long-term Dependencies with Gradient Descent Is Difficult," IEEE Transactions on Neural Networks, vol. 5, pp. 157-166, DOI: 10.1109/72.279181.

[23] R. Pascanu, T. Mikolov and Y. Bengio (2013), "On the Difficulty of Training Recurrent Neural Networks," Proc. of the 30th Int. Conf. on Machine Learning, JMLR: W&CP, vol. 28, Atlanta, Georgia, USA.

[24] X. Glorot, A. Bordes and Y. Bengio (2011), "Deep Sparse Rectifier Neural Networks," Proc. of the 14th Int. Conf. on Artificial Intelligence and Statistics, vol. 15, pp. 315-323, Fort Lauderdale, FL, USA.

[25] Y. Lecun, L. Bottou, G. Orr and K.-R. Müller (1998), "Efficient BackProp," Chapter in Book: Neural Networks: Tricks of the Trade, vol. 7700, pp. 9-48, DOI: 10.1007\/3-540-49430-8_2.

[26] X. Glorot and Y. Bengio (2010), "Understanding the Difficulty of Training Deep Feedforward Neural Networks," Journal of Machine Learning Research, vol. 9, pp. 249-256.

[27] S. Ioffe and C. Szegedy (2015), "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:448-456, https://proceedings.mlr.press/v37/ioffe15.html.

[28] Y. Lecun et al.(1989), "Backpropagation Applied to Handwritten Zip Code Recognition," Neural Computation, vol. 1, pp. 541-551, DOI: 10.1162 neco.1989.1.4.541.

[29] K. He, X. Zhang, S. Ren and J. Sun (2016), "Deep Residual Learning for Image Recognition," Proc. of the IEEE Conf. on Comp. Vision and Pattern Recog. (CVPR), pp. 770-778, DOI: 10.1109 CVPR.2016.90.

[30] H. Noh, T. You, J. Mun and B. Han (2017), "Regularizing Deep Neural Networks by Noise: Its Interpretation and Optimization," Proc. of the 31st Conf. on Neural Inf. Process. Sys. (NIPS), Long Beach, USA.

[31] S. Enrique, J. Hare and M. Niranjan (2018), "Deep Cascade Learning," IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 11, pp. 5475 – 5485, DOI: 10.1109/TNNLS.2018.2805098.

[32] J. Metzen, M. Edgington, Y. Kassahun and F. Kirchner (2008), "Performance Evaluation of EANT in the Robocup Keepaway Benchmark," Proc. of the 6th Int. Conf. on Machine Learning and Applications (ICMLA 2007), pp. 342-347, DOI: 10.1109/ICMLA.2007.23.

[33] K. Stanley and R. Miikkulainen (2002), "Evolving Neural Networks through Augmenting Topologies," Evolutionary Computation, vol. 10, pp. 99-127, DOI: 10.1162/106365602320169811.

[34] E. Real, A. Aggarwal, Y. Huang and Q. Le (2018), "Regularized Evolution for Image Classifier Architecture Search," Proc. of AAAI Conf. on Artificial Intellig., vol. 33, DOI: 10.1609/aaai.v33i01.33014780.

[35] A. Gaier and D. Ha (2019), "Weight Agnostic Neural Networks," arXiv: 1906.04358, DOI: 10.13140/RG.2.2.16025.88169.

[36] C. Shannon and W. Weaver (1963), The Mathematical Theory of Communication, Note 78, p. 44.

[37] J. Schmidhuber (1992), "Learning Complex, Extended Sequences Using the Principle of History Compression," Neural Computation, vol. 4, pp. 234-242, DOI: 10.1162/neco.1992.4.2.234.

[38] O. Granmo et al. (2019), "The Convolutional Tsetlin Machine," arXiv: 1905.09688v5, DOI: 10.48550/arXiv.1905.09688.

[39] Frank Benford (1938), « The law of anomalous numbers », Proceedings of the American Philosophical Society, vol. 78, p. 551-572.

[40] Vecoven N, Ernst D, Wehenkel A, Drion G (2020), “Introducing neuromodulation in deep neural networks to learn adaptive behaviours”. PLOS ONE 15(1): e0227922. https://doi.org/10.1371/journal.pone.0227922.

[41] Glafkides JP, Sher G, Akdag H (2022), "PHYLOGENETIC REPLAY LEARNING IN DEEP NEURAL NETWORKS", Jordanian Journal of Computers and Information Technology (JJCIT) ,Volume 08, Number 03, pp. 218 - 231, doi: 10.5455/jjcit.71-1643583878

LISTE DES TABLEAUX ET FIGURES

TABLEAUX :

FIGURES :

ANNEXES

ANNEXE 1 : DESCRIPTION D'UN APPRENTISSAGE DLL AU QUOTIDIEN D'UN EXPERT

Un expert en apprentissage profond travaille généralement soit sur des modèles déjà existants, soit créé ses propres modèles en s'appuyant sur les meilleures pratiques en vigueur ou des travaux préexistants.

Utilisation de Modèles Existants

Dans le cas de l'utilisation de modèles existants, l'expert commence par chercher des travaux pertinents en ligne, souvent dans des publications scientifiques, des catalogues de modèles comme Hugging Face ou des plateformes de compétitions comme Kaggle. Une fois un modèle approprié trouvé, l'expert l'adapte à son propre cas d'usage. Si les besoins sont très similaires, une simple personnalisation, ou "fine tuning", peut être suffisante.

Création de Modèles

En revanche, lorsqu'il crée un modèle de zéro ou lorsque le modèle existant ne répond pas aux besoins, l'expert se trouve face à un manque d'historique de la construction et de l'évolution topologique du modèle. Cet historique pourrait inclure les changements dans la structure du réseau (ajout de couches, changement de fonctions d'activation, etc.), ce qui peut être précieux pour comprendre le comportement du modèle.

Apprentissage du Modèle

L'expert procède ensuite à l'entraînement du modèle sur son propre jeu de données. Les méthodes peuvent varier, allant du transfert de connaissances ("transfer learning") à l'apprentissage à partir de zéro. Le protocole DDL sert à guider ce processus lorsque l'évolution topologique du modèle n'est pas envisagée, c'est-à-dire lorsque la structure du réseau est fixée dès le départ.

Importance de l'Histoire de Construction

Ce que cette thèse démontre, c'est que l'historique de construction et d'évolution topologique des modèles, bien qu'habituellement négligé, peut être très utile pour comprendre la performance et les limitations des modèles, et pourrait donc être précieusement conservé…

ANNEXE 2 : MODELES PHYLOGENETIQUE DU CHAMPION 2 DE L'EXPERIMENTATION 2

Voici l'arbre depuis l'étape 0 initiale :

En rouge les mutations de l'étape par rapport à la génération précédente.

GÉNÉRATION 00

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 27, 27, 6)         30
_________________________________________________________________
DV_500_500_1 (MaxPooling2D)  (None, 9, 9, 6)           0
_________________________________________________________________
DV_750_500_1 (Flatten)       (None, 486)               0
_________________________________________________________________
DV_1000_500_1 (Dense)        (None, 10)                4870
=================================================================
Total params: 4,900
_________________________________________________________________
```

GÉNÉRATION 01

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_250_500_3 (Conv2D)        (None, 27, 27, 10)        50
_________________________________________________________________
DV_500_500_2 (MaxPooling2D)  (None, 9, 9, 10)          0
_________________________________________________________________
DV_750_500_2 (Flatten)       (None, 810)               0
_________________________________________________________________
DV_1000_500_2 (Dense)        (None, 10)                8110
=================================================================
Total params: 8,160
_________________________________________________________________
```

GÉNÉRATION 02

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_250_500_3 (Conv2D)        (None, 27, 27, 10)        50
_________________________________________________________________
DV_500_500_3 (Dense)         (None, 27, 27, 4)         40
_________________________________________________________________
DV_750_500_3 (Flatten)       (None, 2916)              0
_________________________________________________________________
DV_1000_500_3 (Dense)        (None, 10)                29170
=================================================================
Total params: 29,260
_________________________________________________________________
```

GÉNÉRATION 03

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_250_500_3 (Conv2D)        (None, 27, 27, 10)        50
_________________________________________________________________
DV_500_500_3 (Dense)         (None, 27, 27, 4)         40
_________________________________________________________________
DV_625_500_1 (DepthwiseConv2 (None, 27, 27, 4)         64
_________________________________________________________________
DV_750_500_4 (Flatten)       (None, 2916)              0
_________________________________________________________________
DV_1000_500_4 (Dense)        (None, 10)                29170
=================================================================
Total params: 29,324
_________________________________________________________________
```

GÉNÉRATION 04

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape          Param #     Connected to
==================================================================================================
DV_0_500_1 (InputLayer)         (None, 28, 28, 1)     0
__________________________________________________________________________________________________
DV_250_500_3 (Conv2D)           (None, 27, 27, 10)    50          DV_0_500_1[0][0]
__________________________________________________________________________________________________
DV_500_500_4 (Conv2D)           (None, 14, 14, 4)     644         DV_250_500_3[0][0]
__________________________________________________________________________________________________
DV_500_750_1 (DepthwiseConv2D)  (None, 14, 14, 10)    100         DV_250_500_3[0][0]
__________________________________________________________________________________________________
DV_625_500_3_C (Concatenate)    (None, 14, 14, 14)    0           DV_500_500_4[0][0]
                                                                  DV_500_750_1[0][0]
__________________________________________________________________________________________________
DV_625_500_3 (DepthwiseConv2D)  (None, 14, 14, 14)    224         DV_625_500_3_C[0][0]
__________________________________________________________________________________________________
DV_750_500_6 (Flatten)          (None, 2744)          0           DV_625_500_3[0][0]
__________________________________________________________________________________________________
DV_1000_500_6 (Dense)           (None, 10)            27450       DV_750_500_6[0][0]
==================================================================================================
Total params: 28,468
__________________________________________________________________________________________________
```

GÉNÉRATION 05

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_3 (Conv2D)        (None, 27, 27, 10)        50          DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_500_500_4 (Conv2D)        (None, 14, 14, 4)         644         DV_250_500_3[0][0]
_________________________________________________________________________________________________
DV_500_750_1 (DepthwiseConv2D) (None, 14, 14, 10)      100         DV_250_500_3[0][0]
_________________________________________________________________________________________________
DV_625_500_3_C (Concatenate) (None, 14, 14, 14)        0           DV_500_500_4[0][0]
                                                                   DV_500_750_1[0][0]
_________________________________________________________________________________________________
DV_625_500_3 (DepthwiseConv2D) (None, 14, 14, 14)      224         DV_625_500_3_C[0][0]
_________________________________________________________________________________________________
DV_688_500_1 (DepthwiseConv2D) (None, 14, 14, 14)      126         DV_625_500_3[0][0]
_________________________________________________________________________________________________
DV_750_500_7 (Flatten)       (None, 2744)              0           DV_688_500_1[0][0]
_________________________________________________________________________________________________
DV_1000_500_7 (Dense)        (None, 10)                27450       DV_750_500_7[0][0]
=================================================================================================
Total params: 28,594
_________________________________________________________________________________________________
```

GÉNÉRATION 06

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_3 (Conv2D)        (None, 27, 27, 10)        50          DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_500_500_4 (Conv2D)        (None, 14, 14, 4)         644         DV_250_500_3[0][0]
_________________________________________________________________________________________________
DV_500_750_1 (DepthwiseConv2D) (None, 14, 14, 10)      100         DV_250_500_3[0][0]
_________________________________________________________________________________________________
DV_625_500_3_C (Concatenate) (None, 14, 14, 14)        0           DV_500_500_4[0][0]
                                                                   DV_500_750_1[0][0]
_________________________________________________________________________________________________
DV_625_500_3 (DepthwiseConv2D) (None, 14, 14, 14)      224         DV_625_500_3_C[0][0]
_________________________________________________________________________________________________
DV_688_500_3_BatchNorm (BatchNo (None, 14, 14, 14)     56          DV_625_500_3[0][0]
_________________________________________________________________________________________________
DV_688_500_3 (DepthwiseConv2D) (None, 14, 14, 14)      126         DV_688_500_3_BatchNorm[0][0]
_________________________________________________________________________________________________
DV_750_500_8 (Flatten)       (None, 2744)              0           DV_688_500_3[0][0]
_________________________________________________________________________________________________
DV_1000_500_8 (Dense)        (None, 10)                27450       DV_750_500_8[0][0]
=================================================================================================
Total params: 28,650
_________________________________________________________________________________________________
```

GÉNÉRATION 07

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_3 (Conv2D)	(None, 27, 27, 10)	50	DV_0_500_1[0][0]
DV_500_500_4 (Conv2D)	(None, 14, 14, 4)	644	DV_250_500_3[0][0]
DV_500_750_1 (DepthwiseConv2D)	(None, 14, 14, 10)	100	DV_250_500_3[0][0]
DV_625_500_3_C (Concatenate)	(None, 14, 14, 14)	0	DV_500_500_4[0][0] DV_500_750_1[0][0]
DV_625_500_3 (DepthwiseConv2D)	(None, 14, 14, 14)	224	DV_625_500_3_C[0][0]
DV_656_500_1 (AveragePooling2D)	(None, 7, 7, 14)	0	DV_625_500_3[0][0]
DV_688_500_4_BatchNorm (BatchNo	(None, 7, 7, 14)	56	DV_656_500_1[0][0]
DV_688_500_4 (DepthwiseConv2D)	(None, 7, 7, 14)	126	DV_688_500_4_BatchNorm[0][0]
DV_750_500_9 (Flatten)	(None, 686)	0	DV_688_500_4[0][0]
DV_1000_500_9 (Dense)	(None, 10)	6870	DV_750_500_9[0][0]

Total params: 8,070

GÉNÉRATION 08

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_3 (Conv2D)	(None, 27, 27, 10)	50	DV_0_500_1[0][0]
DV_500_500_5 (MaxPooling2D)	(None, 14, 14, 10)	0	DV_250_500_3[0][0]
DV_500_750_2 (DepthwiseConv2D)	(None, 14, 14, 10)	100	DV_250_500_3[0][0]
DV_625_500_4_C (Concatenate)	(None, 14, 14, 20)	0	DV_500_500_5[0][0] DV_500_750_2[0][0]
DV_625_500_4 (DepthwiseConv2D)	(None, 14, 14, 20)	320	DV_625_500_4_C[0][0]
DV_656_500_2 (AveragePooling2D)	(None, 7, 7, 20)	0	DV_625_500_4[0][0]
DV_688_500_5_BatchNorm (BatchNo	(None, 7, 7, 20)	80	DV_656_500_2[0][0]
DV_688_500_5 (DepthwiseConv2D)	(None, 7, 7, 20)	180	DV_688_500_5_BatchNorm[0][0]
DV_750_500_10 (Flatten)	(None, 980)	0	DV_688_500_5[0][0]
DV_1000_500_10 (Dense)	(None, 10)	9810	DV_750_500_10[0][0]

Total params: 10,540

GÉNÉRATION 09

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_3 (Conv2D)	(None, 27, 27, 10)	50	DV_0_500_1[0][0]
DV_500_500_6 (DepthwiseConv2D)	(None, 14, 14, 10)	10	DV_250_500_3[0][0]
DV_500_750_3 (DepthwiseConv2D)	(None, 14, 14, 10)	100	DV_250_500_3[0][0]
DV_625_500_5_C (Concatenate)	(None, 14, 14, 20)	0	DV_500_500_6[0][0] DV_500_750_3[0][0]
DV_625_500_5 (DepthwiseConv2D)	(None, 14, 14, 20)	320	DV_625_500_5_C[0][0]
DV_656_500_4_C (Concatenate)	(None, 14, 14, 30)	0	DV_625_500_5[0][0] DV_500_750_3[0][0]
DV_656_500_4 (AveragePooling2D)	(None, 7, 7, 30)	0	DV_656_500_4_C[0][0]
DV_688_500_7_BatchNorm (BatchNo	(None, 7, 7, 30)	120	DV_656_500_4[0][0]
DV_688_500_7 (DepthwiseConv2D)	(None, 7, 7, 30)	270	DV_688_500_7_BatchNorm[0][0]
DV_750_500_12 (Flatten)	(None, 1470)	0	DV_688_500_7[0][0]
DV_1000_500_12 (Dense)	(None, 10)	14710	DV_750_500_12[0][0]

Total params: 15,580

GÉNÉRATION 10

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_3 (Conv2D)	(None, 27, 27, 10)	50	DV_0_500_1[0][0]
DV_500_500_6 (DepthwiseConv2D)	(None, 14, 14, 10)	10	DV_250_500_3[0][0]
DV_500_750_3 (DepthwiseConv2D)	(None, 14, 14, 10)	100	DV_250_500_3[0][0]
DV_625_500_5_C (Concatenate)	(None, 14, 14, 20)	0	DV_500_500_6[0][0] DV_500_750_3[0][0]
DV_625_500_5 (DepthwiseConv2D)	(None, 14, 14, 20)	320	DV_625_500_5_C[0][0]
DV_656_500_4_C (Concatenate)	(None, 14, 14, 30)	0	DV_625_500_5[0][0] DV_500_750_3[0][0]
DV_656_500_4 (AveragePooling2D)	(None, 7, 7, 30)	0	DV_656_500_4_C[0][0]
DV_688_500_7_BatchNorm (BatchNo	(None, 7, 7, 30)	120	DV_656_500_4[0][0]
DV_688_500_7 (DepthwiseConv2D)	(None, 7, 7, 30)	270	DV_688_500_7_BatchNorm[0][0]
DV_688_750_1 (SeparableConv2D)	(None, 7, 7, 4)	204	DV_500_500_6[0][0]
DV_750_500_13_C (Concatenate)	(None, 7, 7, 34)	0	DV_688_500_7[0][0] DV_688_750_1[0][0]
DV_750_500_13 (Flatten)	(None, 1666)	0	DV_750_500_13_C[0][0]
DV_1000_500_13 (Dense)	(None, 10)	16670	DV_750_500_13[0][0]

Total params: 17,744

GÉNÉRATION 11

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_656_500_6_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
DV_656_750_2_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
DV_656_500_6 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_500_6_C[0][0]
DV_656_750_2 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_2_C[0][0]
DV_688_500_9_C (Concatenate)	(None, 7, 7, 180)	0	DV_656_500_6[0][0] DV_656_750_2[0][0]
DV_688_500_9_BatchNorm (BatchNo	(None, 7, 7, 180)	720	DV_688_500_9_C[0][0]
DV_688_500_9 (DepthwiseConv2D)	(None, 7, 7, 180)	1620	DV_688_500_9_BatchNorm[0][0]
DV_688_750_3 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_750_500_15_C (Concatenate)	(None, 7, 7, 184)	0	DV_688_500_9[0][0] DV_688_750_3[0][0]
DV_750_500_15 (Flatten)	(None, 9016)	0	DV_750_500_15_C[0][0]
DV_1000_500_15 (Dense)	(None, 10)	90170	DV_750_500_15[0][0]

Total params: 94,554

GÉNÉRATION 12

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_656_500_6_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
DV_656_750_2_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
DV_656_500_6 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_500_6_C[0][0]
DV_656_750_2 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_2_C[0][0]
DV_688_500_9_C (Concatenate)	(None, 7, 7, 180)	0	DV_656_500_6[0][0] DV_656_750_2[0][0]
DV_688_500_9_BatchNorm (BatchNo	(None, 7, 7, 180)	720	DV_688_500_9_C[0][0]
DV_688_500_9 (DepthwiseConv2D)	(None, 7, 7, 180)	1620	DV_688_500_9_BatchNorm[0][0]
DV_688_750_3 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_750_500_15_C (Concatenate)	(None, 7, 7, 184)	0	DV_688_500_9[0][0] DV_688_750_3[0][0]
DV_750_500_15 (Flatten)	(None, 9016)	0	DV_750_500_15_C[0][0]
DV_688_625_1 (GlobalMaxPooling2	(None, 90)	0	DV_656_750_2[0][0]
DV_1000_500_16_C (Concatenate)	(None, 9106)	0	DV_750_500_15[0][0] DV_688_625_1[0][0]
DV_1000_500_16 (Dense)	(None, 10)	91070	DV_1000_500_16_C[0][0]

Total params: 95,454

GENERATION 13

Changement de function d'activation non visible au résumé

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_656_500_6_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
DV_656_750_2_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
DV_656_500_6 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_500_6_C[0][0]
DV_656_750_2 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_2_C[0][0]
DV_688_500_9_C (Concatenate)	(None, 7, 7, 180)	0	DV_656_500_6[0][0] DV_656_750_2[0][0]
DV_688_500_9_BatchNorm (BatchNo	(None, 7, 7, 180)	720	DV_688_500_9_C[0][0]
DV_688_500_9 (DepthwiseConv2D)	(None, 7, 7, 180)	1620	DV_688_500_9_BatchNorm[0][0]
DV_688_750_4 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_750_500_15_C (Concatenate)	(None, 7, 7, 184)	0	DV_688_500_9[0][0] DV_688_750_4[0][0]
DV_750_500_15 (Flatten)	(None, 9016)	0	DV_750_500_15_C[0][0]
DV_688_625_1 (GlobalMaxPooling2	(None, 90)	0	DV_656_750_2[0][0]
DV_1000_500_16_C (Concatenate)	(None, 9106)	0	DV_750_500_15[0][0] DV_688_625_1[0][0]
DV_1000_500_16 (Dense)	(None, 10)	91070	DV_1000_500_16_C[0][0]

Total params: 95,454

GÉNÉRATION 14

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_7_C (Concatenate)	(None, 14, 14, 64)	0	DV_625_500_6[0][0] DV_625_750_1[0][0]
DV_656_750_3_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
DV_656_500_7 (AveragePooling2D)	(None, 7, 7, 64)	0	DV_656_500_7_C[0][0]
DV_656_750_3 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_3_C[0][0]
DV_688_500_10_C (Concatenate)	(None, 7, 7, 154)	0	DV_656_500_7[0][0] DV_656_750_3[0][0]
DV_688_500_10_BatchNorm (BatchN	(None, 7, 7, 154)	616	DV_688_500_10_C[0][0]
DV_688_500_10 (DepthwiseConv2D)	(None, 7, 7, 154)	1386	DV_688_500_10_BatchNorm[0][0]
DV_688_750_5 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_750_500_16_C (Concatenate)	(None, 7, 7, 158)	0	DV_688_500_10[0][0] DV_688_750_5[0][0]
DV_750_500_16 (Flatten)	(None, 7742)	0	DV_750_500_16_C[0][0]
DV_688_625_2 (GlobalMaxPooling2	(None, 90)	0	DV_656_750_3[0][0]
DV_1000_500_17_C (Concatenate)	(None, 7832)	0	DV_750_500_16[0][0] DV_688_625_2[0][0]
DV_1000_500_17 (Dense)	(None, 10)	78330	DV_1000_500_17_C[0][0]

Total params: 82,496

GÉNÉRATION 15

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_7_C (Concatenate)	(None, 14, 14, 64)	0	DV_625_500_6[0][0] DV_625_750_1[0][0]
DV_656_500_7 (AveragePooling2D)	(None, 7, 7, 64)	0	DV_656_500_7_C[0][0]
DV_656_750_3_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_1 (Dropout)	(None, 7, 7, 64)	0	DV_656_500_7[0][0]
DV_656_750_3 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_3_C[0][0]
DV_688_500_10_C (Concatenate)	(None, 7, 7, 154)	0	dropout_1[0][0] DV_656_750_3[0][0]
DV_688_500_10_BatchNorm (BatchN	(None, 7, 7, 154)	616	DV_688_500_10_C[0][0]
DV_688_500_10 (DepthwiseConv2D)	(None, 7, 7, 154)	1386	DV_688_500_10_BatchNorm[0][0]
DV_688_750_5 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_750_500_16_C (Concatenate)	(None, 7, 7, 158)	0	DV_688_500_10[0][0] DV_688_750_5[0][0]
DV_750_500_16 (Flatten)	(None, 7742)	0	DV_750_500_16_C[0][0]
DV_688_625_2 (GlobalMaxPooling2	(None, 90)	0	DV_656_750_3[0][0]
DV_750_250_1 (GlobalMaxPooling2	(None, 60)	0	DV_625_500_6[0][0]
DV_1000_500_18_C (Concatenate)	(None, 7892)	0	DV_750_500_16[0][0] DV_688_625_2[0][0] DV_750_250_1[0][0]
DV_1000_500_18 (Dense)	(None, 10)	78930	DV_1000_500_18_C[0][0]

Total params: 83,096

GENERATION 16

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_1[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_2 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_688_500_11_C (Concatenate)	(None, 7, 7, 184)	0	dropout_2[0][0] DV_656_750_4[0][0]
DV_688_500_11_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_11_C[0][0]
DV_688_500_11 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_11_BatchNorm[0][0]
DV_688_750_6 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_750_500_18_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_11[0][0] DV_688_750_6[0][0]
DV_750_250_3_F (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
DV_750_500_18 (Flatten)	(None, 9212)	0	DV_750_500_18_C[0][0]
DV_688_625_3 (GlobalMaxPooling2	(None, 90)	0	DV_656_750_4[0][0]
DV_750_250_3 (Dense)	(None, 4)	47044	DV_750_250_3_F[0][0]
DV_1000_500_20_C (Concatenate)	(None, 9306)	0	DV_750_500_18[0][0] DV_688_625_3[0][0] DV_750_250_3[0][0]
DV_1000_500_20 (Dense)	(None, 10)	93070	DV_1000_500_20_C[0][0]

Total params: 144,670

GENERATION 17

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_1[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_3 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_688_500_12_C (Concatenate)	(None, 7, 7, 184)	0	dropout_3[0][0] DV_656_750_4[0][0]
DV_688_500_12_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_12_C[0][0]
DV_688_500_12 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_12_BatchNorm[0][0]
DV_688_750_7 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_750_500_19_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_12[0][0] DV_688_750_7[0][0]
DV_672_500_1 (Dense)	(None, 7, 7, 2)	182	DV_656_750_4[0][0]
DV_750_250_4_F (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
DV_750_500_19 (Flatten)	(None, 9212)	0	DV_750_500_19_C[0][0]
DV_688_625_4 (GlobalMaxPooling2	(None, 2)	0	DV_672_500_1[0][0]
DV_750_250_4 (Dense)	(None, 4)	47044	DV_750_250_4_F[0][0]
DV_1000_500_21_C (Concatenate)	(None, 9218)	0	DV_750_500_19[0][0] DV_688_625_4[0][0] DV_750_250_4[0][0]
DV_1000_500_21 (Dense)	(None, 10)	92190	DV_1000_500_21_C[0][0]

Total params: 143,972

GÉNÉRATION 18

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_1[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_4 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_688_500_12_C (Concatenate)	(None, 7, 7, 184)	0	dropout_4[0][0] DV_656_750_4[0][0]
DV_688_500_12_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_12_C[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_12 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_12_BatchNorm[0][0]
DV_688_750_7 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
flatten_1 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_2 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
DV_750_500_20_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_12[0][0] DV_688_750_7[0][0]
DV_672_500_1 (Dense)	(None, 7, 7, 2)	182	DV_656_750_4[0][0]
DV_750_250_5_C (Concatenate)	(None, 12544)	0	flatten_1[0][0] flatten_2[0][0]
DV_750_500_20 (Flatten)	(None, 9212)	0	DV_750_500_20_C[0][0]
DV_688_625_4 (GlobalMaxPooling2	(None, 2)	0	DV_672_500_1[0][0]
DV_750_250_5 (Dense)	(None, 4)	50180	DV_750_250_5_C[0][0]
DV_1000_500_22_C (Concatenate)	(None, 9218)	0	DV_750_500_20[0][0] DV_688_625_4[0][0] DV_750_250_5[0][0]
DV_1000_500_22 (Dense)	(None, 10)	92190	DV_1000_500_22_C[0][0]

Total params: 147,124

GÉNÉRATION 19

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_1[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_5 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_688_500_13_C (Concatenate)	(None, 7, 7, 184)	0	dropout_5[0][0] DV_656_750_4[0][0]
DV_688_500_13_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_13_C[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_13 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_13_BatchNorm[0][0]
DV_688_750_8 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_664_500_1 (SeparableConv2D)	(None, 7, 7, 2)	992	DV_656_750_4[0][0]
flatten_3 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_4 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
DV_750_500_21_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_13[0][0] DV_688_750_8[0][0]
DV_672_500_2 (Dense)	(None, 7, 7, 2)	6	DV_664_500_1[0][0]
DV_750_250_6_C (Concatenate)	(None, 12544)	0	flatten_3[0][0] flatten_4[0][0]
DV_750_500_21 (Flatten)	(None, 9212)	0	DV_750_500_21_C[0][0]
DV_688_625_5 (GlobalMaxPooling2	(None, 2)	0	DV_672_500_2[0][0]
DV_750_250_6 (Dense)	(None, 4)	50180	DV_750_250_6_C[0][0]
DV_1000_500_23_C (Concatenate)	(None, 9218)	0	DV_750_500_21[0][0] DV_688_625_5[0][0] DV_750_250_6[0][0]
DV_1000_500_23 (Dense)	(None, 10)	92190	DV_1000_500_23_C[0][0]

```
==========================================================================================
Total params: 147,940
__________________________________________________________________________________________
```

GÉNÉRATION 20

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_1[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_6 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_688_500_14_C (Concatenate)	(None, 7, 7, 184)	0	dropout_6[0][0] DV_656_750_4[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_14_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_14_C[0][0]
DV_680_500_1 (MaxPooling2D)	(None, 14, 14, 1)	0	DV_375_500_1[0][0]
DV_688_500_14 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_14_BatchNorm[0][0]
DV_688_750_9 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_664_500_1 (SeparableConv2D)	(None, 7, 7, 2)	992	DV_656_750_4[0][0]
flatten_5 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_6 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
flatten_7 (Flatten)	(None, 196)	0	DV_680_500_1[0][0]
DV_750_500_24_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_14[0][0] DV_688_750_9[0][0]
DV_672_500_2 (Dense)	(None, 7, 7, 2)	6	DV_664_500_1[0][0]
DV_750_250_10_C (Concatenate)	(None, 12740)	0	flatten_5[0][0] flatten_6[0][0] flatten_7[0][0]
DV_750_500_24 (Flatten)	(None, 9212)	0	DV_750_500_24_C[0][0]
DV_688_625_6 (GlobalMaxPooling2	(None, 2)	0	DV_672_500_2[0][0]
DV_750_250_10 (Dense)	(None, 10)	127410	DV_750_250_10_C[0][0]

```
DV_688_688_2 (GlobalMaxPooling2 (None, 2)            0           DV_672_500_2[0][0]
__________________________________________________________________________________________________
DV_1000_500_26_C (Concatenate)  (None, 9226)         0           DV_750_500_24[0][0]
                                                                 DV_688_625_6[0][0]
                                                                 DV_750_250_10[0][0]
                                                                 DV_688_688_2[0][0]
__________________________________________________________________________________________________
DV_1000_500_26 (Dense)          (None, 10)           92270       DV_1000_500_26_C[0][0]
==================================================================================================
Total params: 225,250
__________________________________________________________________________________________________
```

GÉNÉRATION 21

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_1[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_7 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_14_C (Concatenate)	(None, 7, 7, 184)	0	dropout_7[0][0] DV_656_750_4[0][0]
DV_680_500_1 (MaxPooling2D)	(None, 14, 14, 1)	0	DV_375_500_1[0][0]
DV_688_500_14_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_14_C[0][0]
flatten_8 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_9 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
flatten_10 (Flatten)	(None, 196)	0	DV_680_500_1[0][0]
DV_688_500_14 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_14_BatchNorm[0][0]
DV_688_750_9 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_664_500_1 (SeparableConv2D)	(None, 7, 7, 2)	992	DV_656_750_4[0][0]
DV_750_250_10_C (Concatenate)	(None, 12740)	0	flatten_8[0][0] flatten_9[0][0] flatten_10[0][0]
DV_750_500_24_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_14[0][0] DV_688_750_9[0][0]
DV_672_500_2 (Dense)	(None, 7, 7, 2)	6	DV_664_500_1[0][0]
DV_750_250_10 (Dense)	(None, 10)	127410	DV_750_250_10_C[0][0]
DV_750_500_24 (Flatten)	(None, 9212)	0	DV_750_500_24_C[0][0]
DV_688_625_6 (GlobalMaxPooling2	(None, 2)	0	DV_672_500_2[0][0]

```
DV_688_688_2 (GlobalMaxPooling2 (None, 2)            0          DV_672_500_2[0][0]
_____________________________________________________________________________________________
DV_875_500_1 (Dense)            (None, 6)            66         DV_750_250_10[0][0]
_____________________________________________________________________________________________
DV_1000_500_27_C (Concatenate)  (None, 9222)         0          DV_750_500_24[0][0]
                                                                DV_688_625_6[0][0]
                                                                DV_688_688_2[0][0]
                                                                DV_875_500_1[0][0]
_____________________________________________________________________________________________
DV_1000_500_27 (Dense)          (None, 10)           92230      DV_1000_500_27_C[0][0]
=============================================================================================
Total params: 225,276
_____________________________________________________________________________________________
```

GÉNÉRATION 22

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_1[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_8 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_15_C (Concatenate)	(None, 7, 7, 184)	0	dropout_8[0][0] DV_656_750_4[0][0]
DV_680_500_1 (MaxPooling2D)	(None, 14, 14, 1)	0	DV_375_500_1[0][0]
DV_688_500_15_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_15_C[0][0]
DV_664_500_1 (SeparableConv2D)	(None, 7, 7, 2)	992	DV_656_750_4[0][0]
flatten_11 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_12 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
flatten_13 (Flatten)	(None, 196)	0	DV_680_500_1[0][0]
DV_688_500_15 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_15_BatchNorm[0][0]
DV_688_750_10 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_672_500_2 (Dense)	(None, 7, 7, 2)	6	DV_664_500_1[0][0]
DV_750_250_11_C (Concatenate)	(None, 12740)	0	flatten_11[0][0] flatten_12[0][0] flatten_13[0][0]
DV_750_500_25_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_15[0][0] DV_688_750_10[0][0]
DV_688_625_7_F (Flatten)	(None, 98)	0	DV_672_500_2[0][0]
DV_750_250_11 (Dense)	(None, 10)	127410	DV_750_250_11_C[0][0]
DV_750_500_25 (Flatten)	(None, 9212)	0	DV_750_500_25_C[0][0]

```
DV_688_625_7 (Dense)             (None, 6)            594         DV_688_625_7_F[0][0]
__________________________________________________________________________________________________
DV_688_688_3 (GlobalMaxPooling2  (None, 2)            0           DV_672_500_2[0][0]
__________________________________________________________________________________________________
DV_875_500_4 (Dense)             (None, 26)           286         DV_750_250_11[0][0]
__________________________________________________________________________________________________
DV_938_500_2 (GlobalAveragePool  (None, 184)          0           DV_688_500_15[0][0]
__________________________________________________________________________________________________
DV_1000_500_30_C (Concatenate)   (None, 9430)         0           DV_750_500_25[0][0]
                                                                  DV_688_625_7[0][0]
                                                                  DV_688_688_3[0][0]
                                                                  DV_875_500_4[0][0]
                                                                  DV_938_500_2[0][0]
__________________________________________________________________________________________________
DV_1000_500_30 (Dense)           (None, 10)           94310       DV_1000_500_30_C[0][0]
==================================================================================================
Total params: 228,170
__________________________________________________________________________________________________
```

GÉNÉRATION 23

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_1 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_1[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_9 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_16_C (Concatenate)	(None, 7, 7, 184)	0	dropout_9[0][0] DV_656_750_4[0][0]
DV_660_500_1 (Conv2D)	(None, 7, 7, 2)	182	DV_656_750_4[0][0]
DV_680_500_2 (MaxPooling2D)	(None, 14, 14, 1)	0	DV_375_500_1[0][0]
DV_688_500_16_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_16_C[0][0]
DV_664_500_2 (SeparableConv2D)	(None, 7, 7, 2)	24	DV_660_500_1[0][0]
flatten_14 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_15 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
flatten_16 (Flatten)	(None, 196)	0	DV_680_500_2[0][0]
DV_688_500_16 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_16_BatchNorm[0][0]
DV_688_750_11 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_672_500_3 (Dense)	(None, 7, 7, 2)	6	DV_664_500_2[0][0]
DV_750_250_12_C (Concatenate)	(None, 12740)	0	flatten_14[0][0] flatten_15[0][0] flatten_16[0][0]
DV_750_500_26_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_16[0][0] DV_688_750_11[0][0]
DV_688_625_8_F (Flatten)	(None, 98)	0	DV_672_500_3[0][0]
DV_750_250_12 (Dense)	(None, 10)	127410	DV_750_250_12_C[0][0]

DV_750_500_26 (Flatten)	(None, 9212)	0	DV_750_500_26_C[0][0]
DV_688_625_8 (Dense)	(None, 6)	594	DV_688_625_8_F[0][0]
DV_688_688_4 (GlobalMaxPooling2	(None, 2)	0	DV_672_500_3[0][0]
DV_875_500_5 (Dense)	(None, 26)	286	DV_750_250_12[0][0]
DV_938_500_3 (GlobalAveragePool	(None, 184)	0	DV_688_500_16[0][0]
DV_1000_500_33_C (Concatenate)	(None, 9430)	0	DV_750_500_26[0][0] DV_688_625_8[0][0] DV_688_688_4[0][0] DV_875_500_5[0][0] DV_938_500_3[0][0]
DV_1000_500_33 (Dense)	(None, 10)	94300	DV_1000_500_33_C[0][0]

Total params: 227,374

GÉNÉRATION 24

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_2 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_2[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_10 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_17_C (Concatenate)	(None, 7, 7, 184)	0	dropout_10[0][0] DV_656_750_4[0][0]
DV_660_500_1 (Conv2D)	(None, 7, 7, 2)	182	DV_656_750_4[0][0]
DV_680_500_3 (MaxPooling2D)	(None, 14, 14, 1)	0	DV_375_500_1[0][0]
DV_688_500_17_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_17_C[0][0]
DV_664_500_3 (Dropout)	(None, 7, 7, 2)	0	DV_660_500_1[0][0]
flatten_17 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]

```
flatten_18 (Flatten)            (None, 784)          0          DV_375_500_1[0][0]
flatten_19 (Flatten)            (None, 196)          0          DV_680_500_3[0][0]
DV_688_500_17 (DepthwiseConv2D) (None, 7, 7, 184)    1656       DV_688_500_17_BatchNorm[0][0]
DV_688_750_12 (SeparableConv2D) (None, 7, 7, 4)      604        DV_500_500_7[0][0]
DV_672_500_4 (Dense)            (None, 7, 7, 2)      6          DV_664_500_3[0][0]
DV_750_250_13_C (Concatenate)   (None, 12740)        0          flatten_17[0][0]
                                                                flatten_18[0][0]
                                                                flatten_19[0][0]
DV_750_500_27_C (Concatenate)   (None, 7, 7, 188)    0          DV_688_500_17[0][0]
                                                                DV_688_750_12[0][0]
DV_688_625_9_F (Flatten)        (None, 98)           0          DV_672_500_4[0][0]
DV_750_250_13 (Dense)           (None, 10)           127410     DV_750_250_13_C[0][0]
DV_938_500_5_C (Concatenate)    (None, 7, 7, 184)    0          dropout_10[0][0]
                                                                DV_656_750_4[0][0]
DV_750_500_27 (Flatten)         (None, 9212)         0          DV_750_500_27_C[0][0]
DV_688_625_9 (Dense)            (None, 6)            594        DV_688_625_9_F[0][0]
DV_688_688_5 (GlobalMaxPooling2 (None, 2)            0          DV_672_500_4[0][0]
DV_875_500_6 (Dense)            (None, 26)           286        DV_750_250_13[0][0]
DV_938_500_5 (GlobalAveragePool (None, 184)          0          DV_938_500_5_C[0][0]
DV_1000_500_35_C (Concatenate)  (None, 9430)         0          DV_750_500_27[0][0]
                                                                DV_688_625_9[0][0]
                                                                DV_688_688_5[0][0]
                                                                DV_875_500_6[0][0]
                                                                DV_938_500_5[0][0]
DV_1000_500_35 (Dense)          (None, 10)           94300      DV_1000_500_35_C[0][0]
==================================================================================================
Total params: 227,350
```

GÉNÉRATION 25

```
Layer (type)                    Output Shape         Param #    Connected to
==================================================================================================
DV_0_500_1 (InputLayer)         (None, 28, 28, 1)    0
DV_250_500_5 (Conv2D)           (None, 27, 27, 30)   150        DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)  (None, 14, 14, 30)   30         DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)  (None, 14, 14, 30)   300        DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)    (None, 14, 14, 60)   0          DV_500_500_7[0][0]
                                                                DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)  (None, 14, 14, 60)   960        DV_625_500_6_C[0][0]
DV_625_750_2 (Dense)            (None, 14, 14, 4)    120        DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)    (None, 14, 14, 94)   0          DV_625_500_6[0][0]
                                                                DV_625_750_2[0][0]
                                                                DV_500_750_4[0][0]
```

DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
dropout_11 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_17_C (Concatenate)	(None, 7, 7, 184)	0	dropout_11[0][0] DV_656_750_4[0][0]
DV_660_500_1 (Conv2D)	(None, 7, 7, 2)	182	DV_656_750_4[0][0]
DV_680_500_3 (MaxPooling2D)	(None, 14, 14, 1)	0	DV_375_500_1[0][0]
DV_688_500_17_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_17_C[0][0]
DV_664_500_3 (Dropout)	(None, 7, 7, 2)	0	DV_660_500_1[0][0]
flatten_20 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_21 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
flatten_22 (Flatten)	(None, 196)	0	DV_680_500_3[0][0]
DV_688_500_17 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_17_BatchNorm[0][0]
DV_688_750_12 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_672_500_4 (Dense)	(None, 7, 7, 2)	6	DV_664_500_3[0][0]
DV_750_250_13_C (Concatenate)	(None, 12740)	0	flatten_20[0][0] flatten_21[0][0] flatten_22[0][0]
DV_750_500_27_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_17[0][0] DV_688_750_12[0][0]
DV_688_625_9_F (Flatten)	(None, 98)	0	DV_672_500_4[0][0]
DV_750_250_13 (Dense)	(None, 10)	127410	DV_750_250_13_C[0][0]
DV_938_500_6_C (Concatenate)	(None, 7, 7, 184)	0	dropout_11[0][0] DV_656_750_4[0][0]
DV_750_500_27 (Flatten)	(None, 9212)	0	DV_750_500_27_C[0][0]
DV_688_625_9 (Dense)	(None, 6)	594	DV_688_625_9_F[0][0]
DV_688_688_5 (GlobalMaxPooling2	(None, 2)	0	DV_672_500_4[0][0]
DV_875_500_7 (Dropout)	(None, 10)	0	DV_750_250_13[0][0]
DV_938_500_6 (GlobalAveragePool	(None, 184)	0	DV_938_500_6_C[0][0]
DV_1000_500_36_C (Concatenate)	(None, 9414)	0	DV_750_500_27[0][0] DV_688_625_9[0][0] DV_688_688_5[0][0] DV_875_500_7[0][0] DV_938_500_6[0][0]
DV_1000_500_36 (Dense)	(None, 10)	94140	DV_1000_500_36_C[0][0]

Total params: 226,904

GÉNÉRATION 26

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]
DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_2 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_2[0][0] DV_500_750_4[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
dropout_12 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_660_500_1 (Conv2D)	(None, 7, 7, 2)	182	DV_656_750_4[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_19_C (Concatenate)	(None, 7, 7, 184)	0	dropout_12[0][0] DV_656_750_4[0][0]
DV_664_500_3 (Dropout)	(None, 7, 7, 2)	0	DV_660_500_1[0][0]
DV_680_500_3 (MaxPooling2D)	(None, 14, 14, 1)	0	DV_375_500_1[0][0]
DV_688_500_19_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_19_C[0][0]
DV_672_500_4 (Dense)	(None, 7, 7, 2)	6	DV_664_500_3[0][0]
flatten_23 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_24 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
flatten_25 (Flatten)	(None, 196)	0	DV_680_500_3[0][0]
DV_688_500_19 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_19_BatchNorm[0][0]
DV_688_750_14 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_680_250_1 (Dropout)	(None, 7, 7, 2)	0	DV_672_500_4[0][0]
DV_750_250_15_C (Concatenate)	(None, 12740)	0	flatten_23[0][0] flatten_24[0][0] flatten_25[0][0]
DV_750_500_29_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_19[0][0] DV_688_750_14[0][0]
DV_688_625_11_F (Flatten)	(None, 98)	0	DV_672_500_4[0][0]

```
DV_688_688_7_F (Flatten)         (None, 98)           0          DV_680_250_1[0][0]
_________________________________________________________________________________________________
DV_750_250_15 (Dense)            (None, 10)           127410     DV_750_250_15_C[0][0]
_________________________________________________________________________________________________
DV_938_500_8_C (Concatenate)     (None, 7, 7, 184)    0          dropout_12[0][0]
                                                                 DV_656_750_4[0][0]
_________________________________________________________________________________________________
DV_750_500_29 (Flatten)          (None, 9212)         0          DV_750_500_29_C[0][0]
_________________________________________________________________________________________________
DV_688_625_11 (Dense)            (None, 6)            594        DV_688_625_11_F[0][0]
_________________________________________________________________________________________________
DV_688_688_7 (Dense)             (None, 4)            392        DV_688_688_7_F[0][0]
_________________________________________________________________________________________________
DV_875_500_9 (Dropout)           (None, 10)           0          DV_750_250_15[0][0]
_________________________________________________________________________________________________
DV_938_500_8 (GlobalAveragePool (None, 184)          0          DV_938_500_8_C[0][0]
_________________________________________________________________________________________________
DV_1000_500_38_C (Concatenate)   (None, 9416)         0          DV_750_500_29[0][0]
                                                                 DV_688_625_11[0][0]
                                                                 DV_688_688_7[0][0]
                                                                 DV_875_500_9[0][0]
                                                                 DV_938_500_8[0][0]
_________________________________________________________________________________________________
DV_1000_500_38 (Dense)           (None, 10)           94160      DV_1000_500_38_C[0][0]
=================================================================================================
Total params: 227,316
_________________________________________________________________________________________________
```

GÉNÉRATION 27

```
_________________________________________________________________________________________________
Layer (type)                     Output Shape         Param #    Connected to
=================================================================================================
DV_0_500_1 (InputLayer)          (None, 28, 28, 1)    0
_________________________________________________________________________________________________
DV_250_500_5 (Conv2D)            (None, 27, 27, 30)   150        DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_500_500_7 (DepthwiseConv2D)   (None, 14, 14, 30)   30         DV_250_500_5[0][0]
_________________________________________________________________________________________________
DV_500_750_4 (DepthwiseConv2D)   (None, 14, 14, 30)   300        DV_250_500_5[0][0]
_________________________________________________________________________________________________
DV_625_500_6_C (Concatenate)     (None, 14, 14, 60)   0          DV_500_500_7[0][0]
                                                                 DV_500_750_4[0][0]
_________________________________________________________________________________________________
DV_625_500_6 (DepthwiseConv2D)   (None, 14, 14, 60)   960        DV_625_500_6_C[0][0]
_________________________________________________________________________________________________
DV_625_750_2 (Dense)             (None, 14, 14, 4)    120        DV_500_750_4[0][0]
_________________________________________________________________________________________________
DV_656_500_8_C (Concatenate)     (None, 14, 14, 94)   0          DV_625_500_6[0][0]
                                                                 DV_625_750_2[0][0]
                                                                 DV_500_750_4[0][0]
_________________________________________________________________________________________________
DV_656_750_4_C (Concatenate)     (None, 14, 14, 90)   0          DV_625_500_6[0][0]
                                                                 DV_500_750_4[0][0]
_________________________________________________________________________________________________
DV_656_500_8 (AveragePooling2D)  (None, 7, 7, 94)     0          DV_656_500_8_C[0][0]
_________________________________________________________________________________________________
DV_656_750_4 (AveragePooling2D)  (None, 7, 7, 90)     0          DV_656_750_4_C[0][0]
_________________________________________________________________________________________________
dropout_13 (Dropout)             (None, 7, 7, 94)     0          DV_656_500_8[0][0]
_________________________________________________________________________________________________
DV_660_500_1 (Conv2D)            (None, 7, 7, 2)      182        DV_656_750_4[0][0]
_________________________________________________________________________________________________
DV_375_500_1 (DepthwiseConv2D)   (None, 28, 28, 1)    16         DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_688_500_19_C (Concatenate)    (None, 7, 7, 184)    0          dropout_13[0][0]
                                                                 DV_656_750_4[0][0]
_________________________________________________________________________________________________
```

DV_664_500_3 (Dropout)	(None, 7, 7, 2)	0	DV_660_500_1[0][0]
DV_680_500_3 (MaxPooling2D)	(None, 14, 14, 1)	0	DV_375_500_1[0][0]
DV_688_500_19_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_19_C[0][0]
DV_672_500_4 (Dense)	(None, 7, 7, 2)	6	DV_664_500_3[0][0]
flatten_26 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_27 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
flatten_28 (Flatten)	(None, 196)	0	DV_680_500_3[0][0]
DV_688_500_19 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_19_BatchNorm[0][0]
DV_688_750_14 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_680_250_1 (Dropout)	(None, 7, 7, 2)	0	DV_672_500_4[0][0]
DV_750_250_15_C (Concatenate)	(None, 12740)	0	flatten_26[0][0] flatten_27[0][0] flatten_28[0][0]
DV_750_500_29_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_19[0][0] DV_688_750_14[0][0]
DV_688_625_11_F (Flatten)	(None, 98)	0	DV_672_500_4[0][0]
DV_688_688_7_F (Flatten)	(None, 98)	0	DV_680_250_1[0][0]
DV_750_250_15 (Dense)	(None, 10)	127410	DV_750_250_15_C[0][0]
DV_938_500_8_C (Concatenate)	(None, 7, 7, 184)	0	dropout_13[0][0] DV_656_750_4[0][0]
DV_875_250_1_F (Flatten)	(None, 98)	0	DV_660_500_1[0][0]
DV_750_500_29 (Flatten)	(None, 9212)	0	DV_750_500_29_C[0][0]
DV_688_625_11 (Dense)	(None, 6)	594	DV_688_625_11_F[0][0]
DV_688_688_7 (Dense)	(None, 4)	392	DV_688_688_7_F[0][0]
DV_875_500_9 (Dropout)	(None, 10)	0	DV_750_250_15[0][0]
DV_938_500_8 (GlobalAveragePool	(None, 184)	0	DV_938_500_8_C[0][0]
DV_875_250_1 (Dense)	(None, 2)	196	DV_875_250_1_F[0][0]
DV_1000_500_39_C (Concatenate)	(None, 9418)	0	DV_750_500_29[0][0] DV_688_625_11[0][0] DV_688_688_7[0][0] DV_875_500_9[0][0] DV_938_500_8[0][0] DV_875_250_1[0][0]
DV_1000_500_39 (Dense)	(None, 10)	94180	DV_1000_500_39_C[0][0]

Total params: 227,532

GÉNÉRATION 28 : CHAMPION TOPOLOGY (ID: 1576176933526592000)

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_5 (Conv2D)	(None, 27, 27, 30)	150	DV_0_500_1[0][0]

DV_500_500_7 (DepthwiseConv2D)	(None, 14, 14, 30)	30	DV_250_500_5[0][0]
DV_500_750_4 (DepthwiseConv2D)	(None, 14, 14, 30)	300	DV_250_500_5[0][0]
DV_625_500_6_C (Concatenate)	(None, 14, 14, 60)	0	DV_500_500_7[0][0] DV_500_750_4[0][0]
DV_625_500_6 (DepthwiseConv2D)	(None, 14, 14, 60)	960	DV_625_500_6_C[0][0]
DV_625_750_2 (Dense)	(None, 14, 14, 4)	120	DV_500_750_4[0][0]
DV_656_500_8_C (Concatenate)	(None, 14, 14, 94)	0	DV_625_500_6[0][0] DV_625_750_2[0][0] DV_500_750_4[0][0]
DV_656_750_4_C (Concatenate)	(None, 14, 14, 90)	0	DV_625_500_6[0][0] DV_500_750_4[0][0]
DV_656_500_8 (AveragePooling2D)	(None, 7, 7, 94)	0	DV_656_500_8_C[0][0]
DV_656_750_4 (AveragePooling2D)	(None, 7, 7, 90)	0	DV_656_750_4_C[0][0]
dropout_14 (Dropout)	(None, 7, 7, 94)	0	DV_656_500_8[0][0]
DV_660_500_1 (Conv2D)	(None, 7, 7, 2)	182	DV_656_750_4[0][0]
DV_375_500_1 (DepthwiseConv2D)	(None, 28, 28, 1)	16	DV_0_500_1[0][0]
DV_688_500_19_C (Concatenate)	(None, 7, 7, 184)	0	dropout_14[0][0] DV_656_750_4[0][0]
DV_664_500_3 (Dropout)	(None, 7, 7, 2)	0	DV_660_500_1[0][0]
DV_680_500_3 (MaxPooling2D)	(None, 14, 14, 1)	0	DV_375_500_1[0][0]
DV_688_500_19_BatchNorm (BatchN	(None, 7, 7, 184)	736	DV_688_500_19_C[0][0]
DV_672_500_4 (Dense)	(None, 7, 7, 2)	6	DV_664_500_3[0][0]
flatten_29 (Flatten)	(None, 11760)	0	DV_625_500_6[0][0]
flatten_30 (Flatten)	(None, 784)	0	DV_375_500_1[0][0]
flatten_31 (Flatten)	(None, 196)	0	DV_680_500_3[0][0]
DV_688_500_19 (DepthwiseConv2D)	(None, 7, 7, 184)	1656	DV_688_500_19_BatchNorm[0][0]
DV_688_750_14 (SeparableConv2D)	(None, 7, 7, 4)	604	DV_500_500_7[0][0]
DV_680_250_1 (Dropout)	(None, 7, 7, 2)	0	DV_672_500_4[0][0]
DV_750_250_15_C (Concatenate)	(None, 12740)	0	flatten_29[0][0] flatten_30[0][0] flatten_31[0][0]
DV_750_500_29_C (Concatenate)	(None, 7, 7, 188)	0	DV_688_500_19[0][0] DV_688_750_14[0][0]
DV_688_625_11_F (Flatten)	(None, 98)	0	DV_672_500_4[0][0]
DV_688_688_7_F (Flatten)	(None, 98)	0	DV_680_250_1[0][0]
DV_750_250_15 (Dense)	(None, 10)	127410	DV_750_250_15_C[0][0]
DV_875_250_1_F (Flatten)	(None, 98)	0	DV_660_500_1[0][0]
DV_750_500_29 (Flatten)	(None, 9212)	0	DV_750_500_29_C[0][0]
DV_688_625_11 (Dense)	(None, 6)	594	DV_688_625_11_F[0][0]

DV_688_688_7 (Dense)	(None, 4)	392	DV_688_688_7_F[0][0]
DV_875_500_9 (Dropout)	(None, 10)	0	DV_750_250_15[0][0]
DV_938_500_9 (GlobalAveragePool	(None, 90)	0	DV_656_750_4[0][0]
DV_875_250_1 (Dense)	(None, 2)	196	DV_875_250_1_F[0][0]
DV_1000_500_40_C (Concatenate)	(None, 9324)	0	DV_750_500_29[0][0] DV_688_625_11[0][0] DV_688_688_7[0][0] DV_875_500_9[0][0] DV_938_500_9[0][0] DV_875_250_1[0][0]
DV_1000_500_40 (Dense)	(None, 10)	93240	DV_1000_500_40_C[0][0]

Total params: 226,592

ANNEXE 3 : TRACES D'UN APPRENTISSAGE PRL

Résultats du code d'un apprentissage PRL permettant de voir les évolutions et scores au cours du temps ainsi que les informations temporelles.

```
2020-07-08 09:24:50 INFO    : ###### Start Evolutionary Step Index: 0 Test : 32/40 ############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_500_500_1 (Flatten)       (None, 784)               0
_________________________________________________________________
DV_1000_500_1 (Dense)        (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
None
2020-07-08 09:24:50 INFO     : Original scores - TrnFitness: 0.08789999783039093 ValFitness: False TstFitness:
0.08789999783039093 Scaled_TstFitness: 0.9944000244140625
2020-07-08  09:24:50.779938:  I  tensorflow/core/common_runtime/gpu/gpu_device.cc:1640]  Found  device  0  with
properties:
name: Tesla V100-PCIE-32GB major: 7 minor: 0 memoryClockRate(GHz): 1.38
pciBusID: 0000:af:00.0
2020-07-08  09:24:50.781296:  I  tensorflow/core/common_runtime/gpu/gpu_device.cc:1640]  Found  device  1  with
properties:
name: Tesla V100-PCIE-32GB major: 7 minor: 0 memoryClockRate(GHz): 1.38
pciBusID: 0000:d8:00.0
2020-07-08 09:24:50.781361: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfully opened
dynamic library libcudart.so.10.1
2020-07-08 09:24:50.781376: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfully opened
dynamic library libcublas.so.10
2020-07-08 09:24:50.781391: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfully opened
dynamic library libcufft.so.10
2020-07-08 09:24:50.781405: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfully opened
dynamic library libcurand.so.10
2020-07-08 09:24:50.781416: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfully opened
dynamic library libcusolver.so.10
2020-07-08 09:24:50.781429: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfully opened
dynamic library libcusparse.so.10
2020-07-08 09:24:50.781444: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfully opened
dynamic library libcudnn.so.7
2020-07-08 09:24:50.784678: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1763] Adding visible gpu devices:
0, 1
2020-07-08  09:24:50.784720:  I  tensorflow/core/common_runtime/gpu/gpu_device.cc:1181]  Device  interconnect
StreamExecutor with strength 1 edge matrix:
2020-07-08 09:24:50.784727: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1187]      0 1
2020-07-08 09:24:50.784732: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1200] 0:   N Y
2020-07-08 09:24:50.784737: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1200] 1:   Y N
2020-07-08  09:24:50.786874:  I  tensorflow/core/common_runtime/gpu/gpu_device.cc:1326]  Created  TensorFlow  device
(/job:localhost/replica:0/task:0/device:GPU:0 with 30591 MB memory) -> physical GPU (device: 0, name: Tesla V100-
PCIE-32GB, pci bus id: 0000:af:00.0, compute capability: 7.0)
```

```
2020-07-08 09:24:50.788221: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1326] Created TensorFlow device
(/job:localhost/replica:0/task:0/device:GPU:1 with 30591 MB memory) -> physical GPU (device: 1, name: Tesla V100-
PCIE-32GB, pci bus id: 0000:d8:00.0, compute capability: 7.0)
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 0s - loss: 1.4766 - acc: 0.6252 - val_loss: 0.8959 - val_acc: 0.8190
Epoch 2/20 - 0s - loss: 0.7505 - acc: 0.8348 - val_loss: 0.6078 - val_acc: 0.8636
Epoch 3/20 - 0s - loss: 0.5713 - acc: 0.8647 - val_loss: 0.4985 - val_acc: 0.8837
Epoch 4/20 - 0s - loss: 0.4891 - acc: 0.8791 - val_loss: 0.4406 - val_acc: 0.8941
Epoch 5/20 - 0s - loss: 0.4407 - acc: 0.8880 - val_loss: 0.4044 - val_acc: 0.8990
Epoch 6/20 - 0s - loss: 0.4091 - acc: 0.8937 - val_loss: 0.3793 - val_acc: 0.9028
Epoch 7/20 - 0s - loss: 0.3858 - acc: 0.8987 - val_loss: 0.3602 - val_acc: 0.9074
Epoch 8/20 - 0s - loss: 0.3685 - acc: 0.9020 - val_loss: 0.3461 - val_acc: 0.9092
Epoch 9/20 - 0s - loss: 0.3546 - acc: 0.9053 - val_loss: 0.3343 - val_acc: 0.9125
Epoch 10/20 - 0s - loss: 0.3437 - acc: 0.9071 - val_loss: 0.3253 - val_acc: 0.9131
Epoch 11/20 - 0s - loss: 0.3344 - acc: 0.9096 - val_loss: 0.3186 - val_acc: 0.9140
Epoch 12/20 - 0s - loss: 0.3266 - acc: 0.9113 - val_loss: 0.3120 - val_acc: 0.9155
Epoch 13/20 - 0s - loss: 0.3201 - acc: 0.9131 - val_loss: 0.3071 - val_acc: 0.9163
Epoch 14/20 - 0s - loss: 0.3143 - acc: 0.9146 - val_loss: 0.3024 - val_acc: 0.9171
Epoch 15/20 - 0s - loss: 0.3092 - acc: 0.9158 - val_loss: 0.2991 - val_acc: 0.9190
Epoch 16/20 - 0s - loss: 0.3046 - acc: 0.9168 - val_loss: 0.2955 - val_acc: 0.9180
Epoch 17/20 - 0s - loss: 0.3008 - acc: 0.9175 - val_loss: 0.2926 - val_acc: 0.9199
Epoch 18/20 - 0s - loss: 0.2971 - acc: 0.9184 - val_loss: 0.2894 - val_acc: 0.9196
Epoch 19/20 - 0s - loss: 0.2938 - acc: 0.9192 - val_loss: 0.2879 - val_acc: 0.9204
Epoch 20/20 - 0s - loss: 0.2908 - acc: 0.9202 - val_loss: 0.2850 - val_acc: 0.9212
2020-07-08 09:25:01 DEBUG   : Metrics(Test loss & Test Accuracy): [0.28500217837691305, 0.9211999773979187]
2020-07-08 09:25:01 DEBUG   : Mutant ID: 1594174369341240832
2020-07-08 09:25:01 INFO     : Current scores - TrnFitness: 0.9201666712760925 TstFitness: 0.9211999773979187
Scaled_TstFitness: 0.9211999773979187
    val_loss  val_acc      loss       acc     lr
0   0.895883   0.8190  1.476648  0.625150  0.001
1   0.607756   0.8636  0.750542  0.834817  0.001
2   0.498501   0.8837  0.571250  0.864700  0.001
3   0.440588   0.8941  0.489086  0.879100  0.001
4   0.404427   0.8990  0.440711  0.888033  0.001
5   0.379308   0.9028  0.409121  0.893733  0.001
6   0.360159   0.9074  0.385789  0.898733  0.001
7   0.346070   0.9092  0.368479  0.902017  0.001
8   0.334342   0.9125  0.354643  0.905283  0.001
9   0.325276   0.9131  0.343730  0.907117  0.001
10  0.318648   0.9140  0.334392  0.909600  0.001
11  0.312022   0.9155  0.326649  0.911267  0.001
12  0.307080   0.9163  0.320112  0.913117  0.001
13  0.302396   0.9171  0.314294  0.914600  0.001
14  0.299052   0.9190  0.309202  0.915783  0.001
15  0.295502   0.9180  0.304649  0.916850  0.001
16  0.292641   0.9199  0.300817  0.917450  0.001
17  0.289438   0.9196  0.297085  0.918383  0.001
18  0.287936   0.9204  0.293793  0.919233  0.001
19  0.285002   0.9212  0.290779  0.920167  0.001
2020-07-08 09:25:01 DEBUG   : get_Shannon(Model)
2020-07-08 09:25:01 DEBUG   : Model Entropy: 12.510672569274902
2020-07-08 09:25:01 DEBUG   : Model Entropy normalized: 0.015957490522034313
2020-07-08 09:25:01 DEBUG   : Layers Sum Entropy: 12.510672569274902
2020-07-08 09:25:01 DEBUG   : Layer Entropy: ['12.510673']
2020-07-08 09:25:01 DEBUG   : Layer Entropy normalized: ['0.015957490522034313']
2020-07-08 09:25:01 DEBUG   : Layer list: ['DV_1000_500_1']
2020-07-08 09:25:01 DEBUG   : out get_Shannon(Model)
2020-07-08 09:25:01 INFO    : ###### End of step index : 0 of test 32 ##############################
2020-07-08 09:25:01 INFO    : ________________________________________________________________________
2020-07-08 09:25:01 INFO    : ###### Start Evolutionary Step Index: 1 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
```

```
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816
_________________________________________________________________
DV_500_500_2 (Flatten)       (None, 9408)              0
_________________________________________________________________
DV_1000_500_2 (Dense)        (None, 10)                94090
=================================================================
Total params: 94,906
Trainable params: 94,906
Non-trainable params: 0
_________________________________________________________________
None
2020-07-08 09:25:01 INFO     : Original scores - TrnFitness: 0.9750999808311462 ValFitness: False TstFitness:
0.9750999808311462 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 1s - loss: 1.2057 - acc: 0.6639 - val_loss: 0.4262 - val_acc: 0.8801
Epoch 2/20 - 1s - loss: 0.3720 - acc: 0.8921 - val_loss: 0.3190 - val_acc: 0.9097
Epoch 3/20 - 1s - loss: 0.3111 - acc: 0.9089 - val_loss: 0.2848 - val_acc: 0.9177
Epoch 4/20 - 1s - loss: 0.2890 - acc: 0.9172 - val_loss: 0.2695 - val_acc: 0.9197
Epoch 5/20 - 1s - loss: 0.2730 - acc: 0.9218 - val_loss: 0.2610 - val_acc: 0.9258
Epoch 6/20 - 1s - loss: 0.2598 - acc: 0.9257 - val_loss: 0.2470 - val_acc: 0.9301
Epoch 7/20 - 1s - loss: 0.2462 - acc: 0.9295 - val_loss: 0.2370 - val_acc: 0.9330
Epoch 8/20 - 1s - loss: 0.2327 - acc: 0.9341 - val_loss: 0.2275 - val_acc: 0.9351
Epoch 9/20 - 1s - loss: 0.2175 - acc: 0.9388 - val_loss: 0.2116 - val_acc: 0.9419
Epoch 10/20 - 1s - loss: 0.2005 - acc: 0.9437 - val_loss: 0.1926 - val_acc: 0.9478
Epoch 11/20 - 1s - loss: 0.1838 - acc: 0.9484 - val_loss: 0.1775 - val_acc: 0.9524
Epoch 12/20 - 1s - loss: 0.1682 - acc: 0.9531 - val_loss: 0.1610 - val_acc: 0.9560
Epoch 13/20 - 1s - loss: 0.1535 - acc: 0.9576 - val_loss: 0.1461 - val_acc: 0.9599
Epoch 14/20 - 1s - loss: 0.1391 - acc: 0.9617 - val_loss: 0.1378 - val_acc: 0.9625
Epoch 15/20 - 1s - loss: 0.1275 - acc: 0.9654 - val_loss: 0.1243 - val_acc: 0.9663
Epoch 16/20 - 1s - loss: 0.1173 - acc: 0.9671 - val_loss: 0.1179 - val_acc: 0.9682
Epoch 17/20 - 1s - loss: 0.1096 - acc: 0.9696 - val_loss: 0.1088 - val_acc: 0.9704
Epoch 18/20 - 1s - loss: 0.1023 - acc: 0.9721 - val_loss: 0.1019 - val_acc: 0.9714
Epoch 19/20 - 1s - loss: 0.0949 - acc: 0.9745 - val_loss: 0.0980 - val_acc: 0.9713
Epoch 20/20 - 1s - loss: 0.0912 - acc: 0.9748 - val_loss: 0.0971 - val_acc: 0.9710
2020-07-08 09:25:12 DEBUG    : Metrics(Test loss & Test Accuracy): [0.09713027883917093, 0.9710000157356262]
2020-07-08 09:25:12 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:25:12 INFO     : Current scores - TrnFitness: 0.9748166799545288 TstFitness: 0.9710000157356262
Scaled_TstFitness: 0.9710000157356262
    val_loss  val_acc      loss       acc     lr
0   0.426165   0.8801  1.205655  0.663850  0.001
1   0.319043   0.9097  0.372022  0.892083  0.001
2   0.284757   0.9177  0.311134  0.908883  0.001
3   0.269461   0.9197  0.288989  0.917167  0.001
4   0.261022   0.9258  0.272999  0.921817  0.001
5   0.247023   0.9301  0.259835  0.925683  0.001
6   0.237004   0.9330  0.246189  0.929550  0.001
7   0.227505   0.9351  0.232731  0.934100  0.001
8   0.211633   0.9419  0.217525  0.938767  0.001
9   0.192566   0.9478  0.200489  0.943733  0.001
10  0.177454   0.9524  0.183781  0.948417  0.001
11  0.161039   0.9560  0.168198  0.953100  0.001
12  0.146100   0.9599  0.153501  0.957583  0.001
13  0.137755   0.9625  0.139062  0.961700  0.001
14  0.124316   0.9663  0.127459  0.965383  0.001
15  0.117876   0.9682  0.117339  0.967133  0.001
16  0.108753   0.9704  0.109568  0.969650  0.001
17  0.101853   0.9714  0.102294  0.972067  0.001
18  0.098033   0.9713  0.094882  0.974533  0.001
19  0.097130   0.9710  0.091184  0.974817  0.001
2020-07-08 09:25:13 DEBUG    : get_Shannon(Model)
2020-07-08 09:25:13 DEBUG    : Model Entropy: 8.944354057312012
2020-07-08 09:25:13 DEBUG    : Model Entropy normalized: 0.0009503138607428827
2020-07-08 09:25:13 DEBUG    : Layers Sum Entropy: 24.477349281311035
```

```
2020-07-08 09:25:13 DEBUG   : Layer Entropy: ['8.944354', '15.532995']
2020-07-08 09:25:13 DEBUG   : Layer Entropy normalized: ['2.236088514328003', '0.0016510411590135016']
2020-07-08 09:25:13 DEBUG   : Layer list: ['DV_250_500_1', 'DV_1000_500_2']
2020-07-08 09:25:13 DEBUG   : out get_Shannon(Model)
2020-07-08 09:25:13 INFO    : ###### End of step index : 1 of test 32 ##############################
2020-07-08 09:25:13 INFO    : _____________________________________________________________________
2020-07-08 09:25:13 INFO    : ###### Start Evolutionary Step Index: 2 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816
_________________________________________________________________
DV_375_500_1 (SeparableConv2 (None, 7, 7, 48)          2736
_________________________________________________________________
DV_500_500_3 (Flatten)       (None, 2352)              0
_________________________________________________________________
DV_1000_500_3 (Dense)        (None, 10)                23530
=================================================================
Total params: 27,082
Trainable params: 27,082
Non-trainable params: 0
_________________________________________________________________
None
2020-07-08 09:25:13 INFO     : Original scores - TrnFitness: 0.9843000173568726 ValFitness: False TstFitness:
0.9843000173568726 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 1s - loss: 1.1689 - acc: 0.7333 - val_loss: 0.4161 - val_acc: 0.8785
Epoch 2/20 - 1s - loss: 0.3538 - acc: 0.8971 - val_loss: 0.2891 - val_acc: 0.9179
Epoch 3/20 - 1s - loss: 0.2776 - acc: 0.9200 - val_loss: 0.2321 - val_acc: 0.9324
Epoch 4/20 - 1s - loss: 0.2213 - acc: 0.9359 - val_loss: 0.1802 - val_acc: 0.9476
Epoch 5/20 - 1s - loss: 0.1726 - acc: 0.9503 - val_loss: 0.1390 - val_acc: 0.9582
Epoch 6/20 - 1s - loss: 0.1393 - acc: 0.9597 - val_loss: 0.1154 - val_acc: 0.9667
Epoch 7/20 - 1s - loss: 0.1182 - acc: 0.9659 - val_loss: 0.0983 - val_acc: 0.9701
Epoch 8/20 - 1s - loss: 0.1045 - acc: 0.9694 - val_loss: 0.0870 - val_acc: 0.9737
Epoch 9/20 - 1s - loss: 0.0958 - acc: 0.9721 - val_loss: 0.0782 - val_acc: 0.9761
Epoch 10/20 - 1s - loss: 0.0884 - acc: 0.9739 - val_loss: 0.0746 - val_acc: 0.9765
Epoch 11/20 - 1s - loss: 0.0821 - acc: 0.9760 - val_loss: 0.0670 - val_acc: 0.9795
Epoch 12/20 - 1s - loss: 0.0772 - acc: 0.9772 - val_loss: 0.0640 - val_acc: 0.9803
Epoch 13/20 - 1s - loss: 0.0745 - acc: 0.9778 - val_loss: 0.0649 - val_acc: 0.9799

Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 14/20 - 1s - loss: 0.0701 - acc: 0.9792 - val_loss: 0.0588 - val_acc: 0.9815
Epoch 15/20 - 1s - loss: 0.0664 - acc: 0.9798 - val_loss: 0.0594 - val_acc: 0.9819

Epoch 00015: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 16/20 - 1s - loss: 0.0643 - acc: 0.9806 - val_loss: 0.0607 - val_acc: 0.9810

Epoch 00016: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 17/20 - 1s - loss: 0.0602 - acc: 0.9819 - val_loss: 0.0555 - val_acc: 0.9822
Epoch 18/20 - 1s - loss: 0.0590 - acc: 0.9822 - val_loss: 0.0572 - val_acc: 0.9827

Epoch 00018: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 19/20 - 1s - loss: 0.0561 - acc: 0.9828 - val_loss: 0.0503 - val_acc: 0.9836
Epoch 20/20 - 1s - loss: 0.0543 - acc: 0.9832 - val_loss: 0.0511 - val_acc: 0.9835
```

```
Epoch 00020: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:25:32 DEBUG   : Metrics(Test loss & Test Accuracy): [0.051100625203805976, 0.9835000038146973]
2020-07-08 09:25:32 DEBUG   : Mutant ID: 1594174369341240832
2020-07-08 09:25:32 INFO    : Current scores - TrnFitness: 0.9832166433334351 TstFitness: 0.9835000038146973
Scaled_TstFitness: 0.9835000038146973
    val_loss  val_acc      loss       acc     lr
0   0.416119   0.8785  1.168881  0.733283  0.001
1   0.289134   0.9179  0.353817  0.897150  0.001
2   0.232117   0.9324  0.277573  0.919967  0.001
3   0.180238   0.9476  0.221333  0.935850  0.001
4   0.138960   0.9582  0.172635  0.950283  0.001
5   0.115416   0.9667  0.139302  0.959667  0.001
6   0.098306   0.9701  0.118178  0.965900  0.001
7   0.086969   0.9737  0.104534  0.969417  0.001
8   0.078178   0.9761  0.095839  0.972133  0.001
9   0.074620   0.9765  0.088383  0.973850  0.001
10  0.066989   0.9795  0.082075  0.976017  0.001
11  0.064041   0.9803  0.077204  0.977183  0.001
12  0.064888   0.9799  0.074477  0.977750  0.001
13  0.058757   0.9815  0.070093  0.979217  0.001
14  0.059421   0.9819  0.066439  0.979833  0.001
15  0.060729   0.9810  0.064279  0.980600  0.001
16  0.055475   0.9822  0.060199  0.981900  0.001
17  0.057240   0.9827  0.058956  0.982150  0.001
18  0.050294   0.9836  0.056140  0.982767  0.001
19  0.051101   0.9835  0.054313  0.983217  0.001
2020-07-08 09:25:32 DEBUG   : get_Shannon(Model)
2020-07-08 09:25:32 DEBUG   : Model Entropy: 8.910970687866211
2020-07-08 09:25:32 DEBUG   : Model Entropy normalized: 0.003777435645555833
2020-07-08 09:25:32 DEBUG   : Layers Sum Entropy: 31.317404747009277
2020-07-08 09:25:32 DEBUG   : Layer Entropy: ['8.910971', '8.368004', '14.03843']
2020-07-08 09:25:32 DEBUG   : Layer Entropy normalized: ['2.2277426719665527', '2.7893346150716147',
'0.005968720329051115']
2020-07-08 09:25:32 DEBUG   : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_1000_500_3']
2020-07-08 09:25:32 DEBUG   : out get_Shannon(Model)
2020-07-08 09:25:32 INFO    : ###### End of step index : 2 of test 32 ##############################
2020-07-08 09:25:32 INFO    : ________________________________________
2020-07-08 09:25:32 INFO    : ###### Start Evolutionary Step Index: 3 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: LeakyReLU
NodeDict['From']: [(0.75, 0.5)] Node_Type: Dense
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816
_________________________________________________________________
DV_375_500_1 (SeparableConv2 (None, 7, 7, 48)          2736
_________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216
_________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0
_________________________________________________________________
DV_750_500_2 (LeakyReLU)     (None, 576)               0
_________________________________________________________________
```

```
DV_1000_500_5 (Dense)        (None, 10)                5770
=================================================================
Total params: 58,538
Trainable params: 58,538
Non-trainable params: 0

_________________________________________________________________
None
2020-07-08 09:25:32 INFO     : Original scores - TrnFitness: 0.9896000027656555 ValFitness: False TstFitness:
0.9896000027656555 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 1s - loss: 0.4570 - acc: 0.8758 - val_loss: 0.1223 - val_acc: 0.9636
Epoch 2/20 - 1s - loss: 0.1107 - acc: 0.9678 - val_loss: 0.0783 - val_acc: 0.9758
Epoch 3/20 - 1s - loss: 0.0808 - acc: 0.9758 - val_loss: 0.0617 - val_acc: 0.9810
Epoch 4/20 - 1s - loss: 0.0644 - acc: 0.9807 - val_loss: 0.0515 - val_acc: 0.9833
Epoch 5/20 - 1s - loss: 0.0562 - acc: 0.9832 - val_loss: 0.0491 - val_acc: 0.9845
Epoch 6/20 - 1s - loss: 0.0477 - acc: 0.9855 - val_loss: 0.0450 - val_acc: 0.9852
Epoch 7/20 - 1s - loss: 0.0444 - acc: 0.9867 - val_loss: 0.0438 - val_acc: 0.9846
Epoch 8/20 - 1s - loss: 0.0411 - acc: 0.9870 - val_loss: 0.0381 - val_acc: 0.9874
Epoch 9/20 - 1s - loss: 0.0373 - acc: 0.9887 - val_loss: 0.0398 - val_acc: 0.9865

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0352 - acc: 0.9891 - val_loss: 0.0373 - val_acc: 0.9879
Epoch 11/20 - 1s - loss: 0.0329 - acc: 0.9898 - val_loss: 0.0345 - val_acc: 0.9887
Epoch 12/20 - 1s - loss: 0.0297 - acc: 0.9907 - val_loss: 0.0369 - val_acc: 0.9878

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 13/20 - 1s - loss: 0.0279 - acc: 0.9915 - val_loss: 0.0382 - val_acc: 0.9881

Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 14/20 - 1s - loss: 0.0255 - acc: 0.9924 - val_loss: 0.0390 - val_acc: 0.9873

Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:25:47 DEBUG    : Metrics(Test loss & Test Accuracy): [0.034473197813367, 0.9886999726295471]
2020-07-08 09:25:47 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:25:47 INFO     : Current scores - TrnFitness: 0.9923999905586243 TstFitness: 0.9886999726295471
Scaled_TstFitness: 0.9886999726295471
    val_loss  val_acc      loss       acc     lr
0   0.122325   0.9636  0.456990  0.875750  0.001
1   0.078255   0.9758  0.110654  0.967750  0.001
2   0.061693   0.9810  0.080759  0.975833  0.001
3   0.051456   0.9833  0.064430  0.980683  0.001
4   0.049063   0.9845  0.056158  0.983167  0.001
5   0.045025   0.9852  0.047683  0.985550  0.001
6   0.043757   0.9846  0.044444  0.986667  0.001
7   0.038086   0.9874  0.041104  0.987000  0.001
8   0.039764   0.9865  0.037332  0.988650  0.001
9   0.037271   0.9879  0.035225  0.989067  0.001
10  0.034473   0.9887  0.032936  0.989767  0.001
11  0.036944   0.9878  0.029740  0.990733  0.001
12  0.038205   0.9881  0.027904  0.991450  0.001
13  0.038962   0.9873  0.025470  0.992400  0.001
2020-07-08 09:25:48 DEBUG    : get_Shannon(Model)
2020-07-08 09:25:48 DEBUG    : Model Entropy: 8.921566009521484
2020-07-08 09:25:48 DEBUG    : Model Entropy normalized: 0.01519857923257493
2020-07-08 09:25:48 DEBUG    : Layers Sum Entropy: 44.55455780029297
2020-07-08 09:25:48 DEBUG    : Layer Entropy: ['8.921566', '8.32647', '15.210262', '12.096259']
2020-07-08 09:25:48 DEBUG    : Layer Entropy normalized: ['2.230391502380371', '2.775490125020345',
'3.802565574645996', '0.021000449856122334']
2020-07-08 09:25:48 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_1000_500_5']
2020-07-08 09:25:48 DEBUG    : out get_Shannon(Model)
2020-07-08 09:25:48 INFO     : ###### End of step index : 3 of test 32 ##############################
2020-07-08 09:25:48 INFO     : _____________________________________________________________________
2020-07-08 09:25:48 INFO     : ###### Start Evolutionary Step Index: 4 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.75, 0.5)] Node_Type: Dense
{'Age': 0,
 'Coordinate': (2, 0.5),
```

```
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816
_________________________________________________________________
DV_375_500_1 (SeparableConv2 (None, 7, 7, 48)          2736
_________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216
_________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0
_________________________________________________________________
DV_750_500_3 (Dense)         (None, 64)                36928
_________________________________________________________________
DV_1000_500_6 (Dense)        (None, 10)                650
=================================================================
Total params: 90,346
Trainable params: 90,346
Non-trainable params: 0
_________________________________________________________________
None
2020-07-08 09:25:48 INFO     : Original scores - TrnFitness: 0.9894000291824341 ValFitness: False TstFitness:
0.9902999997138977 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 1s - loss: 0.4371 - acc: 0.8935 - val_loss: 0.1309 - val_acc: 0.9692
Epoch 2/20 - 1s - loss: 0.1073 - acc: 0.9739 - val_loss: 0.0825 - val_acc: 0.9790
Epoch 3/20 - 1s - loss: 0.0738 - acc: 0.9816 - val_loss: 0.0634 - val_acc: 0.9836
Epoch 4/20 - 1s - loss: 0.0581 - acc: 0.9855 - val_loss: 0.0522 - val_acc: 0.9858
Epoch 5/20 - 1s - loss: 0.0481 - acc: 0.9880 - val_loss: 0.0482 - val_acc: 0.9861
Epoch 6/20 - 1s - loss: 0.0416 - acc: 0.9896 - val_loss: 0.0435 - val_acc: 0.9882
Epoch 7/20 - 1s - loss: 0.0366 - acc: 0.9909 - val_loss: 0.0415 - val_acc: 0.9884
Epoch 8/20 - 1s - loss: 0.0321 - acc: 0.9917 - val_loss: 0.0394 - val_acc: 0.9879
Epoch 9/20 - 1s - loss: 0.0285 - acc: 0.9929 - val_loss: 0.0375 - val_acc: 0.9882
Epoch 10/20 - 1s - loss: 0.0257 - acc: 0.9938 - val_loss: 0.0371 - val_acc: 0.9880
Epoch 11/20 - 1s - loss: 0.0234 - acc: 0.9941 - val_loss: 0.0341 - val_acc: 0.9895
Epoch 12/20 - 1s - loss: 0.0217 - acc: 0.9943 - val_loss: 0.0353 - val_acc: 0.9889

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 13/20 - 1s - loss: 0.0191 - acc: 0.9952 - val_loss: 0.0330 - val_acc: 0.9890
Epoch 14/20 - 1s - loss: 0.0176 - acc: 0.9955 - val_loss: 0.0341 - val_acc: 0.9900

Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 15/20 - 1s - loss: 0.0157 - acc: 0.9963 - val_loss: 0.0326 - val_acc: 0.9886
Epoch 16/20 - 1s - loss: 0.0155 - acc: 0.9962 - val_loss: 0.0308 - val_acc: 0.9901
Epoch 17/20 - 1s - loss: 0.0139 - acc: 0.9966 - val_loss: 0.0316 - val_acc: 0.9900

Epoch 00017: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 18/20 - 1s - loss: 0.0118 - acc: 0.9977 - val_loss: 0.0307 - val_acc: 0.9907
Epoch 19/20 - 1s - loss: 0.0112 - acc: 0.9977 - val_loss: 0.0325 - val_acc: 0.9890

Epoch 00019: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 20/20 - 1s - loss: 0.0094 - acc: 0.9982 - val_loss: 0.0319 - val_acc: 0.9896

Epoch 00020: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:26:10 DEBUG    : Metrics(Test loss & Test Accuracy): [0.03194184391456656, 0.9896000027656555]
2020-07-08 09:26:10 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:26:10 INFO     : Current scores - TrnFitness: 0.9982333183288574 TstFitness: 0.9896000027656555
Scaled_TstFitness: 0.9896000027656555
    val_loss  val_acc      loss       acc     lr
0   0.130898   0.9692  0.437109  0.893450  0.001
1   0.082522   0.9790  0.107300  0.973917  0.001
```

```
2   0.063401   0.9836  0.073823  0.981633  0.001
3   0.052248   0.9858  0.058106  0.985533  0.001
4   0.048153   0.9861  0.048082  0.987950  0.001
5   0.043456   0.9882  0.041623  0.989600  0.001
6   0.041454   0.9884  0.036636  0.990883  0.001
7   0.039422   0.9879  0.032123  0.991700  0.001
8   0.037475   0.9882  0.028542  0.992933  0.001
9   0.037134   0.9880  0.025746  0.993783  0.001
10  0.034140   0.9895  0.023377  0.994100  0.001
11  0.035271   0.9889  0.021657  0.994300  0.001
12  0.032953   0.9890  0.019144  0.995200  0.001
13  0.034080   0.9900  0.017600  0.995550  0.001
14  0.032605   0.9886  0.015652  0.996350  0.001
15  0.030823   0.9901  0.015455  0.996233  0.001
16  0.031643   0.9900  0.013911  0.996550  0.001
17  0.030693   0.9907  0.011803  0.997650  0.001
18  0.032508   0.9890  0.011211  0.997733  0.001
19  0.031942   0.9896  0.009365  0.998233  0.001
2020-07-08 09:26:10 DEBUG   : get_Shannon(Model)
2020-07-08 09:26:10 DEBUG   : Model Entropy: 8.892518997192383
2020-07-08 09:26:10 DEBUG   : Model Entropy normalized: 0.013659783405825472
2020-07-08 09:26:10 DEBUG   : Layers Sum Entropy: 56.19849395751953
2020-07-08 09:26:10 DEBUG   : Layer Entropy: ['8.892519', '8.286877', '15.173878', '14.829126', '9.016094']
2020-07-08  09:26:10  DEBUG      :  Layer  Entropy  normalized:  ['2.2231297492980957',  '2.762292226155599',
'3.7934694290161133', '0.025745011038250394', '0.14087647199630737']
2020-07-08 09:26:10 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_750_500_3',
'DV_1000_500_6']
2020-07-08 09:26:10 DEBUG   : out get_Shannon(Model)
2020-07-08 09:26:10 INFO    : ###### End of step index : 4 of test 32 ##############################
2020-07-08 09:26:10 INFO    : ________________________________________________________________________
2020-07-08 09:26:10 INFO    : ###### Start Evolutionary Step Index: 5 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.625, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.75, 0.5)] Node_Type: Dense
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816
_________________________________________________________________
DV_375_500_1 (SeparableConv2 (None, 7, 7, 48)          2736
_________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216
_________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0
_________________________________________________________________
DV_625_500_1 (Dense)         (None, 64)                36864
_________________________________________________________________
DV_750_500_5 (Activation)    (None, 64)                0
_________________________________________________________________
DV_1000_500_8 (Dense)        (None, 10)                650
=================================================================
Total params: 90,282
Trainable params: 90,282
Non-trainable params: 0
```

```
_________________________________________________________________
None
2020-07-08 09:26:10 INFO     : Original scores - TrnFitness: 0.9902999997138977 ValFitness: False TstFitness:
0.9902999997138977 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 1s - loss: 0.7420 - acc: 0.8731 - val_loss: 0.2789 - val_acc: 0.9673
Epoch 2/20 - 1s - loss: 0.2005 - acc: 0.9736 - val_loss: 0.1451 - val_acc: 0.9781
Epoch 3/20 - 1s - loss: 0.1206 - acc: 0.9815 - val_loss: 0.1000 - val_acc: 0.9825
Epoch 4/20 - 1s - loss: 0.0865 - acc: 0.9858 - val_loss: 0.0767 - val_acc: 0.9860
Epoch 5/20 - 1s - loss: 0.0670 - acc: 0.9880 - val_loss: 0.0641 - val_acc: 0.9873
Epoch 6/20 - 1s - loss: 0.0549 - acc: 0.9898 - val_loss: 0.0567 - val_acc: 0.9873
Epoch 7/20 - 1s - loss: 0.0465 - acc: 0.9909 - val_loss: 0.0493 - val_acc: 0.9888
Epoch 8/20 - 1s - loss: 0.0398 - acc: 0.9922 - val_loss: 0.0494 - val_acc: 0.9876

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 9/20 - 1s - loss: 0.0346 - acc: 0.9933 - val_loss: 0.0443 - val_acc: 0.9883
Epoch 10/20 - 1s - loss: 0.0311 - acc: 0.9938 - val_loss: 0.0427 - val_acc: 0.9883
Epoch 11/20 - 1s - loss: 0.0285 - acc: 0.9943 - val_loss: 0.0411 - val_acc: 0.9887
Epoch 12/20 - 1s - loss: 0.0249 - acc: 0.9952 - val_loss: 0.0362 - val_acc: 0.9898
Epoch 13/20 - 1s - loss: 0.0222 - acc: 0.9954 - val_loss: 0.0362 - val_acc: 0.9897

Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 14/20 - 1s - loss: 0.0192 - acc: 0.9966 - val_loss: 0.0343 - val_acc: 0.9901
Epoch 15/20 - 1s - loss: 0.0174 - acc: 0.9966 - val_loss: 0.0355 - val_acc: 0.9896

Epoch 00015: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 16/20 - 1s - loss: 0.0158 - acc: 0.9973 - val_loss: 0.0339 - val_acc: 0.9893
Epoch 17/20 - 1s - loss: 0.0144 - acc: 0.9976 - val_loss: 0.0335 - val_acc: 0.9893
Epoch 18/20 - 1s - loss: 0.0125 - acc: 0.9980 - val_loss: 0.0320 - val_acc: 0.9902
Epoch 19/20 - 1s - loss: 0.0115 - acc: 0.9983 - val_loss: 0.0332 - val_acc: 0.9900

Epoch 00019: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 20/20 - 1s - loss: 0.0107 - acc: 0.9984 - val_loss: 0.0335 - val_acc: 0.9895

Epoch 00020: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:26:32 DEBUG    : Metrics(Test loss & Test Accuracy): [0.03351729686707258, 0.9894999861717224]
2020-07-08 09:26:32 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:26:32 INFO     : Current scores - TrnFitness: 0.9983500242233276 TstFitness: 0.9894999861717224
Scaled_TstFitness: 0.9894999861717224
    val_loss  val_acc      loss       acc     lr
0   0.278945   0.9673  0.741968  0.873100  0.001
1   0.145111   0.9781  0.200455  0.973567  0.001
2   0.099991   0.9825  0.120553  0.981550  0.001
3   0.076699   0.9860  0.086493  0.985833  0.001
4   0.064055   0.9873  0.066973  0.987983  0.001
5   0.056701   0.9873  0.054950  0.989750  0.001
6   0.049295   0.9888  0.046461  0.990883  0.001
7   0.049406   0.9876  0.039799  0.992217  0.001
8   0.044317   0.9883  0.034614  0.993267  0.001
9   0.042748   0.9883  0.031085  0.993817  0.001
10  0.041137   0.9887  0.028453  0.994300  0.001
11  0.036161   0.9898  0.024864  0.995183  0.001
12  0.036241   0.9897  0.022247  0.995417  0.001
13  0.034299   0.9901  0.019190  0.996600  0.001
14  0.035488   0.9896  0.017371  0.996650  0.001
15  0.033867   0.9893  0.015768  0.997350  0.001
16  0.033525   0.9893  0.014407  0.997583  0.001
17  0.032014   0.9902  0.012467  0.998000  0.001
18  0.033227   0.9900  0.011467  0.998300  0.001
19  0.033517   0.9895  0.010716  0.998350  0.001
2020-07-08 09:26:33 DEBUG    : get_Shannon(Model)
2020-07-08 09:26:33 DEBUG    : Model Entropy: 8.862110137939453
2020-07-08 09:26:33 DEBUG    : Model Entropy normalized: 0.013613072408509145
2020-07-08 09:26:33 DEBUG    : Layers Sum Entropy: 56.197896003723145
2020-07-08 09:26:33 DEBUG    : Layer Entropy: ['8.86211', '8.264378', '15.158968', '14.818532', '9.093908']
2020-07-08 09:26:33 DEBUG    : Layer Entropy normalized: ['2.2155275344848633', '2.7547925313313804',
'3.7897419929504395', '0.025726618038283453', '0.14209231734275818']
2020-07-08 09:26:33 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_500_1',
'DV_1000_500_8']
2020-07-08 09:26:33 DEBUG    : out get_Shannon(Model)
2020-07-08 09:26:33 INFO     : ###### End of step index : 5 of test 32 ##############################
```

```
2020-07-08 09:26:33 INFO    : _____________________________________________________________________
2020-07-08 09:26:33 INFO    : ###### Start Evolutionary Step Index: 6 Test : 32/40 ############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Activation
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_7"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816         DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)       2736        DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216       DV_375_500_1[0][0]
_________________________________________________________________________________________________
DV_625_750_1 (Conv2D)        (None, 7, 7, 32)          24576       DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0           DV_438_500_1[0][0]
_________________________________________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0           DV_438_500_1[0][0]
_________________________________________________________________________________________________
flatten_2 (Flatten)          (None, 1568)              0           DV_625_750_1[0][0]
_________________________________________________________________________________________________
DV_625_500_1 (Dense)         (None, 64)                36864       DV_500_500_4[0][0]
_________________________________________________________________________________________________
DV_750_250_2_C (Concatenate) (None, 2144)              0           flatten_1[0][0]
                                                                   flatten_2[0][0]
_________________________________________________________________________________________________
DV_750_500_6 (Activation)    (None, 64)                0           DV_625_500_1[0][0]
_________________________________________________________________________________________________
DV_750_250_2 (Dense)         (None, 32)                68640       DV_750_250_2_C[0][0]
_________________________________________________________________________________________________
DV_1000_500_10_C (Concatenate)  (None, 96)             0           DV_750_500_6[0][0]
                                                                   DV_750_250_2[0][0]
_________________________________________________________________________________________________
DV_1000_500_10 (Dense)       (None, 10)                970         DV_1000_500_10_C[0][0]
=================================================================================================
Total params: 183,818
Trainable params: 183,818
Non-trainable params: 0
_________________________________________________________________________________________________
None
2020-07-08 09:26:33 INFO     : Original scores - TrnFitness: 0.9919000267982483 ValFitness: False TstFitness:
0.9922999739646912 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 1s - loss: 0.3520 - acc: 0.9243 - val_loss: 0.0882 - val_acc: 0.9812
Epoch 2/20 - 1s - loss: 0.0668 - acc: 0.9867 - val_loss: 0.0557 - val_acc: 0.9877
Epoch 3/20 - 1s - loss: 0.0439 - acc: 0.9915 - val_loss: 0.0446 - val_acc: 0.9888
Epoch 4/20 - 1s - loss: 0.0329 - acc: 0.9933 - val_loss: 0.0385 - val_acc: 0.9897
Epoch 5/20 - 1s - loss: 0.0263 - acc: 0.9947 - val_loss: 0.0337 - val_acc: 0.9907
Epoch 6/20 - 1s - loss: 0.0210 - acc: 0.9961 - val_loss: 0.0335 - val_acc: 0.9905
Epoch 7/20 - 1s - loss: 0.0178 - acc: 0.9966 - val_loss: 0.0306 - val_acc: 0.9908
Epoch 8/20 - 1s - loss: 0.0145 - acc: 0.9975 - val_loss: 0.0286 - val_acc: 0.9917
Epoch 9/20 - 1s - loss: 0.0125 - acc: 0.9979 - val_loss: 0.0278 - val_acc: 0.9915
Epoch 10/20 - 1s - loss: 0.0108 - acc: 0.9983 - val_loss: 0.0283 - val_acc: 0.9912
```

```
Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 11/20 - 1s - loss: 0.0093 - acc: 0.9986 - val_loss: 0.0289 - val_acc: 0.9906

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 12/20 - 1s - loss: 0.0087 - acc: 0.9986 - val_loss: 0.0282 - val_acc: 0.9906

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:26:50 DEBUG   : Metrics(Test loss & Test Accuracy): [0.027817875245213508, 0.9915000200271606]
2020-07-08 09:26:50 DEBUG   : Mutant ID: 1594174369341240832
2020-07-08 09:26:50 INFO    : Current scores - TrnFitness: 0.9986333250999451 TstFitness: 0.9915000200271606
Scaled_TstFitness: 0.9915000200271606
    val_loss  val_acc      loss       acc     lr
0   0.088199   0.9812  0.351992  0.924267  0.001
1   0.055702   0.9877  0.066792  0.986683  0.001
2   0.044551   0.9888  0.043900  0.991450  0.001
3   0.038493   0.9897  0.032889  0.993333  0.001
4   0.033732   0.9907  0.026251  0.994650  0.001
5   0.033475   0.9905  0.021026  0.996133  0.001
6   0.030583   0.9908  0.017811  0.996600  0.001
7   0.028565   0.9917  0.014479  0.997483  0.001
8   0.027818   0.9915  0.012536  0.997900  0.001
9   0.028298   0.9912  0.010775  0.998333  0.001
10  0.028885   0.9906  0.009324  0.998633  0.001
11  0.028171   0.9906  0.008687  0.998633  0.001
2020-07-08 09:26:50 DEBUG   : get_Shannon(Model)
2020-07-08 09:26:50 DEBUG   : Model Entropy: 8.85185432434082
2020-07-08 09:26:50 DEBUG   : Model Entropy normalized: 0.0031267588570613987
2020-07-08 09:26:50 DEBUG   : Layers Sum Entropy: 86.64771270751953
2020-07-08 09:26:50 DEBUG    : Layer Entropy: ['8.851854', '8.259366', '15.151393', '14.288543', '14.798992',
'15.720822', '9.576742']
2020-07-08 09:26:50 DEBUG    : Layer Entropy normalized: ['2.212963581085205', '2.753122011820475',
'3.7878482341766357', '3.5721356868743896', '0.02569269471698337', '0.007332473103679828', '0.09975773096084595']
2020-07-08 09:26:50 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_750_1',
'DV_625_500_1', 'DV_750_250_2', 'DV_1000_500_10']
2020-07-08 09:26:50 DEBUG   : out get_Shannon(Model)
2020-07-08 09:26:50 INFO    : ###### End of step index : 6 of test 32 ##############################
2020-07-08 09:26:50 INFO    : _____________________________________________________________________
2020-07-08 09:26:50 INFO    : ###### Start Evolutionary Step Index: 7 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Activation
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_8"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816         DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)       2736        DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216       DV_375_500_1[0][0]
_________________________________________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0           DV_438_500_1[0][0]
_________________________________________________________________________________________________
DV_625_750_1 (Conv2D)        (None, 7, 7, 32)          24576       DV_250_500_1[0][0]
```

```
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 576)          0           DV_500_500_4[0][0]
__________________________________________________________________________________________________
flatten_3 (Flatten)             (None, 576)          0           DV_438_500_1[0][0]
__________________________________________________________________________________________________
flatten_4 (Flatten)             (None, 1568)         0           DV_625_750_1[0][0]
__________________________________________________________________________________________________
DV_625_500_1 (Dense)            (None, 64)           36864       dropout_1[0][0]
__________________________________________________________________________________________________
DV_750_250_2_C (Concatenate)    (None, 2144)         0           flatten_3[0][0]
                                                                 flatten_4[0][0]
__________________________________________________________________________________________________
DV_750_500_6 (Activation)       (None, 64)           0           DV_625_500_1[0][0]
__________________________________________________________________________________________________
DV_750_250_2 (Dense)            (None, 32)           68640       DV_750_250_2_C[0][0]
__________________________________________________________________________________________________
DV_1000_500_10_C (Concatenate)  (None, 96)           0           DV_750_500_6[0][0]
                                                                 DV_750_250_2[0][0]
__________________________________________________________________________________________________
DV_1000_500_10 (Dense)          (None, 10)           970         DV_1000_500_10_C[0][0]
==================================================================================================
Total params: 183,818
Trainable params: 183,818
Non-trainable params: 0
__________________________________________________________________________________________________
None
2020-07-08 09:26:50 INFO     : Original scores - TrnFitness: 0.9908999800682068 ValFitness: False TstFitness:
0.9919000267982483 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 1s - loss: 0.0190 - acc: 0.9954 - val_loss: 0.0302 - val_acc: 0.9906
Epoch 2/20 - 1s - loss: 0.0135 - acc: 0.9967 - val_loss: 0.0287 - val_acc: 0.9910
Epoch 3/20 - 1s - loss: 0.0114 - acc: 0.9972 - val_loss: 0.0290 - val_acc: 0.9901

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 4/20 - 1s - loss: 0.0098 - acc: 0.9974 - val_loss: 0.0290 - val_acc: 0.9901

Epoch 00004: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 5/20 - 1s - loss: 0.0072 - acc: 0.9984 - val_loss: 0.0286 - val_acc: 0.9910
Epoch 6/20 - 1s - loss: 0.0064 - acc: 0.9985 - val_loss: 0.0296 - val_acc: 0.9906

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 7/20 - 1s - loss: 0.0054 - acc: 0.9988 - val_loss: 0.0340 - val_acc: 0.9897

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 8/20 - 1s - loss: 0.0058 - acc: 0.9987 - val_loss: 0.0278 - val_acc: 0.9915
Epoch 9/20 - 1s - loss: 0.0042 - acc: 0.9992 - val_loss: 0.0297 - val_acc: 0.9912

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0036 - acc: 0.9994 - val_loss: 0.0279 - val_acc: 0.9918

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 11/20 - 1s - loss: 0.0029 - acc: 0.9995 - val_loss: 0.0279 - val_acc: 0.9917

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:27:06 DEBUG    : Metrics(Test loss & Test Accuracy): [0.027835547250567468, 0.9915000200271606]
2020-07-08 09:27:06 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:27:06 INFO     : Current scores - TrnFitness: 0.9995499849319458 TstFitness: 0.9915000200271606
Scaled_TstFitness: 0.9915000200271606
    val_loss  val_acc      loss       acc     lr
0   0.030249   0.9906  0.018986  0.995367  0.001
1   0.028695   0.9910  0.013485  0.996700  0.001
2   0.028953   0.9901  0.011450  0.997183  0.001
3   0.029045   0.9901  0.009803  0.997433  0.001
4   0.028590   0.9910  0.007176  0.998450  0.001
5   0.029625   0.9906  0.006370  0.998533  0.001
6   0.033998   0.9897  0.005403  0.998850  0.001
7   0.027836   0.9915  0.005764  0.998650  0.001
8   0.029678   0.9912  0.004191  0.999167  0.001
9   0.027930   0.9918  0.003614  0.999350  0.001
10  0.027904   0.9917  0.002899  0.999550  0.001
2020-07-08 09:27:06 DEBUG    : get_Shannon(Model)
```

```
2020-07-08 09:27:06 DEBUG   : Model Entropy: 8.838857650756836
2020-07-08 09:27:06 DEBUG   : Model Entropy normalized: 0.0031221680151030857
2020-07-08 09:27:06 DEBUG   : Layers Sum Entropy: 86.55589294433594
2020-07-08 09:27:06 DEBUG    : Layer Entropy: ['8.838858', '8.252718', '15.149298', '14.259788', '14.785739',
'15.671017', '9.598476']
2020-07-08 09:27:06 DEBUG       : Layer Entropy normalized: ['2.209714412689209', '2.750905990600586',
'3.7873244285583496', '3.5649468898773193', '0.0256696856684154 94', '0.007309242860594792', '0.0999841292699178']
2020-07-08 09:27:06 DEBUG     : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_750_1',
'DV_625_500_1', 'DV_750_250_2', 'DV_1000_500_10']
2020-07-08 09:27:06 DEBUG   : out get_Shannon(Model)
2020-07-08 09:27:06 INFO    : ###### End of step index : 7 of test 32 ##############################
2020-07-08 09:27:06 INFO    : ________________________________________________________________________
2020-07-08 09:27:06 INFO    : ###### Start Evolutionary Step Index: 8 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Activation
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_9"
```

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_1 (Conv2D)	(None, 14, 14, 48)	816	DV_0_500_1[0][0]
DV_375_500_1 (SeparableConv2D)	(None, 7, 7, 48)	2736	DV_250_500_1[0][0]
DV_438_500_1 (Conv2D)	(None, 3, 3, 64)	49216	DV_375_500_1[0][0]
DV_500_500_4 (Flatten)	(None, 576)	0	DV_438_500_1[0][0]
DV_625_750_2 (Conv2D)	(None, 7, 7, 32)	24576	DV_250_500_1[0][0]
DV_625_625_2 (Conv2D)	(None, 7, 7, 32)	24576	DV_250_500_1[0][0]
dropout_2 (Dropout)	(None, 576)	0	DV_500_500_4[0][0]
flatten_5 (Flatten)	(None, 576)	0	DV_438_500_1[0][0]
flatten_6 (Flatten)	(None, 1568)	0	DV_625_750_2[0][0]
flatten_7 (Flatten)	(None, 1568)	0	DV_625_625_2[0][0]
DV_625_500_2 (Dense)	(None, 64)	36864	dropout_2[0][0]
DV_750_250_3_C (Concatenate)	(None, 3712)	0	flatten_5[0][0] flatten_6[0][0] flatten_7[0][0]
DV_750_500_7 (Activation)	(None, 64)	0	DV_625_500_2[0][0]
DV_750_250_3 (Dense)	(None, 32)	118816	DV_750_250_3_C[0][0]
DV_1000_500_11_C (Concatenate)	(None, 96)	0	DV_750_500_7[0][0] DV_750_250_3[0][0]
DV_1000_500_11 (Dense)	(None, 10)	970	DV_1000_500_11_C[0][0]

```
=================================================================================================
Total params: 258,570
Trainable params: 258,570
Non-trainable params: 0
_________________________________________________________________________________________________
None
2020-07-08 09:27:07 INFO     : Original scores - TrnFitness: 0.9919999837875366 ValFitness: False TstFitness:
0.9923999905586243 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.3123 - acc: 0.9245 - val_loss: 0.0702 - val_acc: 0.9816
Epoch 2/20 - 1s - loss: 0.0609 - acc: 0.9863 - val_loss: 0.0465 - val_acc: 0.9873
Epoch 3/20 - 1s - loss: 0.0410 - acc: 0.9907 - val_loss: 0.0381 - val_acc: 0.9892
Epoch 4/20 - 1s - loss: 0.0306 - acc: 0.9934 - val_loss: 0.0339 - val_acc: 0.9902
Epoch 5/20 - 1s - loss: 0.0246 - acc: 0.9943 - val_loss: 0.0305 - val_acc: 0.9909
Epoch 6/20 - 1s - loss: 0.0198 - acc: 0.9957 - val_loss: 0.0295 - val_acc: 0.9909
Epoch 7/20 - 1s - loss: 0.0164 - acc: 0.9964 - val_loss: 0.0297 - val_acc: 0.9896

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 8/20 - 1s - loss: 0.0138 - acc: 0.9970 - val_loss: 0.0278 - val_acc: 0.9909
Epoch 9/20 - 1s - loss: 0.0118 - acc: 0.9976 - val_loss: 0.0285 - val_acc: 0.9912

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0105 - acc: 0.9979 - val_loss: 0.0282 - val_acc: 0.9911

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 11/20 - 1s - loss: 0.0084 - acc: 0.9985 - val_loss: 0.0266 - val_acc: 0.9919
Epoch 12/20 - 1s - loss: 0.0075 - acc: 0.9987 - val_loss: 0.0265 - val_acc: 0.9916

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 13/20 - 1s - loss: 0.0066 - acc: 0.9989 - val_loss: 0.0264 - val_acc: 0.9917
Epoch 14/20 - 1s - loss: 0.0055 - acc: 0.9993 - val_loss: 0.0260 - val_acc: 0.9922
Epoch 15/20 - 1s - loss: 0.0054 - acc: 0.9991 - val_loss: 0.0280 - val_acc: 0.9919

Epoch 00015: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 16/20 - 1s - loss: 0.0047 - acc: 0.9994 - val_loss: 0.0259 - val_acc: 0.9927

Epoch 00016: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 17/20 - 1s - loss: 0.0039 - acc: 0.9995 - val_loss: 0.0267 - val_acc: 0.9917

Epoch 00017: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 18/20 - 1s - loss: 0.0034 - acc: 0.9998 - val_loss: 0.0278 - val_acc: 0.9916

Epoch 00018: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 19/20 - 1s - loss: 0.0029 - acc: 0.9998 - val_loss: 0.0265 - val_acc: 0.9919

Epoch 00019: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:27:34 DEBUG    : Metrics(Test loss & Test Accuracy): [0.025917229831265286, 0.9926999807357788]
2020-07-08 09:27:34 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:27:34 INFO     : Current scores - TrnFitness: 0.999750018119812 TstFitness: 0.9926999807357788
Scaled_TstFitness: 0.9926999807357788
    val_loss  val_acc      loss       acc     lr
0   0.070211   0.9816  0.312268  0.924550  0.001
1   0.046484   0.9873  0.060889  0.986283  0.001
2   0.038112   0.9892  0.041035  0.990717  0.001
3   0.033854   0.9902  0.030629  0.993433  0.001
4   0.030453   0.9909  0.024595  0.994333  0.001
5   0.029472   0.9909  0.019821  0.995717  0.001
6   0.029694   0.9896  0.016409  0.996417  0.001
7   0.027791   0.9909  0.013760  0.997017  0.001
8   0.028485   0.9912  0.011848  0.997633  0.001
9   0.028223   0.9911  0.010496  0.997850  0.001
10  0.026564   0.9919  0.008443  0.998533  0.001
11  0.026505   0.9916  0.007543  0.998717  0.001
12  0.026364   0.9917  0.006613  0.998883  0.001
13  0.025951   0.9922  0.005481  0.999267  0.001
14  0.027985   0.9919  0.005423  0.999133  0.001
15  0.025917   0.9927  0.004657  0.999433  0.001
16  0.026688   0.9917  0.003898  0.999550  0.001
17  0.027756   0.9916  0.003354  0.999750  0.001
18  0.026545   0.9919  0.002912  0.999750  0.001
2020-07-08 09:27:35 DEBUG    : get_Shannon(Model)
```

```
2020-07-08 09:27:35 DEBUG   : Model Entropy: 8.829680442810059
2020-07-08 09:27:35 DEBUG   : Model Entropy normalized: 0.0020053782518305832
2020-07-08 09:27:35 DEBUG   : Layers Sum Entropy: 101.754563331604
2020-07-08 09:27:35 DEBUG    : Layer Entropy: ['8.82968', '8.242498', '15.142519', '14.292492', '14.288188',
'14.844903', '16.483028', '9.631254']
2020-07-08 09:27:35 DEBUG       : Layer Entropy normalized: ['2.2074201107025146', '2.747499465942383',
'3.7856297492980957', '3.573122978210449', '3.572046995162964', '0.02577240102820926', '0.004440471016127488',
'0.100325564543406l7']
2020-07-08 09:27:35 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_750_2',
'DV_625_625_2', 'DV_625_500_2', 'DV_750_250_3', 'DV_1000_500_11']
2020-07-08 09:27:35 DEBUG   : out get_Shannon(Model)
2020-07-08 09:27:35 INFO    : ###### End of step index : 8 of test 32 ##############################
2020-07-08 09:27:35 INFO    : _____________________________________________________________________
2020-07-08 09:27:35 INFO    : ###### Start Evolutionary Step Index: 9 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Activation
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_10"
_____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
=====================================================================================================
DV_0_500_1 (InputLayer)         (None, 28, 28, 1)    0
_____________________________________________________________________________________________________
DV_250_500_1 (Conv2D)           (None, 14, 14, 48)   816         DV_0_500_1[0][0]
_____________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)     2736        DV_250_500_1[0][0]
_____________________________________________________________________________________________________
DV_438_500_1 (Conv2D)           (None, 3, 3, 64)     49216       DV_375_500_1[0][0]
_____________________________________________________________________________________________________
DV_500_500_4 (Flatten)          (None, 576)          0           DV_438_500_1[0][0]
_____________________________________________________________________________________________________
DV_625_750_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
_____________________________________________________________________________________________________
DV_625_625_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
_____________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 576)          0           DV_500_500_4[0][0]
_____________________________________________________________________________________________________
flatten_8 (Flatten)             (None, 576)          0           DV_438_500_1[0][0]
_____________________________________________________________________________________________________
flatten_9 (Flatten)             (None, 1568)         0           DV_625_750_2[0][0]
_____________________________________________________________________________________________________
flatten_10 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
_____________________________________________________________________________________________________
DV_625_500_2 (Dense)            (None, 64)           36864       dropout_3[0][0]
_____________________________________________________________________________________________________
DV_750_250_3_C (Concatenate)    (None, 3712)         0           flatten_8[0][0]
                                                                 flatten_9[0][0]
                                                                 flatten_10[0][0]
_____________________________________________________________________________________________________
DV_750_500_7 (Activation)       (None, 64)           0           DV_625_500_2[0][0]
_____________________________________________________________________________________________________
DV_750_250_3 (Dense)            (None, 32)           118816      DV_750_250_3_C[0][0]
_____________________________________________________________________________________________________
DV_1000_500_12_C (Concatenate)  (None, 672)          0           DV_750_500_7[0][0]
                                                                 DV_750_250_3[0][0]
                                                                 dropout_3[0][0]
```

```
_________________________________________________________________________________________________
DV_1000_500_12 (Dense)          (None, 10)           6730        DV_1000_500_12_C[0][0]
=================================================================================================
Total params: 264,330
Trainable params: 264,330
Non-trainable params: 0
_________________________________________________________________________________________________
None
2020-07-08 09:27:35 INFO     : Original scores - TrnFitness: 0.9919999837875366 ValFitness: False TstFitness:
0.992900013923645 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.4005 - acc: 0.8966 - val_loss: 0.0596 - val_acc: 0.9815
Epoch 2/20 - 1s - loss: 0.0595 - acc: 0.9818 - val_loss: 0.0411 - val_acc: 0.9873
Epoch 3/20 - 1s - loss: 0.0404 - acc: 0.9875 - val_loss: 0.0368 - val_acc: 0.9883
Epoch 4/20 - 1s - loss: 0.0323 - acc: 0.9904 - val_loss: 0.0319 - val_acc: 0.9890
Epoch 5/20 - 1s - loss: 0.0254 - acc: 0.9922 - val_loss: 0.0306 - val_acc: 0.9899
Epoch 6/20 - 1s - loss: 0.0221 - acc: 0.9933 - val_loss: 0.0292 - val_acc: 0.9905
Epoch 7/20 - 1s - loss: 0.0182 - acc: 0.9950 - val_loss: 0.0293 - val_acc: 0.9899

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 8/20 - 1s - loss: 0.0162 - acc: 0.9954 - val_loss: 0.0274 - val_acc: 0.9903
Epoch 9/20 - 1s - loss: 0.0138 - acc: 0.9961 - val_loss: 0.0287 - val_acc: 0.9902

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0115 - acc: 0.9969 - val_loss: 0.0267 - val_acc: 0.9908
Epoch 11/20 - 1s - loss: 0.0108 - acc: 0.9972 - val_loss: 0.0269 - val_acc: 0.9912

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 12/20 - 1s - loss: 0.0093 - acc: 0.9976 - val_loss: 0.0283 - val_acc: 0.9903

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 13/20 - 1s - loss: 0.0086 - acc: 0.9978 - val_loss: 0.0285 - val_acc: 0.9898

Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:27:56 DEBUG    : Metrics(Test loss & Test Accuracy): [0.026715012642150395, 0.9908000230789185]
2020-07-08 09:27:56 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:27:56 INFO     : Current scores - TrnFitness: 0.9977999925613403 TstFitness: 0.9908000230789185
Scaled_TstFitness: 0.9908000230789185
    val_loss  val_acc      loss       acc     lr
0   0.059588   0.9815  0.400505  0.896617  0.001
1   0.041133   0.9873  0.059544  0.981817  0.001
2   0.036825   0.9883  0.040423  0.987500  0.001
3   0.031873   0.9890  0.032270  0.990417  0.001
4   0.030649   0.9899  0.025373  0.992217  0.001
5   0.029165   0.9905  0.022150  0.993283  0.001
6   0.029322   0.9899  0.018200  0.994983  0.001
7   0.027418   0.9903  0.016230  0.995400  0.001
8   0.028681   0.9902  0.013847  0.996050  0.001
9   0.026715   0.9908  0.011512  0.996917  0.001
10  0.026920   0.9912  0.010779  0.997150  0.001
11  0.028259   0.9903  0.009336  0.997583  0.001
12  0.028542   0.9898  0.008556  0.997800  0.001
2020-07-08 09:27:56 DEBUG    : get_Shannon(Model)
2020-07-08 09:27:56 DEBUG    : Model Entropy: 8.825361251831055
2020-07-08 09:27:56 DEBUG    : Model Entropy normalized: 0.0017725168210144717
2020-07-08 09:27:56 DEBUG    : Layers Sum Entropy: 104.30861473083496
2020-07-08 09:27:56 DEBUG     : Layer Entropy: ['8.825361', '8.24152', '15.140001', '14.265633', '14.262846',
'14.816523', '16.428823', '12.327908']
2020-07-08 09:27:56 DEBUG     : Layer Entropy normalized: ['2.2063403129577637', '2.747173309326172',
'3.7850003242492676', '3.566408157348633', '3.565711498260498', '0.0257231295108795 17', '0.004425868391990662',
'0.018345100539071218']
2020-07-08 09:27:56 DEBUG     : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_750_2',
'DV_625_625_2', 'DV_625_500_2', 'DV_750_250_3', 'DV_1000_500_12']
2020-07-08 09:27:56 DEBUG    : out get_Shannon(Model)
2020-07-08 09:27:56 INFO     : ###### End of step index : 9 of test 32 ##############################
2020-07-08 09:27:56 INFO     : ________________________________________________________________
2020-07-08 09:27:56 INFO     : ###### Start Evolutionary Step Index: 10 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
```

```
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Activation
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_11"
```

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_1 (Conv2D)	(None, 14, 14, 48)	816	DV_0_500_1[0][0]
DV_375_500_1 (SeparableConv2D)	(None, 7, 7, 48)	2736	DV_250_500_1[0][0]
DV_438_500_1 (Conv2D)	(None, 3, 3, 64)	49216	DV_375_500_1[0][0]
DV_500_500_4 (Flatten)	(None, 576)	0	DV_438_500_1[0][0]
DV_625_750_2 (Conv2D)	(None, 7, 7, 32)	24576	DV_250_500_1[0][0]
DV_625_625_2 (Conv2D)	(None, 7, 7, 32)	24576	DV_250_500_1[0][0]
dropout_4 (Dropout)	(None, 576)	0	DV_500_500_4[0][0]
flatten_11 (Flatten)	(None, 576)	0	DV_438_500_1[0][0]
flatten_12 (Flatten)	(None, 1568)	0	DV_625_750_2[0][0]
flatten_13 (Flatten)	(None, 1568)	0	DV_625_625_2[0][0]
DV_625_500_2 (Dense)	(None, 64)	36864	dropout_4[0][0]
DV_750_250_3_C (Concatenate)	(None, 3712)	0	flatten_11[0][0] flatten_12[0][0] flatten_13[0][0]
DV_750_500_7 (Activation)	(None, 64)	0	DV_625_500_2[0][0]
DV_750_250_3 (Dense)	(None, 32)	118816	DV_750_250_3_C[0][0]
DV_1000_500_13_C (Concatenate)	(None, 736)	0	DV_750_500_7[0][0] DV_750_250_3[0][0] dropout_4[0][0] DV_625_500_2[0][0]
DV_1000_500_13 (Dense)	(None, 10)	7370	DV_1000_500_13_C[0][0]

```
Total params: 264,970
Trainable params: 264,970
Non-trainable params: 0
```

```
None
2020-07-08 09:27:57 INFO     : Original scores - TrnFitness: 0.9923999905586243 ValFitness: False TstFitness:
0.9929999709129333 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.4716 - acc: 0.8784 - val_loss: 0.0515 - val_acc: 0.9839
Epoch 2/20 - 1s - loss: 0.0499 - acc: 0.9850 - val_loss: 0.0383 - val_acc: 0.9874
Epoch 3/20 - 1s - loss: 0.0343 - acc: 0.9902 - val_loss: 0.0320 - val_acc: 0.9894
Epoch 4/20 - 1s - loss: 0.0267 - acc: 0.9921 - val_loss: 0.0291 - val_acc: 0.9903
Epoch 5/20 - 1s - loss: 0.0220 - acc: 0.9936 - val_loss: 0.0265 - val_acc: 0.9909
Epoch 6/20 - 1s - loss: 0.0178 - acc: 0.9949 - val_loss: 0.0249 - val_acc: 0.9915
```

```
Epoch 7/20 - 1s - loss: 0.0156 - acc: 0.9959 - val_loss: 0.0245 - val_acc: 0.9921
Epoch 8/20 - 1s - loss: 0.0134 - acc: 0.9965 - val_loss: 0.0254 - val_acc: 0.9914

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 9/20 - 1s - loss: 0.0113 - acc: 0.9969 - val_loss: 0.0246 - val_acc: 0.9918

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0104 - acc: 0.9974 - val_loss: 0.0246 - val_acc: 0.9919

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:28:13 DEBUG   : Metrics(Test loss & Test Accuracy): [0.024456385533342836, 0.9921000003814697]
2020-07-08 09:28:13 DEBUG   : Mutant ID: 1594174369341240832
2020-07-08 09:28:13 INFO    : Current scores - TrnFitness: 0.9973666667938232 TstFitness: 0.9921000003814697
Scaled_TstFitness: 0.9921000003814697
   val_loss  val_acc      loss       acc     lr
0  0.051461   0.9839  0.471587  0.878350  0.001
1  0.038332   0.9874  0.049941  0.985017  0.001
2  0.032013   0.9894  0.034325  0.990167  0.001
3  0.029053   0.9903  0.026733  0.992067  0.001
4  0.026472   0.9909  0.022028  0.993550  0.001
5  0.024905   0.9915  0.017819  0.994950  0.001
6  0.024456   0.9921  0.015573  0.995867  0.001
7  0.025403   0.9914  0.013383  0.996500  0.001
8  0.024560   0.9918  0.011260  0.996933  0.001
9  0.024554   0.9919  0.010444  0.997367  0.001
2020-07-08 09:28:14 DEBUG   : get_Shannon(Model)
2020-07-08 09:28:14 DEBUG   : Model Entropy: 8.812922477722168
2020-07-08 09:28:14 DEBUG   : Model Entropy normalized: 0.0017475555180888694
2020-07-08 09:28:14 DEBUG   : Layers Sum Entropy: 104.36413288116455
2020-07-08 09:28:14 DEBUG    : Layer Entropy: ['8.8129225', '8.24308', '15.139083', '14.246231', '14.248705',
'14.798869', '16.409634', '12.465609']
2020-07-08 09:28:14 DEBUG       : Layer Entropy normalized: ['2.203230619430542', '2.7476933797200522',
'3.7847707271575928', '3.5615577697753906', '3.56217622756958', '0.025692481133672927', '0.004420698716722685',
'0.0169369682021763']
2020-07-08 09:28:14 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_750_2',
'DV_625_625_2', 'DV_625_500_2', 'DV_750_250_3', 'DV_1000_500_13']
2020-07-08 09:28:14 DEBUG   : out get_Shannon(Model)
2020-07-08 09:28:14 INFO    : ###### End of step index : 10 of test 32 ##############################
2020-07-08 09:28:14 INFO    : ____________________________________________________________
2020-07-08 09:28:14 INFO    : ###### Start Evolutionary Step Index: 11 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.75, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.8125, 0.5)] Node_Type: GaussianDropout
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_12"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816         DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)       2736        DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216       DV_375_500_1[0][0]
```

```
____________________________________________________________________________________________________
DV_500_500_4 (Flatten)          (None, 576)          0           DV_438_500_1[0][0]
____________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 576)          0           DV_500_500_4[0][0]
____________________________________________________________________________________________________
DV_625_750_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
____________________________________________________________________________________________________
DV_625_625_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
____________________________________________________________________________________________________
DV_625_500_2 (Dense)            (None, 64)           36864       dropout_5[0][0]
____________________________________________________________________________________________________
flatten_14 (Flatten)            (None, 576)          0           DV_438_500_1[0][0]
____________________________________________________________________________________________________
flatten_15 (Flatten)            (None, 1568)         0           DV_625_750_2[0][0]
____________________________________________________________________________________________________
flatten_16 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
____________________________________________________________________________________________________
DV_750_500_7 (Activation)       (None, 64)           0           DV_625_500_2[0][0]
____________________________________________________________________________________________________
DV_750_250_3_C (Concatenate)    (None, 3712)         0           flatten_14[0][0]
                                                                 flatten_15[0][0]
                                                                 flatten_16[0][0]
____________________________________________________________________________________________________
DV_812_500_1 (Dense)            (None, 64)           4160        DV_750_500_7[0][0]
____________________________________________________________________________________________________
DV_750_250_3 (Dense)            (None, 32)           118816      DV_750_250_3_C[0][0]
____________________________________________________________________________________________________
DV_875_500_2 (GaussianDropout)  (None, 64)           0           DV_812_500_1[0][0]
____________________________________________________________________________________________________
DV_1000_500_15_C (Concatenate)  (None, 736)          0           DV_750_250_3[0][0]
                                                                 dropout_5[0][0]
                                                                 DV_625_500_2[0][0]
                                                                 DV_875_500_2[0][0]
____________________________________________________________________________________________________
DV_1000_500_15 (Dense)          (None, 10)           7370        DV_1000_500_15_C[0][0]
====================================================================================================
Total params: 269,130
Trainable params: 269,130
Non-trainable params: 0
____________________________________________________________________________________________________
None
2020-07-08 09:28:15 INFO     : Original scores - TrnFitness: 0.9926999807357788 ValFitness: False TstFitness:
0.9926999807357788 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.3611 - acc: 0.8974 - val_loss: 0.0446 - val_acc: 0.9855
Epoch 2/20 - 1s - loss: 0.0449 - acc: 0.9865 - val_loss: 0.0319 - val_acc: 0.9897
Epoch 3/20 - 1s - loss: 0.0312 - acc: 0.9908 - val_loss: 0.0284 - val_acc: 0.9902
Epoch 4/20 - 1s - loss: 0.0239 - acc: 0.9926 - val_loss: 0.0252 - val_acc: 0.9917
Epoch 5/20 - 1s - loss: 0.0204 - acc: 0.9939 - val_loss: 0.0246 - val_acc: 0.9917
Epoch 6/20 - 1s - loss: 0.0165 - acc: 0.9952 - val_loss: 0.0237 - val_acc: 0.9919
Epoch 7/20 - 1s - loss: 0.0142 - acc: 0.9959 - val_loss: 0.0232 - val_acc: 0.9925
Epoch 8/20 - 1s - loss: 0.0124 - acc: 0.9965 - val_loss: 0.0230 - val_acc: 0.9929
Epoch 9/20 - 1s - loss: 0.0101 - acc: 0.9973 - val_loss: 0.0227 - val_acc: 0.9928
Epoch 10/20 - 1s - loss: 0.0087 - acc: 0.9976 - val_loss: 0.0218 - val_acc: 0.9931
Epoch 11/20 - 1s - loss: 0.0077 - acc: 0.9981 - val_loss: 0.0216 - val_acc: 0.9929
Epoch 12/20 - 1s - loss: 0.0071 - acc: 0.9983 - val_loss: 0.0226 - val_acc: 0.9933

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 13/20 - 1s - loss: 0.0068 - acc: 0.9983 - val_loss: 0.0222 - val_acc: 0.9933

Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 14/20 - 1s - loss: 0.0054 - acc: 0.9988 - val_loss: 0.0220 - val_acc: 0.9931

Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:28:40 DEBUG    : Metrics(Test loss & Test Accuracy): [0.021605231425212697, 0.992900013923645]
2020-07-08 09:28:40 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:28:40 INFO     : Current scores - TrnFitness: 0.9987999796867371 TstFitness: 0.992900013923645
Scaled_TstFitness: 0.992900013923645
    val_loss  val_acc      loss       acc     lr
0   0.044563   0.9855  0.361072  0.897417  0.001
1   0.031896   0.9897  0.044883  0.986500  0.001
```

```
2   0.028402   0.9902  0.031245  0.990850  0.001
3   0.025247   0.9917  0.023894  0.992600  0.001
4   0.024588   0.9917  0.020395  0.993900  0.001
5   0.023653   0.9919  0.016484  0.995233  0.001
6   0.023225   0.9925  0.014170  0.995950  0.001
7   0.022989   0.9929  0.012405  0.996483  0.001
8   0.022736   0.9928  0.010121  0.997250  0.001
9   0.021756   0.9931  0.008748  0.997600  0.001
10  0.021605   0.9929  0.007715  0.998117  0.001
11  0.022572   0.9933  0.007071  0.998317  0.001
12  0.022174   0.9933  0.006763  0.998283  0.001
13  0.021964   0.9931  0.005397  0.998800  0.001
2020-07-08 09:28:41 DEBUG   : get_Shannon(Model)
2020-07-08 09:28:41 DEBUG   : Model Entropy: 8.806816101074219
2020-07-08 09:28:41 DEBUG   : Model Entropy normalized: 0.0017244597809035088
2020-07-08 09:28:41 DEBUG   : Layers Sum Entropy: 115.93941116333008
2020-07-08 09:28:41 DEBUG    : Layer Entropy: ['8.806816', '8.240873', '15.136676', '14.22297', '14.220619',
'14.7776165', '11.705934', '16.384174', '12.443732']
2020-07-08 09:28:41 DEBUG       : Layer Entropy normalized: ['2.2017040252685547', '2.746957778930664',
'3.7841689586639404', '3.5557425022125244', '3.555154800415039', '0.02565558420287238', '0.18290521204471588',
'0.0044138400719083 59', '0.016907244920730 59']
2020-07-08 09:28:41 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_750_2',
'DV_625_625_2', 'DV_625_500_2', 'DV_812_500_1', 'DV_750_250_3', 'DV_1000_500_15']
2020-07-08 09:28:41 DEBUG   : out get_Shannon(Model)
2020-07-08 09:28:41 INFO    : ###### End of step index : 11 of test 32 ##############################
2020-07-08 09:28:41 INFO    : ____________________________________________________________________
2020-07-08 09:28:41 INFO    : ###### Start Evolutionary Step Index: 12 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.75, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.8125, 0.5)] Node_Type: GaussianDropout
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_13"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816         DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)       2736        DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216       DV_375_500_1[0][0]
_________________________________________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0           DV_438_500_1[0][0]
_________________________________________________________________________________________________
dropout_6 (Dropout)          (None, 576)               0           DV_500_500_4[0][0]
_________________________________________________________________________________________________
DV_625_750_2 (Conv2D)        (None, 7, 7, 32)          24576       DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_625_625_2 (Conv2D)        (None, 7, 7, 32)          24576       DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_625_500_2 (Dense)         (None, 64)                36864       dropout_6[0][0]
_________________________________________________________________________________________________
flatten_17 (Flatten)         (None, 1568)              0           DV_625_750_2[0][0]
```

```
__________________________________________________________________________________________________
flatten_18 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
__________________________________________________________________________________________________
flatten_19 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
__________________________________________________________________________________________________
DV_750_500_8 (Activation)       (None, 64)           0           DV_625_500_2[0][0]
__________________________________________________________________________________________________
DV_750_250_4_C (Concatenate)    (None, 4704)         0           flatten_17[0][0]
                                                                 flatten_18[0][0]
                                                                 flatten_19[0][0]
__________________________________________________________________________________________________
DV_812_500_2 (Dense)            (None, 64)           4160        DV_750_500_8[0][0]
__________________________________________________________________________________________________
DV_750_250_4 (Dense)            (None, 32)           150560      DV_750_250_4_C[0][0]
__________________________________________________________________________________________________
DV_875_500_3 (GaussianDropout)  (None, 64)           0           DV_812_500_2[0][0]
__________________________________________________________________________________________________
DV_1000_500_16_C (Concatenate)  (None, 736)          0           DV_750_250_4[0][0]
                                                                 dropout_6[0][0]
                                                                 DV_625_500_2[0][0]
                                                                 DV_875_500_3[0][0]
__________________________________________________________________________________________________
DV_1000_500_16 (Dense)          (None, 10)           7370        DV_1000_500_16_C[0][0]
==================================================================================================
Total params: 300,874
Trainable params: 300,874
Non-trainable params: 0
__________________________________________________________________________________________________
None
2020-07-08 09:28:41 INFO     : Original scores - TrnFitness: 0.9921000003814697 ValFitness: False TstFitness:
0.9927999973297119 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.2826 - acc: 0.9194 - val_loss: 0.0456 - val_acc: 0.9846
Epoch 2/20 - 1s - loss: 0.0450 - acc: 0.9865 - val_loss: 0.0311 - val_acc: 0.9894
Epoch 3/20 - 1s - loss: 0.0321 - acc: 0.9902 - val_loss: 0.0271 - val_acc: 0.9912
Epoch 4/20 - 1s - loss: 0.0253 - acc: 0.9927 - val_loss: 0.0264 - val_acc: 0.9915
Epoch 5/20 - 1s - loss: 0.0202 - acc: 0.9940 - val_loss: 0.0244 - val_acc: 0.9913
Epoch 6/20 - 1s - loss: 0.0164 - acc: 0.9950 - val_loss: 0.0247 - val_acc: 0.9922

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 7/20 - 1s - loss: 0.0145 - acc: 0.9959 - val_loss: 0.0229 - val_acc: 0.9927
Epoch 8/20 - 1s - loss: 0.0123 - acc: 0.9967 - val_loss: 0.0237 - val_acc: 0.9925

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 9/20 - 1s - loss: 0.0115 - acc: 0.9969 - val_loss: 0.0236 - val_acc: 0.9924

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0105 - acc: 0.9968 - val_loss: 0.0230 - val_acc: 0.9926

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:28:59 DEBUG    : Metrics(Test loss & Test Accuracy): [0.022913675841756048, 0.9926999807357788]
2020-07-08 09:28:59 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:28:59 INFO     : Current scores - TrnFitness: 0.9968166947364807 TstFitness: 0.9926999807357788
Scaled_TstFitness: 0.9926999807357788
   val_loss  val_acc      loss       acc     lr
0  0.045628   0.9846  0.282630  0.919400  0.001
1  0.031071   0.9894  0.045017  0.986500  0.001
2  0.027071   0.9912  0.032115  0.990200  0.001
3  0.026361   0.9915  0.025268  0.992733  0.001
4  0.024409   0.9913  0.020209  0.993967  0.001
5  0.024688   0.9922  0.016418  0.995017  0.001
6  0.022914   0.9927  0.014482  0.995850  0.001
7  0.023664   0.9925  0.012283  0.996667  0.001
8  0.023593   0.9924  0.011529  0.996867  0.001
9  0.023028   0.9926  0.010464  0.996817  0.001
2020-07-08 09:29:00 DEBUG    : get_Shannon(Model)
2020-07-08 09:29:00 DEBUG    : Model Entropy: 8.805408477783203
2020-07-08 09:29:00 DEBUG    : Model Entropy normalized: 0.0014437462662376132
2020-07-08 09:29:00 DEBUG    : Layers Sum Entropy: 116.36462020874023
2020-07-08 09:29:00 DEBUG    : Layer Entropy: ['8.8054085', '8.240545', '15.135711', '14.209562', '14.205687',
'14.76477', '11.705944', '16.841177', '12.455816']
```

```
2020-07-08 09:29:00 DEBUG : Layer Entropy normalized: ['2.201352119445801', '2.7468484242757163',
'3.7839276790618896', '3.5523905754089355', '3.551421642303467', '0.025633280475934345', '0.182905375957489',
'0.0035801821825455645', '0.016923663408859917']
2020-07-08 09:29:00 DEBUG : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_750_2',
'DV_625_625_2', 'DV_625_500_2', 'DV_812_500_2', 'DV_750_250_4', 'DV_1000_500_16']
2020-07-08 09:29:00 DEBUG : out get_Shannon(Model)
2020-07-08 09:29:00 INFO : ###### End of step index : 12 of test 32 ##############################
2020-07-08 09:29:00 INFO : _____
2020-07-08 09:29:00 INFO : ###### Start Evolutionary Step Index: 13 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Dropout
NodeDict['From']: [(0.75, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.8125, 0.5)] Node_Type: GaussianDropout
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_14"
```

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_1 (Conv2D)	(None, 14, 14, 48)	816	DV_0_500_1[0][0]
DV_375_500_1 (SeparableConv2D)	(None, 7, 7, 48)	2736	DV_250_500_1[0][0]
DV_438_500_1 (Conv2D)	(None, 3, 3, 64)	49216	DV_375_500_1[0][0]
DV_500_500_4 (Flatten)	(None, 576)	0	DV_438_500_1[0][0]
dropout_7 (Dropout)	(None, 576)	0	DV_500_500_4[0][0]
DV_625_750_2 (Conv2D)	(None, 7, 7, 32)	24576	DV_250_500_1[0][0]
DV_625_625_2 (Conv2D)	(None, 7, 7, 32)	24576	DV_250_500_1[0][0]
DV_625_500_2 (Dense)	(None, 64)	36864	dropout_7[0][0]
flatten_20 (Flatten)	(None, 1568)	0	DV_625_750_2[0][0]
flatten_21 (Flatten)	(None, 1568)	0	DV_625_625_2[0][0]
flatten_22 (Flatten)	(None, 1568)	0	DV_625_625_2[0][0]
DV_750_500_9 (Dropout)	(None, 64)	0	DV_625_500_2[0][0]
DV_750_250_5_C (Concatenate)	(None, 4704)	0	flatten_20[0][0] flatten_21[0][0] flatten_22[0][0]
DV_812_500_3 (Dense)	(None, 64)	4160	DV_750_500_9[0][0]
DV_750_250_5 (Dense)	(None, 32)	150560	DV_750_250_5_C[0][0]
DV_875_500_4 (GaussianDropout)	(None, 64)	0	DV_812_500_3[0][0]
DV_1000_500_17_C (Concatenate)	(None, 736)	0	DV_750_250_5[0][0]

```
                                                                 dropout_7[0][0]
                                                                 DV_625_500_2[0][0]
                                                                 DV_875_500_4[0][0]
____________________________________________________________________________________________________
DV_1000_500_17 (Dense)          (None, 10)           7370        DV_1000_500_17_C[0][0]
====================================================================================================
Total params: 300,874
Trainable params: 300,874
Non-trainable params: 0
____________________________________________________________________________________________________
None
2020-07-08 09:29:01 INFO     : Original scores - TrnFitness: 0.9925000071525574 ValFitness: False TstFitness:
0.9933000206947327 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.2863 - acc: 0.9176 - val_loss: 0.0458 - val_acc: 0.9858
Epoch 2/20 - 1s - loss: 0.0414 - acc: 0.9872 - val_loss: 0.0324 - val_acc: 0.9890
Epoch 3/20 - 1s - loss: 0.0301 - acc: 0.9910 - val_loss: 0.0280 - val_acc: 0.9910
Epoch 4/20 - 1s - loss: 0.0225 - acc: 0.9933 - val_loss: 0.0267 - val_acc: 0.9912
Epoch 5/20 - 1s - loss: 0.0185 - acc: 0.9944 - val_loss: 0.0257 - val_acc: 0.9918
Epoch 6/20 - 1s - loss: 0.0152 - acc: 0.9956 - val_loss: 0.0247 - val_acc: 0.9924
Epoch 7/20 - 1s - loss: 0.0131 - acc: 0.9961 - val_loss: 0.0250 - val_acc: 0.9920

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 8/20 - 1s - loss: 0.0115 - acc: 0.9965 - val_loss: 0.0252 - val_acc: 0.9922

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 9/20 - 1s - loss: 0.0102 - acc: 0.9972 - val_loss: 0.0251 - val_acc: 0.9920

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:29:18 DEBUG    : Metrics(Test loss & Test Accuracy): [0.024680099019553745, 0.9923999905586243]
2020-07-08 09:29:18 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:29:18 INFO     : Current scores - TrnFitness: 0.9971666932106018 TstFitness: 0.9923999905586243
Scaled_TstFitness: 0.9923999905586243
   val_loss  val_acc      loss       acc     lr
0  0.045785   0.9858  0.286273  0.917567  0.001
1  0.032379   0.9890  0.041394  0.987150  0.001
2  0.028022   0.9910  0.030124  0.991000  0.001
3  0.026747   0.9912  0.022549  0.993267  0.001
4  0.025740   0.9918  0.018455  0.994450  0.001
5  0.024680   0.9924  0.015177  0.995567  0.001
6  0.025043   0.9920  0.013126  0.996083  0.001
7  0.025196   0.9922  0.011503  0.996533  0.001
8  0.025092   0.9920  0.010178  0.997167  0.001
2020-07-08 09:29:19 DEBUG    : get_Shannon(Model)
2020-07-08 09:29:19 DEBUG    : Model Entropy: 8.805061340332031
2020-07-08 09:29:19 DEBUG    : Model Entropy normalized: 0.0014436893491280589
2020-07-08 09:29:19 DEBUG    : Layers Sum Entropy: 116.3702278137207
2020-07-08 09:29:19 DEBUG    : Layer Entropy: ['8.805061', '8.242043', '15.137336', '14.201311', '14.195278',
'14.75647', '11.705758', '16.86192', '12.465052']
2020-07-08 09:29:19 DEBUG    : Layer Entropy normalized: ['2.201265335083008', '2.7473475138346353',
'3.7843339443206787', '3.550327777862549', '3.5488195419311523', '0.025618871053059895', '0.18290247023105621',
'0.0035845917098376217', '0.016936211482338284']
2020-07-08 09:29:19 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_750_2',
'DV_625_625_2', 'DV_625_500_2', 'DV_812_500_3', 'DV_750_250_5', 'DV_1000_500_17']
2020-07-08 09:29:19 DEBUG    : out get_Shannon(Model)
2020-07-08 09:29:19 INFO     : ###### End of step index : 13 of test 32 ##############################
2020-07-08 09:29:19 INFO     : ____________________________________________________________________
2020-07-08 09:29:19 INFO     : ###### Start Evolutionary Step Index: 14 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Dropout
NodeDict['From']: [(0.75, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.8125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.84375, 0.5)] Node_Type: GaussianDropout
{'Age': 0,
```

```
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_15"
```

Layer (type)	Output Shape	Param #	Connected to
DV_0_500_1 (InputLayer)	(None, 28, 28, 1)	0	
DV_250_500_1 (Conv2D)	(None, 14, 14, 48)	816	DV_0_500_1[0][0]
DV_375_500_1 (SeparableConv2D)	(None, 7, 7, 48)	2736	DV_250_500_1[0][0]
DV_438_500_1 (Conv2D)	(None, 3, 3, 64)	49216	DV_375_500_1[0][0]
DV_500_500_4 (Flatten)	(None, 576)	0	DV_438_500_1[0][0]
dropout_8 (Dropout)	(None, 576)	0	DV_500_500_4[0][0]
DV_625_500_2 (Dense)	(None, 64)	36864	dropout_8[0][0]
DV_625_750_2 (Conv2D)	(None, 7, 7, 32)	24576	DV_250_500_1[0][0]
DV_625_625_2 (Conv2D)	(None, 7, 7, 32)	24576	DV_250_500_1[0][0]
DV_750_500_9 (Dropout)	(None, 64)	0	DV_625_500_2[0][0]
flatten_23 (Flatten)	(None, 1568)	0	DV_625_750_2[0][0]
flatten_24 (Flatten)	(None, 1568)	0	DV_625_625_2[0][0]
flatten_25 (Flatten)	(None, 1568)	0	DV_625_625_2[0][0]
DV_812_500_3 (Dense)	(None, 64)	4160	DV_750_500_9[0][0]
DV_750_250_5_C (Concatenate)	(None, 4704)	0	flatten_23[0][0] flatten_24[0][0] flatten_25[0][0]
DV_844_500_1 (Dense)	(None, 64)	4096	DV_812_500_3[0][0]
DV_750_250_5 (Dense)	(None, 32)	150560	DV_750_250_5_C[0][0]
DV_875_500_5 (GaussianDropout)	(None, 64)	0	DV_844_500_1[0][0]
DV_1000_500_18_C (Concatenate)	(None, 736)	0	DV_750_250_5[0][0] dropout_8[0][0] DV_625_500_2[0][0] DV_875_500_5[0][0]
DV_1000_500_18 (Dense)	(None, 10)	7370	DV_1000_500_18_C[0][0]

```
Total params: 304,970
Trainable params: 304,970
Non-trainable params: 0
```

```
None
2020-07-08 09:29:21 INFO     : Original scores - TrnFitness: 0.9937999844551086 ValFitness: False TstFitness:
0.993399977684021 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.3011 - acc: 0.9155 - val_loss: 0.0400 - val_acc: 0.9873
Epoch 2/20 - 1s - loss: 0.0398 - acc: 0.9886 - val_loss: 0.0304 - val_acc: 0.9906
Epoch 3/20 - 1s - loss: 0.0284 - acc: 0.9918 - val_loss: 0.0262 - val_acc: 0.9919
Epoch 4/20 - 1s - loss: 0.0224 - acc: 0.9932 - val_loss: 0.0242 - val_acc: 0.9920
Epoch 5/20 - 1s - loss: 0.0190 - acc: 0.9943 - val_loss: 0.0236 - val_acc: 0.9920
```

```
Epoch 6/20 - 1s - loss: 0.0155 - acc: 0.9956 - val_loss: 0.0241 - val_acc: 0.9921

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 7/20 - 1s - loss: 0.0127 - acc: 0.9963 - val_loss: 0.0225 - val_acc: 0.9927
Epoch 8/20 - 1s - loss: 0.0112 - acc: 0.9966 - val_loss: 0.0223 - val_acc: 0.9928
Epoch 9/20 - 1s - loss: 0.0100 - acc: 0.9973 - val_loss: 0.0240 - val_acc: 0.9929

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0098 - acc: 0.9971 - val_loss: 0.0237 - val_acc: 0.9930

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 11/20 - 1s - loss: 0.0091 - acc: 0.9972 - val_loss: 0.0231 - val_acc: 0.9936

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:29:41 DEBUG   : Metrics(Test loss & Test Accuracy): [0.022280257258782514, 0.9927999973297119]
2020-07-08 09:29:41 DEBUG   : Mutant ID: 1594174369341240832
2020-07-08 09:29:41 INFO    : Current scores - TrnFitness: 0.9971500039100647 TstFitness: 0.9927999973297119
Scaled_TstFitness: 0.9927999973297119
    val_loss  val_acc      loss       acc     lr
0   0.040006   0.9873  0.301086  0.915500  0.001
1   0.030367   0.9906  0.039783  0.988567  0.001
2   0.026245   0.9919  0.028382  0.991767  0.001
3   0.024176   0.9920  0.022397  0.993217  0.001
4   0.023569   0.9920  0.018961  0.994267  0.001
5   0.024119   0.9921  0.015478  0.995650  0.001
6   0.022505   0.9927  0.012734  0.996333  0.001
7   0.022280   0.9928  0.011231  0.996550  0.001
8   0.024039   0.9929  0.009963  0.997283  0.001
9   0.023721   0.9930  0.009824  0.997117  0.001
10  0.023141   0.9936  0.009071  0.997150  0.001
2020-07-08 09:29:42 DEBUG   : get_Shannon(Model)
2020-07-08 09:29:42 DEBUG   : Model Entropy: 8.804018020629883
2020-07-08 09:29:42 DEBUG   : Model Entropy normalized: 0.0014285279929628237
2020-07-08 09:29:42 DEBUG   : Layers Sum Entropy: 128.0015172958374
2020-07-08 09:29:42 DEBUG    : Layer Entropy: ['8.804018', '8.243589', '15.137894', '14.748881', '14.188121',
'14.183282', '11.693251', '11.7189665', '16.827774', '12.455741']
2020-07-08 09:29:42 DEBUG       : Layer Entropy normalized: ['2.2010045051574707', '2.7478631337483725',
'3.784473419189453', '0.025605696770879958', '3.547030210494995', '3.545820474624634', '0.1827070415019989',
'0.183108851313591', '0.0035773329183357914', '0.01692356104436128']
2020-07-08 09:29:42 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_500_2',
'DV_625_750_2', 'DV_625_625_2', 'DV_812_500_3', 'DV_844_500_1', 'DV_750_250_5', 'DV_1000_500_18']
2020-07-08 09:29:42 DEBUG   : out get_Shannon(Model)
2020-07-08 09:29:42 INFO    : ###### End of step index : 14 of test 32 ##############################
2020-07-08 09:29:42 INFO    : ________________________________________________________________________
2020-07-08 09:29:42 INFO    : ###### Start Evolutionary Step Index: 15 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Dropout
NodeDict['From']: [(0.75, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.78125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.8125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.84375, 0.5)] Node_Type: GaussianDropout
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_16"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
```

```
==================================================================================================
DV_0_500_1 (InputLayer)         (None, 28, 28, 1)    0
__________________________________________________________________________________________________
DV_250_500_1 (Conv2D)           (None, 14, 14, 48)   816         DV_0_500_1[0][0]
__________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)     2736        DV_250_500_1[0][0]
__________________________________________________________________________________________________
DV_438_500_1 (Conv2D)           (None, 3, 3, 64)     49216       DV_375_500_1[0][0]
__________________________________________________________________________________________________
DV_500_500_4 (Flatten)          (None, 576)          0           DV_438_500_1[0][0]
__________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 576)          0           DV_500_500_4[0][0]
__________________________________________________________________________________________________
DV_625_500_2 (Dense)            (None, 64)           36864       dropout_9[0][0]
__________________________________________________________________________________________________
DV_750_500_9 (Dropout)          (None, 64)           0           DV_625_500_2[0][0]
__________________________________________________________________________________________________
DV_625_750_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
__________________________________________________________________________________________________
DV_625_625_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
__________________________________________________________________________________________________
DV_781_500_2 (Activation)       (None, 64)           0           DV_750_500_9[0][0]
__________________________________________________________________________________________________
flatten_26 (Flatten)            (None, 1568)         0           DV_625_750_2[0][0]
__________________________________________________________________________________________________
flatten_27 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
__________________________________________________________________________________________________
flatten_28 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
__________________________________________________________________________________________________
DV_812_500_5 (Dense)            (None, 64)           4160        DV_781_500_2[0][0]
__________________________________________________________________________________________________
DV_750_250_5_C (Concatenate)    (None, 4704)         0           flatten_26[0][0]
                                                                 flatten_27[0][0]
                                                                 flatten_28[0][0]
__________________________________________________________________________________________________
DV_844_500_3 (Dense)            (None, 64)           4096        DV_812_500_5[0][0]
__________________________________________________________________________________________________
DV_750_250_5 (Dense)            (None, 32)           150560      DV_750_250_5_C[0][0]
__________________________________________________________________________________________________
DV_875_500_7 (GaussianDropout)  (None, 64)           0           DV_844_500_3[0][0]
__________________________________________________________________________________________________
DV_1000_500_20_C (Concatenate)  (None, 736)          0           DV_750_250_5[0][0]
                                                                 dropout_9[0][0]
                                                                 DV_625_500_2[0][0]
                                                                 DV_875_500_7[0][0]
__________________________________________________________________________________________________
DV_1000_500_20 (Dense)          (None, 10)           7370        DV_1000_500_20_C[0][0]
==================================================================================================
Total params: 304,970
Trainable params: 304,970
Non-trainable params: 0
__________________________________________________________________________________________________
None
2020-07-08 09:29:43 INFO     : Original scores - TrnFitness: 0.992900013923645 ValFitness: False TstFitness:
0.993399977684021 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.3375 - acc: 0.9046 - val_loss: 0.0447 - val_acc: 0.9854
Epoch 2/20 - 1s - loss: 0.0418 - acc: 0.9877 - val_loss: 0.0325 - val_acc: 0.9888
Epoch 3/20 - 1s - loss: 0.0279 - acc: 0.9918 - val_loss: 0.0291 - val_acc: 0.9903
Epoch 4/20 - 1s - loss: 0.0222 - acc: 0.9933 - val_loss: 0.0267 - val_acc: 0.9915
Epoch 5/20 - 1s - loss: 0.0173 - acc: 0.9951 - val_loss: 0.0241 - val_acc: 0.9921
Epoch 6/20 - 1s - loss: 0.0140 - acc: 0.9958 - val_loss: 0.0236 - val_acc: 0.9917
Epoch 7/20 - 1s - loss: 0.0126 - acc: 0.9964 - val_loss: 0.0247 - val_acc: 0.9924

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 8/20 - 1s - loss: 0.0109 - acc: 0.9967 - val_loss: 0.0252 - val_acc: 0.9922

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 9/20 - 1s - loss: 0.0103 - acc: 0.9970 - val_loss: 0.0244 - val_acc: 0.9921

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
```

```
2020-07-08 09:30:01 DEBUG   : Metrics(Test loss & Test Accuracy): [0.023640528858255128, 0.9916999936103821]
2020-07-08 09:30:01 DEBUG   : Mutant ID: 1594174369341240832
2020-07-08 09:30:01 INFO    : Current scores - TrnFitness: 0.9969666600227356 TstFitness: 0.9916999936103821
Scaled_TstFitness: 0.9916999936103821
   val_loss  val_acc      loss       acc     lr
0  0.044672   0.9854  0.337525  0.904600  0.001
1  0.032454   0.9888  0.041791  0.987700  0.001
2  0.029128   0.9903  0.027916  0.991800  0.001
3  0.026721   0.9915  0.022151  0.993250  0.001
4  0.024103   0.9921  0.017267  0.995100  0.001
5  0.023641   0.9917  0.014030  0.995817  0.001
6  0.024709   0.9924  0.012633  0.996433  0.001
7  0.025238   0.9922  0.010858  0.996700  0.001
8  0.024434   0.9921  0.010323  0.996967  0.001
2020-07-08 09:30:03 DEBUG   : get_Shannon(Model)
2020-07-08 09:30:03 DEBUG   : Model Entropy: 8.80427360534668
2020-07-08 09:30:03 DEBUG   : Model Entropy normalized: 0.0014285694637914458
2020-07-08 09:30:03 DEBUG   : Layers Sum Entropy: 127.94102954864502
2020-07-08 09:30:03 DEBUG    : Layer Entropy: ['8.804274', '8.23775', '15.139832', '14.738966', '14.17736',
'14.172667', '11.703041', '11.706749', '16.802256', '12.458137']
2020-07-08 09:30:03 DEBUG    : Layer Entropy normalized: ['2.20106840133667', '2.7459166844685874',
'3.7849578857421875', '0.02558848261833191', '3.544339895248413', '3.5431666374206543', '0.18286001682281494',
'0.18291795253753662', '0.0035719080847136826', '0.016926815976267277']
2020-07-08 09:30:03 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_500_2',
'DV_625_750_2', 'DV_625_625_2', 'DV_812_500_5', 'DV_844_500_3', 'DV_750_250_5', 'DV_1000_500_20']
2020-07-08 09:30:03 DEBUG   : out get_Shannon(Model)
2020-07-08 09:30:03 INFO    : ###### End of step index : 15 of test 32 ##############################
2020-07-08 09:30:03 INFO    : _________________________________________________________________
2020-07-08 09:30:03 INFO    : ###### Start Evolutionary Step Index: 16 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Dropout
NodeDict['From']: [(0.75, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.78125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.8125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.84375, 0.5)] Node_Type: GaussianDropout
NodeDict['From']: [(0.875, 0.5)] Node_Type: AlphaDropout
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_17"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816         DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)       2736        DV_250_500_1[0][0]
_________________________________________________________________________________________________
dropout_10 (Dropout)         (None, 7, 7, 48)          0           DV_375_500_1[0][0]
_________________________________________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216       dropout_10[0][0]
_________________________________________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0           DV_438_500_1[0][0]
_________________________________________________________________________________________________
dropout_11 (Dropout)         (None, 576)               0           DV_500_500_4[0][0]
_________________________________________________________________________________________________
```

```
DV_625_500_2 (Dense)            (None, 64)           36864       dropout_11[0][0]
_____________________________________________________________________________________________
DV_750_500_9 (Dropout)          (None, 64)           0           DV_625_500_2[0][0]
_____________________________________________________________________________________________
DV_781_500_2 (Activation)       (None, 64)           0           DV_750_500_9[0][0]
_____________________________________________________________________________________________
DV_625_750_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
_____________________________________________________________________________________________
DV_625_625_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
_____________________________________________________________________________________________
DV_812_500_5 (Dense)            (None, 64)           4160        DV_781_500_2[0][0]
_____________________________________________________________________________________________
flatten_29 (Flatten)            (None, 1568)         0           DV_625_750_2[0][0]
_____________________________________________________________________________________________
flatten_30 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
_____________________________________________________________________________________________
flatten_31 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
_____________________________________________________________________________________________
DV_844_500_3 (Dense)            (None, 64)           4096        DV_812_500_5[0][0]
_____________________________________________________________________________________________
DV_750_250_5_C (Concatenate)    (None, 4704)         0           flatten_29[0][0]
                                                                 flatten_30[0][0]
                                                                 flatten_31[0][0]
_____________________________________________________________________________________________
DV_875_500_7 (GaussianDropout)  (None, 64)           0           DV_844_500_3[0][0]
_____________________________________________________________________________________________
DV_750_250_5 (Dense)            (None, 32)           150560      DV_750_250_5_C[0][0]
_____________________________________________________________________________________________
DV_938_500_1 (AlphaDropout)     (None, 64)           0           DV_875_500_7[0][0]
_____________________________________________________________________________________________
DV_1000_500_21_C (Concatenate)  (None, 736)          0           DV_750_250_5[0][0]
                                                                 dropout_11[0][0]
                                                                 DV_625_500_2[0][0]
                                                                 DV_938_500_1[0][0]
_____________________________________________________________________________________________
DV_1000_500_21 (Dense)          (None, 10)           7370        DV_1000_500_21_C[0][0]
=============================================================================================
Total params: 304,970
Trainable params: 304,970
Non-trainable params: 0
_____________________________________________________________________________________________
None
2020-07-08 09:30:04 INFO     : Original scores - TrnFitness: 0.9932000041007996 ValFitness: False TstFitness:
0.9934999942779541 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.3526 - acc: 0.9000 - val_loss: 0.0427 - val_acc: 0.9869
Epoch 2/20 - 1s - loss: 0.0513 - acc: 0.9839 - val_loss: 0.0300 - val_acc: 0.9900
Epoch 3/20 - 1s - loss: 0.0391 - acc: 0.9880 - val_loss: 0.0273 - val_acc: 0.9913
Epoch 4/20 - 1s - loss: 0.0330 - acc: 0.9900 - val_loss: 0.0261 - val_acc: 0.9918
Epoch 5/20 - 1s - loss: 0.0268 - acc: 0.9913 - val_loss: 0.0263 - val_acc: 0.9914

Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 6/20 - 1s - loss: 0.0230 - acc: 0.9925 - val_loss: 0.0243 - val_acc: 0.9916
Epoch 7/20 - 1s - loss: 0.0214 - acc: 0.9930 - val_loss: 0.0247 - val_acc: 0.9918

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 8/20 - 1s - loss: 0.0189 - acc: 0.9938 - val_loss: 0.0244 - val_acc: 0.9925

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 9/20 - 1s - loss: 0.0168 - acc: 0.9948 - val_loss: 0.0247 - val_acc: 0.9921

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:30:22 DEBUG    : Metrics(Test loss & Test Accuracy): [0.024283330682094675, 0.991599977016449]
2020-07-08 09:30:22 DEBUG    : Mutant ID: 1594174369341240832
2020-07-08 09:30:22 INFO     : Current scores - TrnFitness: 0.9947500228881836 TstFitness: 0.991599977016449
Scaled_TstFitness: 0.991599977016449
   val_loss  val_acc      loss       acc     lr
0  0.042687   0.9869  0.352634  0.900000  0.001
1  0.030024   0.9900  0.051251  0.983933  0.001
2  0.027310   0.9913  0.039094  0.987950  0.001
3  0.026117   0.9918  0.032985  0.989950  0.001
```

```
4  0.026283   0.9914  0.026760  0.991267  0.001
5  0.024283   0.9916  0.023010  0.992500  0.001
6  0.024709   0.9918  0.021424  0.992967  0.001
7  0.024427   0.9925  0.018947  0.993800  0.001
8  0.024724   0.9921  0.016838  0.994750  0.001
2020-07-08 09:30:24 DEBUG   : get_Shannon(Model)
2020-07-08 09:30:24 DEBUG   : Model Entropy: 8.793004035949707
2020-07-08 09:30:24 DEBUG   : Model Entropy normalized: 0.0014267408787846352
2020-07-08 09:30:24 DEBUG   : Layers Sum Entropy: 127.84410381317139
2020-07-08 09:30:24 DEBUG    : Layer Entropy: ['8.793004', '8.238329', '15.139215', '14.731662', '14.16304',
'14.155197', '11.686183', '11.694334', '16.77867', '12.46447']
2020-07-08 09:30:24 DEBUG       : Layer Entropy normalized: ['2.1982510089874268', '2.74610964457194',
'3.784803867340088', '0.025575801730155945', '3.540760040283203', '3.538799285888672', '0.18259660899639913',
'0.1827239692211151', '0.0035668939960246185', '0.016935421072918434']
2020-07-08 09:30:24 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_500_2',
'DV_625_750_2', 'DV_625_625_2', 'DV_812_500_5', 'DV_844_500_3', 'DV_750_250_5', 'DV_1000_500_21']
2020-07-08 09:30:24 DEBUG   : out get_Shannon(Model)
2020-07-08 09:30:24 INFO    : ###### End of step index : 16 of test 32 ##############################
2020-07-08 09:30:24 INFO    : ________________________________________________________________________
2020-07-08 09:30:24 INFO    : ###### Start Evolutionary Step Index: 17 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Dropout
NodeDict['From']: [(0.75, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.78125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.8125, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.828125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.84375, 0.5)] Node_Type: GaussianDropout
NodeDict['From']: [(0.875, 0.5)] Node_Type: AlphaDropout
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
 'Version': 1}
Model: "model_18"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816         DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)       2736        DV_250_500_1[0][0]
_________________________________________________________________________________________________
dropout_12 (Dropout)         (None, 7, 7, 48)          0           DV_375_500_1[0][0]
_________________________________________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216       dropout_12[0][0]
_________________________________________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0           DV_438_500_1[0][0]
_________________________________________________________________________________________________
dropout_13 (Dropout)         (None, 576)               0           DV_500_500_4[0][0]
_________________________________________________________________________________________________
DV_625_500_2 (Dense)         (None, 64)                36864       dropout_13[0][0]
_________________________________________________________________________________________________
DV_750_500_9 (Dropout)       (None, 64)                0           DV_625_500_2[0][0]
_________________________________________________________________________________________________
DV_781_500_2 (Activation)    (None, 64)                0           DV_750_500_9[0][0]
_________________________________________________________________________________________________
DV_812_500_5 (Dense)         (None, 64)                4160        DV_781_500_2[0][0]
_________________________________________________________________________________________________
```

```
DV_625_750_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
__________________________________________________________________________________________________
DV_625_625_2 (Conv2D)           (None, 7, 7, 32)     24576       DV_250_500_1[0][0]
__________________________________________________________________________________________________
DV_828_500_1 (Activation)       (None, 64)           0           DV_812_500_5[0][0]
__________________________________________________________________________________________________
flatten_32 (Flatten)            (None, 1568)         0           DV_625_750_2[0][0]
__________________________________________________________________________________________________
flatten_33 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
__________________________________________________________________________________________________
flatten_34 (Flatten)            (None, 1568)         0           DV_625_625_2[0][0]
__________________________________________________________________________________________________
DV_844_500_4 (Dense)            (None, 64)           4096        DV_828_500_1[0][0]
__________________________________________________________________________________________________
DV_750_250_5_C (Concatenate)    (None, 4704)         0           flatten_32[0][0]
                                                                 flatten_33[0][0]
                                                                 flatten_34[0][0]
__________________________________________________________________________________________________
DV_875_500_8 (GaussianDropout)  (None, 64)           0           DV_844_500_4[0][0]
__________________________________________________________________________________________________
DV_750_250_5 (Dense)            (None, 32)           150560      DV_750_250_5_C[0][0]
__________________________________________________________________________________________________
DV_938_500_2 (AlphaDropout)     (None, 64)           0           DV_875_500_8[0][0]
__________________________________________________________________________________________________
DV_1000_500_22_C (Concatenate)  (None, 736)          0           DV_750_250_5[0][0]
                                                                 dropout_13[0][0]
                                                                 DV_625_500_2[0][0]
                                                                 DV_938_500_2[0][0]
__________________________________________________________________________________________________
DV_1000_500_22 (Dense)          (None, 10)           7370        DV_1000_500_22_C[0][0]
==================================================================================================
Total params: 304,970
Trainable params: 304,970
Non-trainable params: 0
__________________________________________________________________________________________________
None
2020-07-08 09:30:26 INFO     : Original scores - TrnFitness: 0.9936000108718872 ValFitness: False TstFitness:
0.9936000108718872 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.3787 - acc: 0.8882 - val_loss: 0.0461 - val_acc: 0.9862
Epoch 2/20 - 1s - loss: 0.0605 - acc: 0.9816 - val_loss: 0.0341 - val_acc: 0.9894
Epoch 3/20 - 1s - loss: 0.0430 - acc: 0.9868 - val_loss: 0.0271 - val_acc: 0.9910
Epoch 4/20 - 1s - loss: 0.0343 - acc: 0.9892 - val_loss: 0.0253 - val_acc: 0.9916
Epoch 5/20 - 1s - loss: 0.0284 - acc: 0.9917 - val_loss: 0.0249 - val_acc: 0.9923
Epoch 6/20 - 1s - loss: 0.0258 - acc: 0.9920 - val_loss: 0.0245 - val_acc: 0.9917
Epoch 7/20 - 1s - loss: 0.0212 - acc: 0.9933 - val_loss: 0.0246 - val_acc: 0.9925

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 8/20 - 1s - loss: 0.0194 - acc: 0.9940 - val_loss: 0.0240 - val_acc: 0.9927
Epoch 9/20 - 1s - loss: 0.0181 - acc: 0.9942 - val_loss: 0.0240 - val_acc: 0.9935

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0161 - acc: 0.9948 - val_loss: 0.0232 - val_acc: 0.9931
Epoch 11/20 - 1s - loss: 0.0145 - acc: 0.9956 - val_loss: 0.0228 - val_acc: 0.9935
Epoch 12/20 - 1s - loss: 0.0130 - acc: 0.9958 - val_loss: 0.0238 - val_acc: 0.9926

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 13/20 - 1s - loss: 0.0132 - acc: 0.9959 - val_loss: 0.0233 - val_acc: 0.9922

Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 14/20 - 1s - loss: 0.0118 - acc: 0.9964 - val_loss: 0.0226 - val_acc: 0.9925
Epoch 15/20 - 1s - loss: 0.0108 - acc: 0.9968 - val_loss: 0.0231 - val_acc: 0.9921

Epoch 00015: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 16/20 - 1s - loss: 0.0103 - acc: 0.9968 - val_loss: 0.0236 - val_acc: 0.9923

Epoch 00016: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 17/20 - 1s - loss: 0.0101 - acc: 0.9970 - val_loss: 0.0223 - val_acc: 0.9930
Epoch 18/20 - 1s - loss: 0.0081 - acc: 0.9976 - val_loss: 0.0234 - val_acc: 0.9925

Epoch 00018: ReduceLROnPlateau reducing learning rate to 0.001.
```

```
Epoch 19/20 - 1s - loss: 0.0092 - acc: 0.9971 - val_loss: 0.0234 - val_acc: 0.9931

Epoch 00019: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 20/20 - 1s - loss: 0.0072 - acc: 0.9979 - val_loss: 0.0230 - val_acc: 0.9929

Epoch 00020: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:30:59 DEBUG   : Metrics(Test loss & Test Accuracy): [0.022286349659936967, 0.9929999709129333]
2020-07-08 09:30:59 DEBUG   : Mutant ID: 1594174369341240832
2020-07-08 09:30:59 INFO    : Current scores - TrnFitness: 0.9979000091552734 TstFitness: 0.9929999709129333
Scaled_TstFitness: 0.9929999709129333
    val_loss  val_acc      loss       acc     lr
0   0.046127   0.9862  0.378722  0.888183  0.001
1   0.034115   0.9894  0.060486  0.981633  0.001
2   0.027079   0.9910  0.042975  0.986750  0.001
3   0.025305   0.9916  0.034269  0.989200  0.001
4   0.024907   0.9923  0.028406  0.991667  0.001
5   0.024467   0.9917  0.025763  0.992050  0.001
6   0.024609   0.9925  0.021168  0.993283  0.001
7   0.023964   0.9927  0.019368  0.994000  0.001
8   0.024006   0.9935  0.018134  0.994167  0.001
9   0.023168   0.9931  0.016070  0.994850  0.001
10  0.022761   0.9935  0.014471  0.995567  0.001
11  0.023797   0.9926  0.012993  0.995817  0.001
12  0.023289   0.9922  0.013192  0.995900  0.001
13  0.022557   0.9925  0.011834  0.996417  0.001
14  0.023121   0.9921  0.010753  0.996767  0.001
15  0.023588   0.9923  0.010335  0.996817  0.001
16  0.022286   0.9930  0.010078  0.997050  0.001
17  0.023367   0.9925  0.008122  0.997583  0.001
18  0.023446   0.9931  0.009218  0.997083  0.001
19  0.023028   0.9929  0.007235  0.997900  0.001
2020-07-08 09:31:01 DEBUG   : get_Shannon(Model)
2020-07-08 09:31:01 DEBUG   : Model Entropy: 8.783628463745117
2020-07-08 09:31:01 DEBUG   : Model Entropy normalized: 0.001425219611186941
2020-07-08 09:31:01 DEBUG   : Layers Sum Entropy: 127.6383867263794
2020-07-08 09:31:01 DEBUG    : Layer Entropy: ['8.783628', '8.232107', '15.136115', '14.732756', '11.657583',
'14.119755', '14.112352', '11.707453', '16.723648', '12.432989']
2020-07-08 09:31:01 DEBUG      : Layer Entropy normalized: ['2.1959071159362793', '2.7440357208251953',
'3.7840287685394287', '0.025577700800365873', '0.182149738073349', '3.5299386978149414', '3.528088092803955',
'0.18292894959449768', '0.0035551972940665523', '0.016892648261526356']
2020-07-08 09:31:01 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_500_2',
'DV_812_500_5', 'DV_625_750_2', 'DV_625_625_2', 'DV_844_500_4', 'DV_750_250_5', 'DV_1000_500_22']
2020-07-08 09:31:01 DEBUG   : out get_Shannon(Model)
2020-07-08 09:31:01 INFO    : ###### End of step index : 17 of test 32 ##############################
2020-07-08 09:31:01 INFO    : ________________________________________________________________________
2020-07-08 09:31:01 INFO    : ###### Start Evolutionary Step Index: 18 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Dropout
NodeDict['From']: [(0.75, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.78125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.8125, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.828125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.84375, 0.5)] Node_Type: GaussianDropout
NodeDict['From']: [(0.875, 0.5)] Node_Type: AlphaDropout
NodeDict['From']: [(0.75, 0.5)] Node_Type: Dense
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
```

```
 'Version': 1}
Model: "model_19"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816         DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)       2736        DV_250_500_1[0][0]
_________________________________________________________________________________________________
dropout_14 (Dropout)         (None, 7, 7, 48)          0           DV_375_500_1[0][0]
_________________________________________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216       dropout_14[0][0]
_________________________________________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0           DV_438_500_1[0][0]
_________________________________________________________________________________________________
dropout_15 (Dropout)         (None, 576)               0           DV_500_500_4[0][0]
_________________________________________________________________________________________________
DV_625_500_2 (Dense)         (None, 64)                36864       dropout_15[0][0]
_________________________________________________________________________________________________
DV_750_500_10 (Dropout)      (None, 64)                0           DV_625_500_2[0][0]
_________________________________________________________________________________________________
DV_781_500_3 (Activation)    (None, 64)                0           DV_750_500_10[0][0]
_________________________________________________________________________________________________
DV_812_500_6 (Dense)         (None, 64)                4160        DV_781_500_3[0][0]
_________________________________________________________________________________________________
DV_625_750_2 (Conv2D)        (None, 7, 7, 32)          24576       DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_625_625_2 (Conv2D)        (None, 7, 7, 32)          24576       DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_828_500_2 (Activation)    (None, 64)                0           DV_812_500_6[0][0]
_________________________________________________________________________________________________
flatten_35 (Flatten)         (None, 1568)              0           DV_625_750_2[0][0]
_________________________________________________________________________________________________
flatten_36 (Flatten)         (None, 1568)              0           DV_625_625_2[0][0]
_________________________________________________________________________________________________
flatten_37 (Flatten)         (None, 1568)              0           DV_625_625_2[0][0]
_________________________________________________________________________________________________
DV_844_500_7 (Dense)         (None, 84)                5376        DV_828_500_2[0][0]
_________________________________________________________________________________________________
DV_750_250_7_C (Concatenate) (None, 4704)              0           flatten_35[0][0]
                                                                   flatten_36[0][0]
                                                                   flatten_37[0][0]
_________________________________________________________________________________________________
DV_875_500_10 (GaussianDropout) (None, 84)             0           DV_844_500_7[0][0]
_________________________________________________________________________________________________
DV_750_250_7 (Dense)         (None, 52)                244660      DV_750_250_7_C[0][0]
_________________________________________________________________________________________________
DV_938_500_4 (AlphaDropout)  (None, 84)                0           DV_875_500_10[0][0]
_________________________________________________________________________________________________
DV_938_750_1 (Dense)         (None, 64)                4096        DV_750_500_10[0][0]
_________________________________________________________________________________________________
DV_1000_500_25_C (Concatenate)  (None, 840)            0           DV_750_250_7[0][0]
                                                                   dropout_15[0][0]
                                                                   DV_625_500_2[0][0]
                                                                   DV_938_500_4[0][0]
                                                                   DV_938_750_1[0][0]
_________________________________________________________________________________________________
DV_1000_500_25 (Dense)       (None, 10)                8410        DV_1000_500_25_C[0][0]
=================================================================================================
Total params: 405,486
Trainable params: 405,486
Non-trainable params: 0
_________________________________________________________________________________________________
None
2020-07-08 09:31:03 INFO     : Original scores - TrnFitness: 0.9932000041007996 ValFitness: False TstFitness:
0.9940999746322632 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.2786 - acc: 0.9205 - val_loss: 0.0386 - val_acc: 0.9879
Epoch 2/20 - 1s - loss: 0.0409 - acc: 0.9873 - val_loss: 0.0283 - val_acc: 0.9908
```

```
Epoch 3/20 - 1s - loss: 0.0296 - acc: 0.9911 - val_loss: 0.0237 - val_acc: 0.9928
Epoch 4/20 - 1s - loss: 0.0223 - acc: 0.9930 - val_loss: 0.0227 - val_acc: 0.9935
Epoch 5/20 - 1s - loss: 0.0194 - acc: 0.9940 - val_loss: 0.0235 - val_acc: 0.9928

Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 6/20 - 1s - loss: 0.0170 - acc: 0.9949 - val_loss: 0.0223 - val_acc: 0.9927
Epoch 7/20 - 1s - loss: 0.0143 - acc: 0.9954 - val_loss: 0.0217 - val_acc: 0.9935
Epoch 8/20 - 1s - loss: 0.0127 - acc: 0.9961 - val_loss: 0.0228 - val_acc: 0.9927

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 9/20 - 1s - loss: 0.0129 - acc: 0.9959 - val_loss: 0.0223 - val_acc: 0.9934

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0099 - acc: 0.9971 - val_loss: 0.0220 - val_acc: 0.9935

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:31:23 DEBUG   : Metrics(Test loss & Test Accuracy): [0.021735972817311995, 0.9934999942779541]
2020-07-08 09:31:23 DEBUG   : Mutant ID: 1594174369341240832
2020-07-08 09:31:23 INFO    : Current scores - TrnFitness: 0.9970666766166687 TstFitness: 0.9934999942779541
Scaled_TstFitness: 0.9934999942779541
   val_loss  val_acc      loss       acc     lr
0  0.038622   0.9879  0.278572  0.920483  0.001
1  0.028302   0.9908  0.040911  0.987333  0.001
2  0.023730   0.9928  0.029634  0.991050  0.001
3  0.022738   0.9935  0.022293  0.992967  0.001
4  0.023473   0.9928  0.019388  0.994017  0.001
5  0.022286   0.9927  0.017044  0.994867  0.001
6  0.021736   0.9935  0.014343  0.995367  0.001
7  0.022818   0.9927  0.012741  0.996050  0.001
8  0.022330   0.9934  0.012906  0.995883  0.001
9  0.021981   0.9935  0.009880  0.997067  0.001
2020-07-08 09:31:25 DEBUG   : get_Shannon(Model)
2020-07-08 09:31:25 DEBUG   : Model Entropy: 8.782720565795898
2020-07-08 09:31:25 DEBUG   : Model Entropy normalized: 0.0013872564469745536
2020-07-08 09:31:25 DEBUG   : Layers Sum Entropy: 140.8023567199707
2020-07-08 09:31:25 DEBUG    : Layer Entropy: ['8.782721', '8.232946', '15.135464', '14.727745', '11.707179',
'14.107707', '14.097927', '12.107032', '17.550873', '11.704294', '12.648469']
2020-07-08 09:31:25 DEBUG       : Layer Entropy normalized: ['2.1956801414489746', '2.7443154652913413',
'3.7838659286499023', '0.02556900183359782', '0.18292467296123505', '3.5269267559051514', '3.524481773376465',
'0.18917237222194672', '0.00373105289173775', '0.18287959694862366', '0.015057701156252906']
2020-07-08 09:31:25 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_500_2',
'DV_812_500_6', 'DV_625_750_2', 'DV_625_625_2', 'DV_844_500_7', 'DV_750_250_7', 'DV_938_750_1', 'DV_1000_500_25']
2020-07-08 09:31:25 DEBUG   : out get_Shannon(Model)
2020-07-08 09:31:25 INFO    : ###### End of step index : 18 of test 32 ##############################
2020-07-08 09:31:25 INFO    : ________________________________________________________________________
2020-07-08 09:31:25 INFO    : ###### Start Evolutionary Step Index: 19 Test : 32/40 ##############################
NodeDict['From']: [(0.0, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: SeparableConv2D
NodeDict['From']: [(0.375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Flatten
NodeDict['From']: [(0.5, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.4375, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.25, 0.5)] Node_Type: Conv2D
NodeDict['From']: [(0.625, 0.5)] Node_Type: Dropout
NodeDict['From']: [(0.75, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.78125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.8125, 0.5)] Node_Type: Activation
NodeDict['From']: [(0.828125, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.84375, 0.5)] Node_Type: Dense
NodeDict['From']: [(0.875, 0.5)] Node_Type: AlphaDropout
NodeDict['From']: [(0.75, 0.5)] Node_Type: Dense
{'Age': 0,
 'Coordinate': (2, 0.5),
 'FromCoordinate': (1.0, 0.5),
 'Id': '',
 'ModelName': '',
 'ModelPath': '',
 'Parameters': [],
 'Signal_Shape': [10],
 'Type': 'Signal_Channel',
```

```
 'Version': 1}
Model: "model_20"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
DV_0_500_1 (InputLayer)      (None, 28, 28, 1)         0
_________________________________________________________________________________________________
DV_250_500_1 (Conv2D)        (None, 14, 14, 48)        816         DV_0_500_1[0][0]
_________________________________________________________________________________________________
DV_375_500_1 (SeparableConv2D)  (None, 7, 7, 48)       2736        DV_250_500_1[0][0]
_________________________________________________________________________________________________
dropout_16 (Dropout)         (None, 7, 7, 48)          0           DV_375_500_1[0][0]
_________________________________________________________________________________________________
DV_438_500_1 (Conv2D)        (None, 3, 3, 64)          49216       dropout_16[0][0]
_________________________________________________________________________________________________
DV_500_500_4 (Flatten)       (None, 576)               0           DV_438_500_1[0][0]
_________________________________________________________________________________________________
dropout_17 (Dropout)         (None, 576)               0           DV_500_500_4[0][0]
_________________________________________________________________________________________________
DV_625_500_2 (Dense)         (None, 64)                36864       dropout_17[0][0]
_________________________________________________________________________________________________
DV_750_500_10 (Dropout)      (None, 64)                0           DV_625_500_2[0][0]
_________________________________________________________________________________________________
DV_781_500_3 (Activation)    (None, 64)                0           DV_750_500_10[0][0]
_________________________________________________________________________________________________
DV_812_500_6 (Dense)         (None, 64)                4160        DV_781_500_3[0][0]
_________________________________________________________________________________________________
DV_625_750_2 (Conv2D)        (None, 7, 7, 32)          24576       DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_625_625_2 (Conv2D)        (None, 7, 7, 32)          24576       DV_250_500_1[0][0]
_________________________________________________________________________________________________
DV_828_500_2 (Activation)    (None, 64)                0           DV_812_500_6[0][0]
_________________________________________________________________________________________________
flatten_38 (Flatten)         (None, 1568)              0           DV_625_750_2[0][0]
_________________________________________________________________________________________________
flatten_39 (Flatten)         (None, 1568)              0           DV_625_625_2[0][0]
_________________________________________________________________________________________________
flatten_40 (Flatten)         (None, 1568)              0           DV_625_625_2[0][0]
_________________________________________________________________________________________________
DV_844_500_7 (Dense)         (None, 84)                5376        DV_828_500_2[0][0]
_________________________________________________________________________________________________
DV_750_250_7_C (Concatenate) (None, 4704)              0           flatten_38[0][0]
                                                                   flatten_39[0][0]
                                                                   flatten_40[0][0]
_________________________________________________________________________________________________
DV_875_500_11 (Dense)        (None, 48)                4032        DV_844_500_7[0][0]
_________________________________________________________________________________________________
DV_750_250_7 (Dense)         (None, 52)                244660      DV_750_250_7_C[0][0]
_________________________________________________________________________________________________
DV_938_500_5 (AlphaDropout)  (None, 48)                0           DV_875_500_11[0][0]
_________________________________________________________________________________________________
DV_938_750_2 (Dense)         (None, 64)                4096        DV_750_500_10[0][0]
_________________________________________________________________________________________________
DV_1000_500_26_C (Concatenate)  (None, 804)            0           DV_750_250_7[0][0]
                                                                   dropout_17[0][0]
                                                                   DV_625_500_2[0][0]
                                                                   DV_938_500_5[0][0]
                                                                   DV_938_750_2[0][0]
_________________________________________________________________________________________________
DV_1000_500_26 (Dense)       (None, 10)                8050        DV_1000_500_26_C[0][0]
=================================================================================================
Total params: 409,158
Trainable params: 409,158
Non-trainable params: 0
_________________________________________________________________________________________________
None
2020-07-08 09:31:27 INFO     : Original scores - TrnFitness: 0.9940000176429749 ValFitness: False TstFitness:
0.9944000244140625 Scaled_TstFitness: 0.9944000244140625
Train on 60000 samples, validate on 10000 samples
Epoch 1/20 - 2s - loss: 0.2959 - acc: 0.9147 - val_loss: 0.0347 - val_acc: 0.9889
Epoch 2/20 - 1s - loss: 0.0404 - acc: 0.9876 - val_loss: 0.0270 - val_acc: 0.9916
```

```
Epoch 3/20 - 1s - loss: 0.0294 - acc: 0.9909 - val_loss: 0.0239 - val_acc: 0.9927
Epoch 4/20 - 1s - loss: 0.0222 - acc: 0.9931 - val_loss: 0.0230 - val_acc: 0.9929
Epoch 5/20 - 1s - loss: 0.0194 - acc: 0.9940 - val_loss: 0.0218 - val_acc: 0.9936
Epoch 6/20 - 1s - loss: 0.0159 - acc: 0.9950 - val_loss: 0.0215 - val_acc: 0.9931
Epoch 7/20 - 1s - loss: 0.0152 - acc: 0.9953 - val_loss: 0.0212 - val_acc: 0.9932
Epoch 8/20 - 1s - loss: 0.0133 - acc: 0.9958 - val_loss: 0.0207 - val_acc: 0.9940
Epoch 9/20 - 1s - loss: 0.0119 - acc: 0.9964 - val_loss: 0.0237 - val_acc: 0.9926

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 10/20 - 1s - loss: 0.0115 - acc: 0.9963 - val_loss: 0.0228 - val_acc: 0.9926

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.001.
Epoch 11/20 - 1s - loss: 0.0114 - acc: 0.9962 - val_loss: 0.0224 - val_acc: 0.9930

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.001.
2020-07-08 09:31:50 DEBUG   : Metrics(Test loss & Test Accuracy): [0.020655150778993266, 0.9940000176429749]
2020-07-08 09:31:50 DEBUG   : Mutant ID: 1594174369341240832
2020-07-08 09:31:50 INFO    : Current scores - TrnFitness: 0.9962000250816345 TstFitness: 0.9940000176429749
Scaled_TstFitness: 0.9940000176429749
    val_loss  val_acc      loss       acc     lr
0   0.034732   0.9889  0.295854  0.914683  0.001
1   0.026992   0.9916  0.040369  0.987650  0.001
2   0.023855   0.9927  0.029405  0.990883  0.001
3   0.022983   0.9929  0.022197  0.993117  0.001
4   0.021803   0.9936  0.019390  0.994000  0.001
5   0.021470   0.9931  0.015890  0.995017  0.001
6   0.021186   0.9932  0.015208  0.995333  0.001
7   0.020655   0.9940  0.013341  0.995833  0.001
8   0.023716   0.9926  0.011889  0.996433  0.001
9   0.022828   0.9926  0.011467  0.996267  0.001
10  0.022439   0.9930  0.011358  0.996200  0.001
2020-07-08 09:31:52 DEBUG   : get_Shannon(Model)
2020-07-08 09:31:52 DEBUG   : Model Entropy: 8.777618408203125
2020-07-08 09:31:52 DEBUG   : Model Entropy normalized: 0.0013760179351313881
2020-07-08 09:31:52 DEBUG   : Layers Sum Entropy: 152.30174922943115
2020-07-08 09:31:52 DEBUG    : Layer Entropy: ['8.777618', '8.231894', '15.134873', '14.723663', '11.680616',
'14.091345', '14.079248', '12.099478', '11.682067', '17.514065', '11.7081375', '12.578744']
2020-07-08 09:31:52 DEBUG       : Layer Entropy normalized: ['2.1944046020507812', '2.743964513142904',
'3.7837183475494385', '0.0255619155503607854', '0.1825096309185028', '3.522836208343506', '3.5198121070861816',
'0.18905434012413025',        '0.13907222520737422',        '0.0037232280588474402',        '0.18293964862823486',
'0.01564520389879521']
2020-07-08 09:31:52 DEBUG    : Layer list: ['DV_250_500_1', 'DV_375_500_1', 'DV_438_500_1', 'DV_625_500_2',
'DV_812_500_6', 'DV_625_750_2', 'DV_625_625_2', 'DV_844_500_7', 'DV_875_500_11', 'DV_750_250_7', 'DV_938_750_2',
'DV_1000_500_26']
2020-07-08 09:31:52 DEBUG   : out get_Shannon(Model)
2020-07-08 09:31:52 INFO    : ###### End of step index : 19 of test 32 ##############################
```