Classification de modulation automatique utilisant les réseaux de neurones

LOUBAR Ahcene
Université des Sciences et de la Technologie Houari Boumediene
Faculté de Genie Electrique
Master Réseaux & Télécommunications
Email : ahcene.loubar@usthb.etu.dz

***Résumé*—Ce document de recherche présente une nouvelle approche pour reconnaître et prédire le type de modulation dans un ensemble de données composé de trames de données modulées. Initialement, un réseau de neurones artificiels (ANN) classique a été utilisé pour reconnaître et prédire le type de modulation , mais en raison de la faible précision de 86 %, un réseau de neurones convolutifs (CNN) a été utilisé à la place . Après avoir formé le modèle CNN, il a atteint une précision de 100 %, démontrant ses performances supérieures par rapport au modèle ANN. Cet article donne un aperçu des méthodes utilisées, des résultats obtenus et des applications et implications potentielles du modèle.**

I. Introduction

Dans ce papier de recherche, nous proposons une approche innovante pour reconnaître et prédire le type de modulation à partir de données prétraitées. Nous avons conçu un système d'intelligence artificielle (IA) capable de reconnaître neuf types de modulations différentes. Notre système d'IA est basé sur un réseau de neurones artificiels (ANN) et un jeu de données comprenant 1800 enregistrements pour l'ensemble d'entraînement (200 pour chaque type de modulation) et 540 enregistrements pour l'ensemble de test (60 pour chaque type de modulation). Les données ont été prétraitées à l'aide d'une normalisation puis un vecteur caractéristiques a été extrait manuellement à partir des données pour etre envoyé dans notre réseau de neurones. Malgré cela, les résultats obtenus ne sont pas satisfaisants.

Afin d'améliorer les résultats, nous avons envisagé d'utiliser un réseau de neurones convolutif (CNN). La mise en œuvre d'un CNN dans notre système d'IA permettra non seulement d'améliorer les résultats, mais aussi de réduire le temps et efforts fournis car contrairement aux ANNs Les réseaux de neurones convolutifs (CNN) ont la capacité unique de fusionner l'extraction de caractéristiques et la classification en un seul apprentissage et éliminent ainsi le besoin de créer un vecteur caractéristique manuellement.

Dans ce papier, nous décrirons et expliquerons les algorithmes et méthodes utilisés dans le cadre de notre système d'IA, ainsi que les résultats obtenus grâce à l'utilisation du CNN.

II. Outils mathématiques

A. Réseaux de neurones artificielles (ANN)

Les réseaux de neurones artificiels (ANN) sont une forme sophistiquée d'apprentissage automatique qui s'inspire fortement des réseaux neuronaux biologiques. Les ANN sont composés d'un grand nombre de neurones reliés entre eux formant des couches connectées, et leurs connexions sont réglées par des poids. La propagation se fait en avant à travers les couches du réseau, et les poids sont réglés à l'aide d'un algorithme d'apprentissage pour permettre à l'ANN d'apprendre à partir des données fournies.

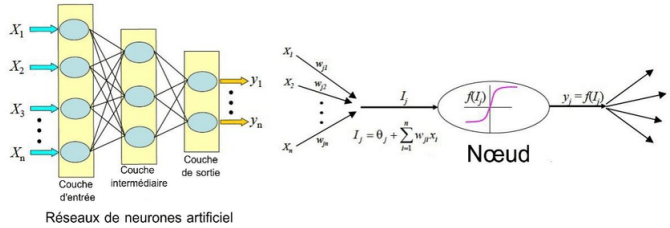


FIGURE 1. Réseau de neurones artificiel[1]

B. Réseaux de neurones convolutifs (CNN)

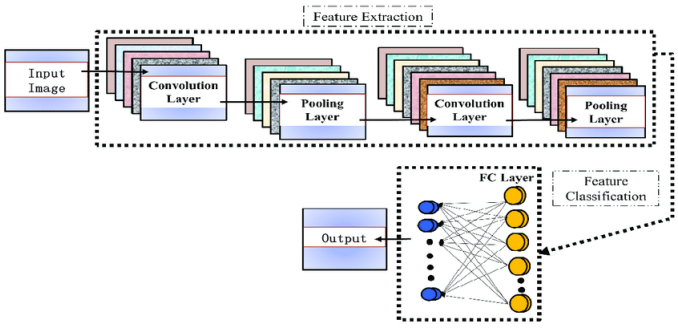


FIGURE 2. Réseau de neurones convolutif[2]

Un Réseau Neuronal Convolutionnel (CNN) est un type de réseau neuronal artificiel utilisé en apprentissage profond. Il est composé de couches hiérarchisées de neurones, avec chaque couche responsable d'une tâche différente. Les couches de

convolution sont responsables du filtrage des données et des couches de pooling aident à réduire le nombre de paramètres et à réduire la taille des données. Les données sont ensuite envoyées à une couche de neurones entièrement connectée qui est responsable de la classification et de la prédiction. Les résultats sont ensuite envoyés à la couche de sortie, qui fournit des informations sur le résultat de la prédiction.

C. Outils mathématiques

Les outils mathématiques qui sont utilisés dans les ANN incluent les fonctions d'activation, qui déterminent la sortie de chaque neurone en fonction de l'entrée. Les fonctions d'activation les plus courantes sont :

la fonction ***sigmoïde***, qui est une fonction sigmoïdale à deux étages qui produit une sortie entre 0 et 1.

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{1}$$

la fonction ***ReLU*** (Rectified Linear Unit), qui est une fonction linéaire qui produit une sortie égale à l'entrée si celle-ci est positive, et 0 sinon.

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{2}$$

la fonction ***softmax,*** est un type de fonction sigmoïde, elle permet de convertir les poids en probabilités, ce qui est utile pour les problèmes de classification multi-classe, où chaque classe a une probabilité d'être la catégorie de sortie.

$$f(x) = \frac{e^{x}}{\sum_{j} e^{xj}} \tag{3}$$

Les fonctions de perte sont utilisées pour mesurer l'erreur du réseau et orienter l'ajustement des poids. Les fonctions de perte couramment utilisées sont la fonction de coût quadratique moyenne (MSE), qui est une mesure de l'erreur entre la sortie attendue et la sortie réelle, et la fonction de cross-entropie, qui mesure l'erreur entre la prédiction et la valeur cible.

Les optimiseurs sont utilisés pour ajuster les poids du réseau.

III. MÉTHODOLOGIE

Dans le cadre ce mini-projet, on cherche à implémenter un système IA qui va nous permettre de classifier le type de modulation présent dans des signaux de télécommunications. Ce genre d'application est une solution pour permettre, par exemple, à un récepteur de détecter le type de modulation dans les trames qu'il reçoit pour qu'il puisse les démoduler. Pour cela, nous disposons d'un jeu de données qui comprend un ensemble de données d'entraînement et un ensemble de données de test. Chaque ensemble contient des signaux de télécommunication enregistrés sur des trames de 1024 échantillons, chacun d'une durée de 5,12 millisecondes et échantillonné à une période de 5 millisecondes. Chaque signal est représenté par ses composantes réelles et imaginaires.

Nos données sont distribuées de manière équilibrée selon 9 types de modulations : 8PSK, 16QAM, 64QAM, B-FM, BPSK, CPFSK, GFSK, PAM4, QPSK, 200 trames par modulation pour les données d'entraînement et 60 trames par modulation pour les données test.

Ce travail va être réalisé sur un PC portable ayant les spécificités suivantes :

- Processeur : AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz
- Mémoire installée (RAM) : 8,00 Go (5,93 Go utilisable)
- Type du systéme : Systéme d'exploitation 64 bits, processeur x64

L'implémentation se fait sur un environnement de programmation intégré (EDI), Spyder, pour Pyhton en utilisant les bibliothèques open source :

- ***Scikit-learn***, également appelé sklearn, est une bibliothèque pour le machine learning en Python.
- ***Keras***, une bibliothèque open source de réseaux de neurones pour le machine learning et le deep learning, et est intégrable à des bibliothèques telles que TensorFlow et Theano.
- ***NumPy***, une bibliothèque de calcul scientifique permettent des manipuler des structures de données telles que les tableaux et les matrices.
- ***Matplotlib***, une bibliothèque open source de visualisation de données en Python.

A. Première approche : AMC par réseaux de neurones classiques

1) Pré-traitement : La classification par réseaux de neurones nécessite la génération des caractéristiques à partir du signal. Pour ce faire, plusieurs caractéristiques sont extraites du signal temporel et fréquentiel pour chaque composante réelle et imaginaire. Les caractéristiques extraites sont ensuite concaténées pour former un vecteur caractéristique global (matrice) pour chaque trame. Le vecteur indices de classes (étiquettes, on affecte un indice pour chaque modulation) est aussi concaténé à la matrice de données pour assurer la supervision.

Pour cela on a crée un module ***Mat_Cara*** contenant la fonction de génération du vecteur caractéristique de chaque données puis sauvegarde et retourne une matrice caractéristique dont les lignes représentent les données et les colonnes représentent les caractéristiques, la dernière colonne est celle des étiquettes.

pour que la fonction ***Mat_Cara*** puisse calculer chaque caractéristique, un module ***Feature vector*** contenant 2 fonctions est utilisé :

- ***feature_extraction*** qui, pour une trame, calcule comme caractéristiques temporelles : L'amplitude maximale, l'amplitude minimale, la moyenne, la racine de la moyenne quadratique, la variance du signal, l'écart type, la puissance du signal, le pic du signal, l'écart entre les pics maximum et minimum, le facteur de Crest, l'étalement du signal, le kurtosis du signal, le facteur de forme et l'indicateur de pulse. Pour les caractéristiques

fréquentielles la fonction calcule : le pic de la puissance du spectre, puissance totale du signal, moyenne spectrale du signal, variance spectrale du signal, l'étalement du spectre, le kurtosis du spectre et le pic relatif de la puissance du signal.

— ***feature_vector*** qui va appliquer feature_extraction aux parties réelle puis imaginaire d'une trame puis va faire leur concaténation et retourne le résultat qui va représenter donc une ligne dans notre matrice caractéristique.

2) Génération du modèle: pour créer un système IA basé sur les réseaux de neurones, notre jeu d'instructions qui de base est représenté par nos données apprentissage et données test va inclure par le suite les données de validation qui représentent 25% de données retranchées des données d'apprentissage.

On se retrouve donc avec 3 ensemble de données pour entraîner et tester notre modèle :

Les données d'apprentissage sont utilisées pour entraîner le modèle . Les données de validation sont ensuite utilisées pour évaluer la précision du modèle et pour vérifier qu'il n'est pas sur-entraîné. Les données de test sont enfin utilisées pour vérifier la précision du modèle (et donc d'évaluer le modèle finale) sur des données non vues auparavant.

La toute première étape de réalisation est donc de loader, avec l'aide de la fonction *loadtxt* de *NumPy,* les ensembles de données apprentissage et test qui ont tous deux été générés et sauvegardés en pré-traitement en utilisant le module Mat_Cara. Maintenant il s'agit de créer les données de validation : pour cela, on va utiliser la fonction *train_test_split* qu'on aurait importer de la bibliothèque *sklearn.model_selection.*

Ensuite, il s'agit de créer architecture a partir de laquelle on peut générer notre modèle. Avec Python, on peut créer des réseaux de neurones artificiels à l'aide de la bibliothèque Keras. on va choisir un type de réseau à couches séquentielles où chaque couche est connectée à la couche précédente et à la couche suivante. On choisit une architecture à une seule couche cachée, c-à-d, une seule couche cachée entre la couche d'entrée et la couche de sortie (une architecture ANN à un seul niveau profond).

On utilise la classe *sequential()* de la bibliothèque *keras.models* pour initialiser le modèle puis la méthode *add()* pour rajouter les couches au modèle. La création de couches connectées se fait en utilisant la méthode *dense()* de *keras.layers* La couche d'entrée est crée automatiquement avec l'initialisation du modèle avec *sequential().* On precise la taille du vectru caractéristiue avec *input_shape*.

La méthode *compile()* de *sequential()* permet de compiler le modèle de réseau de neurones artificiels. La compilation consiste à configurer le modèle pour le processus d'apprentissage, en définissant l'optimiseur ***adam*** , la fonction de perte sparse ***categorical crossentropy*** et la métrique ***sparse categorical accuracy***.

Une fois le modèle configuré, il peut être entraîné en utilisant la méthode *fit()* pour ajuster les paramètres du modèle en fonction des données d'entraînement.

La méthode *fit()* de *sequential()* s'utilise pour entraîner le modèle. Cette méthode prend en entrée les données d'entraînement et les étiquettes et effectue des mises à jour des paramètres du modèle afin de minimiser la fonction de perte. La méthode *fit()* peut être appelée avec des arguments supplémentaires pour contrôler le processus d'apprentissage, tels que le nombre d'époques, la taille du lot et la vitesse d'apprentissage.

L'apprentissage dans notre cas à été fait avec 500 epoches et une taille de lot de 5.

Après ce qu'on vient de dire, notre modèle est entraîné et il ne reste qu'à tester ces performances.

B. Deuxième approche : AMC par réseaux de neurones convolutifs

Comme l'indique leurs nom, les CNNs c'est des architectures de réseaux de neurones auxquelles on ajoute des couches supplémentaires comme on en a parlé auparavant.

En pré-traitement, on ne fait pas d'extraction de caractéristiques, on va seulement faire en sorte , à l'aide d'une fonction *Data_Gen,* de regrouper les parties réelle et imaginaire de chaque donnée dans une matrice qu'on va ensuite envoyer dans notre réseaux de neurones pour entraîner notre modèle.

On va donc aller utiliser la même architecture pour le ANN à la quelle on ajoute le couches suivantes qui la précédent.

On va rajouter 3 couches de convolution 1D qui sont généralement utilisées pour les données unidimensionnelles. Tout comme les couches *Dense()*, les couches *Conv1D()* sont importées de *keras.layers* et elles prennent plus de paramètres : le nombre de filtres qui va mettre à 64, 128, 64 , puis la taille des filtres, *kernel_size,* qu'on va mettre 3, 5, 11 respectivement à chaque couche. Puis on met le *padding* à ***same*** et la fonction d'activation à ***ReLu***.

Aprés chaque couche de convolution on rajoute une couche *MaxPool1D* qu'on va mettre à 4, 32, 16 respectivement, suivi d'une couche de *BatchNormalization* pour normaliser les données selon une loi gaussienne. à la sortie de la dernière couche de convolution, on rajoute une couche *Flatten()* qui va permettre de générer un seule et unique vecteur caractéristique qui va etre envoyé au perception multicouche connecté et pour éviter et minimiser les risque de sur-ajustement on va ajouter une couche *Dropout()* avec un taux de 30% de neurones désactivés.

à cela on va rajouter le perceptron multicouche (à une seule couche connecté) qu'on avait utilisé en ANN sauf que la le nombre de neurones de la couche est mis à 64.

Pour l'apprentissage on procède de la même façon avec *compile()* et *fit()*.

IV. RÉSULTATS

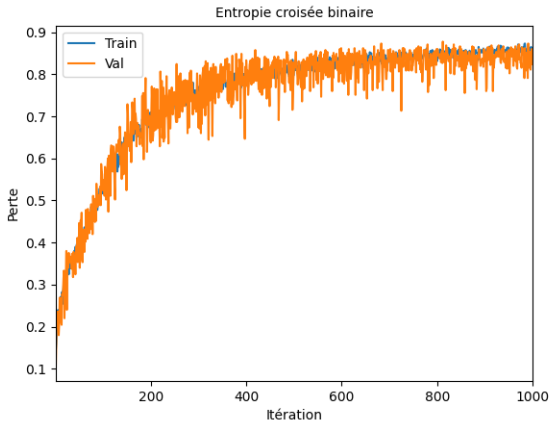
A. Performances du ANN

1) Courbes de taux de reconnaissance et pertes :

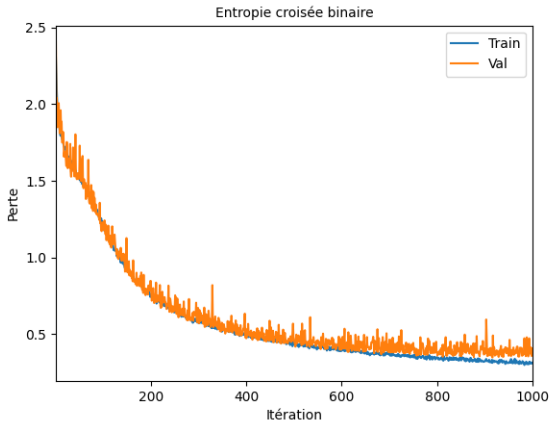
Remarque 1. Comme indiqué sur le rapport de classification, le taux de reconnaissance est à 86% et donc n'atteint pas 100% et la courbe de pertes lui est inversement proportionnelle, en

plus de cela, les courbes sont très fluctuantes et donc sont instables.

Les résultats sur les données de validation, qui sont des données ayant été tirés des données d'apprentissage à partir desquels le modèle a été crée, n'ont pas été satisfaisants ce qui nous mène à déduire que le modèle n'est pas assez prometteur pour le tester sur de nouvelles données.



(a) Taux de reconnaissance



(b) Courbes des pertes

FIGURE 3. Taux de reconnaissance et pertes

2) Rapport de classification et évaluation:

Remarque 2. D'après le rapport de classification, le modèle a obtenu une précision globale de 85,33% et une perte globale de 37%. Les scores F1 pour les différentes classes allaient de 0,58 pour la classe 0 à 1,00 pour les classes 4 et 5. Ce qui est intéressant à remarquer, c'est que le score F1 pour la classe 8 était de 0,40. Cela suggère que le modèle a eu plus de difficultés à prédire cette classe que les autres classes. En général, ces résultats ne sont pas mauvais, c-à-d, que le modèle à pu prédire la majorité des classes mais ils ne sont pas optimaux.

Problème 3. Dans notre domaine de travail, 86% de reconnaissance n'est pas assez suffisant, l'augmentation du nombre

Rapport de classification :

	precision	recall	f1-score	support
0.0	0.46	0.79	0.58	42
1.0	0.77	0.91	0.83	54
2.0	0.90	0.73	0.81	63
3.0	1.00	0.98	0.99	46
4.0	1.00	0.94	0.97	50
5.0	1.00	1.00	1.00	42
6.0	1.00	1.00	1.00	57
7.0	1.00	1.00	1.00	49
8.0	0.61	0.30	0.40	47
accuracy			0.85	450
macro avg	0.86	0.85	0.84	450
weighted avg	0.87	0.85	0.85	450

(a) Rapport de classification

```
sparse_categorical_accuracy: 0.8533
Perte globale : 0.37
Justesse du modèle (Accuracy) : 85.33
```

(b) Évaluation

FIGURE 4. Rapport de classification et évaluation

d'epoches et des neurones dans la couches cachées n'a pas amélioré les résultats, donc, le problème pourrait se dans le choix et le nombre de caractéristiques calculé manuellement.

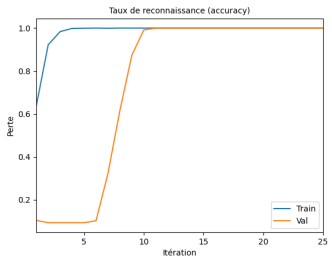
Solution 4. Une structure automatique qui déterminerai elle même les bons caractéristiques, comme les réseaux de neurones convolutifs, donnerait peut être de meilleurs résultats.

B. Performances du CNN

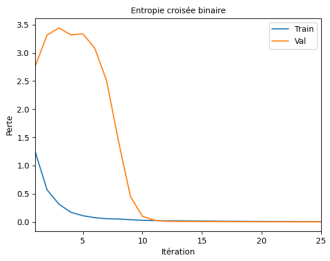
1) Courbes de taux de reconnaissance et pertes :

Remarque 5. Contrairement aux courbes du réseau ANN du taux de reconnaissance, celles du CNN sont plus stables et elles convergent vers 100% aux alentours de la dixième itération.

Les courbes sont parfaitement lisses à la convergence contrairement au cas précédent ce qui montre une certaine stabilité dans l'apprentissage. Ce résultat est commun au deux courbes ce qui montre que le modèle généralise bien et qu'on a pas de sur-apprentissage.



(a) Taux de reconnaissance



(b) Pertes

FIGURE 5. Courbe de taux de reconnaissance et pertes CNN

2) Rapport de classification et évaluation:

Remarque 6. D'après le rapport de validation de classification de votre réseau CNN, il semble que le modèle fonctionne très bien. La précision, le rappel et le F1-score sont tous les trois à 1,00, indiquant que le modèle classe correctement chaque échantillon. De plus, l'exactitude est également à 1,00, suggérant que le modèle est appliqué au bon ensemble de données. En fin de compte, il semble que votre CNN fonctionne très bien et est capable de classer correctement tous les échantillons sur lesquels il est entraîné.

Rapport de classification Val:

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	42
1.0	1.00	1.00	1.00	54
2.0	1.00	1.00	1.00	63
3.0	1.00	1.00	1.00	46
4.0	1.00	1.00	1.00	50
5.0	1.00	1.00	1.00	42
6.0	1.00	1.00	1.00	57
7.0	1.00	1.00	1.00	49
8.0	1.00	1.00	1.00	47
accuracy			1.00	450
macro avg	1.00	1.00	1.00	450
weighted avg	1.00	1.00	1.00	450

(a) Rapport de classification

```
sparse_categorical_accuracy: 1.0000
Perte globale val : 0.00
Justesse du modèle val (Accuracy) : 100.00
```

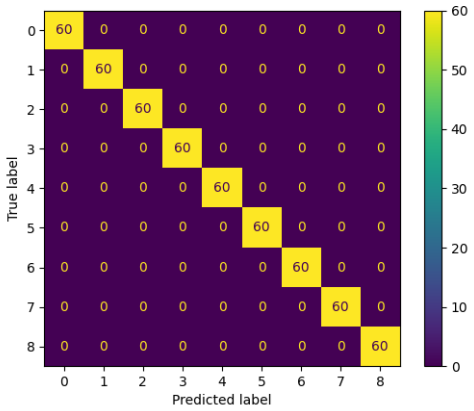
(b) évaluation

FIGURE 6. Rapport de classification et évaluation

3) Test du modèle sur de nouvelles données :

Remarque 7. Le taux de reconnaissance sur les données test est de 100% et les pertes sont à 0% comme pour les données de validation.

La matrice de confusion nous montre qu'il n y a eu 0 erreurs quand à la classification ce qui confirme que le modèle à bien été optimisé par l'ajout de couches de convolution.



(a) Matrice de confusion

```
sparse_categorical_accuracy: 1.0000
Perte globale test: 0.00
Justesse du modèle test (Accuracy) : 100.00
```

(b) évaluation

FIGURE 7. Performances du modèle sur les données test

V. CONCLUSIONS

En conclusion, ce mini projet a démontré que les réseaux de neurones convolutifs offrent des performances supérieures pour la reconnaissance et la prédiction du type de modulation par rapport aux réseaux de neurones artificiels. Cela est dû à la capacité des CNN à prendre en compte l'information spatiale et la structure des signaux et aussi à générer, de manière automatique, suffisamment de caractéristiques pour l'apprentissage contrairement aux méthodes manuelles, qui leur permet de mieux gérer la classification des données modulées que les ANN. De plus, les CNN sont plus efficaces car ils peuvent traiter plusieurs couches de données en même temps. Ces résultats peuvent avoir des implications considérables pour les domaines de l'apprentissage automatique et du traitement du signal.

RÉFÉRENCES

[1] Mohammed Amin, Benbouras & , Ratiba & Debiche, Fatiha & Nassim, Hallal & Lagaguine, Maroua & Petrisor, Alexandru-Ionut. (2019). A review of Artificial Neural Networks used for estimating Mechanical Soil Parameters.

[2] Ali, Saqib & Li, Jianqiang & Pei, Yan & Aslam, Muhammad & Shaukat, Zeeshan & Azeem, Muhammad. (2020). An Effective and Improved CNN-ELM Classifier for Handwritten Digits Recognition and Classification. Symmetry. 12. 15. 10.3390/sym12101742.