Amélioration de la recherche d'architecture neuronale en combinant un algorithme FireFly avec une évaluation sans apprentissage

N. Mokhtari, A. Nédélec, M. Gilles, P. De Loor

Lab-STICC (CNRS UMR 6285) - ENIB
Centre Européen de Réalité Virtuelle
Brest, France

{nassim.mokhtari, alexis.nedelec, marlene.gilles, pierre.deloor}@enib.fr

Résumé

Les algorithmes de recherche d'architecture neuronale (Neural Architecture Search - NAS) sont utilisés pour automatiser la conception de réseaux de neurones profonds. Trouver la meilleure architecture pour un jeu de données peut prendre beaucoup de temps car ces algorithmes doivent explorer un grand nombre de réseaux et les évaluer pour choisir le plus approprié. Dans ce travail, nous proposons une nouvelle métrique qui utilise le score Intra-Cluster Distance (ICD) pour évaluer la capacité d'un modèle non entraîné à distinguer les données sans avoir à les entraîner.Pour la recherche de l'architecture, nous utilisons une méta-heuristique de type FireFly améliorée, plus robuste face au problème des optimums locaux que l'algorithme FireFly de base, comme technique de recherche pour trouver le meilleur modèle de réseau de neurones pour un ensemble spécifique de données. Les résultats expérimentaux sur les différents NAS Benchmarks montrent que notre métrique est valable pour l'évaluation des réseaux de neurones convolutifs ainsi que des réseaux de neurones récurrents, et que l'algorithme FireFly que nous proposons peut améliorer les résultats obtenus par les méthodes sans entraînement les plus récentes.

Mots-clés

Apprentissage profond, recherche d'architecture neuronale, score sans entraînement, distance intra-groupe, apprentissage automatique, méta-heuristiques, algorithme FireFly.

Abstract

Neural Architecture Search (NAS) algorithms are used to automate the design of deep neural networks. Finding the best architecture for a given dataset can be time consuming since these algorithms have to explore a large number of networks, and score them according to their performances to choose the most appropriate one. In this work, we propose a novel metric that uses the Intra-Cluster Distance (ICD) score to evaluate the ability of an untrained model to distinguish between data in order to approximate its quality. We also use an improved version of the FireFly algorithm, more robust to the local optimums problem than the baseline FireFly algorithm, as a search technique to find the best neural network model adapted to a specific dataset. Experimental results on the different NAS Benchmarks show that our metric is valid for either scoring CNNs and RNNs, and that our proposed FireFly algorithm can improve the result obtained by the state-of-art training-free methods.

Keywords

Deep Learning, Neural Architecture Search, Training-free Score, Intra Cluster Distance, Machine Learning, Metaheuristics, FireFly algorithm

1 Introduction

Ce document est la traduction de notre travail publié dans *International Joint Conference on Neural Networks (IJCNN) 2022* [1].

Les avancées récentes dans le domaine de la classification d'images et de la reconnaissance vocale liées à la recherche sur l'apprentissage profond, en particulier sur les réseaux de neurones convolutifs, ont montré l'utilité de ces derniers pour l'extraction de caractéristiques et la classification, et semblent les mieux adaptées à de nombreux problèmes de classification [2–5].

En raison de la croissance du nombre d'hyperparamètres (couches cachées, unités cachées, etc.) utilisés pour définir les architectures d'apprentissage profond et, par conséquent, la hausse du nombre d'organisations de réseau possibles (augmentation exponentielle) impliquées par ces hyperparamètres, un nouveau défi consiste à trouver des solutions pour concevoir l'architecture elle-même à l'aide d'algorithmes plutôt que manuellement. Pour ce faire, la communauté de l'apprentissage profond a introduit la recherche d'architecture neuronale *(Neural Architecture Search - NAS)*, des algorithmes capables d'automatiser la découverte d'architectures efficaces [8–13].

Afin de comparer l'efficacité des NAS, la communauté du NAS a conçu des *benchmarks* spécifiques destinés à cet effet : [14–17]. En effet, le choix de l'architecture d'un réseau de neurones peut être vu comme un problème combi-

natoire, où l'objectif est de trouver la combinaison d'hyperparamètres (nombre de couches, taille des couches, etc.) qui offre les meilleures performances. Les *benchmarks* fournissent un espace de recherche fini d'architectures, qui peut être utilisé pour comparer les algorithmes de NAS. Les méta-heuristiques sont souvent utilisées pour résoudre ce type de problèmes, et il existe plusieurs travaux qui exploitent ces méthodes [18–22]. L'utilisation d'une méta-heuristique implique l'évaluation des solutions (architectures de réseaux de neurones) afin d'évaluer leurs qualités.
L'un des principaux problèmes liés à l'évaluation des performances d'un modèle est la phase d'apprentissage qui prend beaucoup de temps, entraînant un temps de recherche énorme qui peut prendre des jours, même en utilisant des centaines de GPU [23]. Il existe donc des méthodes permettant de contourner cette phase d'apprentissage et d'évaluer une architecture à partir de métriques basées sur la distribution de l'activation des neurones par rapport à différentes valeurs d'entrée rassemblées dans un *mini-batch* de données, comme la métrique de Mellor [23] et la proposition de Lopes et al. [24].
Notre contribution dans ce travail peut être résumée comme suit :

1. Une nouvelle métrique ***training-free*** qui peut approcher la qualité d'un réseau de neurones en évaluant sa capacité à distinguer les données. Cette métrique peut être utilisée pour évaluer plus de types de réseaux de neurones que les métriques précédentes.
2. Une version améliorée de l'algorithme *FireFly (Improved Firefly Algorithm - IFA)* qui utilise des opérateurs d'algorithmes génétiques, permettant à notre IFA d'être plus robuste aux optimums locaux que la version standard.
3. Une combinaison de la métrique proposée et de l'algorithme FireFly amélioré afin de trouver le modèle le plus intéressant dans un espace de recherche d'architecture de réseaux de neurones.
4. Une évaluation de notre proposition qui montre qu'elle surpasse l'état de l'art en terme de performance pour trouver l'architecture la plus adaptée à un problème de *machine learning*.

Le reste du document s'organise comme suit : la Section 2 introduit une synthèse des différents travaux réalisés dans le cadre de l'évaluation des réseaux de neurones pour les NAS. La Section 3 présente l'algorithme *FireFly*. Dans la Section 4, nous présentons notre méthode pour évaluer un réseau de neurones non entraîné et une proposition d'amélioration de l'algorithme *FireFly*. En Section 5 nous présentons les résultats expérimentaux obtenus sur 4 *benchmarks NAS*, montrant l'efficacité de notre proposition (valide et améliore les scores de l'état de l'art). Enfin, nous résumerons dans la Section 6 les résultats de ce travail ainsi que les améliorations possibles.

2 Travaux connexes

L'espace de recherche (ensemble de tous les réseaux possibles) étant très grand, l'évaluation de l'efficacité d'un algorithme NAS ne peut être faite de manière exhaustive. Ceci a conduit à la création de plusieurs benchmarks [14–17] qui consistent en des espaces de recherche NAS et des méta-données relatives à l'entraînement de ces réseaux dans cet espace de recherche [23].
Le NAS-Bench-101 est composé de 423 624 réseaux de neurones (CNN) qui ont été entraînés de manière exhaustive, avec trois initialisations différentes, sur le jeu de données CIFAR-10 pour 4, 12, 36 et 108 itérations (423K * 3 * 4 = 5M modèles entraînés au total) [14]. Le NAS-Bench-201 comprend 15 625 réseaux entraînés plusieurs fois sur CIFAR-10, CIFAR-100 et ImageNet-16-120 [15]. Le NAS-BENCH-NLP est composé de 14K réseaux de neurones (RNN) entraînés sur le jeu de données *Penn Tree Bank* et le jeu de données *WikiText-2* [17].
Les algorithmes NAS explorent l'espace de recherche afin de trouver le meilleur réseau. Cependant, l'entraînement de chaque architecture de réseau de neurones pour sélectionner la plus appropriée est un processus qui prend du temps. Par conséquent, la possibilité d'évaluer la qualité d'une architecture sans l'entraîner est une alternative pour trouver le réseau de neurones le plus approprié sans passer des jours à faire des calculs.
Mellor et al. [23] ont proposé une façon d'évaluer un réseau de neurones sans entraînement préalable (les poids du réseaux sont définis aléatoirement) , en identifiant un indicateur binaire qui se concentre uniquement sur les unités linéaires rectifiées (ReLU) du réseau (0 pour une unité inactive, et 1 pour une unité active). L'intuition derrière leur approche est que plus les codes binaires associés à deux entrées sont similaires, plus il est difficile pour le réseau d'apprendre à discriminer ces entrées. Un mini-batch de données X = [x_1, x_2, ..., x_n] est envoyé à travers un réseau de neurones (composé de ReLU et de plusieurs autres types d'unités) afin d'obtenir des codes binaires C = [c_1, c_2, ...,c_n], où chaque c_i (obtenu uniquement à partir des sorties ReLU) fait référence au code binaire de x_i. Pour évaluer le réseau de neurones, Mellor et al. [23] calculent le logarithme du déterminant d'une matrice K_h, où chaque composante est calculée en utilisant la distance de Hamming (Eq. (1)). Plus le score est élevé, meilleur est le réseau

$$K_h[i,j] = N_A - Hamming\_distance(c_i, c_j) \quad (1)$$

où N_A est le nombre d'unités ReLU.
Lopes et al. [24] ont proposé une autre façon d'évaluer un réseau de neurones sans entraînement. En se basant sur les travaux de Mellor et al. [23], ils ont proposé d'utiliser les codes binaires pour calculer une matrice jacobienne (J). L'objectif est de déterminer si un réseau non entraîné peut distinguer les opérateurs linéaires locaux pour chaque point de données, mais aussi obtenir des résultats similaires pour des points de données similaires (appartenant à la même classe dans une approche supervisée). Pour estimer ce comportement, ils évaluent la corrélation des valeurs de J par rapport à leur classe en calculant une matrice de covariance pour chaque classe présente dans J. Les corrélations sont d'abord évaluées individuellement, car elles peuvent avoir

des tailles différentes en raison du nombre de données par classe. Le score final est la somme des scores des matrices de corrélation individuelles.

L'exploration de toutes les architecture du benchmark NAS afin de trouver le meilleur modèle peut être une tâche difficile et longue, même si nous utilisons ce type de métrique pour éviter l'entraînement du réseau, en raison des millions de modèles inclus dans cet espace de recherche. Pour éviter cela, il existe plusieurs travaux qui exploitent des méta-heuristiques pour trouver le réseau le plus approprié à une tâche donnée afin d'éviter une recherche exhaustive. Par exemple, Sun et al. [20] ont utilisé un algorithme génétique pour concevoir automatiquement un réseau de neurones convolutif, en utilisant la précision du réseau entraîné comme fonction objectif de leur algorithme génétique.

Rere et al. [22] ont proposé plusieurs méta-heuristiques (algorithmes SA, DE et HS). Leur fonction objectif était l'erreur standard sur l'ensemble d'entraînement. Carvalho et al. [21] ont proposé une fonction objectif basée sur l'erreur d'entraînement et l'erreur de test, qui a été utilisée avec les algorithmes VNS, SA, GEO et GA. Ayumi et al. [19] ont utilisé un algorithme de recuit microcanonique *(Microcanonical Annealing Algorithm - MAA)* pour concevoir un réseau de neurones, en utilisant l'erreur durant la phase d'apprentissage comme fonction objectif. Strumberger et al. [18] ont préféré utiliser un algorithme FireFly pour concevoir leur CNN, où la fonction objectif utilisée était basée sur l'erreur obtenue sur l'ensemble de test. Mellor et al. [23] ont proposé d'utiliser un algorithme d'évolution régularisée assistée *(Assisted Regularised Evolution Algorithm - AREA)*, une version améliorée de REA proposée par Real et al. [13], combiné à la métrique qu'ils ont proposée pour trouver la meilleure architecture de réseau de neurones.

Dans ce travail, nous nous intéressons à l'utilisation d'une version améliorée de l'algorithme FireFly, combinée avec une métrique d'évaluation de modèle sans entraînement utilisée comme fonction objectif. Notons que cette métrique sera exploitable avec n'importe quel type de cellules.

Notre choix d'utiliser l'algorithme FireFly vient du fait que cette méthode inclut plusieurs méta-heuristiques telles que le recuit simulé *(Simulated Annealing - SA)*, ou l'optimisation par essaims de particules *(Particle Swarm Optimization - PSO)*, en plus de sa convergence rapide vers un optimum [25].

3 Présentation de l'algorithme FireFly (FA)

Inspiré par le comportement des lucioles, cet algorithme a été initialement développé par Xin-She Yang en 2008 [25]. Il est basé sur trois règles :

1. Les lucioles sont unisexes, donc une luciole est attirée par une autre sans tenir compte de son genre.
2. L'attirance est proportionnelle à la luminosité et toutes deux sont inversement proportionnelles à la distance (l'attirance et la luminosité diminuent lorsque la distance augmente). Pour toute paire de lucioles, la moins brillante sera attirée par la plus brillante, et se dirigera donc vers elle. S'il n'y a pas de luciole plus brillante, cette dernière se déplacera au hasard.
3. La luminosité d'une luciole (solution) est donnée par la fonction objectif.

Le mouvement d'une luciole i vers une luciole j plus lumineuse est défini par Eq. (2).

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2}(x_j^t - x_i^t) + \alpha \tau_i^t \qquad (2)$$

— r : distance entre deux lucioles.
— α : poids du déplacement aléatoire.
— β_0 : attirance à r = 0.
— γ : coefficient d'absorption (paramètre à renseigner).
— x_i^t : position de la luciole i à l'instant t.
— τ_i^t : vecteur de nombres aléatoires donnés par une distribution gaussienne à l'instant t pour la luciole i.

Le fonctionnement de l'algorithme FireFly est présenté dans l'algorithme 1.

Algorithm 1 FireFly Algorithm

Générer aléatoirement une population de n lucioles x_i=(i=1,2,...,n).
L'intensité lumineuse I_i de chaque luciole x_i est donnée par f(x_i) où f est la fonction objectif
Définir les coefficients α, β et γ.
while critères d'arrêt non atteint **do**
  for i=1 to n **do**
    for j=1 to n **do**
      if I_i< I_j **then**
        Déplacer la luciole i vers la luciole j
      end if
      Faire varier l'attractivité en fonction de la distance r.
      Évaluer la nouvelle solution et mettre à jour l'intensité lumineuse.
    end for
  end for
  Renvoyer la meilleure solution
end while

4 Méthode proposée

Dans cette partie, nous présentons notre approche basée sur :

1. L'évaluation de la qualité d'un réseau de neurones avec une métrique sans entraînement.
2. L'utilisation d'un algorithme FireFly amélioré pour guider le choix parmi l'énorme quantité d'architectures possibles.

4.1 Évaluation de la qualité d'un modèle

Nous proposons une nouvelle façon d'évaluer un réseau de neurones sans l'entraîner, en faisant passer un mini batch de données par le réseau à évaluer afin d'obtenir le code binaire de chaque donnée d'entrée (0 pour une valeur négative et 1 pour une valeur positive). Nous suivons la méthodologie de Mellor et al. [23] sauf que nous utilisons toutes les unités du réseau, pas seulement celles ayant une fonction ReLU.

Notre intuition est que les unités qui utilisent une fonction d'activation autre que ReLU (ou n'en utilisant pas) peuvent aussi être utilisées pour formaliser la manière dont le modèle interprète les données (codes binaires). Puisque nous cherchons à évaluer la capacité d'un modèle à distinguer des données, nous pouvons éviter de vérifier les similarités (en calculant la co-variance comme le fait Lopes el al. [24] par exemple) et nous concentrer davantage sur le degré de différence entre les représentations. Pour ce faire, nous proposons d'utiliser la distance intra-cluster (ICD), calculée sur les codes binaires, comme métrique pour évaluer le réseau non entraîné. La figure 1 illustre notre processus d'évaluation du réseau, pour un mini-batch de données contenant 4 échantillons.

L'ICD est généralement utilisée pour évaluer la qualité d'une méthode de *clustering* où l'objectif est de trouver un *clustering* fournissant des groupes qui ont une petite distance intra-cluster. L'idée est de calculer la distance moyenne entre chaque point de données et le centre du cluster (moyenne des données).

Dans notre travail, nous évaluons le réseau selon l'ICD calculé en utilisant Eq. (3).

$$ICD = \frac{\sum_{n=1}^{N} d(\bar{c}, c_i)}{N} \tag{3}$$

Où d est la distance euclidienne, c_i est le code binaire de l'entrée x_i, $\bar{c}$ est le centre des codes binaires (moyenne des codes binaires) et N est le nombre total de codes binaires (nombre d'échantillons dans le mini-lot de données).

Une petite valeur ICD signifie que le cluster est compact (composé d'échantillons similaires), alors qu'une grande valeur signifie que le cluster est étiré (les échantillons sont différents). Puisque nous recherchons un réseau capable de fournir différents codes binaires pour différents échantillons, le cluster composé de ces codes binaires doit être le plus étiré possible, ce qui donne une valeur ICD élevée. Plus le score est élevé, meilleur est le réseau.

4.2 Improved FireFly Algorithm

Grâce au mécanisme d'attraction, l'algorithme FireFly atteint rapidement un optimum, ce qui augmente les chances de rester bloqué dans un optimum local, même si l'algorithme FireFly inclut une diversification (la marche aléatoire). Afin d'améliorer la diversification, nous proposons d'utiliser les opérateurs de l'algorithme génétique (sélection, croisement et mutation) qui pourraient permettre une meilleure exploration de l'espace de recherche en combinant les composants des solutions déjà explorées. Plus de détails sur l'implémentation de ces opérateurs de métaheuristiques peuvent être trouvés dans la Section 4.C.

Nous proposons d'exécuter une itération de l'algorithme génétique chaque fois que l'algorithme FireFly est bloqué dans un optimum local, afin de créer une nouvelle population, complètement différente de l'ancienne. Nous considérons que l'algorithme FireFly est bloqué dans un optimum local, s'il n'arrive plus à améliorer la meilleure solution après un certain nombre d'itérations (chances). Avant chaque exécution de l'algorithme génétique, l'optimum actuel est stocké dans une liste (*candidats*), le résultat final de l'exploration (lorsque le critère d'arrêt est atteint) est la meilleure solution de la liste des candidats. L'algorithme 2 illustre l'approche proposée.

L'utilisation de ce mécanisme donne la chance à notre algorithme proposé d'effectuer une diversification "mineure" (marche aléatoire de FireFly) qui lui permet de mieux explorer une région de l'espace de recherche, avant d'en explorer une autre obtenue par une diversification "majeure" grâce à l'algorithme génétique.

La sélection de la meilleure solution dans la liste des *candidats* est effectuée en fonction des performances des modèles après l'entraînement. Comme notre métrique ne peut qu'approximer la qualité du modèle et ne peut la mesurer exactement, le fait de conserver une liste de candidats augmente les chances de trouver une bonne architecture.

Le critère d'arrêt a été fixé en fonction du nombre de populations générées.

Algorithm 2 Improved FireFly Algorithm

```
Générer aléatoirement la population de solution
Définir MaxChances
chances = MaxChances
candidats = [ ]
LocalBest = NULL
while critères d'arrêt non atteint do
    Exécution d'une itération de FireFly
    Best_t = la meilleure solution pour la population actuelle
    if LocalBest = NULL then
        LocalBest = Best_t
    else
        if fitness(Best_t) ≥ fitness(Best_{t-1}) then
            LocalBest = Best_t
        else
            chances - -
        end if
    end if
    if chances = 0 then
        ajouter LocalBest à candidats
        LocalBest = NULL
        Effectuer une itération de l'Algorithme Génétique
        chances = MaxChances
    end if
end while
Déterminer la meilleure solution à partir de la liste des candidats
```

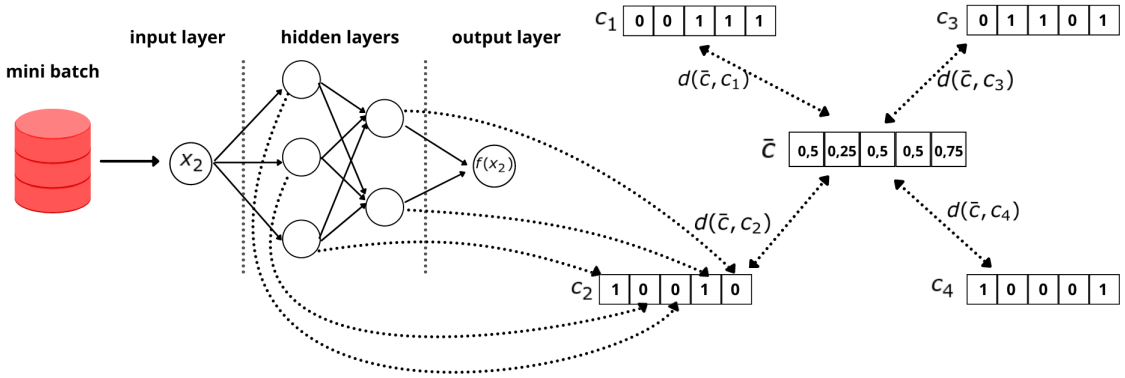


FIGURE 1 – Le processus proposé pour l'évaluation de la qualité d'un réseau de neurones : chaque échantillon x_i du mini-batch de données est utilisé pour générer un code binaire c_i, en utilisant les sorties des unités cachées données par x_i comme entrée du réseau de neurones. La moyenne de tous les codes binaires est $\bar{c}$ et est utilisée pour calculer la valeur du ICD.

4.3 Implémentation

Chaque solution (luciole) représente une architecture réseau contenue dans le NAS benchmark, puisque chaque benchmark a sa propre représentation d'une architecture, nous devons créer une implémentation dédiée pour chacun d'entre eux. Dans ce qui suit, nous décrirons notre choix pour représenter une solution, et l'implémentation des opérateurs de la métaheuristique.

4.3.1 NAS-BENCH-101

Dans le NAS-BENCH-101, tous les réseaux partagent le même squelette, qui est composé de 3 piles, chacune comprenant 3 cellules, comme le montre la partie gauche de la figure 2. Les réseaux sont différents dans le "module" (cellule), qui est représenté par des graphes acycliques dirigés (jusqu'à 7 sommets et 9 arêtes). Les opérations valides à chaque sommet sont "convolution 3x3", "convolution 1x1", et "max-pooling 3x3" [14]. La partie droite de la figure 2 montre un exemple de module dans le NAS-BENCH-101.

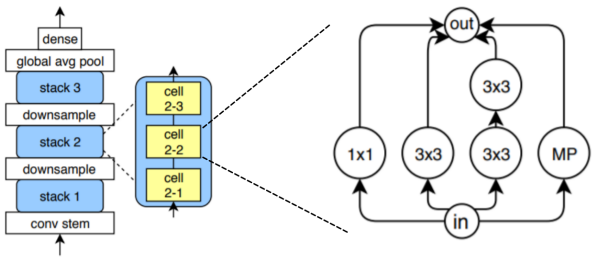


FIGURE 2 – Architecture d'un réseau dans le modèle NAS-BENCH-101 [14] : La partie gauche est le squelette partagé par tous les modèles, la partie centrale représente une pile de cellules alors que la partie droite représente un exemple de cellule (module).

Dans le NAS-BENCH-101, une architecture possible est représentée par la matrice d'adjacence du module, et une liste contenant les opérations à chaque sommet. L'exemple illustré dans la figure 2 peut être représenté à l'aide de la matrice d'adjacence présentée à la figure 3 et de la liste d'opérations suivante : [INPUT, CONV1X1, CONV3X3, CONV3X3, CONV3X3, MAXPOOL3X3, OUTPUT].

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

FIGURE 3 – matrice d'adjacence pour NAS-BENCH-101

Afin de représenter notre solution, nous choisissons de conserver la matrice d'adjacence, et d'encoder l'opération de la chaîne de caractères en nombres réels (entre 0 et 1), afin de rendre possible le déplacement de la luciole. L'encodage se fait comme suit :

- MAXPOOL3X3 = 0
- CONV1X1 = 0.33
- CONV3X3 = 0.66

Les opérateurs de la métaheuristique sont implémentés comme suit :

- Croisement : après la sélection de deux solutions (parents), une nouvelle solution est générée par :
  - Séparation horizontale des matrices des parents en deux parties, puis jointure de la partie supérieure du premier parent avec la partie inférieure du second, afin de créer la nouvelle matrice d'adjacence.
  - Séparation verticale des listes d'opérations codées des parents (aux sommets) en deux parties, puis jointure de la partie gauche du premier parent avec la partie droite du second parent, pour créer la nouvelle liste d'opérations codées.

— Mutation : sélectionner aléatoirement un sommet, puis générer un nouveau réel pour son opération (nombre aléatoire entre 0 et 1).
— Déplacement d'une luciole : le déplacement est effectué en calculant Eq.(2) par élément (pour chaque élément de la matrice et chaque élément de la liste des opérations codées). Après chaque déplacement, les valeurs inférieures à 0 sont mises à 0 et celles qui sont supérieures à 1 sont mises à 1.

Afin de rechercher une nouvelle architecture (donnée par les opérateurs de la métaheuristique) dans le NAS-BENCH-101, les opérateurs de la valeur des sommets (x) sont décodés comme suit :

$$operateur(x) = \begin{cases} MAXPOOL3X3, & \text{si } x < 0.33 \\ CONV1X1, & \text{si } 0.33 \leq x < 0.66 \\ CONV3X3, & \text{otherwise} \end{cases}$$

4.3.2 NAS-BENCH-201

Les architectures du NAS-BENCH-201 partagent le même squelette, qui commence par une convolution 3 par 3 et une couche de normalisation par lots (*batch normalization*). Trois piles de cellules sont ensuite reliées par un bloc résiduel. Le squelette se termine par une couche de *pooling* globale utilisant la moyenne (*global average pooling*), suivie d'une couche de classification utilisant *softmax* [15]. Le NAS-BENCH-201 supporte 5 opérations qui sont codées comme suit :

— none = 0
— skip connection = 1
— 1-by-1 convolution = 2
— 3-by-3 convolution = 3
— 3-by-3 average pooling = 4

De la même manière que le NAS-BENCH-101, les cellules du NAS-BENCH-201 peuvent être représentées sous la forme d'un graphe acyclique dirigé [15], elles peuvent donc être représentées à l'aide d'une matrice d'adjacence (M), où $M[i, j]$ définit l'opération existant entre les noeuds i et j. La figure 5 illustre la matrice d'adjacence utilisée pour représenter la cellule illustrée dans la figure 4.

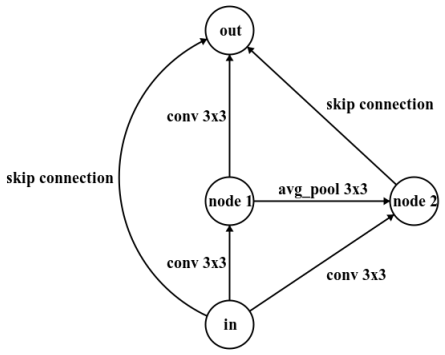


FIGURE 4 – Exemple d'un module (cellule) dans NAS-BENCH-201

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 1 & 3 & 1 & 0 \end{bmatrix}$$

FIGURE 5 – matrice d'adjacence pour NAS-BENCH-201

Les opérateurs de la métaheuristique sont implémentés comme suit :

— Croisement : effectué de la même manière que le croisement des matrices d'adjacence NAS-BENCH-101.
— Mutation : sélectionner aléatoirement un élément de la matrice, puis générer un nouvel entier pour sa valeur (nombre aléatoire entre 0 et 4).
— Déplacement d'une luciole : se fait de la même manière que le déplacement des matrices d'adjacence NAS-BENCH-101. Après chaque déplacement, les valeurs sont d'abord arrondies à l'entier le plus proche, puis, celles inférieures à 0 sont mises à 0 et celles supérieures à 4 sont mises à 4.

5 Expérimentation

Dans cette section, nous allons présenter les résultats expérimentaux obtenus sur différents benchmarks/datasets. Tous les tests ont été réalisés sur un ordinateur portable équipé d'un CPU Intel i7-11850H, de 32 Go de RAM et d'un GPU NVIDIA RTX A3000. Notez que dans l'évaluation suivante du modèle **non-entraîné**, seuls 100 échantillons ont été utilisés pour noter les réseaux neuronaux.

5.1 Validation de la métrique

Afin de valider notre proposition (basée sur l'ICD) pour l'évaluation d'un réseau de neurones sans entraînement, nous suivons le même protocole de test que Mellor et al. dans [23].

Nous avons utilisé 2000 architectures de chacun des benchmarks : NAS-BENCH-101, NAS-BENCH-201, NAS-BENCH-NLP et NDS. Tous ces benchmarks fournissent la précision et l'erreur pour chaque architecture après son entraînement, ce qui nous permet d'évaluer et de valider notre métrique sur un grand nombre de données en peu de temps. Nous avons calculé le τ de Spearman pour vérifier s'il existe une corrélation entre le score du réseau (sans apprentissage) et sa performance finale (après entraînement), qui est la précision dans le cas de NAS-BENCH-101, NAS-BENCH-201 et NDS, et l'erreur dans le cas de NAS-BENCH-NLP. Les résultats obtenus sur les différents benchmarks/datasets sont illustrés dans le tableau 1.

D'après les τ de Spearman présentés dans le tableau 1, nous pouvons dire qu'il existe une corrélation dans tous les benchmarks/datasets testés, atteignant 0,798 et 0,796 dans NAS-BENCH-201 pour CIFAR-10 et CIFAR-100, ce qui signifie qu'il existe une forte corrélation entre notre score avant l'apprentissage et la précision finale après l'apprentissage du réseau. La valeur p est égale à 0, signifiant que la probabilité que la valeur de τ soit due au hasard est nulle. La

Benchmark	Dataset	Spearman's τ	p-value
NAS-BENCH-101	CIFAR-10	0.499497	2.212611e-129
NAS-BENCH-201	CIFAR-10	0.798250	0.0
NAS-BENCH-201	CIFAR-100	0.796226	0.0
NDS (DARTS)	CIFAR-10	0.624632	7.627274e-217
NDS (Amoeba)	CIFAR-10	0.256211	2.408139e-31
NDS (ENAS)	CIFAR-10	0.501247	1.031932e-127
NDS (NASNET)	CIFAR-10	0.387336	1.369921e-72
NDS (PNAS)	CIFAR-10	0.490744	1.044203e-121
NAS-BENCH-NLP	Penn TreeBank	-0.420435	4.588134e-73

TABLE 1 – Valeur du τ de Spearman et valeur p du score ICD sur différents benchmarks/dataset.

plus petite valeur de τ était de 0,256 obtenue sur le benchmark NDS (Amoeba) en utilisant le jeu de données CIFAR-10, signifiant que la corrélation est faible.
Sur le NAS-BENCH-NLP, nous pouvons remarquer que la valeur du τ de Spearman est négative (-0,42) : il existe une corrélation entre le score avant l'entraînement et le résultat final du réseau. Dans ce cas, la corrélation est négative car le réseau est évalué en fonction de l'erreur, donc plus l'erreur est faible, meilleur est le réseau.
La figure 6 illustre les corrélations entre les scores de l'ICD (avant apprentissage) et la précision finale du test (obtenue après la apprentissage) à l'aide d'un nuage de points. Nous pouvons remarquer que dans la plupart des cas, le graphique obtenu a la forme d'une ligne avec une pente positive, montrant que plus le score avant l'apprentissage est élevé, plus la précision du test final est élevée. Cela nous permet de dire que la métrique que nous proposons est valide et peut être utilisée pour évaluer un réseau de neurones sans apprentissage, que le réseau soit un CNN ou un RNN.

5.2 Comparaison avec Mellor et al. [23]

Dans cette partie, nous comparons les résultats obtenus à l'aide de notre métrique avec la proposition de Mellor et al [23]. La comparaison se fait sur les scores τ de Kendall calculés sur les différents benchmarks NAS. Le tableau 2 montre les résultats obtenus.

Benchmark	Dataset	ICD	Mellor et al.
NAS-BENCH-101	CIFAR-10	0.353	0.285
NAS-BENCH-201	CIFAR-10	0.595	0.574
NAS-BENCH-201	CIFAR-100	0.594	0.611
NDS (DARTS)	CIFAR-10	0.457	0.467
NDS (Amoeba)	CIFAR-10	0.185	0.223
NDS (ENAS)	CIFAR-10	0.362	0.365
NDS (NASNET)	CIFAR-10	0.271	0.304
NDS (PNAS)	CIFAR-10	0.352	0.382

TABLE 2 – τ de Kendall pour l'ICD (notre proposition) ainsi que la proposition de Mellor et al. sur différents benchmarks/datasets

Nous pouvons remarquer que les τ de Kendall obtenus par la proposition de Mellor et al. et ceux obtenus par notre score ICD sont similaires de manière générale, avec quelques meilleurs résultats pour la proposition de Mellor dans certains cas à l'exemple du NDS (Amoeba) avec CIFAR-10, et de meilleurs résultats pour notre proposition dans d'autres cas à l'exemple du NAS-BENCH-101 avec CIFAR-10, mais aucune différence significative n'a été observée.
D'après les différents résultats, nous pouvons dire que la métrique proposée par Mellor et al. et notre proposition ICD sont équivalentes pour évaluer les réseaux sur une tâche de classification. Cependant, l'utilisation de toutes les unités du réseau rend notre méthode plus générique, puisqu'elle supporte les modèles RNNs (valable sur le benchmark NAS-BENCH-NLP).

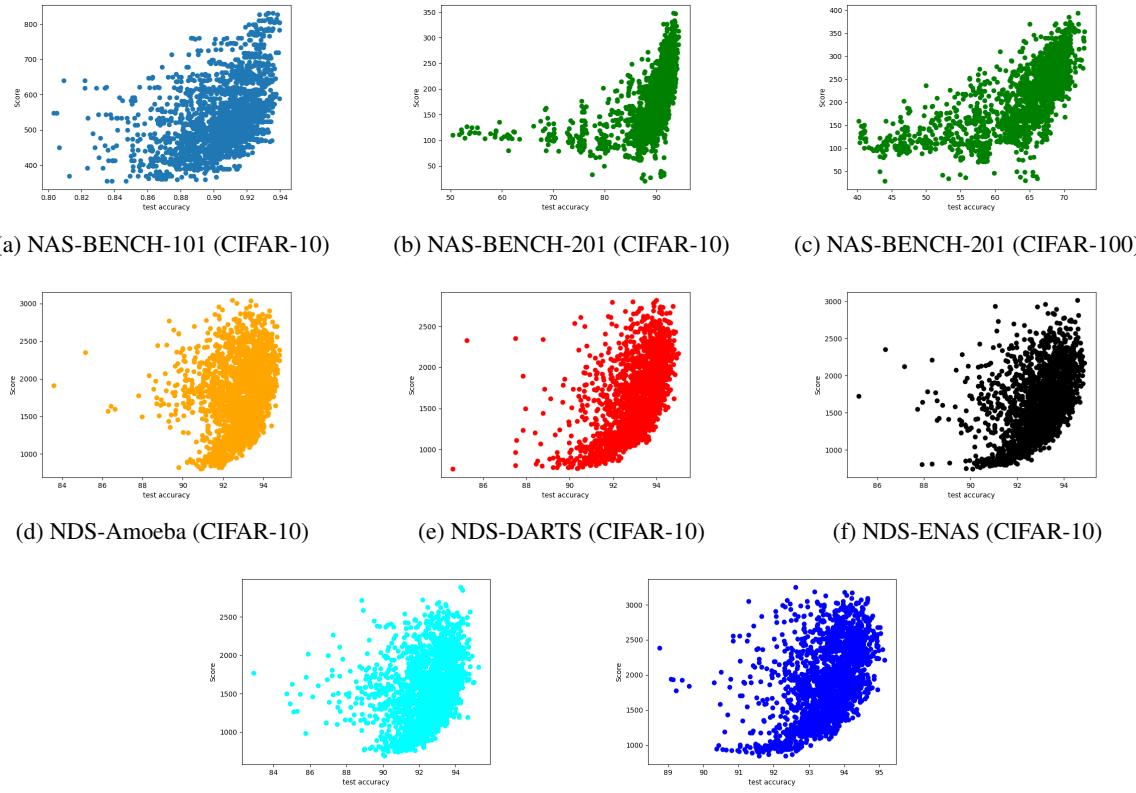
5.3 Comparaison avec d'autres méthodes sans entraînement

Dans cette partie, nous comparons notre proposition : la métrique proposée combinée à l'algorithme FireFly amélioré (IFA), aux méthodes sans entraînement les plus récentes, ainsi qu'à l'algorithme FireFly de base.
Toutes les expériences ont été réalisées en utilisant les paramètres suivants :

- Critères d'arrêt = 100 générations
- Taille de la population = 20
- Max chances = 5 (pour IFA uniquement)
- β_0 = 0.95
- γ = 0.15
- α = 0.5

5.3.1 NAS-BENCH-101

Nous avons comparé nos résultats obtenus (pour 10 exécutions) des FA, GA et IFA sur le benchmark NAS-BENCH-101, utilisant le jeu de données CIFAR-10, avec les méthodes NASWOT et AREA proposées par Mellor et al. [23]. NASWOT consiste à choisir le meilleur réseau parmi les N possibilités générées aléatoirement tandis que REA est la méthode proposée par Rere et al. [22]. Le temps de recherche moyen, la précision moyenne du test et l'écart-type sont résumés dans le tableau 3.
Tout d'abord, nous pouvons remarquer que l'IFA est plus performant que la version basique de l'algorithme FireFly, en donnant une meilleure précision de test avec un plus petit écart-type (plus stable). Ensuite, nous pouvons également remarquer que notre proposition surpasse les autres



(a) NAS-BENCH-101 (CIFAR-10) (b) NAS-BENCH-201 (CIFAR-10) (c) NAS-BENCH-201 (CIFAR-100)

(d) NDS-Amoeba (CIFAR-10) (e) NDS-DARTS (CIFAR-10) (f) NDS-ENAS (CIFAR-10)

(g) NDS-NASNET (CIFAR-10) (h) NDS-PNAS (CIFAR-10)

FIGURE 6 – Graphiques de notre score ICD pour les architectures **non-entraînées** par rapport à la précision des tests lorsqu'elles sont entraînées pour le NAS-Bench-201, NAS-Bench-101 et le NDS . Les entrées lors du calcul du score pour chaque graphique proviennent de CIFAR-10, sauf pour (c) qui utilise CIFAR-100.

Méthode	Recherche (s)	Précision de test
NASWOT (N=100)	23	91.77 ± 0.05
REA	12000	93.87 ± 0.22
AREA	12000	93.91 ± 0.29
FA (our)	3260	93.00 ± 1.44
IFA (our)	3596	$\mathbf{94.03 \pm 0.12}$

TABLE 3 – Comparaison sur le NAS-BENCH-101 (CIFAR-10) de notre proposition et des méthodes de l'état de l'art. Performances présentées en termes de précision avec une moyenne$\pm$std

méthodes, en donnant 0,12 % de plus de précision que la méthode AREA et plus de stabilité (0,12 pour IFA contre 0,22 pour AREA). Tout cela en étant plus de 3 fois plus rapide.

5.3.2 NAS-BENCH-201

Nous avons comparé les résultats obtenus (pour 10 exécutions) de FA, GA et IFA sur le benchmark NAS-BENCH-201, en utilisant les jeux de données CIFAR-10, CIFAR-100, ImageNet16-120, avec les méthodes NASWOT et EPE-NAS. EPE-NAS a été introduit par Lopes et al. [24] pour être combiné à leur métrique, leur algorithme de recherche est similaire à NASWOT (basé sur des sélections aléatoires). Comme Lopes et al. [24], nous exécutons notre proposition sur le jeu de données CIFAR-10, les architectures obtenues sont ensuite entraînées sur le CIFAR-100 et ImageNet16-120. La précision moyenne du test et l'écart-type sont résumés dans le tableau 4.

D'après les résultats obtenus, nous constatons que l'IFA que nous proposons surpasse les méthodes sans apprentissage de l'état de l'art sur les trois jeux de données du NAS-BENCH-201, avec une meilleure moyenne de précision pour chaque jeu de données.

Nous pouvons également remarquer que la version de base de l'algorithme FireFly (FA), qui utilise notre métrique ICD, surpasse les méthodes sans apprentissage de l'état de l'art sur les jeux de données CIFAR-10 et ImageNet16-120, et obtient la même précision moyenne que EPE-NAS proposée par Lopes et al. [24] sur le jeu de données CIFAR-100, mais le FA réussi à produire une valeur std plus petite, signifiant qu'il est plus stable que EPE-NAS.

En comparant FA et IFA, nous pouvons remarquer que l'IFA obtient de meilleurs résultats avec une précision supérieure et un std plus petit. Comme prévu, l'utilisation des opérations génétiques rend notre méthode plus robuste au problème des optimums locaux.

Method	CIFAR-10	CIFAR-100	ImageNet16-120
NAS-WOT (N=10)	92.47 ± 0.04	69.20 ± 1.05	42.20 ± 1.37
EPE-NAS (N=10)	92.63 ± 0.32	$\mathbf{70.10 \pm 1.71}$	41.92 ± 4.25
NAS-WOT (N=100)	91.41 ± 2.24	67.18 ± 4.14	41.42 ± 1.53
EPE-NAS (N=100)	91.59 ± 0.87	67.19 ± 3.82	38.80 ± 5.41
NAS-WOT (N=500)	91.71 ± 1.37	67.54 ± 2.23	39.84 ± 3.68
EPE-NAS (N=500)	92.27 ± 1.75	69.33 ± 0.66	42.05 ± 3.09
NAS-WOT (N=1000)	91.20 ± 2.04	68.95 ± 0.72	38.08 ± 1.58
EPE-NAS (N=1000)	91.31 ± 1.69	69.58 ± 0.83	41.84 ± 2.06
FA (our)	$\mathbf{92.90 \pm 1.07}$	$\mathbf{70.10 \pm 1.56}$	$\mathbf{43.38 \pm 1.80}$
IFA (our)	$\mathbf{93.58 \pm 0.15}$	$\mathbf{70.27 \pm 0.75}$	$\mathbf{44.53 \pm 1.54}$

TABLE 4 – Comparaison sur le NAS-BENCH-201 entre notre proposition et les méthodes de l'état de l'art. Performances indiquées en termes de précision avec moyenne$\pm$std

Benchmark(Dataset)	Méthode	score	notre score
NAS-BENCH-101(CIFAR-10)	GA-NAS [27]	94.23	94.03
NAS-BENCH-201(CIFAR-10)	β-DARTS [26]	94.36	93.58
NAS-BENCH-201(CIFAR-100)	β-DARTS [26]	73.51	70.27
NAS-BENCH-201(ImageNet)	β-DARTS-RS [26]	46.71	44.53

TABLE 5 – Comparaison sur différents benchmarks/datasets entre notre proposition et les méthodes NAS les plus performantes de l'état de l'art en fonction de la précision des tests.

5.4 Comparaison avec les méthodes NAS les plus performantes de l'état de l'art

Dans ce qui suit, nous allons comparer notre méthode proposée avec les méthodes NAS les plus performantes de l'état de l'art (basées sur l'apprentissage), en fonction de la précision des tests.

Le tableau 5 comprend la comparaison avec GA-NAS proposé par Rezaei el al. [27] sur le NAS-BENCH-101, et également la comparaison avec β-DARTS et β-DARTS-RS proposés par Peng et al. [26]. Nous remarquons que, même si la méthode sans entraînement que nous proposons n'atteint pas les mêmes performances que celles utilisant l'entraînement, l'écart entre elles est faible.

6 Conclusion

Les algorithmes NAS sont capables d'automatiser la découverte d'architectures efficaces dans un espace de recherche. Trouver le modèle le plus intéressant peut prendre beaucoup de temps car les algorithmes NAS doivent évaluer les réseaux en fonction de leurs performances pour choisir le plus adapté.

Afin d'éviter l'entraînement des modèles lors de l'exécution d'un algorithme NAS, nous avons proposé une nouvelle métrique, qui permet d'approcher la qualité d'un modèle sans aucun entraînement. Cette dernière est basée sur l'utilisation du score Intra-Cluster Distance (ICD). Les résultats obtenus sur différents benchmarks NAS (NAS-BENCH-101, NAS-BENCH-201, NDS et NAS-BENCH-NLP) montrent que notre métrique est valable pour évaluer les CNN et les RNN.

Nous avons proposé d'utiliser un algorithme Firefly amélioré (IFA) comme technique de recherche pour trouver la meilleure architecture pour un ensemble spécifique de données. Cet algorithme utilise les opérateurs de l'algorithme génétique ce qui lui permet d'être plus robuste que l'algorithme FireFly de base face au problème des optimums locaux. Les résultats expérimentaux sur les benchmarks NAS-BENCH-101 et NAS-BENCH-201 montrent que notre IFA, combiné à la métrique proposée, surpasse les méthodes sans entraînement de l'état de l'art ainsi que la version basique de l'algorithme FireFly.

Dans le cadre de travaux futurs, nous envisageons de trouver un moyen générique pour représenter les hyperparamètres d'une architecture de réseau de neurones (nombre de couches, nombre d'unités, type de cellules, etc.) en y incluant d'autres types de couches en plus des CNNs, ce qui nous permettrait d'exploiter notre proposition dans un problème d'apprentissage autre que la classification d'images.

Remerciements

Ce travail a été réalisé dans le cadre du projet franco-canadien DOMAID financé par l'Agence Nationale de la Recherche (ANR-20-CE26-0014-01) et le FRQSC.

Références

[1] N. Mokhtari, A. Nédélec, M. Gilles and P. De Loor, "Improving Neural Architecture Search by Mixing a FireFly algorithm with a Training Free Evaluation," 2022 International Joint Conference on Neural Networks (IJCNN), Padua, Italy, 2022, pp. 1-8, doi : 10.1109/IJCNN55064.2022.9892861.

[2] Cao, X., Yao, J., Xu, Z., and Meng, D. : Hyperspec-tral image classification with convolutional neural net-work

and active learning.IEEE Transactions on Geo-science and Remote Sensing, 58(7) :4604–4616, 2020.

[3] Martins, V., Kaleita, A., Gelder, B., Silveira, H., and Abe,C. : Exploring multiscale object-based convolutional neural network (multi-ocnn) for remote sensing image classification at high spatial resolution.ISPRS Journal of Photogrammetry and Remote Sensing,168 :56–73,2020.

[4] Mustaqeem and Kwon, S. : Mlt-dnet : Speech emotion recognition using 1d dilated cnn based on multilearning trick approach.Expert Systems with Applications, 167, 2020.

[5] Zhang, N., Wang, J., Wei, W., Qu, X., Cheng, N., and Xiao,J. : Cacnet : Cube attentional cnn for automaticspeech recognition. In2021 International Joint Conference on Neural Networks (IJCNN), pages 1–7, 2021.

[6] Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search : A survey. Journal of Machine Learning Research, 20(55) :1–21, 2019.

[7] Wistuba, M., Rawat, A., and Pedapati, T. A survey on neural architecture search. arXiv preprint arXiv :1905.01392, 2019.

[8] Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In International Conference on Learning Representations, 2017.

[9] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.

[10] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In International Conference on Machine Learning, 2018.

[11] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. MnasNet : Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019.

[12] Liu, H., Simonyan, K., and Yang, Y. DARTS : Differentiable architecture search. In International Conference on Learning Representations, 2019.

[13] Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In Proceedings of the AAAI Conference on Artificial Intelligence, 2019.

[14] Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K. & Hutter, F. : NAS-Bench-101 : Towards Reproducible Neural Architecture Search. Proceedings of the 36th International Conference on Machine Learning, in Proceedings of Machine Learning Research, 2019.

[15] Dong, X. and Yang, Y., "NAS-Bench-201 : Extending the Scope of Reproducible Neural Architecture Search", arXiv e-prints, 2020.

[16] Radosavovic, Ilija & Johnson, Justin & Xie, Saining & Lo, Wan-Yen & Dollár, Piotr : On Network Design Spaces for Visual Recognition, 2019.

[17] Klyuchnikov, Nikita & Trofimov, Ilya & Artemova, Ekaterina & Salnikov, Mikhail & Fedorov, Maxim & Burnaev, Evgeny. NAS-Bench-NLP : Neural Architecture Search Benchmark for Natural Language Processing, 2020.

[18] I. Strumberger, E. Tuba, N. Bacanin,M. Zivkovic, M. Beko, and M. Tuba. Designing convolutional neural network architecture by the firefly algorithm. In2019 International Young Engineers Forum (YEF-ECE), pages 59–65, 2019.

[19] Vina Ayumi, L.M. Rere, Mohamad IvanFanany, and Aniati Arymurthy. Optimization of convolutional neural network using microcanonical annealing algorithm. 102016.

[20] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary Yen. Automatically designing cnn architectures using Genetic Algorithm for image classification, 08 2018.

[21] Adenilson Carvalho, Fernando Ramos, and Antonio Chaves. Metaheuristics for the feed forward artificial neural network (ann)architecture optimization problem.Neural Computing and Applications, 20, 10 2010.

[22] L.M. Rere, Mohamad Ivan Fanany,and Aniati Arymurthy.Metaheuristic algorithms for convolution neuralnetwork.Computational Intelligence and Neuroscience, 2016, 05 2016.

[23] Mellor, J., Turner, J., Storkey, A. Crowley, E.J. : Neural Architecture Search without Training. Proceedings of the 38th International Conference on Machine Learning, in Proceedings of Machine Learning Research,2021

[24] Lopes, Vasco et al. "EPE-NAS : Efficient Performance Estimation Without Training for Neural Architecture Search." ICANN (2021).

[25] Xin-She Yang. 2008. Nature-Inspired Metaheuristic Algorithms. Luniver Press.

[26] Ye, Peng and Li, Baopu and Li, Yikang and Chen, Tao and Fan, Jiayuan and Ouyang, Wanli. β-DARTS : Beta-Decay Regularization for Differentiable Architecture Search,2022

[27] Rezaei, Seyed Saeed Changiz and Han, Fred X and Niu, Di and Salameh, Mohammad and Mills, Keith and Lian, Shuo and Lu, Wei and Jui, Shangling. Generative Adversarial Neural Architecture Search,arXiv,2021