HMIN 339M

Méthodes avancées de la science des données

Classification de l'occupation du sol en Dordogne

Réalisé par:
DIDIER Adrien - 21609811
MARTIN Loïc - 21605214

Sommaire

1. Mise en situation

Pour ce projet, nous avons essayé plusieurs types de réseaux neurones. En effet, nous pouvons noter l'implémentation d'un MLP (Multilayer Perceptron), d'un CNN1D et d'un CNN2D (Convolutional Neural Networks).

Nous pourrons ainsi voir si ces derniers ainsi que l'utilisation de différentes couches ont une influence sur la précision du modèle.
En effet, chaque type de réseaux de neurones aura des couches spécifiques pour effectuer certaines tâches.

Avant d'arriver à la création du modèle, nous avons dû réaliser plusieurs étapes.
Premièrement, nous avons dû lire le jeu de données et normaliser les valeurs entre 0 et 1 pour chaque série temporelle.
Ensuite, nous avons récupéré la vérité terrain et la position des pixels d'entraînement et de test.
Depuis le jeu d'entraînement, nous pouvons créer un jeu de validation qui nous permettra d'estimer la prédiction d'erreurs de notre modèle.

Pour terminer, chaque modèle aura sa propre manière pour lire les valeurs des séries temporelles, cette étape sera décrite dans les parties suivantes.

Pour chaque modèle, nous réaliserons différents tests pour obtenir des valeurs comparables.
Notre analyse des modèles portera sur les F1 score, les valeurs de loss.
Tous nos entraînements ont été effectués avec un batch_size de 256, 20 epochs et l'utilisation de callbacks pour obtenir le meilleur modèle lors de l'évaluation de ce dernier.

2. MLP

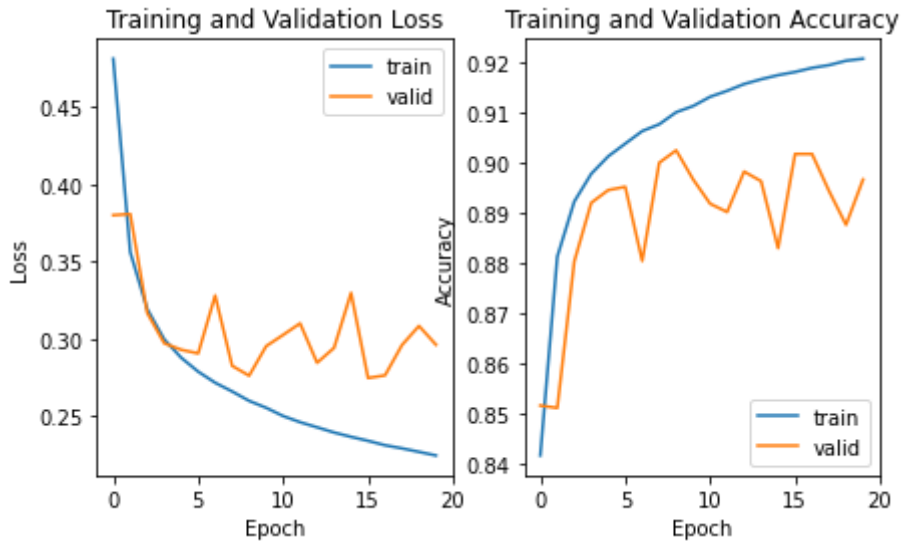
Tout d'abord, nous avons implémenté un perceptron pour comprendre le jeu de données et obtenir un premier résultat. Une fois ce réseau fonctionnel, nous avons pu développer le MLP.

Pour un MLP, il est nécessaire d'obtenir un tableau 2D du type (nombre d'échantillons, nombre de features = (nombre de dates x nombre de bandes)). Ceci se montrera extrêmement important pour créer la couche d'entrée de notre réseau de neurones.

Notre premier modèle ressemblait à ceci :

```
input_shape = (32)
model = tf.keras.models.Sequential()
model.add(tf.keras.Input(shape=(32)))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(len(labels), activation='softmax'))
```

Nous pouvons observer le shape de 32 dans notre couche d'entrée ainsi l'utilisation de 2 couches cachées utilisant la fonction d'activation relu et notre couche de sortie softmax qui possède 5 neurones (nombre de labels).



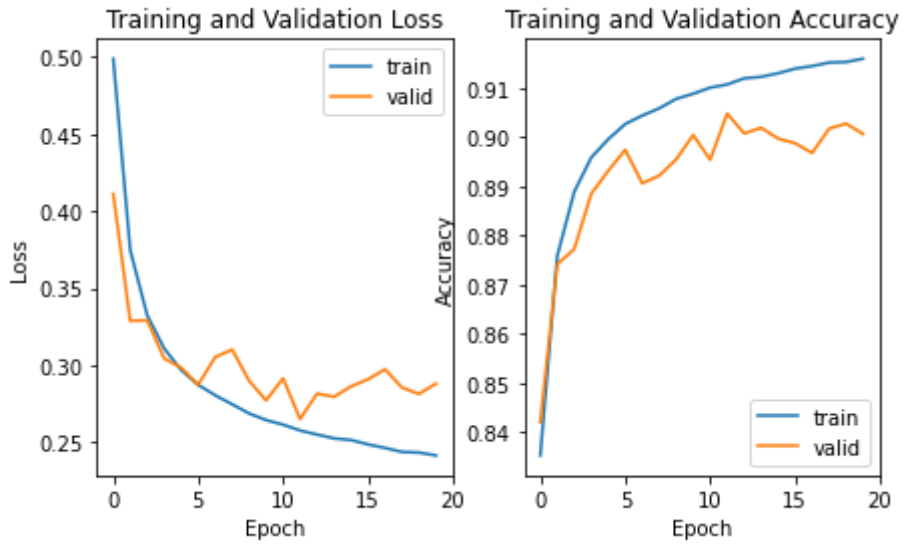
Nous pouvons observer que notre modèle obtient de superbes résultats sur le jeu d'entraînement, toutefois en comparant la courbe obtenue avec celle du jeu de validation, nous pouvons observer que le modèle est en overfitting, pour cela, il est nécessaire d'utiliser du dropout.

Il est également possible de prédire des classes sur le jeu de validation et évaluer en agrégeant au niveau objet.
La valeur obtenue est le F1 score qui est une mesure de la précision d'un test.
Pour ce modèle particulier, nous obtenons un résultat de `0.8401378401179921`.

Ce F1 score confirme bien la tendance du modèle à réaliser du surentraînement, car nous sommes bien loin des 0.93 annoncés par l'apprentissage sur le jeu de d'entraînement.

Pour éviter le surentraînement, nous avons décidé de mettre en place du dropout.
En effet, le dropout est une technique qui est destinée à empêcher le sur-ajustement sur les données de training en abandonnant des unités dans un réseau de neurones. Cela force le modèle à éviter de trop s'appuyer sur un ensemble particulier de features.

```
model = tf.keras.models.Sequential()
model.add(tf.keras.Input(shape=(32)))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(tf.keras.layers.Dense(len(labels), activation='softmax'))
model.output_shape
```

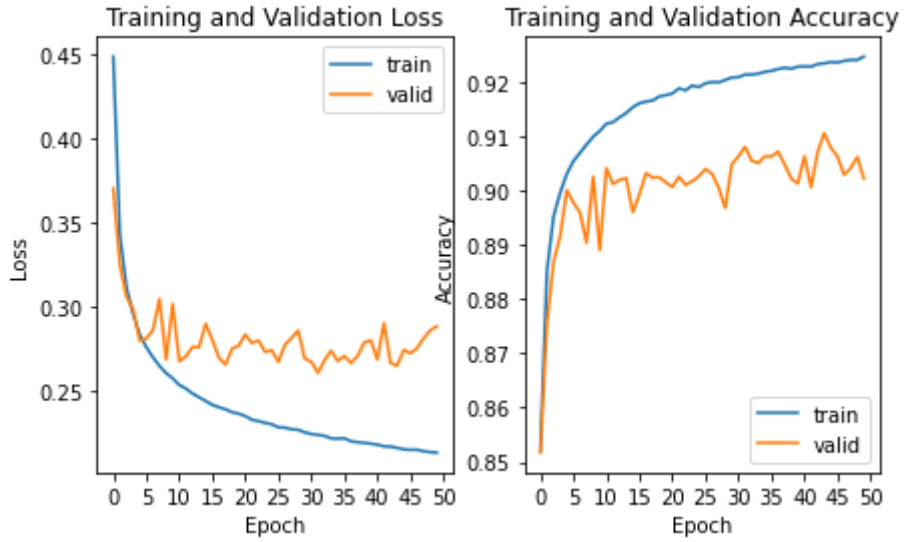


Cette fois-ci, les deux courbes sont plutôt ressemblantes, nous avons donc réussi à résoudre le problème d'overfitting et rendu notre modèle plus fiable en cas de nouveaux jeux de données.

Une fois la prédiction réalisée de nouveau, nous obtenons désormais un F1 score de `0.861028`
Nous observons donc une amélioration de la précision de notre modèle.

Il pourrait cependant être intéressant de modifier le nombre d'epochs et la taille du batch_size pour se rendre compte si ces derniers ont réellement un impact sur les performances du modèle.

Pour ce test, nous allons augmenter le nombre d'epochs à 50 et utiliser un batch_size de 128 et conserver le modèle précédemment implémenté.



F1 score : `0.857733587582501`

Nous pouvons observer que ce changement n'a pas amélioré le modèle, en effet même si la précision du training s'est vue améliorée ce n'est pas le cas pour le jeu de validation qui stagne dès les premières epochs.

Après ces différents tests, il nous semble intéressant de nous concentrer sur l'utilisation des CNN pour vérifier si ces derniers proposent de meilleures performances qu'un MLP dans le cadre de la classification d'images.

Nous avons finalement obtenu une précision de 0.837265 sur les prédictions au niveau objet.

3. CNN1D

En premier lieu, nous allons définir ce qu'est un CNN1D et l'intérêt de ce dernier.
Un CNN1D permet d'identifier des modèles simples dans nos données qui seront ensuite utilisés pour former des modèles plus complexes dans des couches supérieures.

Un CNN1D nécessite un tableau 3D du type (nombre d'échantillons, nombre de dates, nombre de bandes).

```
input_shape = (32)
n_timesteps, n_features, n_outputs = train_X.shape[1],
train_X.shape[2], train_y.shape[0]
```

La couche d'entrée demande une shape (n_timesteps, n_features), nous avons train_X de shape (`451962, 8, 4)` où il y a bien les 8 temporalités avec les 4 spectres différents.

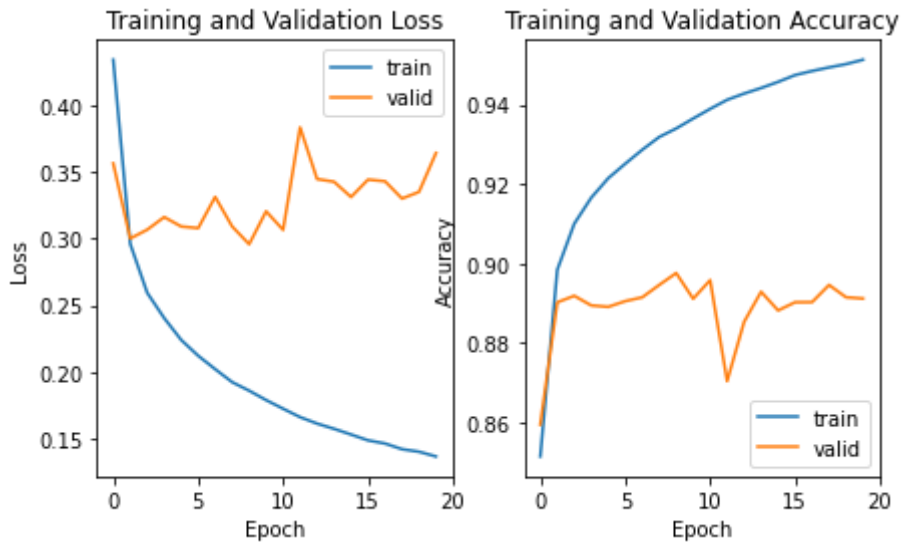
Un CNN1D propose trois couches :

- la couche de convolution qui est la composante clé des réseaux de neurones convolutifs, et constitue toujours au moins leur première couche.
Son but est de repérer la présence d'un ensemble de features dans les images reçues en entrée.
- la couche de pooling qui consiste à réduire la taille des images, tout en préservant leurs caractéristiques importantes.
- la couche de correction ReLU qui permet d'améliorer l'efficacité du traitement en intercalant entre les couches de traitement une couche qui va opérer une fonction mathématique (fonction d'activation) sur les signaux de sortie.

Il nous a donc fallu allier ces différentes couches pour développer des modèles qui sauraient user des propriétés de ces dernières.

Nous avons donc mis en place plusieurs modèles utilisant les différentes couches décrites précédemment. Le meilleur modèle que nous avons pu obtenir sur ce jeu de données est celui-ci :

```
model = Sequential()

model.add(Conv1D(filters=128, kernel_size=3, activation='relu',
input_shape=(n_timesteps,n_features)))
model.add(MaxPooling1D(pool_size=2))

model.add(Conv1D(filters=128, kernel_size=2, activation='relu'))
model.add(MaxPooling1D(pool_size=2))

model.add(Conv1D(filters=128, kernel_size=1, activation='relu'))
model.add(MaxPooling1D(pool_size=1))

model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(len(labels), activation='softmax'))
```



F1_score : 0.8755242469335304

Cette fois-ci, nous avons obtenu, une précision de 0.847705 pour la prédiction des objets.

4. CNN2D

Un CNN 2D ou spatial requiert un tableau 3D du type (nombre d'échantillons, largeur, hauteur, nombre de features= (nombre de dates x nombre de bandes)).

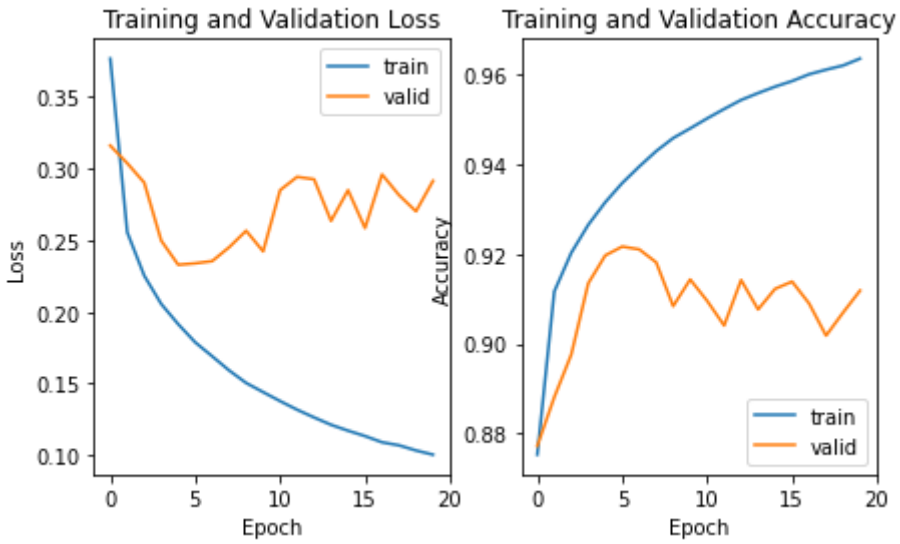
Un CNN2D propose aussi l'utilisation de différentes couches tout comme le CNN1D. Nous pouvons noter la couche de pooling, la couche de convolution. La couche de BatchNormalization qui applique une transformation qui maintient la sortie moyenne proche de 0 et l'écart-type de sortie proche de 1.

Pour un CNN bidimensionnel, il est possible d'extraire des patchs (ici 5x5) autour des pixels à classifier pour obtenir une meilleure prédiction de ces derniers.

Voici le modèle avec lequel nous avons obtenu les meilleurs résultats.
Nous pouvons noter la valeur de l'input_shape, nous utilisons bien les patchs de 5x5 et les 8 temporalités avec les 4 bandes spectrales.

```
model = Sequential()

model.add(Conv2D(64, (3, 3), input_shape=(5, 5, 32)))
model.add(Activation("relu"))
model.add(Dropout(0.2))

model.add(Conv2D(32, (3, 3), strides=(2, 2), padding="same"))
model.add(Activation("relu"))
model.add(Dropout(0.2))

model.add(GlobalAveragePooling2D())

model.add(Dense(5))
model.add(Activation('softmax'))
```

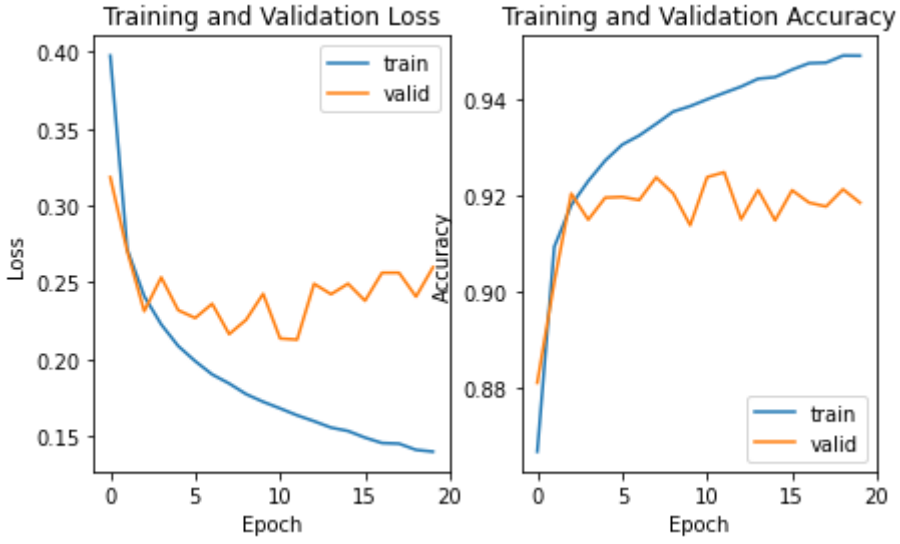
Nous avons donc utilisé des couches de dropout avec une valeur de 0.2 en s'inspirant du CNN1D. Toutefois, nous pouvons penser que le modèle est en overfitting au vu de la différence de loss et d'accuracy entre l'entraînement et la validation.



F1 score : `0.954292`
Prédiction sur les objets : `0.869464`

C'est pourquoi, nous avons décidé d'augmenter la valeur de dropout à 0.5. En regardant le graphe, nous observons moins la tendance du modèle à overfit. Cependant, en comparant le F1 score et la prédiction sur les objets, nous observons un écart qui se creuse. Ce dernier est encore plus important que lors du test précédent. Cette fois-ci, nous pouvons considérer que le modèle a réalisé de l'underfitting.

Le modèle ne se rapproche pas assez de la fonction et est donc incapable de saisir la tendance sous-jacente des données. Il ignore une grande partie des entités et conséquemment produit des étiquettes incorrectes.



F1 score : `0.966478`
Prédiction sur les objets : `0.8470930157148092`

5. Conclusion

Ce rapport présente donc les choix faits lors de notre travail. Il montre comment nous sommes venu à ces résultats et présente les différentes analyses menées pour arriver à améliorer nos modèles.

En passant de l'utilisation de différents réseaux de neurones comme les MLP et les CNN, à l'utilisation de différentes couches, nous avons voulu réaliser de nombreux tests pour obtenir des modèles satisfaisants. Il nous semblait judicieux de tester plusieurs configurations pour obtenir des données intéressantes à analyser et comparer.

D'après nos tests, nous obtenons une meilleure prédiction sur les objets et un meilleur F1 score à l'aide du CNN2D. Toutefois, le MLP se démarque par ses courbes d'accuracy et de loss qui se rapproche énormément entre les jeux de tests et de validation.

D'un point de vue personnel, le domaine de l'intelligence artificielle nous intéresse grandement et nous souhaitons intégrer ce domaine à notre futur métier, c'est pourquoi nous avons pris énormément de plaisir à suivre cette matière.
En effet, étant étudiants en DECOL et amateurs d'intelligence artificielle, le choix de cette UE était évident et intéressant pour nous. Il nous a apporté de multiples aspects sur le point de vue technique. Il nous a permis de redécouvrir le deep learning et l'apprentissage supervisé. Nous avons pu affiner nos connaissances en python et des différentes librairies. Les compétences acquises sont extrêmement bénéfiques pour notre culture informatique.
Les données proposées et le thème du projet nous ont extrêmement plu, en effet l'utilisation de temporalité spatiale et de bandes spectrales a été vraiment intéressante. De plus, l'approche demandée pour le projet nous a demandé de mettre en application nos connaissances pour développer les modèles et pouvoir les analyser.

Les différents codes pour mener à bien ce projet sont disponibles sur le lien Github suivant :
https://github.com/LoicMartin34/ClassificationSolDordogne.git