Rapport de Compétion Kaggle 1
Fondements de l'Apprentissage Machine
IFT 6390

Equipe et membre(s) :
MAMACHE Hamed Nazim

Professeur :
M. RABUSSEAU Guillaume

ID Kaggle :
Nazim_Mamache

Trimestre d'automne 2022

29 octobre 2022

Table des matières

1 Introduction

Pour cette compétition Kaggle, on travaille sur la détection de chiffres manuscrits. L'objectif est de concevoir un algorithme d'apprentissage qui peut classer automatiquement des images. La figure 2 montre un exemple d'image provenant du jeu de données d'entraînement. Toutes les images (y compris ceux du jeu de données de test retenus) sont de dimension 56×28. La classe d'une image est indiquée par la somme de ces chiffres. Par exemple, la classe de l'image de la figure 2 est 5. Le jeu des données a été généré à partir du célèbre jeu de données MNIST. L'ensemble d'entraînement et l'ensemble de test pour ces données se composent respectivement de 50 000 et 10 000 échantillons.

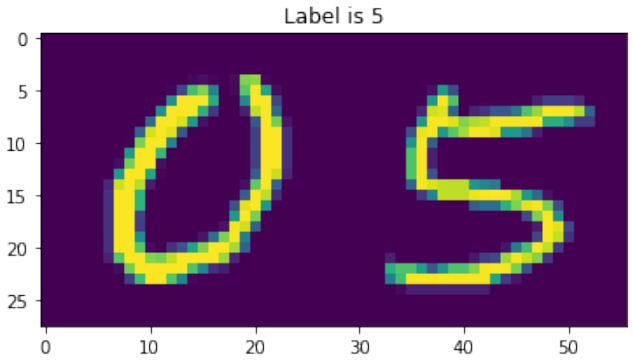

FIGURE 1 – Exemple d'image du jeu de données d'entraînement, avec le label correspondant

Le but de la compétition est d'implémenter et d'entraîner plusieurs algorithmes de classification basés sur le jeu de données fourni.

La compétition et ses résultats sont disponibles sur le lien suivant :

https://www.kaggle.com/t/0d0b1c033ece47ffa1dbc8bd374689ae

2 Feature Design

La conception et la sélection des features s'est faite de la manière suivante. Au départ les données sont chargées.

```
train_x = pd.read_csv('./classification-of-mnist-digits/train.csv')
train_y = pd.read_csv('./classification-of-mnist-digits/train_result.csv')
test_x = pd.read_csv('./classification-of-mnist-digits/test.csv')
```

Listing 1 – Chargement des données

On procède ensuite au split des données : 80% des données sont consacrées pour le train, 20% pour la validation.

```
val_x = train_x[:10000]
val_y = train_y[:10000]
train_x = train_x[10000:]
train_y = train_y[10000:]
```

Listing 2 – Split Train and Valid Data

On procède ensuite au nettoyage des données : ici, on supprime seulement la dernière colonne des features (1569ème colonne du jeu de données qui ne contenaient que des valeurs aberrantes, des NaN par exemple).

```
train_x = train_x.iloc[: , :-1]
val_x = val_x.iloc[: , :-1]
test_x = test_x.iloc[: , :-1]
train_y = train_y.iloc[:,1:]
val_y = val_y.iloc[:,1:]
```

Listing 3 – Nettoyage des données

Enfin, on convertit l'ensemble des données en *np.array*().

```
train_x = train_x.to_numpy()
val_x = val_x.to_numpy()
test_x = test_x.to_numpy()
train_y = train_y.to_numpy()
val_y = val_y.to_numpy()
```

Listing 4 – Conversion en np.array()

Pour certains modèles, comme nous le verrons un peu plus tard, nous aurons besoin de transformer les labels en one hot.

```
def one_hot(a, num_classes):
  return np.squeeze(np.eye(num_classes)[a.reshape(-1)])
train_y = one_hot(train_y, 19)
val_y = one_hot(val_y, 19)
```

Listing 5 – Conversion des labels en one hot

3 Algorithmes

3.1 Régression logistique

L'architecture du modèle est un simple réseau de neurones composé d'un seul perceptron. L'entrée est de taille 1568 (nombre de pixels de l'image 56×28) et la sortie est de taille 19 (vecteur one hot des classes possibles). Ainsi, on procède à la conversion des labels en one hot. Le modèle de Régression Logistique a été construit de zéro.
Voici l'architecture du modèle : (*source :* https://thedatafrog.com/en/articles/logistic-regression/)

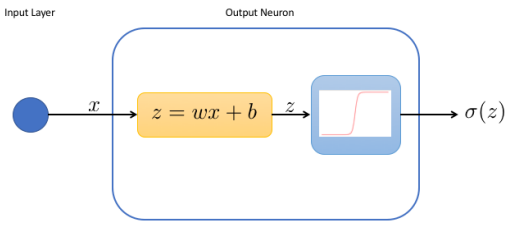

FIGURE 2 – Architecture du réseau à 1 neurone : la régression logistique

3.2 RandomForest

Le second algorithme implémenté est RandomForest de la librairie sklearn. L'implémentation est plûtot simple, le nombre d'arbres utilisés est de 1000.

```
#Create a RandomForest Classifier
clf = RandomForestClassifier(n_estimators=1000)

#Train the model
clf.fit(train_x,train_y)

preds =clf.predict(test_x)
```

Listing 6 – RandomClassifier from sklearn

3.3 MLP

Pour l'implémentation d'un MLP, la librairie keras a été utilisée. Le modèle a été implémenté de la manière suivante.

```
model = Sequential()
model.add(Dense(32, input_dim = 1568, activation= 'relu'))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(128, activation = 'relu'))
model.add(Dense(256, activation = 'relu'))
model.add(Dense(19, activation = 'softmax'))
```

Listing 7 – MLP model with keras

3.4 CNN

Pour l'implémentation d'un CNN, la librairie keras a encore une fois été utilisée. Le modèle a été implémenté de la manière suivante.

```
num_classes = 19
input_shape = (56, 28, 1)

model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Listing 8 – CNN model with keras

Dans la partie suivante, nous discuterons des hyper-paramètres choisis, des astuces d'optimisation etc...

4 Méthodologie

4.1 Régression logistique

Concernant le réseau de neurones de la régression logistique, un learning_rate de 0.1 a été choisi ainsi qu'un facteur de correction des probabilités epsilon égal à 10^{-6}. Ces choix sont assez courants pour les réseaux de neurones (le learning_rate est souvent plus petit, égal à environ 0.01 ou même 0.001, cependant de meilleurs résultats ont été obtenus avec un learning_rate plus grand).

4.2 RandomForest

Pour le modèle RandomForest, le nombre d'arbres utilisés est un hyper-paramètres qui influent énormément sur la performance du modèle. En effet, j'ai pu observer que les résultats obtenus avec le critère de performance (dans notre cas l'accuracy) s'amélioraient plus le nombre d'arbres augmentaient.
Ainsi, le choix a été fait de considérer un grand nombre d'arbres, égal à 1000.

4.3 MLP

Concernant le modèle MLP, programmé avec la librairie keras, la loss a été calculé avec l'entropie croisée, l'optimizer choisi est celui d'Adam (avec un learning_rate de 0.001 et un epsilon de 10^{-6}). Enfin, les différentes fonctions d'activation sont toutes des ReLU. Ces choix sont les plus utilisés pour les réseaux de neurones et notamment lorsqu'on manipule le jeu de données MNIST.
Un batch size égale à 32 a été choisie pour accélérer l'entraînement.
Concernant le nombre de couches, le choix s'est porté vers 4 couches cachées (cette décision est justifiée par de meilleurs résultats d'accuracy avec 4 couches). Au total, dans le modèle, 98.547 paramètres sont en apprentissage.

4.4 CNN

Concernant le modèle CNN, également programmé avec la librairie keras, les mêmes paramètres ont été choisis que pour le MLP (loss, optimizer) pour des raisons similaires.
A noter que le batch size choisi pour cette méthode a été de 128.
Concernant le nombre de couches, le choix s'est porté vers 2 couches de convolutions et 2 couches de MaxPooling. De plus, un drop out a été utilisé. Le drop out est une technique qui sert à éviter des co-adaptations complexes sur les données de l'échantillon d'entraînement. Au total, dans le modèle, 91.795 paramètres sont en apprentissage.

5 Résultats obtenus

Voici les courbes de loss et d'accuracy obtenues avec les modèles : Régression logistique, Random Forest, MLP et CNN :

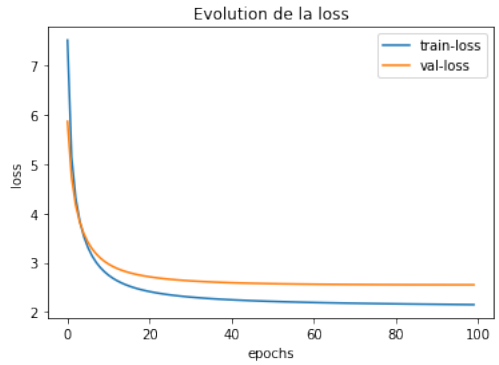

FIGURE 3 – Evolution de la loss pour la régression logistique

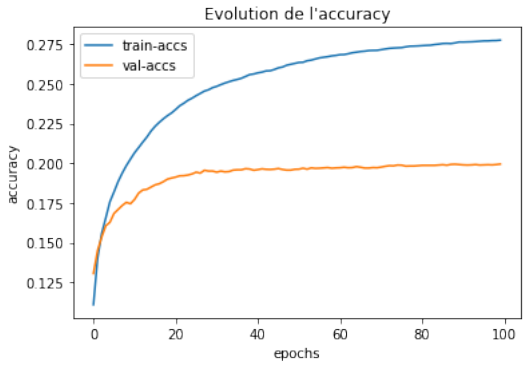

FIGURE 4 – Evolution de l'accuracy pour la régression logistique

L'accuracy obtenue avec le modèle Régression Logistique sur Kaggle a été très faible (0.18240). Le modèle Random Forest a quant à lui permit d'obtenir de meilleur résultats avec une accuracy de 0.72680.

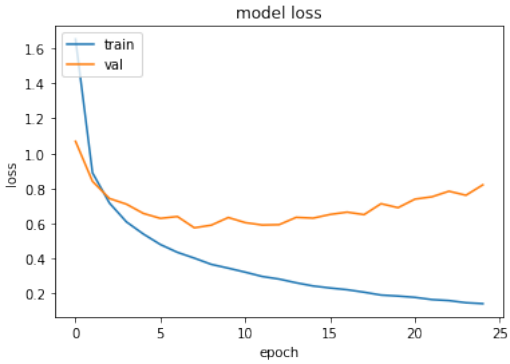

FIGURE 5 – Evolution de la loss pour le MLP

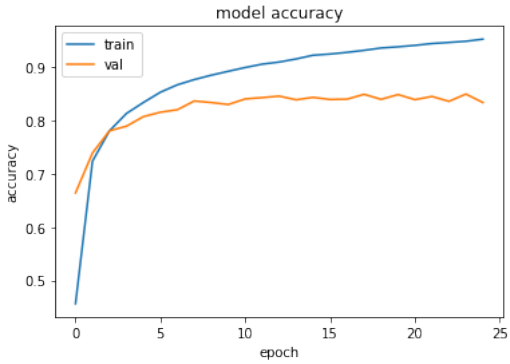

FIGURE 6 – Evolution de l'accuracy pour le MLP

L'accuracy obtenue avec le modèle MLP sur Kaggle a été de 0.84940. Ce sont les meilleurs résultats obtenus parmi les 4 méthodes testées.

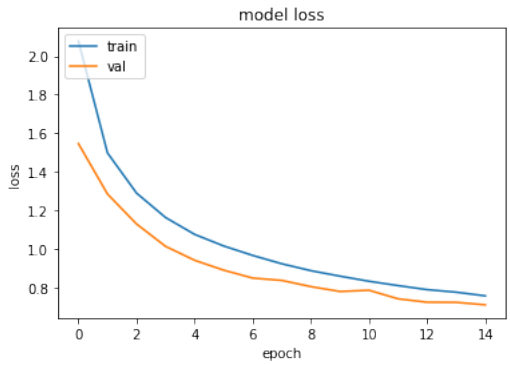

FIGURE 7 – Evolution de la loss pour le CNN

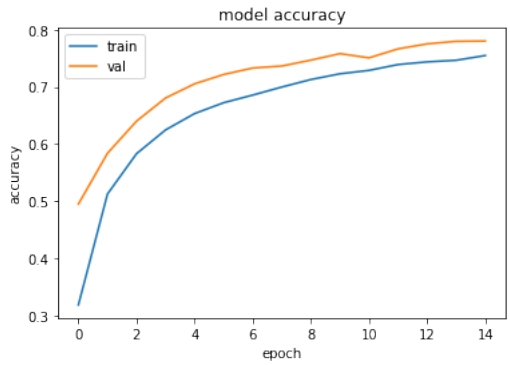

FIGURE 8 – Evolution de l'accuracy pour le CNN

L'accuracy obtenue avec le modèle CNN sur Kaggle a été de 0.78900.

Dans les lignes suivantes, nous comparerons les résultats obtenus lorsqu'on considère la fonction d'activation sigmoid pour le MLP, et un learning rate égale à 0.01 (et non à 0.001).

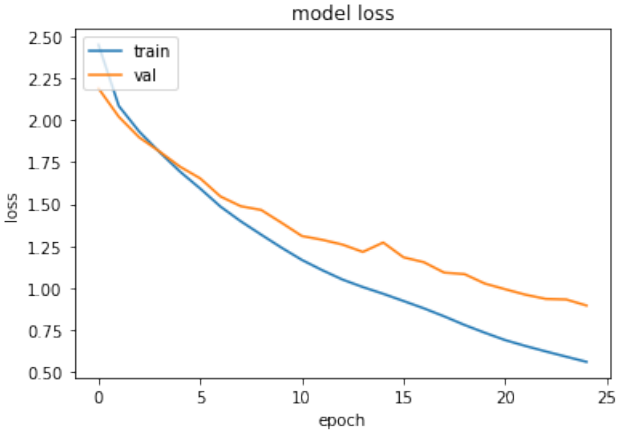

FIGURE 9 – Evolution de la loss pour le MLP (fonction d'activation sigmoid)

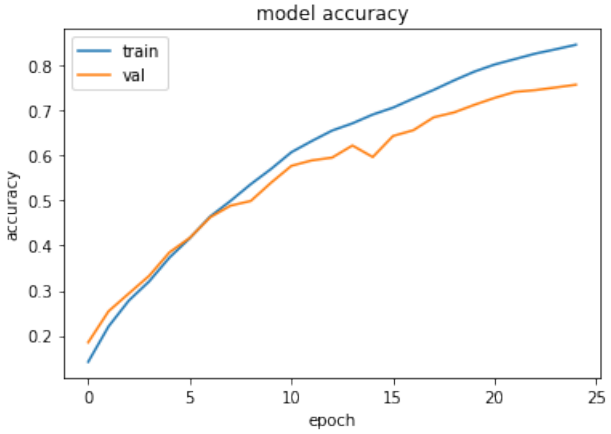

FIGURE 10 – Evolution de l'accuracy pour le MLP (fonction d'activation sigmoid)

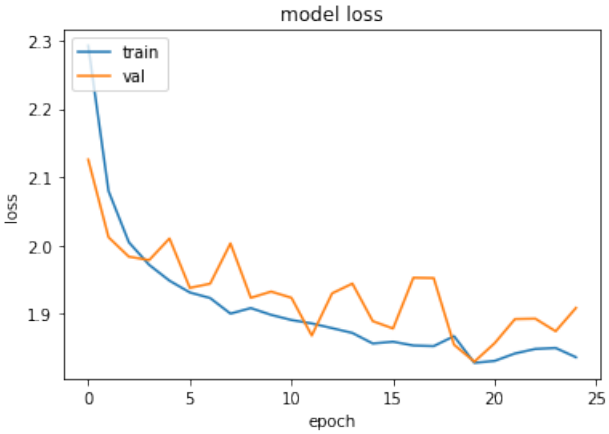

FIGURE 11 – Evolution de la loss pour le MLP (learning rate de 0.01)

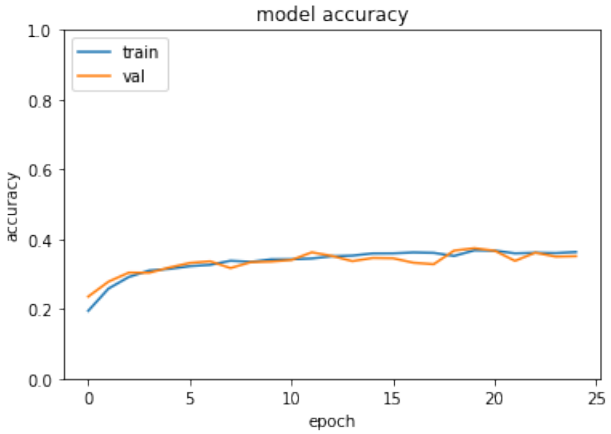

FIGURE 12 – Evolution de l'accuracy pour le MLP (learning rate de 0.01)

Les résultats obtenus ici sont beaucoup moins bons que les résultats obtenus avec la fonction d'activation ReLU et un learning rate de 0.001. Le choix de tels hyperparamètres a été très vite écarté.

6 Analyse, discussions et conclusion

Comme on a pu le voir précedemment, les meilleurs résultats ont pu être obtenus avec le MLP (avec fonction d'activation ReLU et learning rate de 0.001). Ma méthodologie a été de considérer des modèles de plus en plus complexes, en m'attendant à obtenir des résultats de plus en plus bons. La prochaine étape aurait été l'implémentation du réseau de neurones RestNet-18

pour observer les résultats obtenus avec un modèle à grande capacité.
Une idée d'amélioration serait peut-être de procéder à un travail préalable sur les données : comme par exemple ne considérer que les pixels "colorés" de l'image, formant le chiffre (les informations stockées dans les pixels composant la bordure ne servant, à mon sens, à pas grand chose).

7 Division des contribution

Cette compétition a été réalisée en solitaire.
Je déclare par la présente que tous les travaux préesentées dans ce rapport sont ceux de l'auteur.

Références

[1] The 1-Neuron Network : Logistic Regression
https://thedatafrog.com/en/articles/logistic-regression/

[2] Keras : The Sequential model
https://keras.io/guides/sequential_model/

[3] Understanding Random Forests Classifiers in Python Tutorial
https://www.datacamp.com/tutorial/random-forests-classifier-python

[4] How to choose CNN Architecture MNIST
https://www.kaggle.com/code/cdeotte/how-to-choose-cnn-architecture-mnist/notebook

[5] sklearn.ensemble.RandomForestClassifier
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html