Application de reconnaissance de chiffres manuscrits

Présenté par Anne-Sophie, Bastien, Eva et Laurent

Sommaire

- Contexte du projet
- Présentation de MNIST
- Schéma d'architecture
- Choix du bon modèle : Régression Logistique, KNN, SVC ou RandomForest ?
- Computer Vision/Preprocessing avec OpenCV
- Text-To-Speech avec pyttsx3
- Présentation de l'application JoyMNIST

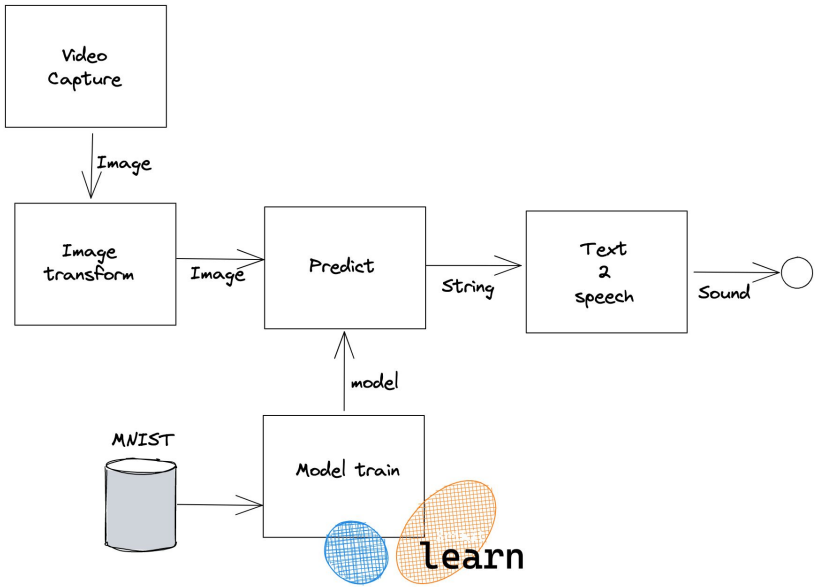
Contexte du projet

Développement d'une application d'Intelligence Artificielle capable au travers d'une webcam de reconnaître des chiffres manuscrits et d'indiquer via synthèse vocale les chiffres reconnus.

Cette application intègre les différentes fonctionnalités suivantes :

- Une fonctionnalité de Computer Vision pour numériser via la Webcam le chiffre présenté ;
- Une fonctionnalité de Preprocessing qui prépare l'image à l'analyse par l'Intelligence Artificielle;
- Une Intelligence Artificielle, entraînée sur la base de données MNIST capable de reconnaître les chiffres manuscrits ;
- Une fonctionnalité de Text-To-Speech qui effectue la synthèse vocale du chiffre manuscrit reconnu.

Schéma d'architecture de l'application IA



MNIST (Modified ou Mixed National Institute of Standards and Technology)

Le “Hello World” du Machine Learning

- Base de données de chiffres écrits à la main
- Regroupe 60000 images d’apprentissage et 10000 images de test
- Les données sont issues d’une base de données antérieure, appelée simplement NIST1
- Images en noir et blanc, normalisées centrées de 28 pixels de côté.

RandomForest

preprocessing, maxAbsScaler

paramétrage avec gridsearch

classification report

cross validation score

```
rf = RandomForestClassifier(bootstrap = False,
                            max_depth = None,
                            max_features = 'auto',
                            min_samples_leaf = 1,
                            min_samples_split = 5,
                            n_estimators = 100)
rf.fit(X_train,y_train)
```

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 100, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [None, 2, 4]
# Minimum number of samples required to split a node
min_samples_split = [2, 5]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2]
# Method of selecting samples for training each tree
bootstrap = [True, False]
```

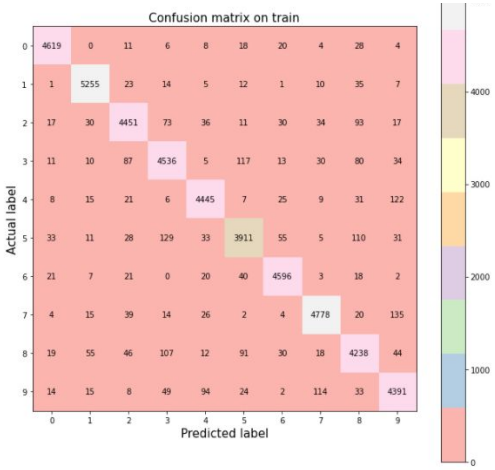
Classification Report

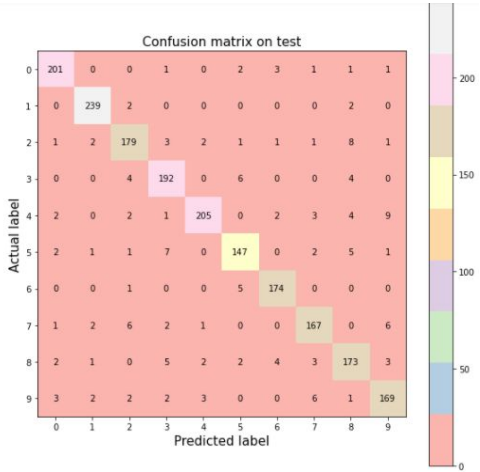
	precision	recall	f1-score	support
0	0.97	0.98	0.97	813
1	0.99	0.98	0.98	961
2	0.96	0.97	0.96	860
3	0.96	0.94	0.95	863
4	0.96	0.96	0.96	827
5	0.97	0.97	0.97	756
6	0.97	0.98	0.98	841
7	0.97	0.96	0.97	899
8	0.95	0.95	0.95	768
9	0.95	0.94	0.94	812
accuracy			0.96	8400
macro avg	0.96	0.96	0.96	8400
weighted avg	0.96	0.96	0.96	8400

```
#Cross validation
score = cross_val_score(rf, df_x, df_y)
print (np.mean(score))
```

```
0.9671666666666667
```

Régression Logistique





Preprocessing avec MaxAbsScaler

```
from sklearn.preprocessing import MaxAbsScaler
from sklearn import preprocessing

max_abs_scaler = preprocessing.MaxAbsScaler()
X_train_maxabs = max_abs_scaler.fit_transform(X_train)
```

Paramétrages avec GridSearchCV

```
# choosing the model and its parameters
# default solver is incredibly slow thats why we change it to lbfgs
model = LogisticRegression(solver='lbfgs', max_iter= 1000, n_jobs= -1)
```

```
# selecting the hyperparameters with GridSearchCV()
from sklearn.model_selection import GridSearchCV

# Set of parameters we want to try out for our model
param_grid_digits = {'C':[0.0001, 0.001, 0.01, 0.1, 1.0] }


model_grid = GridSearchCV(model, param_grid=param_grid_digits, cv=3)
```

```
%%time
model_grid.best_params_
```

```
Wall time: 0 ns

{'C': 0.0001}
```

Régression Logistique

Classification report on Train

	precision	recall	f1-score	support
0	0.97	0.98	0.98	4718.00
1	0.97	0.98	0.98	5363.00
2	0.94	0.93	0.93	4792.00
3	0.92	0.92	0.92	4923.00
4	0.95	0.95	0.95	4689.00
5	0.92	0.90	0.91	4346.00
6	0.96	0.97	0.97	4728.00
7	0.95	0.95	0.95	5037.00
8	0.90	0.91	0.91	4660.00
9	0.92	0.93	0.92	4744.00
accuracy	0.94	0.94	0.94	0.94
macro avg	0.94	0.94	0.94	48000.00
weighted avg	0.94	0.94	0.94	48000.00

Classification report on Test

	precision	recall	f1-score	support
0	0.95	0.96	0.95	210.00
1	0.97	0.98	0.98	243.00
2	0.91	0.90	0.90	199.00
3	0.90	0.93	0.92	206.00
4	0.96	0.90	0.93	228.00
5	0.90	0.89	0.89	166.00
6	0.95	0.97	0.96	180.00
7	0.91	0.90	0.91	185.00
8	0.87	0.89	0.88	195.00
9	0.89	0.90	0.89	188.00
accuracy	0.92	0.92	0.92	0.92
macro avg	0.92	0.92	0.92	2000.00
weighted avg	0.92	0.92	0.92	2000.00

KNN

```
grid_params = {'n_neighbors': list(range(1,15))}

gs = GridSearchCV(KNeighborsClassifier(),
                  grid_params,
                  verbose = 1,
                  n_jobs = -1)

gs_results = gs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
```

```
gs_results.best_score_
```

```
0.9683333333333334
```

```
gs_results.best_estimator_
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
gs_results.best_params_
```

```
{'n_neighbors': 3}
```

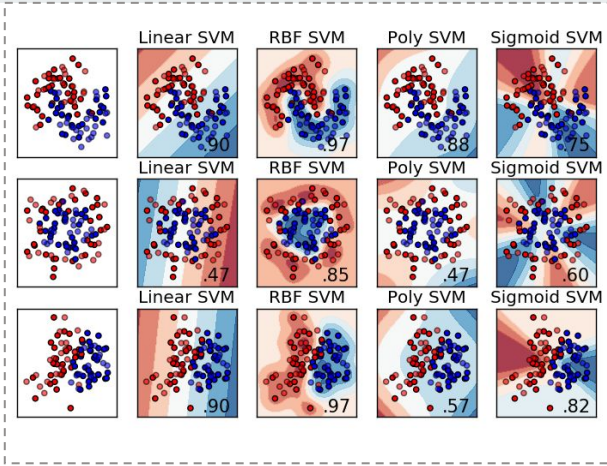
	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.96	1.00	0.98	1135
2	0.98	0.97	0.97	1032
3	0.96	0.97	0.96	1010
4	0.98	0.97	0.97	982
5	0.97	0.96	0.96	892
6	0.98	0.99	0.98	958
7	0.96	0.96	0.96	1028
8	0.99	0.94	0.96	974
9	0.96	0.96	0.96	1009
accuracy			0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000

```
accuracy_score(y_test, y_pred)
```

```
0.9705
```

SVC

Cover's theorem - "given a set of training data that is not linearly separable, with high probability it can be transformed into a linearly separable training set by projecting it into a higher-dimensional space via some non-linear transformation"



Pros of SVM

High stability due to dependency on support vectors and not the data points.
Does not get influenced by Outliers.
No assumptions made of the datasets.
Numeric predictions problem can be dealt with SVM.

Cons of SVM

Blackbox method.
Inclined to overfitting method.
Very rigorous computation.

Accuracy Score

0.9771

SVC(C=6.7046885962229785, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.004147722987833689, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

https://www.analyticssteps.com/blogs/how-does-support-vector-machine-algorithm-works-machine-learning

Analyse de comparaison

- Le score :
  - RandomForest = 0.9673
  - LogisticRegression = 0.9201
  - KNN = 0.9705
  - SVC = 0.9771
- La rapidité
  - RandomForest = 2min 29sec
  - LogisticRegression = instantané
  - KNN = 1min 44sec
  - SVC = 21min 48sec

Computer Vision/Preprocessing avec OpenCV

Nous avons utilisé OpenCV pour la capture et le traitement des images, en s'inspirant largement des exemples présents dans la documentation.

- tracking du chiffre affiché
- mise au format 28x28
- affichage de l'image traitée et de la prédiction à côté du chiffre

Text-To-Speech avec *pyttsx3*

Nous avons utilisé pyttsx3 pour prononcer le chiffre des prédictions sans paramétrage particulier.

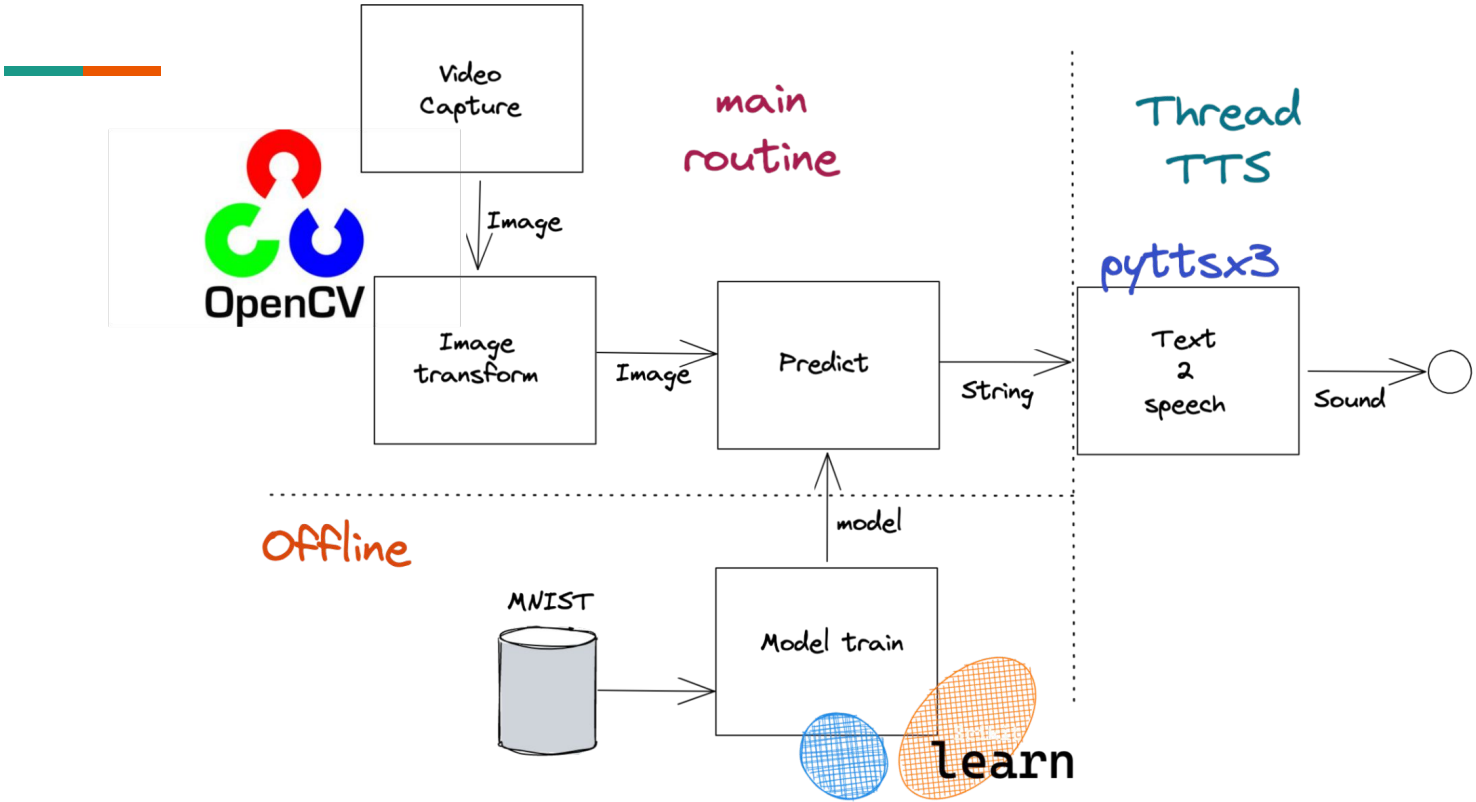
Afin d'améliorer les performances et ne pas bloquer la routine principale, cette tâche tourne en parallèle dans un thread avec laquelle nous communiquons via une queue.

Voting Ensemble

En pratique, selon la graphologie, les 4 modèles se trompent et réussissent différemment.

Pour cette raison, nous avons choisi de prédire les chiffres en fonction du vote majoritaire des 4 algorithmes.

```
# vote for predictions
digit = most_common([lr_digit, knn_digit, rf_digit, svc_digit])
```


Video Capture
main routine
Thread TTS
pyttsx3
OpenCV
Image
Image transform
Image
Predict
String
Text 2 speech
Sound
offline
model
MNIST
Model train
learn

JoyMNIST

https://github.com/evaliza/joymnist