

Titre professionnel

**Développeur en intelligence
artificielle**

RNCP 34757

Rapport E2 compétence C8 et C14

**Reconnaissance de chiffres
manuscrits**

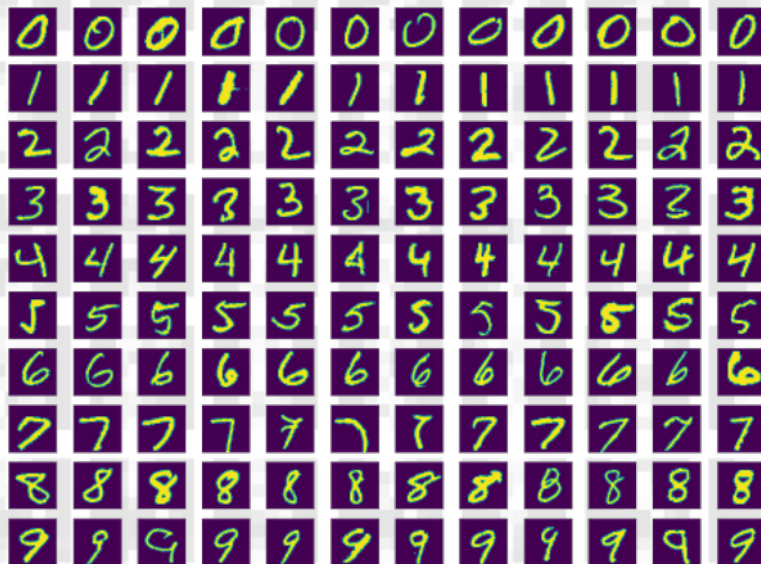


Table des matières

1	Présentation du projet existant.....	3
a	Préambule.....	3
b	Présentation du code.....	3
i	Présentation des données :.....	3
ii	Visualisation des images contenues dans la base de données.....	3
iii	Mise en œuvre de l'algorithme choisi.....	4
c	Conclusion.....	5
2	Amélioration du projet.....	6
a	Amélioration sur la qualité des images.....	6
i	Chargement de la base de données.....	6
ii	Visualisation d'un échantillon d'images.....	6
iii	Réduction du dataset.....	7
iv	Test de l'algorithme dans les mêmes conditions que sur le 1 ^{er} dataset.....	8
v	Amélioration de la régression linéaire.....	8
vi	GridSearchCV.....	10
vii	Learning curve, testons l'hypothèse 3.....	12
viii	Conclusion.....	13
b	Amélioration en testant de nouveaux algorithmes.....	13
i	Test du classifieur KNN.....	13
ii	Test avec un réseau de neurones.....	14
c	Conclusion de l'étude sur les 3 modèles proposés.....	15
3	Ajout de nouvelles fonctionnalités.....	15
a	Cahier des charges :.....	16
b	Test du programme final "lecture_cm.py" :.....	16
4	Test de non régression.....	17
5	Conclusion.....	17
6	Annexes	17
a	Jupyter notebook :.....	17
b	Programme python :.....	17

1 Présentation du projet existant.

a Préambule.

Il s'agit du projet d'étude numéro 5 de reconnaissance de chiffres manuscrits. La base de données utilisée est une base dérivée de la base MNIST (Modified National Institute of Standards and Technology) mais diffère par la dimension des images (8 x 8 pixels contre 28 x 28 pixels), par le codage des images (16 niveau de gris contre 256) ainsi que par la taille de la base de données (1797 échantillons contre 70 000).

La base de scikit-learn contient donc 1797 chiffres manuscrits de 0 à 9 avec environ 180 exemples par chiffre (représenté par une image de 8 x 8 pixels codée en 16 niveau de gris) et permet de tester facilement de nombreux algorithmes sans avoir à disposer d'une grande puissance de calcul.

La reconnaissance de l'écriture manuscrite est un problème difficile, ayant de multiples applications (on pense naturellement au déchiffrement des codes postaux à haute vitesse effectué dans les centres de tri).

La base MNIST est devenue une base de données de référence sur lequel bon nombre d'algorithmes ont été testés. Le site de Yann Lecun (<http://yann.lecun.com/exdb/mnist/>) donne un tableau récapitulatif par type d'algorithme et par algorithme des meilleurs résultats obtenus (le résultat le plus récent date de 2012 avec 0,23% d'erreur).

Selon le site Wikipédia (https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es_MNIST) un record a été établi en 2018 avec un taux d'erreur de 0,18 %.

Autant le dire tout de suite, s'il n'est pas si difficile que cela d'obtenir de bons résultats (au delà de 98%) le passage avec de "vraies" données peut réserver des surprises !

b Présentation du code.

Je ne donnerai pas ici tout le code du brief (l'intégralité du code se trouve en annexe) mais simplement les parties en rapport avec la classification multi-classe. L'exercice de classification mono-classe ainsi que l'exercice de codage d'une descente de gradient (sans utilisation de la librairie scikit-learn) ne feront donc pas partie de ce rapport.

i Présentation des données :

```
1 # Importation des données depuis scikit learn datasets
2 digits = load_digits()
3
4 # Les images sont organisées en matrice de 8x8
5 X = digits.images
6 # Labels des images (chiffre de 0 à 9)
7 Y = digits.target
8 # Les images sont "aplaties" et transformées en un tableau de 64 éléments
9 Xdata = digits.data
10
11 print(f"X Shape: {X.shape}, Xdata Shape: {Xdata.shape}, Y shape: {Y.shape}, "+
12       f"digits.target_name: {digits.target_names}\n")
```

X Shape: (1797, 8, 8), Xdata Shape: (1797, 64), Y shape: (1797,), digits.target_name: [0 1 2 3 4 5 6 7 8 9]

ii Visualisation des images contenues dans la base de données.

Ce bout de code permet d'afficher certaines images contenues dans la base de données choisie de façon aléatoire mais ordonnée par classe (de 0 à 9).

```

1 # Préparation de l'affichage
2 n_digits = np.unique(Y) # = 0,1,2,...9
3 M = 12
4
5 fig, axs = plt.subplots(len(n_digits), M, figsize=(20, 15))
6
7 # Afficher M exemples de tout les digits (de 0 à 9)
8 for i, d in enumerate(n_digits):
9     x = X[Y == d]
10    for j in range(M):
11        num = random.randint(0, x.shape[0]-1)
12        axs[i,j].imshow(X[Y == d][j], cmap="gray")
13        axs[i,j].axis('off')

```



Comme on peut le voir, la qualité des images est plutôt médiocre. Ce sera d'ailleurs un des axes d'amélioration du programme.

iii Mise en œuvre de l'algorithme choisi.

Nous utiliserons la régression logistique de scikit learn pour réaliser la tâche de classification des chiffres. On notera que nous n'avons pas cherché à optimiser le résultat obtenu (nous avons réalisé cet exercice au début de la formation, il s'agissait alors d'apprendre les différentes notions qui seront approfondies plus tard). Par ailleurs, bon nombre de tutos se contentent très souvent des paramètres par défaut des algorithmes qui sont, il faut le noter, plutôt pertinents. Mais il est clair que chercher à optimiser ces paramètres apporte souvent des gains appréciables tant au niveau de la précision, que de la qualité des prédictions (recall). Ce sera l'objet d'un autre axe d'amélioration.

Dans un premier temps, nous partageons les données en un jeu d'entraînement (80%) et un jeu de test (20%), puis nous construisons le modèle, enfin nous testons les prédictions réalisées. Un certain nombre d'erreurs apparaissent.

```

1 # On split le jeux de données
2 X_train, X_test, Y_train, Y_test = train_test_split(Xdata, Y, test_size=0.2, random_state= 511)
3
4 # On construit et on 'fit' le modèle
5 log_regr = LogisticRegression(solver='liblinear')
6 log_regr.fit(X_train, Y_train)
7
8 # on teste sur les données de.... test :)
9 prediction = log_regr.predict(X_test)
10
11 err, pre, d= erreur_prediction(valeurs_predites= prediction, valeurs_reelle= Y_test)
12 print(f"{err} erreurs sur {len(X_test)} images. soit une précision de {pre:.2f} %")
13 print(f"Dictionnaire des erreurs (clé valeur réelle: valeur prédite):\n{d}")
14

```

```

19 erreurs sur 360 images. soit une précision de 94.72 %
Dictionnaire des erreurs (clé valeur réelle: valeur prédite):
{8: [1, 1, 1, 3, 1], 3: [5, 8], 9: [8, 3, 4, 8], 5: [9, 9, 9, 9], 4: [7, 6], 1: [6, 8]}

```

Pour mieux quantifier ces erreurs, nous affichons la précision, la sensibilité (recall) ainsi que le f1 score (qui combine la précision et le recall en une sorte de moyenne pondérée) :

```

1 target_names= ["détection de 0", "détection de 1", "détection de 2", "détection de 3", "détection de 4",
2               "détection de 5", "détection de 6", "détection de 7", "détection de 8", "détection de 9"]
3 print("\n",classification_report(Y_test,prediction, target_names=target_names))
4

```

	precision	recall	f1-score	support
détection de 0	1.00	1.00	1.00	31
détection de 1	0.89	0.94	0.91	33
détection de 2	1.00	1.00	1.00	35
détection de 3	0.94	0.94	0.94	36
détection de 4	0.97	0.95	0.96	41
détection de 5	0.98	0.91	0.94	45
détection de 6	0.95	1.00	0.97	35
détection de 7	0.97	1.00	0.98	31
détection de 8	0.90	0.88	0.89	40
détection de 9	0.88	0.88	0.88	33
accuracy			0.95	360
macro avg	0.95	0.95	0.95	360
weighted avg	0.95	0.95	0.95	360

c Conclusion.

Le score obtenu est plutôt flatteur compte tenu du faible investissement dans la recherche d'optimisation. Mais quelques problèmes incitent à rester méfiant sur ce résultat.

D'abord la faible qualité des images n'augure pas de bons résultats sur des chiffres manuscrits qui ne proviendraient pas de cette base. Ensuite un simple changement de la graine initialisant une suite pseudo aléatoire dans la fonction `train_test_split` fait varier les résultats de façon importante (la précision augmente de 2 % passant à 96,5% avec `random_state= 42` ... le plus célèbre des nombres au lieu de 511). Enfin "lbfgs" qui est l'algorithme d'optimisation par défaut produit un avertissement indiquant qu'il n'a pas pu converger suffisamment (d'où son remplacement par "liblinear"). Pour que ce programme puisse fonctionner de façon correcte dans la « vraie vie » il faut donc l'améliorer quelque peu !

2 Amélioration du projet

Des pistes pour améliorer le projet sont l'amélioration de la qualité de l'image, l'amélioration de l'algorithme utilisé avec optimisation des hyper-paramètres et recherche d'autres algorithmes qui pourraient être utilisés

a Amélioration sur la qualité des images.

Dans un premier temps nous pourrions travailler sur le dataset original avec des images de meilleures qualités. Ce qui nous permettra d'avoir un algorithme utilisable dans des conditions réelles.

i Chargement de la base de données.

Cette base peut être chargée depuis la librairie h5py, mais j'aurais pu tout autant télécharger directement les données depuis un site de dataset.

```
1 # Importation du jeux d'entraînement
2 f = h5py.File("train.hdf5", 'r')
3 train_x, train_y = f['image'][...], f['label'][...]
4 f.close()
5
6 # Importation du jeux de test
7 f = h5py.File("test.hdf5", 'r')
8 test_x, test_y = f['image'][...], f['label'][...]
9 f.close()
10
11 etiquette= np.unique(test_y)
12 print(f"Entrainement: X Shape: {train_x.shape}, Y shape: {train_y.shape}\n"+\
13       f"Test: X Shape: {test_x.shape}, Y shape: {test_y.shape}\n"+\
14       f"Etiquette: {etiquette}")
```

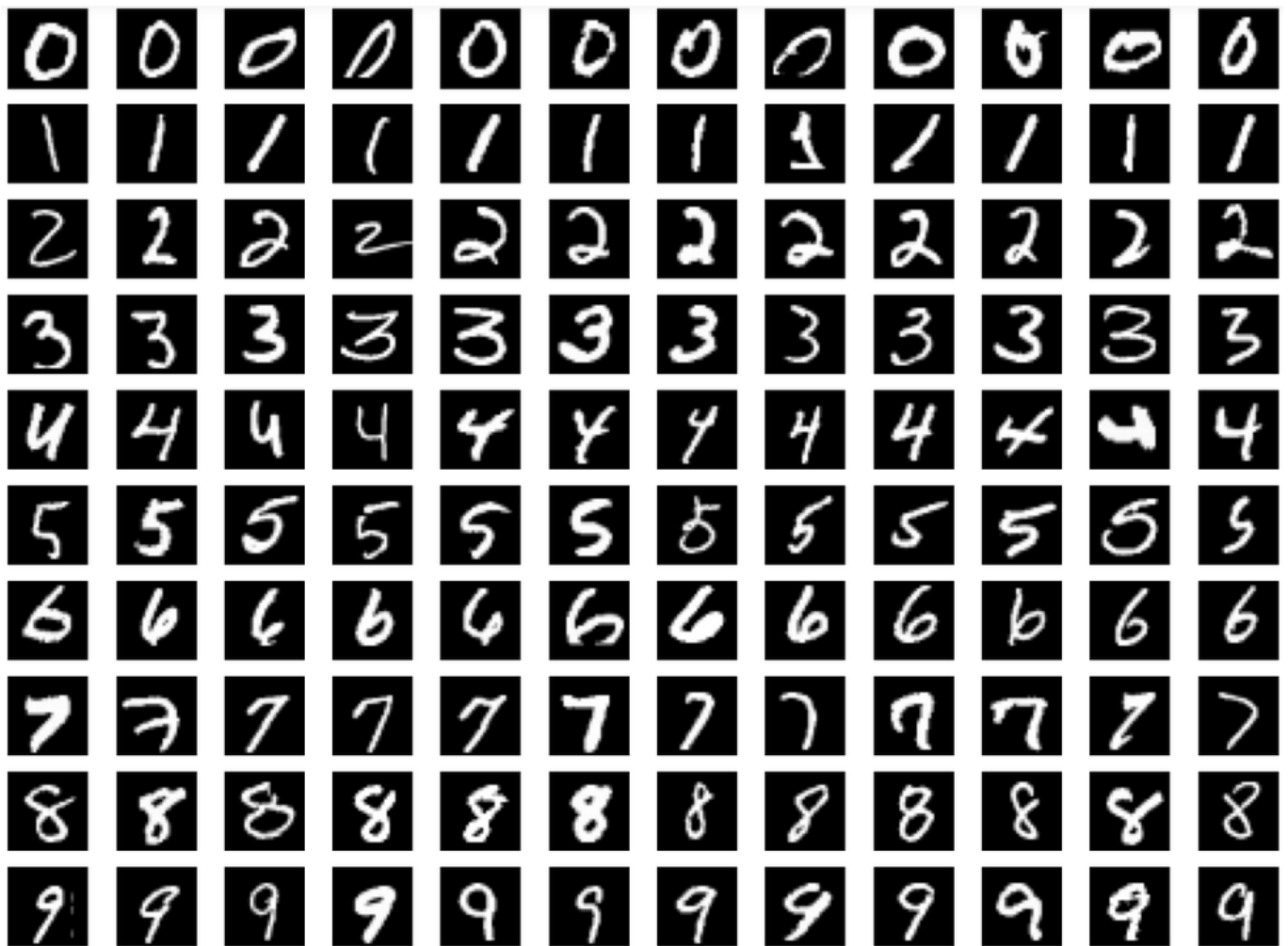
```
Entrainement: X Shape: (60000, 28, 28), Y shape: (60000,)
Test: X Shape: (10000, 28, 28), Y shape: (10000,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]
```

Comme on peut le voir, cette base comprend 70000 données. Compte tenu des limitations de l'ordinateur d'une part, et le fait que nous n'aurons pas besoin de toutes ces données pour réaliser les tests sur les divers algorithmes d'autre part, je ne prendrais qu'une partie de ces données.

ii Visualisation d'un échantillon d'images.

```
1 # Préparation de l'affichage
2 M= 12
3 fig, axs = plt.subplots(len(etiquette), M, figsize=(20, 15))
4
5 # Afficher M exemples de tout les digits (de 0 à 9)
6 for i, d in enumerate(etiquette):
7     x= train_x[train_y == d]
8     for j in range(M):
9         num= random.randint(0, x.shape[0]-1)
10        axs[i,j].imshow(x[num], cmap= "gray")
11        axs[i,j].axis('off')
12
```

Comme on peut le constater sur la figure ci-dessous, les images sont de bien meilleure qualité. On peut aussi constater une grande variété dans l'écriture ce qui signifie que pour livrer un produit efficace, il faudra sans doute intégrer plus de données, peut-être même la totalité du dataset. L'étude de la courbe d'apprentissage (fonction learning curve de scikit learn) pourra être utile. Mais pour l'instant il s'agit de tester l'algorithme de régression linéaire utilisé précédemment.



iii Réduction du dataset.

Le précédent dataset sur lequel j'ai travaillé avait 1797 images. On va en construire un plus grand sans pour autant prendre la totalité du dataset MNIST. En effet, une optimisation des hyperparamètres par grid searchCV en prenant la totalité du dataset nécessiterait une puissance machine trop importante.

```

1 # On récupère les données. On ne prendra qu'une partie des données situé dans le
2 # jeux de train. On en profite pour "aplatir" les images
3 data= train_x.reshape(train_x.shape[0], train_x.shape[1]*train_x.shape[2])
4
5 # On réduit le jeux de données original
6 data_red, _, Y_red, _ = train_test_split(data, train_y, train_size=0.1, random_state= 65)
7
8 # On split le jeux de données
9 X_train_red, X_test_red, Y_train_red, Y_test_red = train_test_split(data_red, Y_red, train_size=0.8,
10                                                                    random_state= 65)
11
12 print(f"Entrainement: X Shape: {X_train_red.shape}, Y shape: {Y_train_red.shape}\n"+\
13       f"Test: X Shape: {X_test_red.shape}, Y shape: {Y_test_red.shape}\n"+\
14       f"Etiquette: {etiquette}")

```

```

Entrainement: X Shape: (4800, 784), Y shape: (4800,)
Test: X Shape: (1200, 784), Y shape: (1200,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]

```

On regarde si ce dataset est correctement réparti.


```

1 # Le jeux de donnée est-il équilibré ?
2 d= {}
3 for i in etiquette:
4     l= len(X_train_red[Y_train_red== i])
5     d[i]= l
6     print(f"Le chiffre {i} est représenté {l} fois", end= " - ")
7 mini, maxi= min(d, key= d.get), max(d, key= d.get)
8 print(f"\nLe chiffre le moins représenté est {mini} représenté {d[mini]} fois.")
9 print(f"Le chiffre le plus représenté est {maxi} représenté {d[maxi]} fois.")

```

Le chiffre 0 est représenté 483 fois - Le chiffre 1 est représenté 546 fois - Le chiffre 2 est représenté 472 fois -
 Le chiffre 3 est représenté 483 fois - Le chiffre 4 est représenté 473 fois - Le chiffre 5 est représenté 431 fois -
 Le chiffre 6 est représenté 482 fois - Le chiffre 7 est représenté 480 fois - Le chiffre 8 est représenté 487 fois -
 Le chiffre 9 est représenté 463 fois -
 Le chiffre le moins représenté est 5 représenté 431 fois.
 Le chiffre le plus représenté est 1 représenté 546 fois.

On peut constater une différence significative entre le chiffre le plus représenté et celui étant le moins représenté. Un paramètre de la fonction de régression logistique permet de tenir compte de ce problème (class_weight) mais ne sera pas utilisé ici pour pouvoir comparer les résultats obtenus avec les deux différents datasets.

iv Test de l'algorithme dans les mêmes conditions que sur le 1^{er} dataset.

```

1 deb= time.time()
2 # On construit et on 'fit' le modèle
3 model_reglog = LogisticRegression(solver='liblinear')
4 model_reglog.fit(X_train_red, Y_train_red)
5 fin= time.time()
6 print(f"temps d'exécution: {fin-deb:.2f} s")

```

temps d'exécution: 144.15 s

/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
 warnings.warn(

Comme on peut le voir, l'algorithme a eu du mal à converger ! Les résultats ne sont d'ailleurs pas très bons.

```

# on teste sur les données de... test :)
prediction = model_reglog.predict(X_test_red)
erreur= 0
longueur= len(Y_test_red)
for i in range(longueur):
    if Y_test_red[i]!= prediction[i]: erreur+= 1
print(f"{erreur} erreurs sur {longueur} images soit une précision de {100*(1-erreur/longueur):.2f} %")

```

225 erreurs sur 1200 images soit une précision de 81.25 %

Pour améliorer les résultats, nous allons prendre un "solver" plus approprié, mais aussi normaliser les données. En effet, beaucoup d'algorithmes ont de bien meilleurs résultats avec des données qui ont été normalisées (même si celles-ci sont cohérentes entre elles puisqu'il s'agit de nombres entiers compris entre 0 et 255) avec l'une des méthodes proposées par scikit learn.

v Amélioration de la régression linéaire.

La première chose que l'on constate à la lecture de la documentation est que le solver "liblinear" n'est peut être pas le mieux adapté. On en testera donc un autre. D'autre part, la documentation de scikit learn donne une autre piste pour améliorer les résultats : La normalisation des données. Plusieurs méthodes existent (standardscaler, minmaxscaler, robustscaler pour ne citer que les plus connues).


```

1 # On récupère les données. On ne prendra qu'une partie des données situé dans le
2 # jeux de train. On en profite pour "aplatir" les images
3 data= train_x.reshape(train_x.shape[0], train_x.shape[1]*train_x.shape[2])
4
5 # On réduit le jeux de données original
6 data_red, _, Y_red, _ = train_test_split(data, train_y, train_size=0.1, random_state= 65)
7
8 # Standardisation
9 scaler=StandardScaler()
10 datastd= scaler.fit_transform(data_red)
11
12 # On split le jeux de données
13 X_train_red, X_test_red, Y_train_red, Y_test_red = train_test_split(datastd, Y_red, train_size=0.8,
14 random_state= 65)
15
16 print(f"Entrainement: X Shape: {X_train_red.shape}, Y shape: {Y_train_red.shape}\n"+\
17       f"Test: X Shape: {X_test_red.shape}, Y shape: {Y_test_red.shape}\n"+\
18       f"Etiquette: {etiquette}")

```

```

Entrainement: X Shape: (4800, 784), Y shape: (4800,)
Test: X Shape: (1200, 784), Y shape: (1200,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]

```

Ensuite le choix d'un autre solver ; il faudra aussi tenir compte du fait que les classes ne contiennent pas le même nombre d'images.

```

1 deb= time.time()
2 # On construit et on 'fit' le modèle
3 model_reglog = LogisticRegression(solver='sag', class_weight= "balanced", max_iter= 2000, verbose= True)
4 model_reglog.fit(X_train_red, Y_train_red)
5 fin= time.time()
6 print(f"temps d'exécution: {fin-deb:.2f} s")

```

```

Epoch 1873, change: 0.00012927
Epoch 1874, change: 0.00012904
Epoch 1875, change: 0.00012909
Epoch 1876, change: 0.00012907
Epoch 1877, change: 0.00012884
Epoch 1878, change: 0.00012878
Epoch 1879, change: 0.00012875
Epoch 1880, change: 0.00012867
Epoch 1881, change: 0.00012849
Epoch 1882, change: 0.00012828
Epoch 1883, change: 0.00012828
Epoch 1884, change: 0.00012830
Epoch 1885, change: 0.00012804
Epoch 1886, chanmax_iter reached after 1366 seconds
temps d'exécution: 1366.36 s

```

```

/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter
er was reached which means the coef_ did not converge
  warnings.warn(
[Parallel(n_jobs=1)]: Done    1 out of    1 | elapsed: 22.8min finished

```

La valeur par défaut du nombre en dessous duquel l'algorithme atteint son seuil de convergence est de $1e-4$. Comme on peut le voir ici, nous avons presque atteint ce chiffre.

La précision fait un bond de presque dix points ce qui est un bon résultat.

```

1 # on teste sur les données de... test :)
2 prediction = model_reglog.predict(X_test_red)
3 erreur= 0
4 longueur= len(Y_test_red)
5 for i in range(longueur):
6     if Y_test_red[i] != prediction[i]: erreur+= 1
7 print(f"{erreur} erreurs sur {longueur} images soit une précision de {100*(1-erreur/longueur):.2f} %")

```

```

125 erreurs sur 1200 images soit une précision de 89.58 %

```

```

1 target_names= ["détection de 0", "détection de 1", "détection de 2", "détection de 3", "détection de 4",
2               "détection de 5", "détection de 6", "détection de 7", "détection de 8", "détection de 9"]
3 print(f"Le score obtenu est de {model_reglog.score(X_test_red, Y_test_red)*100:.2f}")
4 print("\n",classification_report(Y_test_red,prediction, target_names= target_names))
5

```

Le score obtenu est de 89.58

	precision	recall	f1-score	support
détection de 0	0.91	0.94	0.93	126
détection de 1	0.90	0.99	0.94	117
détection de 2	0.92	0.80	0.85	137
détection de 3	0.84	0.88	0.86	112
détection de 4	0.90	0.91	0.90	123
détection de 5	0.88	0.77	0.82	103
détection de 6	0.94	0.97	0.96	135
détection de 7	0.92	0.91	0.92	119
détection de 8	0.84	0.85	0.85	110
détection de 9	0.89	0.92	0.90	118
accuracy			0.90	1200
macro avg	0.89	0.89	0.89	1200
weighted avg	0.90	0.90	0.89	1200

Pour espérer aller plus loin et améliorer le résultat un GridSearchCV sera effectué. On essaiera de trouver le meilleur algorithme de standardisation parmi les 3 principaux. On testera chacun des solvers composant l'algorithme de régression linéaire pour déterminer la meilleure des solutions.

vi GridSearchCV

On va construire un pipeline avec une première partie standardisation (c'est dans cette partie là que seront testés les 3 algorithmes de standardisation) et la seconde partie dans laquelle seul l'algorithme de régression logistique sera testé mais avec tous les solvers disponibles (à l'exception de liblinear qui a montré ses limites). Nous fixerons "class_weight" à "balanced", et laisserons les autres paramètres à leurs valeurs par défauts. C'est un compromis entre temps d'exécution et exhaustivité ! Et puis lorsque le meilleur "solver" sera trouvé on pourra toujours chercher à optimiser les autres paramètres même si cela ne garantit pas, qu'au final, la meilleure des solutions soit trouvée.

```

# On réalise un pipe avec 2 actions: un scaler et un algorithme
pipe = Pipeline(steps= [('scaler', StandardScaler()), ('algo', LogisticRegression())])

# On testera 3 standardisations et pour la régression logistique on testera tout les solvers
param_grid = {'pipeline__scaler': [StandardScaler(), MinMaxScaler(), RobustScaler()],
              'pipeline__algo__solver': ["sag", "newton-cg", "lbfgs", "saga"],
              'pipeline__algo__class_weight': ["balanced"],
              'pipeline__algo__max_iter': [2000]}
# instantiate and run as before:

model= make_pipeline(pipe)

grid = GridSearchCV(model, param_grid, cv=5, verbose= True, n_jobs= -1)

grid

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('pipeline',
                                       Pipeline(steps=[('scaler',
                                                         StandardScaler()),
                                                         ('algo',
                                                         LogisticRegression()))])),
             n_jobs=-1,
             param_grid={'pipeline__algo__class_weight': ['balanced'],
                         'pipeline__algo__max_iter': [2000],
                         'pipeline__algo__solver': ['sag', 'newton-cg', 'lbfgs',
                                                    'saga'],
                         'pipeline__scaler': [StandardScaler(), MinMaxScaler(),
                                              RobustScaler()]},
             verbose=True)

```

Et, après 7heures 8 minutes et 37 secondes le meilleur choix standardisation/solver est :

```

deb= time.time()

grid.fit(X_train_red, Y_train_red)

fin= time.time()
print(f"temps d'exécution: {fin-deb:.2f} s")
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_it
er was reached which means the coef_ did not converge
warnings.warn(
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_it
er was reached which means the coef_ did not converge
warnings.warn(
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_it
er was reached which means the coef_ did not converge
warnings.warn(
temps d'exécution: 25717.32 s

```

```

grid.best_estimator_

Pipeline(steps=[('pipeline',
                 Pipeline(steps=[('scaler', MinMaxScaler()),
                                ('algo',
                                  LogisticRegression(class_weight='balanced',
                                                         max_iter=2000,
                                                         solver='newton-cg')))]))]

```

Le solver est "newton-cg" avec un MinMaxScaler pour la standardisation des données. Le gain de précision obtenu est décevant.

```

print(f"Précision obtenue: {100*grid.best_score_: .2f} %")

Précision obtenue: 88.40 %

```

Voyons ce que cela donne sur le jeu de test :

```

# on teste sur les données de... test :)
prediction = mod_prov.predict(X_test_red)

err, pre, d= erreur_prediction(valeurs_predites= prediction, valeurs_reelle= Y_test_red)
print(f"{err} erreurs sur {len(X_test_red)} images soit une précision de {pre:.2f} %")
print(f"Dictionnaire des erreurs (clé valeur réelle: valeur prédite):\n{d}")

111 erreurs sur 1200 images soit une précision de 90.75 %
Dictionnaire des erreurs (clé valeur réelle: valeur prédite):
{8: [3, 5, 5, 5, 3, 2, 5, 1, 5, 3, 1, 3, 1, 9], 2: [7, 8, 0, 7, 1, 3, 4, 7, 3, 1, 0, 1, 7, 6, 7, 6, 7, 3, 6, 8, 8,
8, 3], 5: [3, 3, 8, 3, 9, 4, 3, 0, 9, 3, 3, 9, 8, 0, 8, 8], 7: [5, 4, 3, 9, 9, 9, 9, 2, 5, 9, 9], 3: [8, 2, 5, 2, 0,
9, 0, 5, 5, 8, 7, 2, 8, 8], 6: [8, 5, 7, 5, 5], 4: [3, 2, 9, 8, 1, 9, 9, 8, 8, 8, 0], 0: [6, 5, 4], 9: [1, 4, 4, 8,
4, 7, 3, 3, 7, 8, 4, 7], 1: [3, 3]}

target_names= ["détection de 0", "détection de 1", "détection de 2", "détection de 3", "détection de 4",
               "détection de 5", "détection de 6", "détection de 7", "détection de 8", "détection de 9"]
print("\n",classification_report(Y_test_red,prediction, target_names= target_names))

```

	precision	recall	f1-score	support
détection de 0	0.95	0.98	0.96	126
détection de 1	0.93	0.98	0.96	117
détection de 2	0.95	0.83	0.89	137
détection de 3	0.83	0.88	0.85	112
détection de 4	0.93	0.91	0.92	123
détection de 5	0.86	0.84	0.85	103
détection de 6	0.97	0.96	0.97	135
détection de 7	0.91	0.91	0.91	119
détection de 8	0.83	0.87	0.85	110
détection de 9	0.88	0.90	0.89	118
accuracy			0.91	1200
macro avg	0.91	0.91	0.91	1200
weighted avg	0.91	0.91	0.91	1200

Comme attendu le gain est modeste 90,75 % contre 89,58 % sur le jeu de test !

Je vois plusieurs explications possibles à cela.

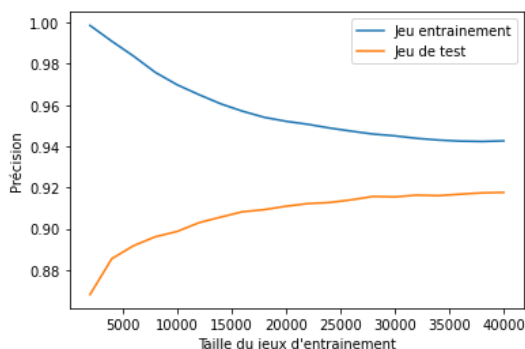
- La première c'est qu'avec GridSearchCV on ne se contente pas d'un seul calcul mais de 5 (paramètre "cv") et une moyenne des résultats est ensuite effectuée. Comme nous l'avons vu dans les conclusions de la première étude avec le dataset fourni par scikit learn, les résultats peuvent varier de façon significative lorsque l'on change d'échantillon (une différence de 2 % avait été constatée).
- La seconde est que l'algorithme atteint ses limites. Quelque soit les valeurs données nous n'obtiendrons jamais de meilleurs résultats. Il faudra alors se tourner vers d'autres algorithmes.
- La troisième hypothèse c'est que nous n'avons pas pris assez de données d'entraînement. La fonction learning curve de scikit learn peut permettre de vérifier (ou non) cette hypothèse.

vii Learning curve, testons l'hypothèse 3.

Pour tester cette hypothèse on prend le meilleur algorithme que l'on entraîne sur la totalité du jeu de train. De même la totalité du jeu de test sera utilisée.

```
train_size, train_score, val_score = learning_curve(pipe_minmax_logreg, data_train, train_y,
                                                    train_sizes = np.linspace(0.05, 1, 20), cv = 3)
```

```
plt.plot(train_size, train_score.mean(axis=1), label = "Jeu entraînement")
plt.plot(train_size, val_score.mean(axis=1), label = "Jeu de test")
plt.xlabel("Taille du jeux d'entraînement")
plt.ylabel("Précision")
_ = plt.legend()
```



Comme on le voit, la précision sur le jeu de test a une allure de faux plat montant que détestent bon nombre de cyclistes ! La réduction du jeu de données a permis d'accélérer les calculs pour la recherche du meilleur algorithme sans que cela ne nuise au résultat.

viii Conclusion.

Il est probable que la régression logistique ait atteint un plafond que l'on peut estimer à 2 ou 3 points au dessus du niveau atteint. Il serait donc judicieux de se tourner vers d'autres algorithmes si l'on souhaite améliorer encore les résultats. En effet, il ne faut pas oublier que même les algorithmes avec un résultat flatteur de plus de 99 % peuvent décevoir sur des cas concrets. Améliorer la précision serait bienvenue ! Parmi tous les classificateurs, le KNN semble être un bon candidat. Mais on peut aussi décider de se tourner vers une autre technologie : Les réseaux de neurones. Un type particulièrement prisé dans la classification d'images est le réseau de neurone convolutif !

Dernier point, et pour pouvoir comparer les différentes méthodes, un modèle entraîné sur la totalité du jeu d'entraînement de MNIST et validé sur la totalité du jeu de test a été enregistré dans le répertoire "modeles". Les données présentées à cet algorithme doivent être des tableaux numpy à 1 dimension (utilisation de ravel) avec des nombres flottants compris entre 0 et 1.

b Amélioration en testant de nouveaux algorithmes.

Référentiel : C8. Modifier les paramètres et composants de l'intelligence artificielle afin d'ajuster aux objectifs du projet les capacités fonctionnelles de l'algorithme à l'aide de techniques d'optimisation.

i Test du classifieur KNN.

Après quelques tests d'optimisation avec la fonction GridSearchCV et en gardant la standardisation avec le MinMaxScaler le classifieur KNN montre une rapidité plus grande et une meilleure précision.

```
# On choisi les meilleurs paramètres en gardant la standardisation min/max
pipe_minmax_knn= make_pipeline(MinMaxScaler(), KNeighborsClassifier(n_neighbors= 6, weights= "distance",
                                                                    algorithm='ball_tree', p= 3))

pipe_minmax_knn.fit(data_train, train_y)
prediction = pipe_minmax_knn.predict(data_test)
```

Et le résultat est très encourageant !

```
err, pre, d= erreur_prediction(valeurs_predites= prediction, valeurs_reelle= test_y)
print(f"{err} erreurs sur {len(data_test)} images soit une précision de {pre:.2f} %")
print(f"Dictionnaire des erreurs (clé valeur réelle: valeur prédite):\n{d}")

263 erreurs sur 10000 images soit une précision de 97.37 %
Dictionnaire des erreurs (clé valeur réelle: valeur prédite):
{4: [0, 9, 6, 1, 9, 9, 9, 6, 9, 1, 9, 8, 9, 9, 9, 6, 6, 9, 9, 9, 1, 7, 1, 9, 7, 1, 1, 9, 9], 3: [5, 7, 7, 5, 5, 9,
7, 4, 5, 2, 5, 5, 5, 7, 7, 5, 7, 5, 5, 8, 2, 7, 6, 8, 9, 5, 9, 5, 9, 5, 5, 9, 8], 9: [8, 7, 8, 4, 0, 3, 5, 3, 4, 2,
6, 1, 4, 7, 0, 7, 1, 7, 4, 1, 1, 3, 3, 2, 7, 7, 2, 4, 1, 4, 5, 3, 5, 0, 0, 0, 7, 7, 4], 2: [7, 8, 7, 6, 7, 7, 0, 7,
0, 7, 0, 7, 1, 0, 7, 6, 1, 1, 3, 7, 7, 3, 7, 7, 7, 0, 7, 0, 1, 7, 0, 0, 0, 8, 7, 8, 0], 6: [4, 0, 5, 0, 1, 4, 5, 1,
0, 4, 0, 1, 0], 7: [4, 1, 4, 2, 1, 1, 1, 9, 9, 1, 1, 1, 1, 9, 1, 1, 1, 1, 9, 9, 9, 9, 1, 1, 1, 9, 9, 9, 9, 1, 1, 2,
2, 2], 8: [0, 1, 3, 3, 3, 9, 4, 3, 3, 7, 4, 5, 3, 2, 4, 2, 3, 3, 6, 6, 3, 5, 0, 0, 5, 2, 7, 9, 3, 7, 6, 0, 6, 2, 4,
9, 9, 7, 9, 4, 5, 5], 5: [9, 4, 3, 6, 9, 7, 3, 6, 8, 8, 3, 9, 9, 0, 8, 4, 6, 3, 6, 9, 6, 6, 0, 6, 0, 6], 1: [2, 2,
6], 0: [6, 2, 7, 5, 6, 1, 6]}
```

Avec la régression logistique optimisée le nombre d'erreurs était de 737 pour 10 000, avec le classifieur KNN optimisé le nombre d'erreurs passe à 263 ce qui est un excellent résultat. La précision grimpe à 97,37 %.

Il est clair que le classifieur est un excellent choix pour ce type de données et pour ce dataset en particulier. Cependant, dans la classification d'image les réseaux de neurones, et particulièrement les CNN semblent faits pour ce type de données.

ii Test avec un réseau de neurones.

La création d'un réseau de neurones de type CNN n'est pas, à proprement parler, difficile. Ce qui est plus compliqué c'est d'en faire un suffisamment "léger" pour qu'il puisse être entraîné facilement tout en étant efficace.

La phase d'apprentissage est une étape clef. Comme pour tous les algorithmes de machine learning, il faut s'assurer aussi qu'il ne soit pas sur-entraîné, perdant ainsi sa capacité de généralisation. On tracera les courbes d'apprentissage en regardant les courbes de perte et de précision sur le jeu d'entraînement, comme sur le jeu de test pour s'assurer que l'on ne soit pas en "overfitting". Le sur-apprentissage peut en effet se repérer quand l'évolution des courbes de perte et de précision sur les 2 jeux divergent.

Plusieurs réseaux ont été créés et leurs performances comparées sur des chiffres manuscrits qui ne sont pas issus de MNIST.

Sans présenter ici tout les réseaux entraînés (ils se trouvent dans le Jupyter notebook en annexe de ce document), voici les étapes suivies :

Conception du réseau :

```
batch_size = 128
epochs = 250

input_shape= (train_x.shape[1], train_x.shape[2], 1)
nb_classe= len(etiquette)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape= input_shape ))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classe, activation='softmax'))
model.compile(loss= keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.summary()
```

Entraînement et évaluation du réseau :

```
hist = model.fit(data_train_pour_cnn, train_y_conv, batch_size=batch_size, epochs=epochs,
                verbose=1, validation_data=(data_test_pour_cnn, test_y_conv), workers= -1)
```

```
score = model.evaluate(data_test_pour_cnn, test_y_conv, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test loss: 0.05704181641340256
Test accuracy: 0.9825000166893005

Courbes d'apprentissage :

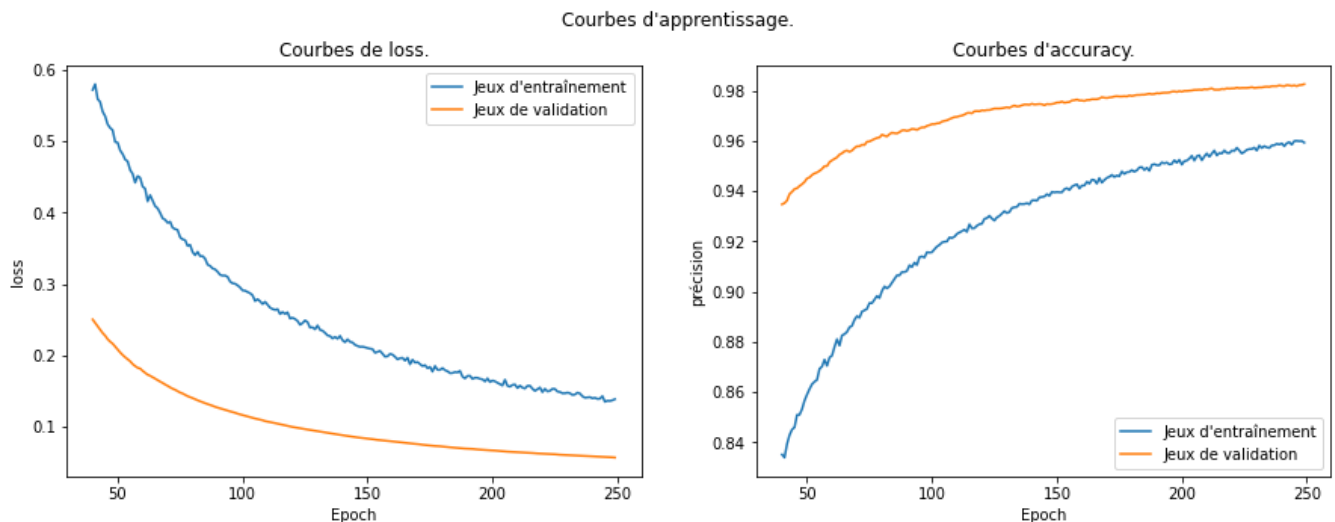
En "zoomant" sur la partie finale pour s'assurer qu'il n'y a pas de phénomène de "sur-apprentissage" qui se révèle en général quand les évolutions des courbes (du jeu d'entraînement et du jeu de validation) divergent.


```
x= range(len(hist.history["loss"]))
debut= 40

fig, axes= plt.subplots(1, 2, figsize= (15,5))
fig.suptitle("Courbes d'apprentissage.")

axes[0].plot(x[debut:], hist.history["loss"][debut:], label="Jeux d'entraînement")
axes[0].plot(x[debut:], hist.history["val_loss"][debut:], label="Jeux de validation")
axes[0].set_title("Courbes de loss.")
axes[0].set_xlabel= "Epoch", ylabel="loss")
axes[0].legend()

axes[1].plot(x[debut:], hist.history["accuracy"][debut:], label="Jeux d'entraînement")
axes[1].plot(x[debut:], hist.history["val_accuracy"][debut:], label="Jeux de validation")
axes[1].set_title("Courbes d'accuracy.")
axes[1].set_xlabel= "Epoch", ylabel="précision")
_ = axes[1].legend()
```



c Conclusion de l'étude sur les 3 modèles proposés

Le premier algorithme a les moins bonnes performances mais son modèle binaire est plutôt léger (environ 65 Ko).

Le second algorithme utilisant le KNN est beaucoup plus performant au prix d'un modèle particulièrement lourd (770 Mo). Nous gagnons en précision, mais au prix d'un besoin en ressource machine important.

Le dernier algorithme à base d'un réseau neuronal de type CNN est à la fois excellent en terme de résultat sans, pour autant, avoir un besoin en ressource machine comparable à celui du KNN. Le modèle entraîné ne pèse "que" 26 Mo (pour le premier réseau entraîné).

3 Ajout de nouvelles fonctionnalités

Référentiel : C14. Améliorer l'application d'intelligence artificielle en développant une évolution fonctionnelle pour répondre à un besoin exprimé par un client ou un utilisateur.

La précision obtenue est plutôt flatteuse (au delà des 98%) mais il s'agit de données issues de la même base, et l'expérience montre qu'il faut savoir être prudent tant que l'on ne valide pas ces résultats dans le monde réel.

On peut se donner comme objectif de réaliser un programme utilisant un navigateur internet pour saisir des chiffres dessinés par un vrai humain. On pourra alors tester plusieurs réseaux de neurones différents sur des cas concrets et se faire une idée plus précise de l'utilisation possible de ces algorithmes.

a Cahier des charges :

- Réaliser un programme utilisant un navigateur et qui pourrait être, dans un second temps, déployé.
- Définir une zone de dessin suffisamment grande pour que l'utilisateur puisse dessiner un chiffre.
- Prévoir une zone de sélection de divers modèles pouvant être testés
- Afficher la prédiction issue de l'algorithme.
- Écrire le programme en python.
- Livrable : Le ".py" ainsi que toute ses dépendances (y compris les modèles utilisés pour la prédiction).
- Prévoir un jeu de test pour réaliser des tests de non régression (même si cela est délicat sur des prévisions basées sur des probabilités).


b Test du programme final "lecture_cm.py" :

Les différents tests menés sur des chiffres dessinés sur le canva montre que nous sommes loin des 98 % affichés par les modèles à base de CNN et même des 92 % à 97 % des modèles de régression logistique et de KNN.

Voici un exemple de prédiction avec 2 modèles différents indiquant 4 pour la prédiction de l'un (correct) et 9 pour l'autre (Comment peut on voir un 9 ? Le charme des CNN et de la décomposition d'image).

Choix du modèle d'algorithme:

KNN



↓ ↶ ↷ 🗑️


Prédiction

4

Chiffre lu: 4

Choix du modèle d'algorithme:

CNN-32-64-F-256-Do



↓ ↶ ↷ 🗑️

Prédiction

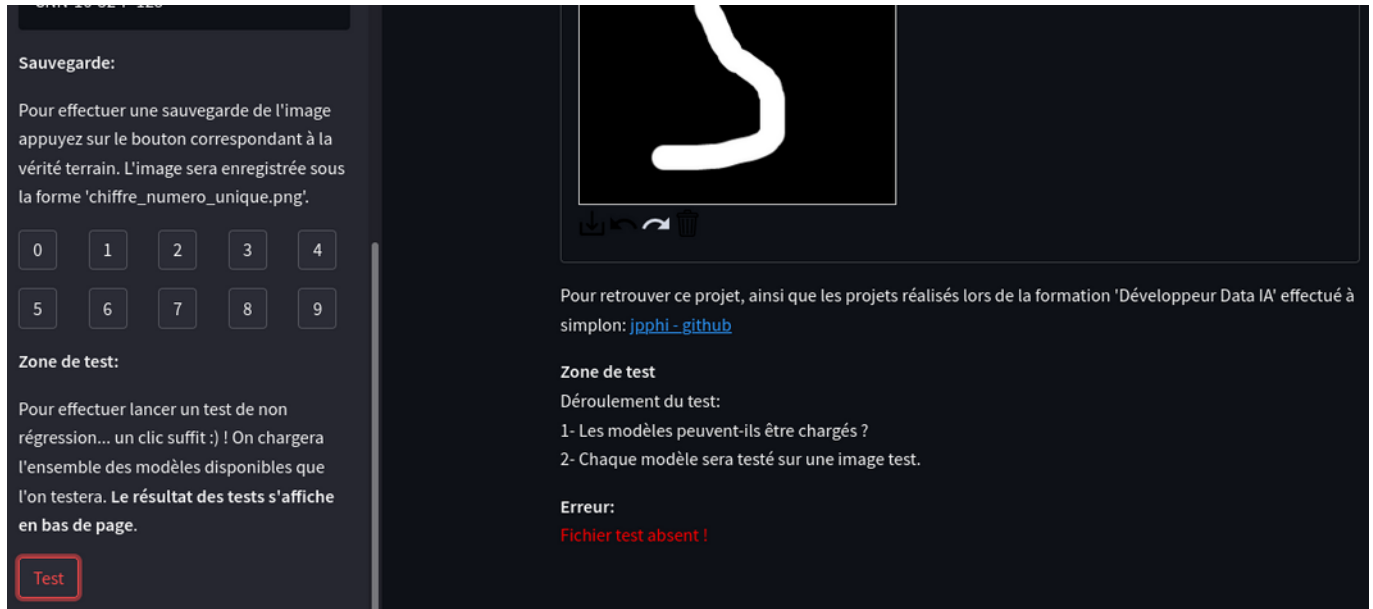
4

Chiffre lu: 9

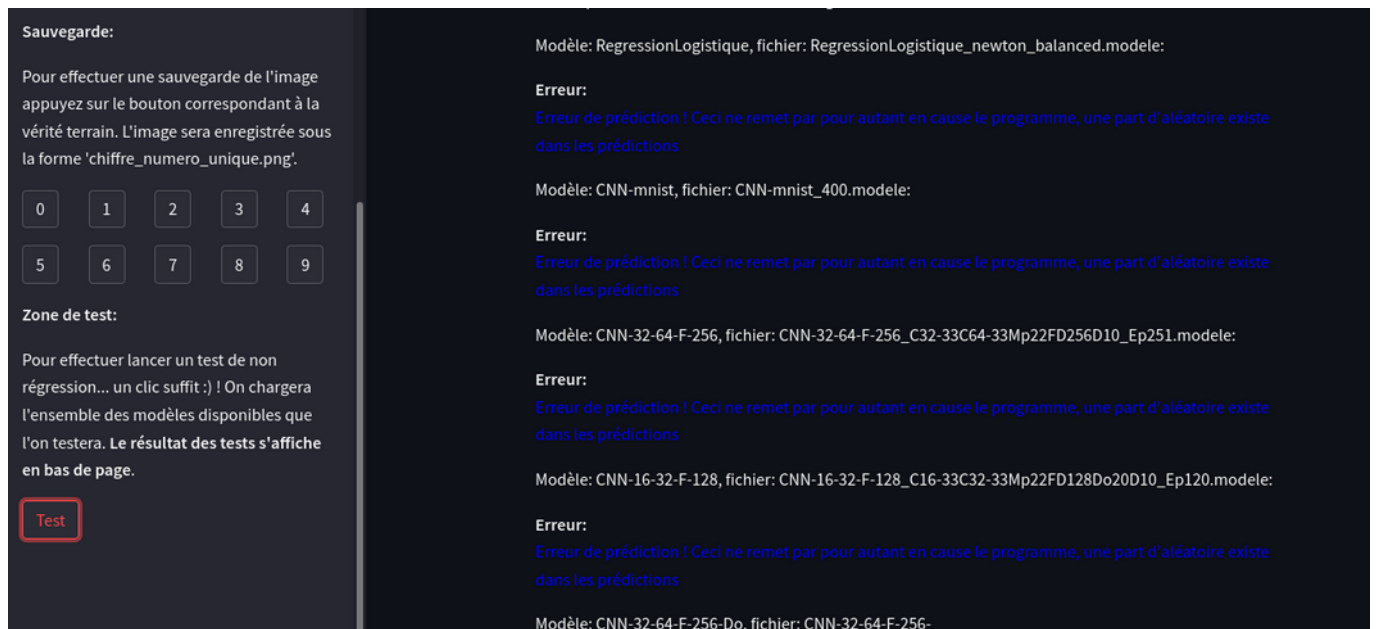
4 Test de fonctionnalité et de non régression.

Une série de tests a été réalisée. Le chargement des algorithmes est testé (test de fonctionnalités), ainsi que leurs réponses à une image test représentant un 5 (test de non régression) et qui est normalement reconnue par tous les modèles. Cependant si un modèle ne reconnaît pas ce chiffre un message s'affichera. Pour le code, voir le fichier "lecture_cm.py" en Annexe.

Voici un exemple d'erreur : Ici le fichier contenant l'image test est absent !



Lorsque l'image test n'est pas reconnue, il ne s'agit pas à proprement parlé d'une erreur ; mais si l'ensemble des algorithmes ne reconnaissent pas l'image, il faut s'en inquiéter !



Et il peut même arriver que les tests se passent bien et sans erreur !

Sauvegarde:

Pour effectuer une sauvegarde de l'image appuyez sur le bouton correspondant à la vérité terrain. L'image sera enregistrée sous la forme 'chiffre_numero_unique.png'.

0

1

2

3

4

5

6

7

8

9

Zone de test:

Pour effectuer lancer un test de non régression... un clic suffit :) ! On chargera l'ensemble des modèles disponibles que l'on testera. Le **résultat des tests s'affiche en bas de page.**

Test



↓ ↶ ↷ 🗑️

Pour retrouver ce projet, ainsi que les projets réalisés lors de la formation 'Développeur Data IA' effectué à simplon: [jpphi - github](https://github.com/jpphi)

Zone de test

Déroulement du test:

- 1- Les modèles peuvent-ils être chargés ?
- 2- Chaque modèle sera testé sur une image test.

Tests passés avec succès, pas d'erreur de fonctionnement

Il peut même arriver, cerise sur le gâteau, de prédire 4 lorsque l'on écrit 4 !

Options:

Épaisseur du trait

1

25

17

Choix du modèle d'algorithme

CNN-32-64-F-256-Do

Sauvegarde:

Pour effectuer une sauvegarde de l'image appuyez sur le bouton correspondant à la vérité terrain. L'image sera enregistrée sous la forme 'chiffre_numero_unique.png'.

0

1

2

3

4

5

6


7

8

9

Zone de test:

Pour effectuer lancer un test de non régression... un clic suffit :) ! On chargera



Prédiction

↓ ↶ ↷ 🗑️

4

Chiffre lu: 4

Tableau des probabilités: p(0)= 9.69 %, p(1)= 9.70 %, p(2)= 9.95 %, p(3)= 10.10 %, p(4)= 10.44 %, p(5)= 9.80 %, p(6)= 9.87 %, p(7)= 9.88 %, p(8)= 10.23 %, p(9)= 10.34 %

fichier modèle: CNN-32-64-F-256-Do_C32-33C64-33Mp22Do25FD256Do50D10_Ep250.modele

Pour retrouver ce projet, ainsi que les projets réalisés lors de la formation 'Développeur Data IA' effectué à simplon: [jpphi - github](https://github.com/jpphi)

5 Conclusion.

Comme on pouvait s'y attendre, il y a toujours une différence significative entre théorie et réalité. Pour autant, ce programme est fonctionnel et pourrait même évoluer en intégrant des fonctionnalités permettant d'utiliser les données issues des dessins pour compléter l'apprentissage réalisé sur la base MNIST. Pour plus de fiabilité, on pourrait faire la détection sur plusieurs modèles ce qui diminuerait le nombre de "faux positifs" mais augmenterait le nombre de "faux négatifs".

Une piste d'amélioration pourrait-être d'utiliser l'analyse en composante principale (PCA). En effet la majorité des pixels compose le fond de l'image et n'apporte pas d'information. En fait on peut même dire que la seule information intéressante est contenue dans les pixels formant le bord du chiffre. C'est d'ailleurs ce que j'espérais des CNN ; qu'ils arrivent à extraire les différentes petites zones du dessin représentant le chiffre.

6 Annexes .

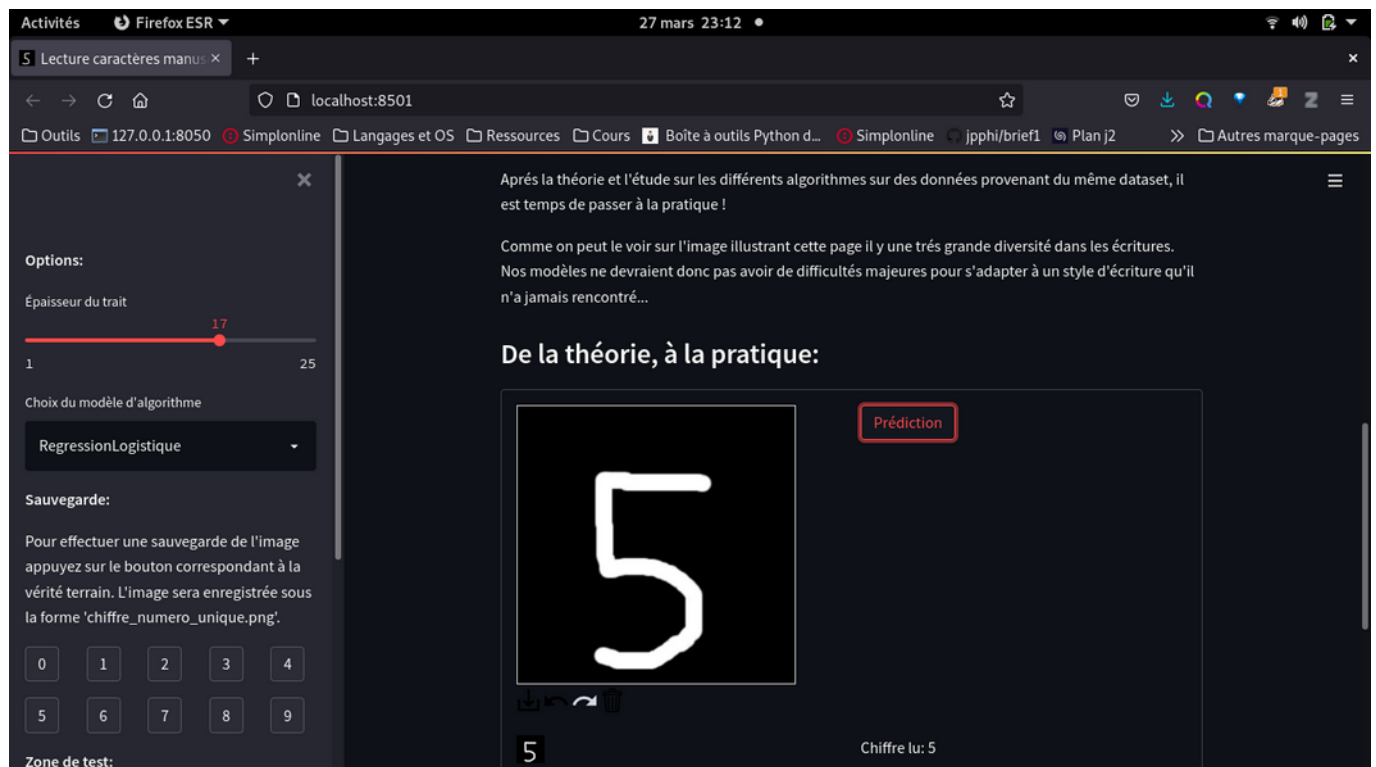
a Jupyter notebook :

C'est le fichier avec lequel a été réalisé la mise au point, l'optimisation ainsi que l'ensemble des tests des différents modèles de machine learning. Si la totalité du code est fournie, la sortie des cellules de code est amputée car le document aurait comporté plus de 60 pages.

b Programme python :

Ce programme est une interface web réalisée grâce à la librairie streamlit. Il permet de dessiner un chiffre (l'épaisseur du trait est paramétrable) puis d'utiliser plusieurs modèles (à choisir dans un boîte de sélection) pour le "décoder". Il permet également d'enregistrer le dessin "avec la vérité terrain" pour constituer des données d'entraînement. Enfin un programme de test de non régression est intégré au programme.

Pour lancer le programme, s'assurer que les dépendances soient installées (sur ce pc, se placer dans **l'environnement prjfe**) puis lancer la commande **streamlit run lecture_cm.py** .



Reprise du projet 5 avec le jeux MNIST original

Imports et fonctions

```
In [2]: import numpy as np

import random

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.model_selection import learning_curve
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.neighbors import KNeighborsClassifier

import h5py

import time

import matplotlib.pyplot as plt

from joblib import dump, load

from PIL import Image

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

-----
ImportError                                Traceback (most recent call last)
/tmp/ipykernel_12278/2242608575.py in <module>
    24 import tensorflow as tf
    25
--> 26 from tensorflow import keras
    27
    28 from tensorflow.keras.models import Sequential

ImportError: cannot import name 'keras' from 'tensorflow' (unknown location)
```

```
In [3]: def erreur_prediction(valeurs_predites= None, valeurs_reelle= None):
    """
    Vérifie que les valeurs prédites sont les mêmes que les valeurs réelles.

    paramètres en entrée:
    valeurs_predites
    valeurs_reelle

    Paramètres de sortie:
    erreur: nombre d'erreur de prédiction
    precision: 100*(1-erreur/(longueur des tableaux))
    liste_erreur: Liste de toutes les erreurs détecté sous forme de dictionnaire
    clé valeur réelle: valeur prédite
    """

    assert len(valeurs_predites) == len(valeurs_reelle), "Erreur de dimension: la dimension des tableaux "+\
        f"doit être identique. {len(valeurs_predites)} # de {len(valeurs_reelle)}"

    erreur= 0
    d= {}
    longueur= len(valeurs_reelle)
    for i in range(longueur):
        if valeurs_reelle[i] != valeurs_predites[i]:
            if valeurs_reelle[i] not in d.keys():
                d[valeurs_reelle[i]] = [valeurs_predites[i]]
            else:
                d[valeurs_reelle[i]].append(valeurs_predites[i])
            erreur+= 1

    return erreur, 100*(1-erreur/longueur), d
```

Reprise partielle du brief

```
In [4]: # Importation des données depuis scikit learn datasets
digits = load_digits()

# Les images sont organisées en matrice de 8x8
X = digits.images
# Labels des images (chiffre de 0 à 9)
Y = digits.target
# Les images sont "aplaties" et transformées en un tableau de 64 éléments
Xdata= digits.data

print(f"X Shape: {X.shape}, Xdata Shape: {Xdata.shape}, Y shape: {Y.shape}, "+\
      f"digits.target_name: {digits.target_names}\n")

X Shape: (1797, 8, 8), Xdata Shape: (1797, 64), Y shape: (1797,), digits.target_name: [0 1 2 3 4 5 6 7 8 9]
```

```
In [5]: # Préparation de l'affichage
n_digits = np.unique(Y) # = 0,1,2,...9
M = 12

fig, axs = plt.subplots(len(n_digits), M, figsize=(20, 15))

# Afficher M exemples de tout les digits (de 0 à 9)
for i, d in enumerate(n_digits):
    x = X[Y == d]
    for j in range(M):
        num = random.randint(0, x.shape[0]-1)
        axs[i, j].imshow(X[Y == d][j], cmap="gray")
        axs[i, j].axis('off')
```



```
In [8]: # On split le jeux de données
X_train, X_test, Y_train, Y_test = train_test_split(Xdata, Y, test_size=0.2, random_state= 511)

# On construit et on 'fit' le modèle
log_regr = LogisticRegression(solver='liblinear')
log_regr.fit(X_train, Y_train)

# on teste sur les données de.... test :)
prediction = log_regr.predict(X_test)
```

```
err, pre, d= erreur_prediction(valeurs_predites= prediction, valeurs_reelle= Y_test)
print(f"{err} erreurs sur {len(X_test)} images. soit une précision de {pre:.2f} %")
print(f"Dictionnaire des erreurs (clé valeur réelle: valeur prédite):\n{d}")
```

```
19 erreurs sur 360 images. soit une précision de 94.72 %
Dictionnaire des erreurs (clé valeur réelle: valeur prédite):
{8: [1, 1, 1, 3, 1], 3: [5, 8], 9: [8, 3, 4, 8], 5: [9, 9, 9, 9], 4: [7, 6], 1: [6, 8]}
```

```
In [6]: target_names= ["détection de 0", "détection de 1", "détection de 2", "détection de 3", "détection de 4",
                        "détection de 5", "détection de 6", "détection de 7", "détection de 8", "détection de 9"]
print("\n", classification_report(Y_test, prediction, target_names=target_names))
```

	precision	recall	f1-score	support
détection de 0	1.00	1.00	1.00	31
détection de 1	0.89	0.94	0.91	33
détection de 2	1.00	1.00	1.00	35
détection de 3	0.94	0.94	0.94	36
détection de 4	0.97	0.95	0.96	41
détection de 5	0.98	0.91	0.94	45
détection de 6	0.95	1.00	0.97	35
détection de 7	0.97	1.00	0.98	31
détection de 8	0.90	0.88	0.89	40
détection de 9	0.88	0.88	0.88	33
accuracy			0.95	360
macro avg	0.95	0.95	0.95	360
weighted avg	0.95	0.95	0.95	360

Reprise du brief 5 avec MNIST

Importation et visualisation des données du dataset MNIST

On notera (voir ci-dessous) que les jeux de données sont quelque peu déséquilibré.

```
In [7]: # Importation du jeux d'entraînement
f = h5py.File("../datas/train.hdf5", 'r')
train_x, train_y = (f['image'][...])/255.0, f['label'][...]
f.close()

# Importation du jeux de test
f = h5py.File("../datas/test.hdf5", 'r')
test_x, test_y = (f['image'][...])/255.0, f['label'][...]
f.close()

etiquette= np.unique(test_y)
print(f"Entrainement: X Shape: {train_x.shape}, Y shape: {train_y.shape}\n"+
      f"Test: X Shape: {test_x.shape}, Y shape: {test_y.shape}\n"+
      f"Etiquette: {etiquette}")
```

Entrainement: X Shape: (60000, 28, 28), Y shape: (60000,)
Test: X Shape: (10000, 28, 28), Y shape: (10000,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]

```
In [8]: # Le jeux de donnée "train" est-il équilibré ?
d= {}
for i in etiquette:
    l= len(train_x[train_y== i])
    d[i]= l
    print(f"Le chiffre {i} est représenté {l} fois", end= " - ")
mini, maxi= min(d, key= d.get), max(d, key= d.get)
print(f"\nLe chiffre le moins représenté est {mini} représenté {d[mini]} fois.")
print(f"Le chiffre le plus représenté est {maxi} représenté {d[maxi]} fois.")
```

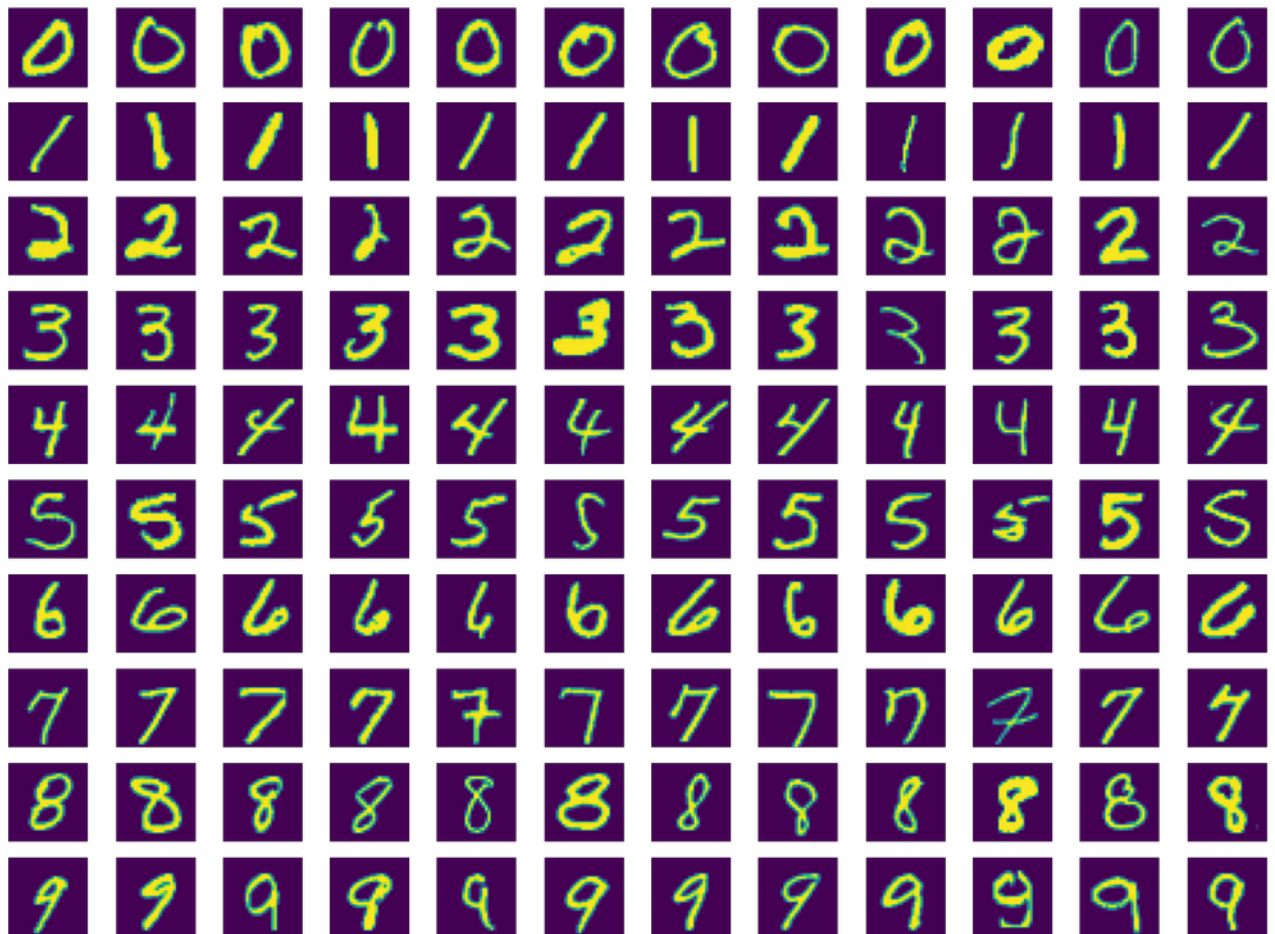
Le chiffre 0 est représenté 5923 fois - Le chiffre 1 est représenté 6742 fois - Le chiffre 2 est représenté 5958 fois - Le chiffre 3 est représenté 6131 fois - Le chiffre 4 est représenté 5842 fois - Le chiffre 5 est représenté 5421 fois - Le chiffre 6 est représenté 5918 fois - Le chiffre 7 est représenté 6265 fois - Le chiffre 8 est représenté 5851 fois - Le chiffre 9 est représenté 5949 fois -
Le chiffre le moins représenté est 5 représenté 5421 fois.
Le chiffre le plus représenté est 1 représenté 6742 fois.

```
In [9]: # Le jeux de donnée "test" est-il équilibré ?
d= {}
for i in etiquette:
    l= len(test_x[test_y== i])
    d[i]= l
    print(f"Le chiffre {i} est représenté {l} fois", end= " - ")
mini, maxi= min(d, key= d.get), max(d, key= d.get)
print(f"\nLe chiffre le moins représenté est {mini} représenté {d[mini]} fois.")
print(f"Le chiffre le plus représenté est {maxi} représenté {d[maxi]} fois.")
```

Le chiffre 0 est représenté 980 fois - Le chiffre 1 est représenté 1135 fois - Le chiffre 2 est représenté 1032 fois - Le chiffre 3 est représenté 1010 fois - Le chiffre 4 est représenté 982 fois - Le chiffre 5 est représenté 892 fois - Le chiffre 6 est représenté 958 fois - Le chiffre 7 est représenté 1028 fois - Le chiffre 8 est représenté 974 fois - Le chiffre 9 est représenté 1009 fois -
Le chiffre le moins représenté est 5 représenté 892 fois.
Le chiffre le plus représenté est 1 représenté 1135 fois.

```
In [10]: # Préparation de l'affichage
M= 12
fig, axs = plt.subplots(len(etiquette), M, figsize=(20, 15))

np.random.seed(51165)
# Afficher M exemples de tout les digits (de 0 à 9)
for i, d in enumerate(etiquette):
    x= train_x[train_y == d]
    for j in range(M):
        num= random.randint(0, x.shape[0]-1)
        axs[i,j].imshow(x[num])
        axs[i,j].axis('off')
```

On test en se plaçant dans les mêmes conditions que pour le brief 5

```
In [11]: # On récupère les données. On ne prendra qu'une partie des données situés dans le
# jeux de train. On en profite pour "aplatir" les images
data= train_x.reshape(train_x.shape[0], train_x.shape[1]*train_x.shape[2])

# On réduit le jeux de données original
data_red, _, Y_red, _ = train_test_split(data, train_y, train_size=0.1, random_state= 65)

# On split le jeux de données
X_train_red, X_test_red, Y_train_red, Y_test_red = train_test_split(data_red, Y_red, train_size=0.8,
                                                                    random_state= 65)

print(f"Entrainement: X Shape: {X_train_red.shape}, Y shape: {Y_train_red.shape}\n"+\
      f"Test: X Shape: {X_test_red.shape}, Y shape: {Y_test_red.shape}\n"+\
      f"Etiquette: {etiquette}")

Entrainement: X Shape: (4800, 784), Y shape: (4800,)
Test: X Shape: (1200, 784), Y shape: (1200,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]
```

```
In [12]: # Le jeux de donnée est-il équilibré ?
d= {}
for i in etiquette:
    l= len(X_train_red[Y_train_red== i])
    d[i]= l
    print(f"Le chiffre {i} est représenté {l} fois", end= " - ")
mini, maxi= min(d, key= d.get), max(d, key= d.get)
print(f"\nLe chiffre le moins représenté est {mini} représenté {d[mini]} fois.")
print(f"Le chiffre le plus représenté est {maxi} représenté {d[maxi]} fois.")

Le chiffre 0 est représenté 483 fois - Le chiffre 1 est représenté 546 fois - Le chiffre 2 est représenté 472 fois - Le chiffre 3 est
représenté 483 fois - Le chiffre 4 est représenté 473 fois - Le chiffre 5 est représenté 431 fois - Le chiffre 6 est représenté 482 fo
is - Le chiffre 7 est représenté 480 fois - Le chiffre 8 est représenté 487 fois - Le chiffre 9 est représenté 463 fois -
Le chiffre le moins représenté est 5 représenté 431 fois.
Le chiffre le plus représenté est 1 représenté 546 fois.
```

```
In [13]: deb= time.time()
# On construit et on 'fit' le modèle
model_reglog = LogisticRegression(solver='liblinear')
model_reglog.fit(X_train_red, Y_train_red)
fin= time.time()
print(f"temps d'exécution: {fin-deb:.2f} s")

temps d'exécution: 3.07 s
```

On teste l'algorithme

```
In [14]: # on teste sur les données de... test :)
prediction = model_reglog.predict(X_test_red)

err, pre, d= erreur_prediction(valeurs_predites= prediction, valeurs_reelle= Y_test_red)
print(f"{err} erreurs sur {len(X_test_red)} images soit une précision de {pre:.2f} %")
print(f"Dictionnaire des erreurs (clé valeur réelle: valeur prédite):\n{d}")

118 erreurs sur 1200 images soit une précision de 90.17 %
Dictionnaire des erreurs (clé valeur réelle: valeur prédite):
{0: [4, 5, 5, 3, 4, 5, 5, 3, 0, 5, 3, 1, 3, 9, 9, 1, 9], 2: [7, 8, 0, 7, 6, 4, 3, 7, 3, 1, 1, 3, 0, 8, 5, 6, 7, 3, 6, 8, 8, 7, 3, 8],
5: [3, 8, 3, 9, 4, 9, 4, 3, 1, 9, 6, 3, 9, 8, 0, 8, 8], 9: [4, 1, 4, 8, 4, 5, 2, 3, 3, 7, 7, 4, 7], 7: [5, 4, 9, 8, 8, 9, 9, 8, 9, 9],
3: [8, 8, 2, 5, 2, 0, 9, 0, 5, 5, 7, 8, 8], 6: [8, 5, 4, 5, 2, 5], 4: [3, 2, 9, 8, 1, 9, 9, 9, 9, 6, 8, 5, 0], 0: [4, 6, 6], 1: [2,
3]}
```

```
In [15]: target_names= ["détection de 0", "détection de 1", "détection de 2", "détection de 3", "détection de 4",
                        "détection de 5", "détection de 6", "détection de 7", "détection de 8", "détection de 9"]
print("Le score obtenu est de {} % de bonne prédictions".
      format(round(model_reglog.score(X_test_red, Y_test_red)*100,2)))
print("\n",classification_report(Y_test_red,prediction, target_names= target_names))
```

Le score obtenu est de 90.17 % de bonne prédictions

	precision	recall	f1-score	support
détection de 0	0.95	0.98	0.96	126
détection de 1	0.94	0.98	0.96	117
détection de 2	0.95	0.82	0.88	137
détection de 3	0.85	0.88	0.87	112
détection de 4	0.90	0.89	0.90	123
détection de 5	0.85	0.83	0.84	103
détection de 6	0.95	0.96	0.95	135
détection de 7	0.92	0.92	0.92	119
détection de 8	0.82	0.85	0.83	110
détection de 9	0.85	0.89	0.87	118
accuracy			0.90	1200
macro avg	0.90	0.90	0.90	1200
weighted avg	0.90	0.90	0.90	1200

Normalisation des données et test d'un nouveau solver

```
In [16]: # On récupère les données. On ne prendra qu'une partie des données situé dans le
# jeux de train. On en profite pour "aplatir" les images
data= train_x.reshape(train_x.shape[0], train_x.shape[1]*train_x.shape[2])

# On réduit le jeux de données original
data_red, _, Y_red, _ = train_test_split(data, train_y, train_size=0.1, random_state= 65)

# Standardisation
scaler=StandardScaler()
datastd= scaler.fit_transform(data_red)

# On split le jeux de données
X_train_red, X_test_red, Y_train_red, Y_test_red = train_test_split(datastd, Y_red, train_size=0.8,
                                                                    random_state= 65)

print(f"Entrainement: X Shape: {X_train_red.shape}, Y shape: {Y_train_red.shape}\n"+\
      f"Test: X Shape: {X_test_red.shape}, Y shape: {Y_test_red.shape}\n"+\
      f"Etiquette: {etiquette}")
```

Entrainement: X Shape: (4800, 784), Y shape: (4800,)
Test: X Shape: (1200, 784), Y shape: (1200,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]

```
In [17]: deb= time.time()
# On construit et on 'fit' le modèle
model_reglog = LogisticRegression(solver='lbfgs', class_weight= "balanced", max_iter= 2000)
model_reglog.fit(X_train_red, Y_train_red)
fin= time.time()
print(f"temps d'exécution: {fin-deb:.2f} s")
```

temps d'exécution: 6.05 s

```
In [18]: # on teste sur les données de... test :)
prediction = model_reglog.predict(X_test_red)

err, pre, d= erreur_prediction(valeurs_predites= prediction, valeurs_reelle= Y_test_red)
print(f"{err} erreurs sur {len(X_test_red)} images soit une précision de {pre:.2f} %")
print(f"Dictionnaire des erreurs (clé valeur réelle: valeur prédite):\n{d}")

126 erreurs sur 1200 images soit une précision de 89.50 %
Dictionnaire des erreurs (clé valeur réelle: valeur prédite):
{8: [3, 5, 5, 3, 6, 2, 5, 2, 1, 3, 1, 3, 2, 1, 9], 2: [3, 8, 3, 7, 6, 1, 3, 4, 4, 3, 1, 0, 3, 1, 0, 7, 6, 7, 4, 7, 3, 6, 0, 9, 8, 3],
5: [3, 3, 8, 0, 3, 9, 4, 3, 1, 0, 9, 3, 6, 8, 4, 3, 9, 8, 9, 8, 0, 8, 8], 3: [5, 9, 8, 2, 8, 2, 0, 0, 5, 5, 7, 2, 8, 8], 9: [4, 1, 3,
4, 3, 4, 4, 2, 4, 3, 8, 8, 4, 7], 6: [8, 5, 0, 8, 5], 4: [3, 8, 2, 9, 8, 1, 9, 2, 8, 0], 1: [2, 3, 2], 0: [6, 5, 4, 6, 6, 7], 7: [4,
3, 9, 4, 9, 9, 1, 5, 9, 9]}
```

```
In [19]: target_names= ["détection de 0", "détection de 1", "détection de 2", "détection de 3", "détection de 4",
                        "détection de 5", "détection de 6", "détection de 7", "détection de 8", "détection de 9"]
print(f"Le score obtenu est de {model_reglog.score(X_test_red, Y_test_red)*100:.2f}%")
print("\n",classification_report(Y_test_red,prediction, target_names= target_names))
```

Le score obtenu est de 89.50

	precision	recall	f1-score	support
détection de 0	0.92	0.95	0.94	126
détection de 1	0.92	0.97	0.95	117

détection de 2	0.91	0.81	0.86	137
détection de 3	0.81	0.88	0.84	112
détection de 4	0.89	0.92	0.90	123
détection de 5	0.89	0.78	0.83	103
détection de 6	0.94	0.96	0.95	135
détection de 7	0.94	0.92	0.93	119
détection de 8	0.83	0.86	0.85	110
détection de 9	0.88	0.88	0.88	118
accuracy			0.90	1200
macro avg	0.89	0.89	0.89	1200
weighted avg	0.90	0.90	0.89	1200

Gridsearch CV pour déterminer la meilleure standardisation et le meilleur solveur pour la régression logistique

```
In [18]: # On récupère les données. On ne prendra qu'une partie des données situés dans le
# jeux de train. On en profite pour "aplatir" les images
data= train_x.reshape(train_x.shape[0], train_x.shape[1]*train_x.shape[2])

# On réduit le jeux de données original
data_red, _, Y_red, _ = train_test_split(data, train_y, train_size=0.1, random_state= 65)

# On split le jeux de données
X_train_red, X_test_red, Y_train_red, Y_test_red = train_test_split(data_red, Y_red, train_size=0.8,
                                                                    random_state= 65)

print(f"Entrainement: X Shape: {X_train_red.shape}, Y shape: {Y_train_red.shape}\n"+\
      f"Test: X Shape: {X_test_red.shape}, Y shape: {Y_test_red.shape}\n"+\
      f"Etiquette: {etiquette}")

Entrainement: X Shape: (4800, 784), Y shape: (4800,)
Test: X Shape: (1200, 784), Y shape: (1200,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]
```

```
In [19]: # On réalise un pipe avec 2 actions: un scaler et un algorithme
pipe = Pipeline(steps=[('scaler', StandardScaler()), ('algo', LogisticRegression())])

# On testera 3 standardisations et pour la régression logistique on testera tout les solvers
param_grid = {'pipeline__scaler': [StandardScaler(), MinMaxScaler(), RobustScaler()],
              'pipeline__algo__solver': ["sag", "newton-cg", "lbfgs", "saga"],
              'pipeline__algo__class_weight': ["balanced"],
              'pipeline__algo__max_iter': [2000]}

# instantiate and run as before:

model= make_pipeline(pipe)

grid = GridSearchCV(model, param_grid, cv=5, n_jobs= -1)

grid
```

```
Out[19]: GridSearchCV(cv=5,
                    estimator=Pipeline(steps=[('pipeline',
                                              Pipeline(steps=[('scaler',
                                                                StandardScaler()),
                                                                ('algo',
                                                                LogisticRegression()))])),
                    n_jobs=-1,
                    param_grid={'pipeline__algo__class_weight': ['balanced'],
                                'pipeline__algo__max_iter': [2000],
                                'pipeline__algo__solver': ['sag', 'newton-cg', 'lbfgs',
                                                            'saga'],
                                'pipeline__scaler': [StandardScaler(), MinMaxScaler(),
                                                    RobustScaler()]})
```

```
In [20]: deb= time.time()

grid.fit(X_train_red, Y_train_red)

fin= time.time()
print(f"temps d'exécution: {fin-deb:.2f} s")
```

```
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn()
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn()
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn()
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn()
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn()
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/home/jpphi/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:


```
In [3]: # Importation du jeux d'entrainement. On réalise directement une normalisation min/max en divisant par 255.0
# Les pixels composant l'image étant compris entre 0 et 255
f = h5py.File("./datas/train.hdf5", 'r')
train_x, train_y = (f['image'][:])/255.0, f['label'][:...]
f.close()

# Importation du jeux de test
f = h5py.File("./datas/test.hdf5", 'r')
test_x, test_y = (f['image'][:])/255.0, f['label'][:...]
f.close()

etiquette= np.unique(test_y)
print(f"Entrainement: X Shape: {train_x.shape}, Y shape: {train_y.shape}\n"+
      f"Test: X Shape: {test_x.shape}, Y shape: {test_y.shape}\n"+
      f"Etiquette: {etiquette}")
```

Entrainement: X Shape: (60000, 28, 28), Y shape: (60000,)
Test: X Shape: (10000, 28, 28), Y shape: (10000,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]

```
In [4]: X_train= train_x.reshape(train_x.shape[0], train_x.shape[1]*train_x.shape[2])
Y_train= train_y

X_test= test_x.reshape(test_x.shape[0], test_x.shape[1]*test_x.shape[2])
Y_test= test_y
```

```
In [7]: model_reglog= LogisticRegression(class_weight='balanced', max_iter=2000, solver='newton-cg')
deb= time.time()
model_reglog.fit(X_train, Y_train)
fin= time.time()
print(f"fit en {fin-deb} s")
```

fit en 328.4471061229706 s

```
In [8]: prediction = model_reglog.predict(X_test)

err, pre, d= erreur_prediction(valeurs_predites= prediction, valeurs_reelle= Y_test)
print(f"{err} erreurs sur {len(X_test)} images soit une précision de {pre:.2f} %")
print(f"Dictionnaire des erreurs (clé valeur réelle: valeur prédite):\n{d}")
```

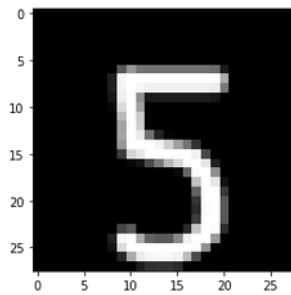
737 erreurs sur 10000 images soit une précision de 92.63 %
Dictionnaire des erreurs (clé valeur réelle: valeur prédite):
{5: [6, 7, 3, 0, 8, 3, 8, 8, 3, 7, 8, 9, 3, 4, 3, 8, 3, 6, 7, 6, 3, 0, 8, 0, 3, 3, 4, 9, 8, 8, 8, 4, 8, 3, 2, 8, 7, 3, 8, 0, 3, 3, 2, 0, 7, 7, 9, 9, 4, 0, 8, 4, 8, 0, 2, 3, 3, 3, 1, 6, 8, 8, 4, 3, 8, 6, 3, 8, 6, 8, 8, 1, 3, 3, 3, 3, 3, 8, 3, 8, 8, 3, 6, 4, 8, 6, 8, 3, 6, 8, 8, 8, 4, 4, 8, 8, 3, 6, 3, 3, 3, 6, 6, 0, 0, 6, 6, 3, 6], 4: [6, 6, 2, 9, 9, 9, 9, 8, 9, 1, 6, 9, 6, 9, 9, 9, 3, 7, 3, 6, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 2, 9, 2, 2, 9, 1, 6, 9, 2, 6, 9, 9, 1, 6, 9, 3, 7, 0, 9, 8, 9, 8, 8, 9, 9, 7, 2, 9, 8], 3: [2, 5, 5, 5, 7, 5, 2, 6, 5, 2, 5, 5, 8, 5, 9, 8, 7, 9, 8, 7, 7, 5, 7, 5, 1, 8, 5, 9, 5, 5, 6, 5, 0, 2, 7, 5, 5, 2, 2, 2, 8, 5, 2, 2, 7, 7, 7, 5, 9, 8, 5, 2, 8, 5, 8, 4, 0, 9, 0, 8, 8, 5, 8, 8, 8, 8, 8, 7, 5, 8, 2, 2, 2, 2, 8, 8, 5, 8, 8, 5, 8, 2, 0, 9, 2, 8, 9, 8], 6: [2, 5, 0, 0, 0, 5, 2, 2, 1, 8, 7, 7, 5, 4, 0, 2, 1, 5, 4, 5, 1, 2, 4, 0, 3, 0, 4, 5, 0, 5, 5, 5, 5, 5, 2, 2, 2, 5, 4, 4, 5, 5, 5, 5, 3, 0], 7: [4, 9, 2, 1, 3, 2, 2, 9, 9, 9, 2, 2, 1, 2, 2, 9, 1, 9, 9, 3, 2, 9, 9, 8, 2, 9, 5, 9, 3, 9, 4, 4, 9, 9, 4, 1, 2, 9, 9, 8, 0, 9, 2, 2, 3, 9, 1, 4, 9, 9, 3, 9, 9, 9, 2, 2, 8, 8, 2, 9, 9, 9, 9, 9, 2, 9, 1, 3, 4, 2, 2, 2, 2, 2, 2, 2, 9, 9, 3, 9], 2: [9, 7, 7, 8, 1, 8, 8, 6, 9, 8, 8, 7, 0, 3, 3, 6, 6, 6, 1, 4, 9, 0, 7, 0, 8, 8, 3, 8, 1, 0, 4, 8, 8, 3, 3, 1, 3, 5, 8, 3, 8, 3, 8, 8, 1, 8, 4, 4, 7, 3, 8, 5, 4, 7, 1, 4, 3, 3, 3, 8, 8, 3, 8, 8, 8, 6, 6, 6, 8, 4, 8, 6, 6, 7, 8, 6, 9, 4, 4, 6, 8, 8, 8, 7, 8, 8, 3, 8, 4, 6, 1, 8, 5, 0, 0, 8, 8, 7, 8, 8], 9: [3, 8, 7, 8, 3, 7, 7, 3, 4, 4, 5, 8, 7, 7, 5, 4, 5, 2, 1, 4, 0, 0, 1, 1, 0, 4, 4, 4, 8, 7, 8, 1, 1, 3, 4, 4, 4, 3, 1, 4, 7, 4, 0, 7, 4, 4, 7, 1, 8, 7, 4, 5, 7, 7, 7, 3, 8, 8, 0, 3, 3, 0, 0, 0, 3, 4, 7, 5, 0, 8, 5, 4, 7, 5, 7, 7, 4, 7, 7, 8, 4, 4, 4], 8: [7, 5, 5, 7, 2, 3, 1, 3, 4, 5, 3, 7, 2, 9, 4, 4, 5, 5, 3, 1, 6, 2, 1, 7, 5, 0, 5, 5, 9, 9, 3, 3, 5, 5, 9, 6, 2, 0, 0, 9, 5, 3, 9, 3, 2, 5, 0, 1, 3, 5, 5, 3, 3, 3, 5, 5, 2, 3, 5, 0, 7, 3, 4, 3, 7, 3, 7, 6, 0, 3, 6, 6, 6, 4, 1, 5, 4, 0, 1, 1, 6, 6, 9, 1, 7, 4, 5, 5, 7, 9, 9, 7, 3, 5, 5, 9, 9, 5, 1, 0, 5, 5, 5, 6, 6, 6, 0, 6, 5, 9, 6, 5, 1, 6, 1], 1: [8, 2, 6, 5, 2, 6, 2, 3, 7, 3, 6, 5, 2, 2, 8, 2, 8, 8, 8, 8, 8, 8, 8, 7], 0: [6, 7, 5, 5, 5, 5, 5, 7, 5, 4, 2, 6, 8, 3, 6, 6, 3, 7, 5, 5, 3, 5, 3, 5, 2]}

```
In [9]: target_names= ["détection de 0", "détection de 1", "détection de 2", "détection de 3", "détection de 4",
                    "détection de 5", "détection de 6", "détection de 7", "détection de 8", "détection de 9"]
print("\n",classification_report(Y_test,prediction, target_names= target_names))
```

	precision	recall	f1-score	support
détection de 0	0.95	0.97	0.96	980
détection de 1	0.97	0.98	0.97	1135
détection de 2	0.93	0.90	0.92	1032
détection de 3	0.91	0.91	0.91	1010
détection de 4	0.94	0.94	0.94	982
détection de 5	0.89	0.88	0.89	892
détection de 6	0.94	0.95	0.95	958
détection de 7	0.94	0.92	0.93	1028
détection de 8	0.88	0.88	0.88	974
détection de 9	0.91	0.92	0.92	1009
accuracy			0.93	10000
macro avg	0.93	0.93	0.93	10000
weighted avg	0.93	0.93	0.93	10000

Test sur un chiffre qui n'est pas issue d'un dataset

```
In [4]: img= Image.open("./images/image_test.png")
        _=plt.imshow(img, cmap= "gray")
```



```
In [12]: chiffre=np.array(img)/255.
chiffre_plat=chiffre.ravel()
tab=list(model_reglog.predict_proba([chiffre_plat])[0])
print(f"Le chiffre le plus probable est: {tab.index(max(tab))}\nListe des probabilités:{tab}")

Le chiffre le plus probable est: 5
Liste des probabilités:[0.00010985476550631836, 6.972696327770812e-07, 0.008636546996151222, 0.369270499667964, 0.0004235051156176312,
0.3821188287201187, 0.00019094647377095294, 1.4248341358281702e-10, 0.23924762938532074, 1.4914634344104556e-06]
```

Enregistrement du modèle

```
In [21]: dump(model_reglog, 'RegressionLogistique_newton_balanced.modele')
```

```
Out[21]: ['RegressionLogistique_newton_balanced.modele']
```

Learning curve

- Le but de cette étude est de montrer que l'on pouvait ne pas prendre la totalité des données pour réaliser l'optimisation des hyperparamètres (qui aurait demandé sinon un temps de calcul très élevé). Cependant, le modèle définitif trouvé a été entraîné sur la totalité des données.
- On fera cette étude en prenant l'intégralité des données avec le meilleur solveur.

```
In [33]: # On récupère les données et on en profite pour "aplatir" les images
data_train=train_x.reshape(train_x.shape[0], train_x.shape[1]*train_x.shape[2])
data_test=test_x.reshape(test_x.shape[0], test_x.shape[1]*test_x.shape[2])

etiquette=np.unique(test_y)

print(f"Entrainement: X Shape: {data_train.shape}, Y shape: {train_y.shape}\n"+
      f"Test: X Shape: {data_test.shape}, Y shape: {test_y.shape}\n"+
      f"Etiquette: {etiquette}")
```

```
Entrainement: X Shape: (60000, 784), Y shape: (60000,)
Test: X Shape: (10000, 784), Y shape: (10000,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]
```

```
In [34]: # On choisi le meilleur solveur avec une standardisation min/max
pipe_minmax_logreg=make_pipeline(MinMaxScaler(), LogisticRegression(class_weight='balanced',
                                                                    max_iter=2000, solver='newton-cg'))

pipe_minmax_logreg.fit(data_train, train_y)
```

```
Out[34]: Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                          ('logisticregression',
                           LogisticRegression(class_weight='balanced', max_iter=2000,
                                                solver='newton-cg'))])
```

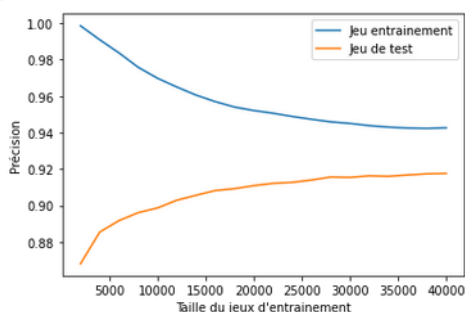
```
In [35]: prediction = pipe_minmax_logreg.predict(data_test)

err, pre, d=erreur_prediction(valeurs_predites=prediction, valeurs_reelle=test_y)
print(f"{err} erreurs sur {len(data_test)} images soit une précision de {pre:.2f} %")
print(f"Dictionnaire des erreurs (clé valeur réelle: valeur prédite):\n{d}")
```

```
737 erreurs sur 10000 images soit une précision de 92.63 %
Dictionnaire des erreurs (clé valeur réelle: valeur prédite):
{5: [6, 7, 3, 0, 8, 3, 8, 8, 3, 7, 8, 9, 3, 4, 3, 8, 3, 6, 7, 6, 3, 0, 8, 0, 3, 3, 4, 9, 8, 8, 8, 4, 8, 3, 2, 8, 7, 3, 8, 0, 3, 3, 2,
0, 7, 7, 9, 9, 4, 0, 8, 4, 8, 0, 2, 3, 3, 3, 1, 6, 8, 8, 4, 3, 8, 6, 3, 8, 6, 8, 8, 1, 3, 3, 3, 3, 3, 3, 8, 3, 8, 8, 3, 6, 4, 8, 6, 8,
3, 6, 8, 8, 8, 4, 4, 8, 8, 3, 6, 3, 3, 3, 6, 6, 0, 0, 6, 6, 3, 6], 4: [6, 6, 2, 9, 9, 9, 9, 8, 9, 1, 6, 9, 6, 9, 9, 9, 3, 7, 3, 6, 8,
9, 9, 9, 9, 7, 9, 9, 9, 2, 9, 9, 2, 2, 9, 1, 9, 2, 6, 9, 9, 1, 6, 9, 3, 7, 0, 9, 8, 8, 8, 9, 9, 9, 7, 2, 9, 8], 3: [2, 5, 5,
5, 7, 5, 2, 6, 5, 2, 5, 5, 8, 5, 9, 8, 7, 9, 8, 7, 7, 5, 7, 5, 1, 8, 5, 9, 5, 5, 6, 5, 0, 2, 7, 5, 5, 2, 2, 2, 8, 5, 2, 2, 7, 7, 7, 5,
9, 8, 5, 2, 8, 5, 8, 4, 2, 0, 9, 0, 8, 8, 5, 8, 8, 8, 8, 8, 7, 5, 8, 2, 2, 2, 2, 8, 8, 5, 8, 2, 0, 9, 2, 8, 9, 8], 6: [2, 5,
0, 0, 0, 5, 2, 2, 1, 8, 7, 7, 5, 4, 0, 2, 1, 5, 4, 5, 1, 2, 4, 0, 3, 0, 4, 5, 0, 5, 5, 5, 5, 2, 2, 5, 4, 4, 5, 5, 5, 3, 0],
7: [4, 9, 2, 1, 3, 2, 2, 9, 9, 9, 2, 2, 1, 2, 2, 9, 1, 9, 9, 3, 2, 9, 9, 8, 2, 9, 5, 9, 3, 9, 4, 4, 9, 9, 4, 1, 2, 9, 9, 8, 0, 9, 2,
2, 3, 9, 1, 4, 9, 9, 3, 9, 9, 9, 2, 2, 8, 8, 2, 9, 9, 9, 9, 9, 2, 9, 1, 3, 4, 2, 2, 2, 2, 2, 2, 2, 9, 9, 3, 9], 2: [9, 7, 7, 8, 1,
8, 8, 6, 9, 8, 8, 7, 0, 3, 3, 6, 6, 6, 1, 4, 9, 0, 7, 0, 8, 8, 3, 8, 1, 0, 4, 8, 8, 3, 3, 1, 3, 5, 8, 3, 8, 8, 1, 8, 4, 4, 7, 3,
8, 5, 4, 7, 1, 4, 3, 3, 3, 8, 8, 3, 8, 8, 8, 6, 6, 6, 8, 4, 8, 6, 6, 7, 8, 6, 9, 4, 4, 6, 8, 8, 8, 7, 8, 8, 3, 8, 4, 6, 1, 8, 5, 0, 0,
8, 8, 7, 8, 8], 9: [3, 8, 7, 8, 3, 7, 7, 3, 4, 4, 5, 8, 7, 7, 5, 4, 5, 2, 1, 4, 0, 0, 1, 1, 0, 4, 4, 4, 8, 7, 8, 1, 1, 3, 4, 4, 4, 3,
1, 4, 7, 4, 0, 7, 4, 4, 7, 1, 8, 7, 4, 5, 7, 7, 3, 8, 8, 0, 3, 3, 0, 0, 0, 3, 4, 7, 5, 0, 8, 5, 4, 7, 5, 7, 7, 4, 7, 7, 8, 4, 4,
4], 8: [7, 5, 5, 7, 2, 3, 1, 3, 4, 5, 3, 7, 2, 9, 4, 4, 5, 5, 3, 1, 6, 2, 1, 7, 5, 0, 5, 5, 9, 9, 3, 3, 5, 5, 9, 6, 2, 0, 0, 9, 5, 3,
9, 3, 2, 5, 0, 1, 3, 5, 5, 3, 3, 3, 3, 5, 5, 5, 2, 3, 5, 0, 7, 3, 4, 3, 7, 3, 7, 6, 0, 6, 6, 6, 4, 1, 5, 4, 0, 1, 1, 6, 6, 9, 1, 7, 4,
5, 5, 7, 9, 9, 7, 3, 5, 5, 9, 9, 5, 1, 0, 5, 5, 5, 6, 6, 6, 0, 6, 5, 9, 6, 5, 1, 6, 1], 1: [8, 2, 6, 5, 2, 6, 2, 3, 7, 3, 6, 5, 2, 2,
8, 2, 8, 8, 8, 8, 8, 8, 8, 8, 7], 0: [6, 7, 5, 5, 5, 5, 5, 7, 5, 4, 2, 6, 8, 3, 6, 6, 3, 7, 5, 5, 3, 5, 3, 5, 2]}
```

```
In [36]: train_size, train_score, val_score=learning_curve(pipe_minmax_logreg, data_train, train_y,
                                                           train_sizes=np.linspace(0.05,1,20), cv=3)
```

```
In [37]: plt.plot(train_size, train_score.mean(axis=1), label= "Jeu entrainement")
plt.plot(train_size, val_score.mean(axis=1), label= "Jeu de test")
plt.xlabel("Taille du jeux d'entrainement")
plt.ylabel("Précision")
_ = plt.legend()
```



Recherche de nouveau algo - KNN classifieur

```
In [39]: # On récupère les données et on en profite pour "aplatir" les images
data_train= train_x.reshape(train_x.shape[0], train_x.shape[1]*train_x.shape[2])
data_test= test_x.reshape(test_x.shape[0], test_x.shape[1]*test_x.shape[2])

etiquette= np.unique(test_y)

print(f"Entrainement: X Shape: {data_train.shape}, Y shape: {train_y.shape}\n"+\
      f"Test: X Shape: {data_test.shape}, Y shape: {test_y.shape}\n"+\
      f"Etiquette: {etiquette}")
```

```
Entrainement: X Shape: (60000, 784), Y shape: (60000,)
Test: X Shape: (10000, 784), Y shape: (10000,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]
```

```
In [40]: # On choisi les meilleurs paramètres en gardant la standardisation min/max
pipe_minmax_knn= make_pipeline(MinMaxScaler(), KNeighborsClassifier(n_neighbors= 6, weights= "distance",
                                                                    algorithm='ball_tree', p= 3))

pipe_minmax_knn.fit(data_train, train_y)
prediction = pipe_minmax_knn.predict(data_test)
```

```
Out[40]: Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                          ('kneighborsclassifier',
                           KNeighborsClassifier(algorithm='ball_tree', n_neighbors=6, p=3,
                                                  weights='distance'))])
```

```
In [42]: err, pre, d= erreur_prediction(valeurs_predites= prediction, valeurs_reelle= test_y)
print(f"{err} erreurs sur {len(data_test)} images soit une précision de {pre:.2f} %")
print(f"Dictionnaire des erreurs (clé valeur réelle: valeur prédite):\n{d}")
```

```
263 erreurs sur 10000 images soit une précision de 97.37 %
Dictionnaire des erreurs (clé valeur réelle: valeur prédite):
{4: [0, 9, 6, 1, 9, 9, 9, 6, 9, 1, 9, 8, 9, 9, 9, 6, 6, 9, 9, 9, 1, 7, 1, 9, 7, 1, 1, 9, 9], 3: [5, 7, 7, 5, 5, 9, 7, 4, 5, 2, 5, 5,
5, 7, 7, 5, 7, 5, 5, 8, 2, 7, 6, 8, 9, 5, 9, 5, 9, 5, 5, 9, 8], 9: [8, 7, 8, 4, 0, 3, 5, 3, 4, 2, 6, 1, 4, 7, 0, 7, 1, 7, 4, 1, 1, 3,
3, 2, 7, 7, 2, 4, 1, 4, 5, 3, 5, 0, 0, 0, 7, 7, 4], 2: [7, 8, 7, 6, 7, 7, 0, 7, 1, 0, 7, 6, 1, 1, 3, 7, 7, 3, 7, 7, 7, 0,
7, 0, 1, 7, 0, 0, 0, 7, 8, 0], 6: [4, 0, 5, 0, 1, 4, 5, 1, 0, 4, 0, 1, 0], 7: [4, 1, 4, 2, 1, 1, 1, 9, 9, 1, 1, 1, 1, 9, 1, 1, 1,
1, 9, 9, 9, 1, 1, 1, 9, 9, 9, 1, 1, 2, 2, 2], 8: [0, 1, 3, 3, 3, 9, 4, 3, 3, 7, 4, 5, 3, 2, 4, 2, 3, 3, 6, 6, 3, 5, 0, 0, 5, 2,
7, 9, 3, 7, 6, 0, 6, 2, 4, 9, 9, 7, 9, 4, 5, 5], 5: [9, 4, 3, 6, 9, 7, 3, 6, 8, 8, 3, 9, 9, 0, 8, 4, 6, 3, 6, 9, 6, 6, 0, 6, 0, 6], 1:
[2, 2, 6], 0: [6, 2, 7, 5, 6, 1, 6]}
```

```
312 erreurs sur 10000 images soit une précision de 96.88 % Dictionnaire des erreurs (clé valeur réelle: valeur prédite): {4: [0, 9, 6, 1, 9, 9, 9, 9, 6, 9, 1, 9, 8, 9, 9, 1, 9, 9, 9, 9, 9, 9, 1, 7, 1, 0, 9, 7, 1, 1, 9, 9, 0], 3: [1, 7, 7, 9, 5, 5, 9, 7, 4, 1, 2, 5, 5, 7, 7, 5, 7, 5, 8, 2, 6, 8, 5, 9, 5, 1, 2, 5, 5, 8, 5, 9, 5], 9: [8, 7, 3, 8, 4, 3, 3, 3, 3, 4, 2, 1, 6, 1, 1, 4, 7, 0,
4, 1, 1, 3, 3, 2, 0, 7, 2, 1, 7, 4, 7, 5, 3, 3, 5, 0, 0, 0, 7, 7, 7, 4], 2: [0, 7, 8, 7, 4, 7, 7, 0, 7, 0, 7, 1, 0, 7, 1, 0, 7, 6, 1, 1, 3, 7, 7, 3, 1, 7, 7, 0, 7, 0, 1, 8, 1, 7, 1, 0, 0, 0, 8, 7, 0], 6: [1, 4, 5, 1, 0, 4, 0, 1, 0], 7: [4, 1, 1, 2, 1, 1, 1, 9, 4, 9, 1, 1, 1, 4, 1, 1, 9, 1, 1, 1, 1, 1, 9, 9, 9, 1, 1, 1, 9, 9, 9, 9, 1, 1, 2, 2, 2], 8: [0, 1, 3, 4, 3, 3, 6, 9, 4, 5, 3, 5, 3, 2, 5, 7, 0,
5, 3, 2, 4, 1, 2, 3, 0, 3, 5, 0, 6, 3, 5, 0, 0, 5, 3, 3, 5, 2, 7, 9, 3, 7, 6, 0, 6, 2, 4, 5, 0, 6, 9, 1, 9, 3, 4, 5, 5], 5: [9, 4, 3, 6, 9, 7, 3, 3, 3, 6, 8, 3, 3, 3, 9, 0, 8, 4, 0, 3, 9, 3, 0, 3, 3, 3, 6, 0,
[2, 2], 0: [2, 7, 5, 6, 1, 6]}
```

Plutôt que d'intégrer le min max dans le modèle on demandera à ce que la normalisation soit faite avant

Pour avoir un modèle compatible avec les autre modèle pour le traitement et la comparaison avec les autres algorithmes, on "sort" la normalisation du modèle.

```
In [3]: # Importation du jeux d'entrainement. On réalise directement une normalisation min/max en divisant par 255.0
# Les pixels composant l'image étant compris entre 0 et 255
f = h5py.File("../datas/train.hdf5", 'r')
train_x, train_y = (f['image']/255.0, f['label'])
f.close()

# Importation du jeux de test
f = h5py.File("../datas/test.hdf5", 'r')
test_x, test_y = (f['image']/255.0, f['label'])
f.close()

etiquette= np.unique(test_y)
print(f"Entrainement: X Shape: {train_x.shape}, Y shape: {train_y.shape}\n"+\
      f"Test: X Shape: {test_x.shape}, Y shape: {test_y.shape}\n"+\
      f"Etiquette: {etiquette}")
```

```
Entrainement: X Shape: (60000, 28, 28), Y shape: (60000,)
Test: X Shape: (10000, 28, 28), Y shape: (10000,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]
```



```
In [4]: X_train= train_x.reshape(train_x.shape[0], train_x.shape[1]*train_x.shape[2])
Y_train= train_y

X_test= test_x.reshape(test_x.shape[0], test_x.shape[1]*test_x.shape[2])
Y_test= test_y
```

```
In [5]: model_knn= KNeighborsClassifier(algorithm='ball_tree', n_neighbors=6, p=3, weights='distance')
deb= time.time()
model_knn.fit(X_train, Y_train)
fin= time.time()
print(f"fit en {fin-deb} s")

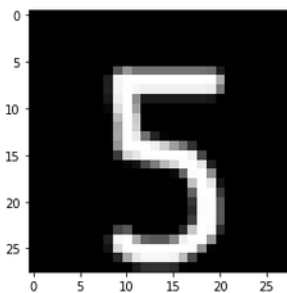
fit en 16.260356187820435 s
```

```
In [6]: dump(model_knn,"KNN_n6p3_distance_balltreep3.modele")
```

```
Out[6]: ['KNN_n6p3_distance_balltreep3.modele']
```

Test sur une image non issue de MNIST

```
In [6]: img= Image.open("./images/image_test.png")
_ = plt.imshow(img, cmap= "gray")
```



```
In [14]: #model_knn= load('./modeles/KNN_n6p3_distance_balltreep3.modele')
chiffre=np.array(img)/255.0
chiffre=chiffre.ravel()
tab= list(model_knn.predict_proba([chiffre])[0])
print(tab.index(max(tab)))
print(100 * np.round(tab,4))
```

```
5
[ 0.  0.  0.  0.  0. 100.  0.  0.  0.  0.]
```

Test d'un CNN

On normalisera le jeu de données en divisant chaque élément par 255.0 ce qui les transforme en flottant compris entre 0 et 1.

```
In [4]: # Importation du jeux d'entraînement
f = h5py.File("./datas/train.hdf5", 'r')
train_x, train_y = (f['image']/255.0, f['label'])[...]
f.close()

# Importation du jeux de test
f = h5py.File("./datas/test.hdf5", 'r')
test_x, test_y = (f['image']/255.0, f['label'])[...]
f.close()
```

```
In [5]: # On récupère les données et on les met en forme pour qu'elles soient compatible avec le CNN.
# la dimension des données de test et de train sont de type:
# (shape[0],shape[1], shape[2]), il sera nécessaire d'y ajouter une dimension supplémentaire
data_train_pour_cnn= train_x.reshape(train_x.shape[0], train_x.shape[1], train_x.shape[2], 1)
data_test_pour_cnn= test_x.reshape(test_x.shape[0], test_x.shape[1], test_x.shape[2], 1)

etiquette= np.unique(test_y)

# convert class vectors to binary class matrices
train_y_conv = keras.utils.to_categorical(train_y, len(etiquette))
test_y_conv = keras.utils.to_categorical(test_y, len(etiquette))

print(f"Entrainement: X Shape: {data_train_pour_cnn.shape}, Y shape: {train_y.shape}\n"+\
      f"Test: X Shape: {data_test_pour_cnn.shape}, Y shape: {test_y.shape}\n"+\
      f"Etiquette: {etiquette}")
```

```
Entrainement: X Shape: (60000, 28, 28, 1), Y shape: (60000,)
Test: X Shape: (10000, 28, 28, 1), Y shape: (10000,)
Etiquette: [0 1 2 3 4 5 6 7 8 9]
```

Conception du réseau

2 versions de se réseau ont été faite, une avec dropout, l'autre sans. Sur les jeux de test et d'entraînement, les courbes sont très proche et les résultats obtenus similaire. A voir sur des cas "réel" si les 2 réseaux se distinguent.

In [70]:

```
batch_size = 128
epochs = 250

input_shape= (train_x.shape[1], train_x.shape[2], 1)
nb_classe= len(etiquette)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape= input_shape ))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classe, activation='softmax'))
model.compile(loss= keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 26, 26, 32)	320
conv2d_9 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_4 (MaxPooling 2D)	(None, 12, 12, 64)	0
dropout_8 (Dropout)	(None, 12, 12, 64)	0
flatten_4 (Flatten)	(None, 9216)	0
dense_8 (Dense)	(None, 256)	2359552
dropout_9 (Dropout)	(None, 256)	0
dense_9 (Dense)	(None, 10)	2570

=====
Total params: 2,380,938
Trainable params: 2,380,938
Non-trainable params: 0

In [9]:

```
hist = model.fit(data_train_pour_cnn, train_y_conv, batch_size=batch_size, epochs=epochs,
                 verbose=1, validation_data=(data_test_pour_cnn, test_y_conv), workers=-1)
```

```
Epoch 1/251
469/469 [=====] - 278s 589ms/step - loss: 5.2512 - accuracy: 0.4805 - val_loss: 1.7019 - val_accuracy: 0.7154
Epoch 2/251
469/469 [=====] - 253s 540ms/step - loss: 1.2893 - accuracy: 0.7795 - val_loss: 0.9239 - val_accuracy: 0.8355
Epoch 3/251
469/469 [=====] - 255s 544ms/step - loss: 0.8350 - accuracy: 0.8492 - val_loss: 0.6893 - val_accuracy: 0.8729
Epoch 4/251
469/469 [=====] - 259s 551ms/step - loss: 0.6502 - accuracy: 0.8787 - val_loss: 0.5698 - val_accuracy: 0.8959
Epoch 5/251
469/469 [=====] - 259s 551ms/step - loss: 0.5416 - accuracy: 0.8957 - val_loss: 0.4912 - val_accuracy: 0.9078
Epoch 6/251
469/469 [=====] - 258s 551ms/step - loss: 0.4677 - accuracy: 0.9079 - val_loss: 0.4409 - val_accuracy: 0.9135
Epoch 7/251
469/469 [=====] - 259s 553ms/step - loss: 0.4127 - accuracy: 0.9160 - val_loss: 0.3986 - val_accuracy: 0.9194
Epoch 8/251
469/469 [=====] - 286s 609ms/step - loss: 0.3704 - accuracy: 0.9229 - val_loss: 0.3674 - val_accuracy: 0.9256
Epoch 9/251
469/469 [=====] - 253s 538ms/step - loss: 0.3355 - accuracy: 0.9294 - val_loss: 0.3433 - val_accuracy: 0.9291
Epoch 10/251
469/469 [=====] - 260s 555ms/step - loss: 0.3063 - accuracy: 0.9343 - val_loss: 0.3236 - val_accuracy: 0.9329
Epoch 11/251
469/469 [=====] - 257s 549ms/step - loss: 0.2822 - accuracy: 0.9384 - val_loss: 0.3013 - val_accuracy: 0.9363
Epoch 12/251
469/469 [=====] - 266s 566ms/step - loss: 0.2614 - accuracy: 0.9414 - val_loss: 0.2828 - val_accuracy: 0.9397
Epoch 13/251
469/469 [=====] - 269s 573ms/step - loss: 0.2427 - accuracy: 0.9451 - val_loss: 0.2710 - val_accuracy: 0.9419
Epoch 14/251
469/469 [=====] - 267s 570ms/step - loss: 0.2260 - accuracy: 0.9478 - val_loss: 0.2590 - val_accuracy: 0.9431
Epoch 15/251
469/469 [=====] - 285s 607ms/step - loss: 0.2128 - accuracy: 0.9510 - val_loss: 0.2473 - val_accuracy: 0.9462
Epoch 16/251
469/469 [=====] - 256s 545ms/step - loss: 0.1994 - accuracy: 0.9535 - val_loss: 0.2362 - val_accuracy: 0.9477
Epoch 17/251
469/469 [=====] - 255s 544ms/step - loss: 0.1881 - accuracy: 0.9553 - val_loss: 0.2271 - val_accuracy: 0.9490
Epoch 18/251
469/469 [=====] - 257s 547ms/step - loss: 0.1775 - accuracy: 0.9578 - val_loss: 0.2187 - val_accuracy: 0.9517
Epoch 19/251
469/469 [=====] - 266s 568ms/step - loss: 0.1682 - accuracy: 0.9595 - val_loss: 0.2129 - val_accuracy: 0.9521
Epoch 20/251
469/469 [=====] - 265s 565ms/step - loss: 0.1597 - accuracy: 0.9610 - val_loss: 0.2060 - val_accuracy: 0.9535
Epoch 21/251
469/469 [=====] - 289s 616ms/step - loss: 0.1516 - accuracy: 0.9629 - val_loss: 0.1990 - val_accuracy: 0.9549
Epoch 22/251
469/469 [=====] - 258s 551ms/step - loss: 0.1445 - accuracy: 0.9648 - val_loss: 0.1925 - val_accuracy: 0.9569
Epoch 23/251
469/469 [=====] - 256s 546ms/step - loss: 0.1375 - accuracy: 0.9659 - val_loss: 0.1877 - val_accuracy: 0.9563
Epoch 24/251
469/469 [=====] - 257s 548ms/step - loss: 0.1316 - accuracy: 0.9669 - val_loss: 0.1832 - val_accuracy: 0.9578
Epoch 25/251
469/469 [=====] - 260s 554ms/step - loss: 0.1255 - accuracy: 0.9687 - val_loss: 0.1800 - val_accuracy: 0.9575
Epoch 26/251
469/469 [=====] - 267s 570ms/step - loss: 0.1203 - accuracy: 0.9699 - val_loss: 0.1736 - val_accuracy: 0.9591
Epoch 27/251
```

```

Epoch 227/251
469/469 [=====] - 432s 921ms/step - loss: 6.8405e-04 - accuracy: 1.0000 - val_loss: 0.0909 - val_accuracy: 0.9808
Epoch 228/251
469/469 [=====] - 335s 715ms/step - loss: 6.7646e-04 - accuracy: 1.0000 - val_loss: 0.0912 - val_accuracy: 0.9801
Epoch 229/251
469/469 [=====] - 286s 609ms/step - loss: 6.5611e-04 - accuracy: 1.0000 - val_loss: 0.0913 - val_accuracy: 0.9806
Epoch 230/251
469/469 [=====] - 316s 673ms/step - loss: 6.5371e-04 - accuracy: 1.0000 - val_loss: 0.0912 - val_accuracy: 0.9806
Epoch 231/251
469/469 [=====] - 306s 652ms/step - loss: 6.6297e-04 - accuracy: 1.0000 - val_loss: 0.0908 - val_accuracy: 0.9803
Epoch 232/251
469/469 [=====] - 300s 639ms/step - loss: 6.4536e-04 - accuracy: 1.0000 - val_loss: 0.0909 - val_accuracy: 0.9808
Epoch 233/251
469/469 [=====] - 321s 685ms/step - loss: 6.2289e-04 - accuracy: 1.0000 - val_loss: 0.0920 - val_accuracy: 0.9800
Epoch 234/251
469/469 [=====] - 422s 901ms/step - loss: 6.2168e-04 - accuracy: 0.9999 - val_loss: 0.0910 - val_accuracy: 0.9806
Epoch 235/251
469/469 [=====] - 323s 689ms/step - loss: 6.0564e-04 - accuracy: 1.0000 - val_loss: 0.0910 - val_accuracy: 0.9810
Epoch 236/251
469/469 [=====] - 267s 568ms/step - loss: 5.9623e-04 - accuracy: 1.0000 - val_loss: 0.0915 - val_accuracy: 0.9803
Epoch 237/251
469/469 [=====] - 296s 632ms/step - loss: 5.9823e-04 - accuracy: 1.0000 - val_loss: 0.0912 - val_accuracy: 0.9804
Epoch 238/251
469/469 [=====] - 294s 628ms/step - loss: 5.7338e-04 - accuracy: 1.0000 - val_loss: 0.0910 - val_accuracy: 0.9810
Epoch 239/251
469/469 [=====] - 388s 827ms/step - loss: 5.7910e-04 - accuracy: 1.0000 - val_loss: 0.0911 - val_accuracy: 0.9808
Epoch 240/251
469/469 [=====] - 365s 779ms/step - loss: 5.4962e-04 - accuracy: 1.0000 - val_loss: 0.0910 - val_accuracy: 0.9808
Epoch 241/251
469/469 [=====] - 262s 559ms/step - loss: 5.5579e-04 - accuracy: 1.0000 - val_loss: 0.0908 - val_accuracy: 0.9810
Epoch 242/251
469/469 [=====] - 290s 619ms/step - loss: 5.3767e-04 - accuracy: 1.0000 - val_loss: 0.0912 - val_accuracy: 0.9810
Epoch 243/251
469/469 [=====] - 282s 600ms/step - loss: 5.3187e-04 - accuracy: 1.0000 - val_loss: 0.0915 - val_accuracy: 0.9812
Epoch 244/251
469/469 [=====] - 330s 704ms/step - loss: 5.2024e-04 - accuracy: 1.0000 - val_loss: 0.0910 - val_accuracy: 0.9808
Epoch 245/251
469/469 [=====] - 425s 907ms/step - loss: 5.2134e-04 - accuracy: 1.0000 - val_loss: 0.0909 - val_accuracy: 0.9807
Epoch 246/251
469/469 [=====] - 309s 659ms/step - loss: 5.1270e-04 - accuracy: 1.0000 - val_loss: 0.0912 - val_accuracy: 0.9812
Epoch 247/251
469/469 [=====] - 268s 572ms/step - loss: 5.0154e-04 - accuracy: 1.0000 - val_loss: 0.0908 - val_accuracy: 0.9808
Epoch 248/251
469/469 [=====] - 290s 619ms/step - loss: 4.8726e-04 - accuracy: 1.0000 - val_loss: 0.0911 - val_accuracy: 0.9807
Epoch 249/251
469/469 [=====] - 273s 583ms/step - loss: 4.7768e-04 - accuracy: 1.0000 - val_loss: 0.0905 - val_accuracy: 0.9813
Epoch 250/251
469/469 [=====] - 297s 633ms/step - loss: 4.9169e-04 - accuracy: 1.0000 - val_loss: 0.0912 - val_accuracy: 0.9809
Epoch 251/251
469/469 [=====] - 298s 636ms/step - loss: 4.6185e-04 - accuracy: 1.0000 - val_loss: 0.0916 - val_accuracy: 0.9809

```

Evaluation et sauvegarde du modèle.

Les noms seront par la suite changer pour pouvoir être manipuler plus simplement en fonction des contraintes du programme de visualition et de test.

```

In [11]: score = model.evaluate(data_test_pour_cnn, test_y_conv, verbose=1)
          print('Test loss:', score[0])
          print('Test accuracy:', score[1])

313/313 [=====] - 16s 52ms/step - loss: 0.0916 - accuracy: 0.9809
Test loss: 0.09160742163658142
Test accuracy: 0.98089998960495

In [ ]: nom_modele= f"mnist_{epochs}.modele"
          model.save(nom_modele)
          print(f"sauvegarde du modèle {nom_modele}.")

In [25]: hist.history

Out[25]: {'loss': [22.571229934692383,
                  10.335871696472168,
                  6.109948635101318,
                  4.1272125244140625,
                  3.1016037464141846,
                  2.4336371421813965,
                  2.041743278503418,
                  1.7853261232376099,
                  1.6038233041763306,

```

```

0.9800000190734863,
0.9801999926567078,
0.9801999926567078,
0.9803000092506409,
0.9801999926567078,
0.9804999828338623,
0.980400025844574,
0.980599994277954,
0.9807999730110168,
0.9803000092506409,
0.9803000092506409,
0.9803000092506409,
0.980599994277954,
0.9804999828338623,
0.9807999730110168,
0.9807000160217285,
0.9807999730110168,
0.9807999730110168,
0.9811000227928162,
0.98089998960495,
0.9810000061988831,
0.9810000061988831,
0.9810000061988831,
0.9811000227928162,
0.9812999963760376,
0.9810000061988831,
0.9811000227928162,
0.9811999797821045,
0.9811999797821045,
0.9814000129699707,
0.9815000295639038,
0.9814000129699707,
0.9817000031471252,
0.9817000031471252,
0.9817000031471252,
0.9819999933242798,
0.9818000197410583,
0.9817000031471252,
0.9821000099182129,
0.9819999933242798,
0.9818000197410583,
0.9819999933242798,
0.9817000031471252,
0.9821000099182129,

```

Tracé des courbes d'évolution de la perte et de la précision

In [63]:

```

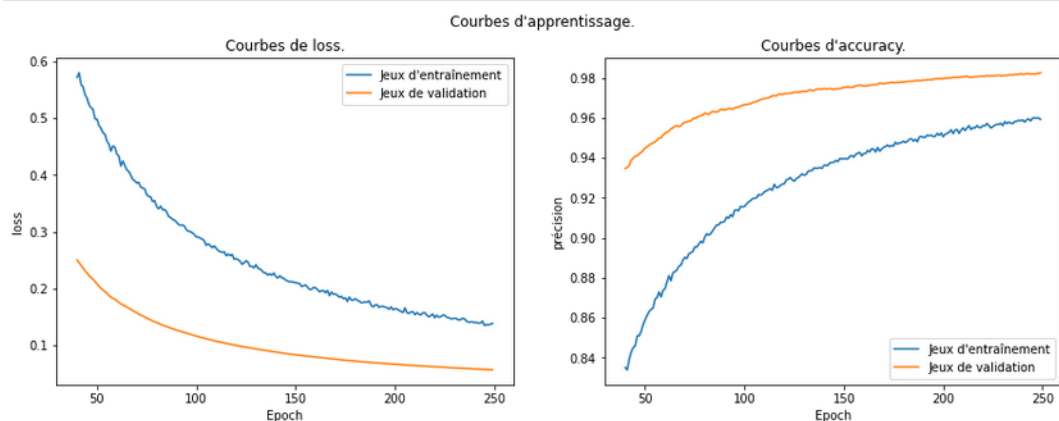
x= range(len(hist.history["loss"]))
debut= 40

fig, axes= plt.subplots(1, 2, figsize= (15,5))
fig.suptitle("Courbes d'apprentissage.")

axes[0].plot(x[debut:], hist.history["loss"][debut:], label="Jeux d'entraînement")
axes[0].plot(x[debut:], hist.history["val_loss"][debut:], label="Jeux de validation")
axes[0].set_title("Courbes de loss.")
axes[0].set_xlabel= "Epoch", ylabel="loss")
axes[0].legend()

axes[1].plot(x[debut:], hist.history["accuracy"][debut:], label="Jeux d'entraînement")
axes[1].plot(x[debut:], hist.history["val_accuracy"][debut:], label="Jeux de validation")
axes[1].set_title("Courbes d'accuracy.")
axes[1].set_xlabel= "Epoch", ylabel="précision")
_= axes[1].legend()

```



Autres réseaux de neurones

In [6]:

```
batch_size = 128
epochs = 200

input_shape= (train_x.shape[1], train_x.shape[2], 1)
nb_classe= len(etiquette)

model2 = Sequential()

model2.add(Conv2D(16, kernel_size=(3, 3),activation='relu',input_shape= input_shape ))

model2.add(Conv2D(32, (3, 3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.2))

model2.add(Conv2D(32, (3, 3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Flatten())

model2.add(Dense(256, activation='relu'))
model2.add(Dropout(0.3))

model2.add(Dense(nb_classe, activation='softmax'))
model2.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

model2.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 16)	160
conv2d_1 (Conv2D)	(None, 24, 24, 32)	4640
max_pooling2d (MaxPooling2D)	(None, 12, 12, 32)	0
dropout (Dropout)	(None, 12, 12, 32)	0
conv2d_2 (Conv2D)	(None, 10, 10, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 32)	0
flatten (Flatten)	(None, 800)	0
dense (Dense)	(None, 256)	205056
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570

=====
Total params: 221,674
Trainable params: 221,674
Non-trainable params: 0

2022-03-28 18:20:43.866453: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcuda.so.1'; dLError: libcuda.so.1: cannot open shared object file: No such file or directory
2022-03-28 18:20:43.866486: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (303)
2022-03-28 18:20:43.866511: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not appear to be running on this host (debian-jpphi): /proc/driver/nvidia/version does not exist
2022-03-28 18:20:43.867898: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

In [7]:

```
batch_size = 128
epochs = 200

input_shape= (train_x.shape[1], train_x.shape[2], 1)
nb_classe= len(etiquette)

model2 = Sequential()

model2.add(Conv2D(16, kernel_size=(3, 3),activation='relu',input_shape= input_shape ))

model2.add(Conv2D(32, (3, 3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.2))

model2.add(Conv2D(32, (3, 3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Flatten())

model2.add(Dense(256, activation='relu'))
model2.add(Dropout(0.3))

model2.add(Dense(nb_classe, activation='softmax'))
model2.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

model2.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 26, 26, 16)	160
conv2d_4 (Conv2D)	(None, 24, 24, 32)	4640
max_pooling2d_2 (MaxPooling2D)	(None, 12, 12, 32)	0

2D)		
dropout_2 (Dropout)	(None, 12, 12, 32)	0
conv2d_5 (Conv2D)	(None, 10, 10, 32)	9248
max_pooling2d_3 (MaxPooling 2D)	(None, 5, 5, 32)	0
flatten_1 (Flatten)	(None, 800)	0
dense_2 (Dense)	(None, 256)	205056
dropout_3 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 10)	2570

=====

Total params: 221,674
Trainable params: 221,674
Non-trainable params: 0

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Projet fin d'étude Simplon

Codé en mars 2022

@auteur: Jean-Pierre Maffre
"""

#----- Import, variables globales et fonctions -----
#
#-----

import streamlit as st
from streamlit_drawable_canvas import st_canvas

import os

import time

from PIL import Image, ImageOps

import tensorflow as tf

from joblib import load

import numpy as np

def charge_modeles(repertoire= None):
    """
    Retourne un dictionnaire dont la clé est le nom de l'algorithme et la valeur
    associée à la clé le nom du modèle binaire à charger.
    Entrée:
        nom du répertoire contenant les fichiers modeles
    Sortie:
        le dictionnaire{algo:nom_du_fichier}
    """
    assert type(repertoire)== str, "Le paramètre repertoire doit être de '+'\
    "type chaîne de caractères"

    liste_fichiers= os.listdir(repertoire)
    # On ne récupère que les fichiers d'extensions .modele
    liste_fichiers = [f for f in liste_fichiers if ".modele" in f]
    # Recherche du nom de l'algorithme contenue dans le nom fichier
    # Les fichiers doivent être écrit avec la syntaxe: algo_caractéristique.modele
    algo= [a.split("_")[0] for i,a in enumerate(liste_fichiers)]
    return {cle: valeur for cle, valeur in zip(algo, liste_fichiers)}

def sauvegarde(nom):
    imgpil= Image.open("./images/tmp.png")
    imgpil.save(nom)

ERR_CHARGEMENT_DICO= False
ERR_DICO_VIDE= False
ERR_TYPE_ALGO= False
ERR_FICHIER_TEST_ABS= False
TEST_LANCER= False

#----- Programme principal -----
#
#-----

st.set_page_config(page_title= "Lecture caractères manuscrits",
                    page_icon= "./images/icône.png")

dict_algo_fichier = charge_modeles(repertoire= "./modeles/")
algos = list(dict_algo_fichier.keys())
fichiers= list(dict_algo_fichier.values())

```



```

#----- Construction de la page - zone latérale -----
st.sidebar.markdown("**Options**")
# Épaisseur du tracé
epaisseur = st.sidebar.slider(label= "Épaisseur du trait", min_value= 1,
                               max_value= 25, value= 15, key= "ept")
nom_algo = st.sidebar.selectbox(label="Choix du modèle d'algorithme",
                                options= algos, key="btselect")
fichier_modele= dict_algo_fichier[nom_algo]

st.sidebar.markdown("**Sauvegarde**")

with st.sidebar.container():
    st.markdown(
        "Pour effectuer une sauvegarde de l'image appuyez sur le bouton "
        "correspondant à la vérité terrain. L'image sera enregistrée sous "
        "la forme 'chiffre_numero_unique.png'."
    )
    c21, c22, c23, c24, c25= st.columns(5)
    with c21:
        chif0= st.button(("0"), key="b0")
        chif5= st.button(("5"), key="b5")
    with c22:
        chif1= st.button(("1"), key="b1")
        chif6= st.button(("6"), key="b6")
    with c23:
        chif2= st.button(("2"), key="b2")
        chif7= st.button(("7"), key="b7")
    with c24:
        chif3= st.button(("3"), key="b3")
        chif8= st.button(("8"), key="b8")
    with c25:
        chif4= st.button(("4"), key="b4")
        chif9= st.button(("9"), key="b9")

    if chif0:
        sauvegarde(f"0_{int(time.time())}.png")
    if chif1:
        sauvegarde(f"1_{int(time.time())}.png")
    if chif2:
        sauvegarde(f"2_{int(time.time())}.png")
    if chif3:
        sauvegarde(f"3_{int(time.time())}.png")
    if chif4:
        sauvegarde(f"4_{int(time.time())}.png")
    if chif5:
        sauvegarde(f"5_{int(time.time())}.png")
    if chif6:
        sauvegarde(f"6_{int(time.time())}.png")
    if chif7:
        sauvegarde(f"7_{int(time.time())}.png")
    if chif8:
        sauvegarde(f"8_{int(time.time())}.png")
    if chif9:
        sauvegarde(f"9_{int(time.time())}.png")

st.sidebar.markdown("**Zone de test**")
st.sidebar.markdown(
    "Pour effectuer lancer un test de non régression... un clic suffit :) ! "
    "On chargera l'ensemble des modèles disponibles que l'on testera. **Le "
    "résultat des tests s'affiche en bas de page**."
)
test= st.sidebar.button("Test", key="btst")

#----- Construction de la page - zone principale -----

st.title("🌿 Lecture caractères manuscrits")
Image_illustration = Image.open("./images/image_titre.png")

```

```

st.image(Image_illustration)
st.markdown(
    "Après la théorie et l'étude sur les différents algorithmes "
    "sur des données provenant du même dataset, il est temps de "
    "passer à la pratique !\n\n"
    "Comme on peut le voir sur l'image illustrant cette page il y "
    "une très grande diversité dans les écritures. Nos modèles ne "
    "devraient donc pas avoir de difficultés majeures pour s'adapter "
    "à un style d'écriture qu'il n'a jamais rencontré...\n\n"
)

st.subheader("De la théorie, à la pratique:")

with st.form("Prédiction"):
    c1, c2= st.columns(2)
    with c1:
        recon_canvas = st_canvas(
            #epaisseur = st.slider("Epaisseur du trait: ", 1, 25, 15)
            # Fixed fill color with some opacity
            fill_color="rgba(0, 165, 0, 0.3)", #"rgba(255, 165, 0, 0.3)"
            stroke_width= epaisseur,
            stroke_color="#FFFFFF",
            background_color="#000000",
            update_streamlit=True,
            height=280,
            width=280,
            drawing_mode= "freedraw",
            key="recon_canvas",
        )
    with c2:
        irma_dit = st.form_submit_button("Prédiction")

    if irma_dit:
        # on charge l'image et on l'a converti
        img= recon_canvas.image_data
        image= Image.fromarray(img)
        image= image.resize((28,28))
        # Conversion de l'image au format rgba en tableau numpy 2d
        image= ImageOps.grayscale(image)
        image.save("./images/tmp.png")

        chiffre= np.array(image)/255.0

        # On charge le modèle
        id_algo= nom_algo[0:3]
        if id_algo== "Reg" or id_algo== "KNN":
            modele= load(f'./modeles/{fichier_modele}')
            tab= list(modele.predict_proba([chiffre.ravel()])[0])
        elif id_algo== "CNN":
            modele = tf.keras.models.load_model(f'./modeles/{fichier_modele}')
            tab= list(modele.predict(chiffre.reshape(1,28,28)).reshape(10))

        prediction= tab.index(max(tab))
        # On affiche la prédiction et le tableau des probabilités
        ch= ""
        for i in range(len(tab)):ch+= f"p({i})= {100*tab[i]:.2f} %, "
        col1, col2= st.columns(2)
        with col1:
            st.image(image)
        with col2:
            st.write(f"Chiffre lu: {prediction}")

        st.write(f"Tableau des probabilités:\n{ch[:-2]}")
        st.write(f"fichier modèle: {fichier_modele}")

st.markdown(
    "Pour retrouver ce projet, ainsi que les projets réalisés lors de la "
    "formation 'Développeur Data IA' effectué à simplon: "
    "[jpphi - github](https://github.com/jpphi)"
)

```

```

#----- tests de non régression -----
#
#-----

if test:
    TEST_LANCER= True
    st.markdown(
        "**Zone de test**<br>Déroulement du test:<br>"
        "1- Les modèles peuvent-ils être chargés ?<br>"
        "2- Chaque modèle sera testé sur une image test."
        , unsafe_allow_html= True)

    try:
        dict_algo_fichier = charge_modeles(repertoire= "./modeles/")
        if dict_algo_fichier== {}: ERR_DICO_VIDE= True
    except:
        ERR_CHARGEMENT_DICO= True

    if ERR_CHARGEMENT_DICO:
        st.markdown(
            "**Erreur:**<br><span style='color:red'> Le répertoire dans "
            "lequel les modèles sont sauvegardés n'existe pas !</span><br>"
            , unsafe_allow_html= True)

    elif ERR_DICO_VIDE:
        st.markdown(
            "**Erreur:**<br><span style='color:red'> Aucun algorithme n'a pu "
            "être chargé !</span><br>", unsafe_allow_html= True)

    else: # on continue les tests
        algos = list(dict_algo_fichier.keys())
        fichiers= list(dict_algo_fichier.values())
        try:
            chiffre_test= Image.open("./images/image_test.png")
            chiffre_test= np.array(chiffre_test)/255.0

        except:
            st.markdown(
                "**Erreur:**<br><span style='color:red'> Fichier test absent "
                "!!</span><br>", unsafe_allow_html= True)
            ERR_FICHIER_TEST_ABS= True

    if not(ERR_FICHIER_TEST_ABS):

        for i in range(len(algos)):
            # on charge le modèle
            id_algo= nom_algo[0:3]
            if id_algo== "Reg" or id_algo== "KNN":
                modele= load(f'./modeles/{fichier_modele}')
                tab= list(modele.predict_proba([chiffre_test.ravel()])[0])
            elif id_algo== "CNN":
                modele = tf.keras.models.load_model(f'./modeles/{fichier_modele}')
                tab= list(modele.predict(chiffre_test.reshape(1,28,28)).reshape(10))
            else:
                st.markdown(
                    "**Erreur:**<br><span style='color:red'> Algorithme de "
                    "type inconnu !</span><br>", unsafe_allow_html= True)
                ERR_TYPE_ALGO= True
                break

        prediction= tab.index(max(tab))
        if prediction!= 5:
            st.write(f"Modèle: {algos[i]}, fichier: {fichiers[i]}:")
            st.markdown(
                "**Erreur:**<br><span style='color:blue'> Erreur de "
                "prédiction ! Ceci ne remet pas pour autant en cause "
                "le programme, une part d'aléatoire existe dans les "

```

```
        "prédictions </span><br/>", unsafe_allow_html= True)

if not(ERR_CHARGEMENT_DICO) and not(ERR_DICO_VIDE) and not(ERR_TYPE_ALGO) and \
    not(ERR_FICHER_TEST_ABS) and TEST_LANCER:
    st.markdown("**Tests passés avec succès, pas d'erreur de fonctionnement**")
    TEST_LANCER= False
```