# Rapport - semaine 2 - juin 2020

## Test du réseau de neurones (CNN)

L’objective est d’entraîner un réseau de neurones avec donnés différentes d’entrée pour vérifier et tester avec quelle type d’entrée on a des mieux résultats, c’est-à-dire, avec quelle données l’algorithme va apprendre mieux. Pour faire répondre cette dernière question on a besoin d’analyser quelques points, par exemple, la précision et le temps d’exécution.

Les données d’entrée seront dans un premier moment des hologrammes et dans un deuxième moment leurs distributions de Wigner.

#### Base des données 1 – Hologrammes

La base des donnés utilisé a 1500 hologrammes au totale, on a 5 classes (nombre de sources ponctuelles) et donc 300 hologrammes par classe.



Figure : base des données

#### Division de la base des données en trainset et testset

Afin d’entrainer le réseau j’ai divisé la base des données en *trainset*, avec 80% de hologrammes de la base d’origine et *testset,* avec les 20% restants. L’image suivante montre les dimensions des nouvelles matrix (à gauche) et aussi les longueurs de la liste des *targets* (à droite)*.*



Figure : division de la base des données

#### Pré traitement des données

Il faut traiter la base des donnés avant, où il faut la remodeler pour être en conformité avec la nôtre modèle. L’image suivante montre les nouvelles dimensions, où le premier chiffre est le nombre des hologrammes, puis vient la forme de chaque matrix (200 x 200) et finalement, le dernier, c’est le chiffre 1, ce qui signifie que les images sont en niveaux de gris.



Figure : Nouvelles dimensions des matrices

Il faut aussi l’implémenter le principe du « *one-hot-encode »* dans notre vecteur avec les *targets.* C’est-à-dire, créer une colonne pour chaque catégorie avec une variable binaire.

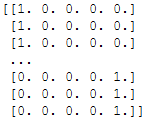


Figure : Dimensions des matrices targets après le redimensionnement

#### Construction du model

J’ai utilisé la bibliothèque Keras, car elle est une API facile à utiliser et est le plus populaire *framework* pour l’apprentissage profond (*Deep learning*).

Le type de model utilisé était le *Sequential,* car ce type permets construire une model couche par couche. Les deux premières couches sont des couches de convolution 2D. J’ai mis 64 et 32 comme les nombres de nœuds dans la première et la deuxième couche, les mêmes valeurs utilisés dans la référence que j’ai utilisé comme guide. Pour la taille du *kernel*, j’ai mis 3, avec le même justificatif d’avant.

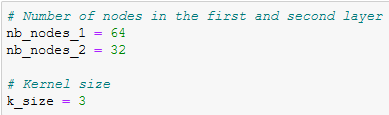


Figure : Quelques paramètres des couches de convolution

On a utilisé l’activation linéaire rectifiée (*relu*) comme fonction d'activation pour les deux premières couches, car elle utilise souvent dans les réseaux des neurones. Après la deuxième couche de convolution on a mis un couche *Flatten¸* cette couche est responsable pour faire la connexion entre la couche de convolution (*Conv2D*) et les couches denses (*Dense*).

La couche *Dense* est utilisée pour la couche de sortie. On a mis 5 nœuds pour la couche de sortie, un pour chaque résultat possible (1-5 sources ponctuelles). De plus, on a utilisé le *softmax* comme fonction d'activation, car il fait passer la sortie à 1 pour que la sortie puisse être interprétée comme des probabilités# peuvent être interprétées comme des probabilités.

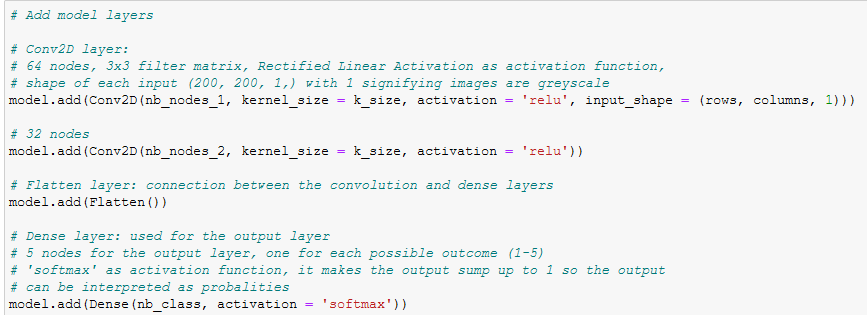


Figure : Construction du model

#### Compilation du model

La compilation de ce modèle nécessite 3 paramètres, le « *optmizer »*, qu’on a mis le « *adam »,* qui il est suivant utilisé ; une fonction de perte, qu’on a mis le *« categorical\_crossentropy* », le choix le plus courant pour la classification ; et finalement la métrique, qu’on a mis la précision (*« metrics »).*

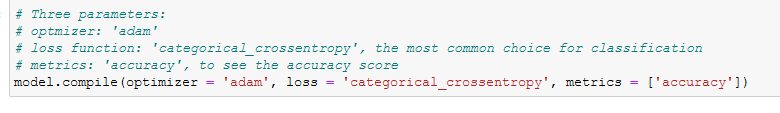


Figure : Paramètres de la compilation du model

#### Entrainer le model

Pour entraîner le model on a mis trois paramètres : la base pour entrainer (X\_train), une matrice avec les résultats (Y\_train), une base pour faire la validation, on a utilisé la base des tests (X\_test, Y\_test) et nombre de « *epochs*, » nombre de fois que l'algorithme d'apprentissage travaillera pendant toute la durée de la formation.

Pour nombre de *« epochs »* égal à 10, on a pris 16 minutes pour arriver à une précision de seulement 24,67%.

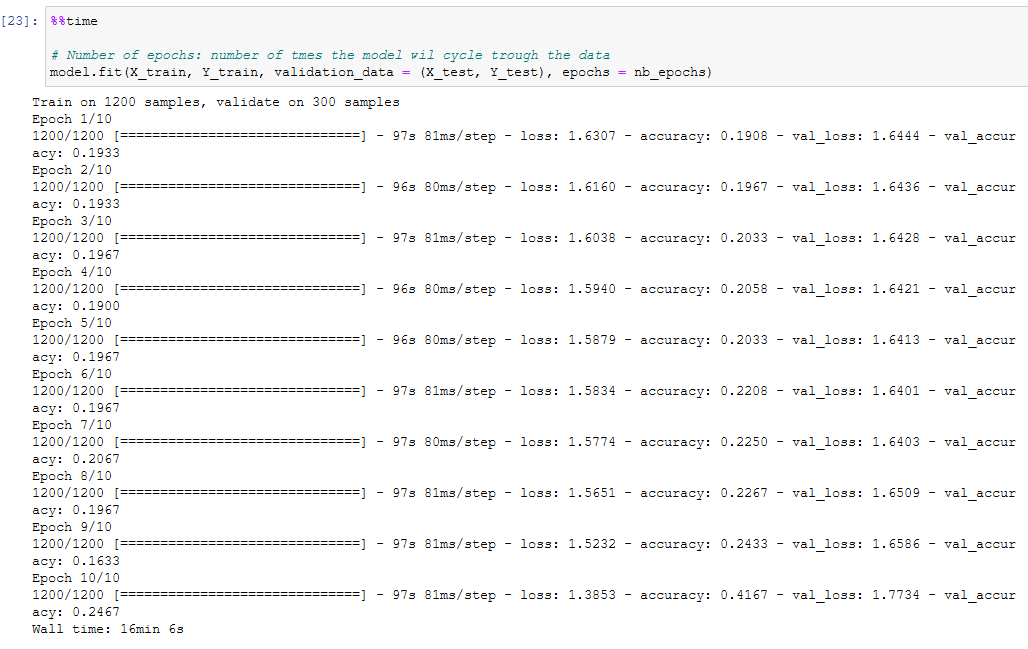


Figure : Entrainement du réseau

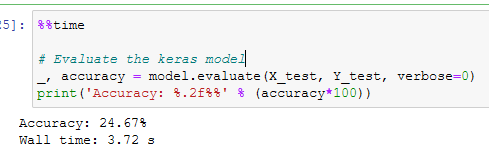


Figure : Précision

Finalement, l’image suivant affiche une synthèse du model :

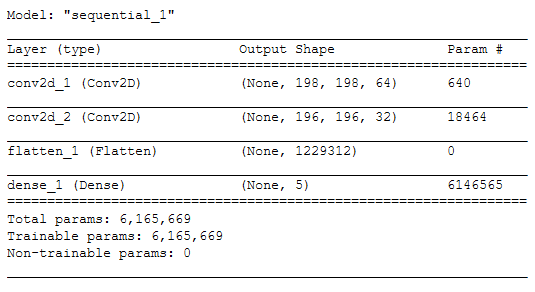


Figure : Résumé du model

## Conclusion

Les premiers résultats avec seulement des hologrammes ne donnent pas de bonnes perspectives. Je vais essayer faire des tests avec un numéro plus grand de *« epochs ».*

## Reference

[1] Bibliothèque Keras :

<https://keras.io/>

[2] Building a Convolutional Neural Network (CNN) in Keras :

<https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5>

[3] Your First Deep Learning Project in Python with Keras Step-By-Step :

<https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>

[4] Jupyter notebook avec les tests :

<https://github.com/fernandolucasaa/computerGeneratedHolography/blob/master/neural_network/jupyterNeuralNetworks.ipynb>