haute école arc ingénierie
neuchâtel berne jura www.he-arc.ch

Traitement d'image

Prédiction d'un chiffre manuscrit avec MNIST

Réalisé par Griesser Gabriel, Feuillade Julien, Fasmeyer Florian et Luca Srdjenovic

Classe INF3dlm-a

6 Mai 2018

Table des matières

Prérequis

Afin de pouvoir utiliser correctement le logiciel, il est très fortement recommandé d'avoir :

- Python 3 ou version plus haut
- OpenCV **3**.0 ou plus haut

Ainsi que les packages suivant :

- Pygame 1.9.3
- Scikit-learn

Introduction

Le but de ce projet est de développer un logiciel capable de lire des chiffres manuscrites. Le logiciel en question permet à l'utilisateur d'écrire ses propres chiffres ou de lire des chiffres directement depuis une image/photo. On peut envoyer une à plusieurs images en même temps.

Des algorithmes essaient ensuite de deviner la valeur de ces chiffres en fonction des images.

Phase de pré-traitement (preprocessing)

Pour que nos algorithmes de prédictions fonctionnent, il nous faut d'abord traiter les images que l'on reçoit, c'est à dire qu'il faut passer d'abord par un traitement de l'image pour récupérer uniquement les éléments qui nous intéressent (phase de segmentation).

A noter que ce traitement ,marche pour nos dessins réalisés depuis notre programme. Par contre, quand il s'agit d'une photo/image importer, il faut faire attention car les nuances de forte couleur ou de luminosité de fonctionne pas.

Esquisse de contour

Pour commencer, on reçoit une image, on va d'abord la convertir en échelle de gris (grayscale). Ensuite on applique **cv2.GaussianBlur()**, c'est à dire, qu'on applique une convolution à l'image en gris avec une fonction Gaussienne. Une fois ceci fait, nous pouvons faire un threshold de l'image en gris. Le threshold consiste à faire un seuillage, donc de la segmentation dans notre image. Une fois ceci fait, on va pouvoir trouver les contours des différents chiffres se trouvant dans l'image grâce à la fonction **cv2.findContours().** Cette fonction nous retourne toutes les contours, donc à partir de ça il me suffit de parcours chaque contour seul qui correspond à un chiffre.

Extraction d'un chiffre

Dans chaque contours se trouve un chiffre, il nous suffit de parcourir l'intérieur de la zone de contour pour extraire notre chiffre. Notre contour ressemble à un rectangle, donc on va récupérer ses sommets et dessiner pour chaque chiffre son rectangle de contour avec la fonction **cv2.rectangle()**. C'est dans ce rectangle que l'on va parcourir pour récupérer chaque chiffre. Il suffit d'indiquer la zone d'intérieur du rectangle (hauteur, largeur) dans notre image récupéré par le threshold. Le threshold va nous permettre de pouvoir faire de des opérations morphologiques directement.

Opérations morphologiques

Dès qu'on récupérer un chiffre depuis une zone de contour, nous allons faire une transformation morphologique, plus précisément une dilatation avec la fonction **cv2.dilate()** qui va permettre de pouvoir augmenter le niveau de blanc. La raison est que nos algorithmes ont besoin de traiter un chiffre blanc sur couleur noir. Une fois fait, je dois redimensionner l'image en 28x28 pixel pour qu'elle soit conforme aux normes des données du MNIST.

Exemple de preprocessing

Exemple de preprocessing d'une image dessinée depuis notre programme :

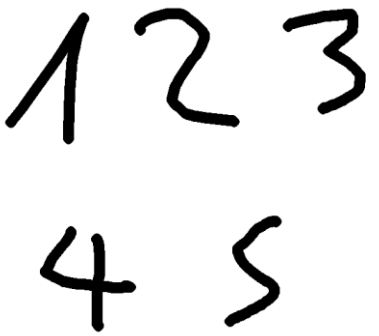


Image originale

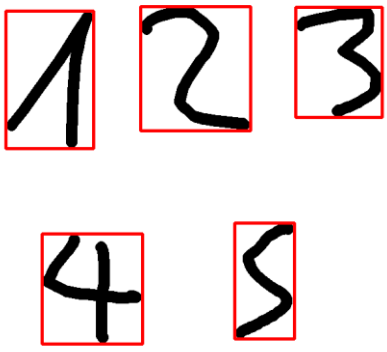
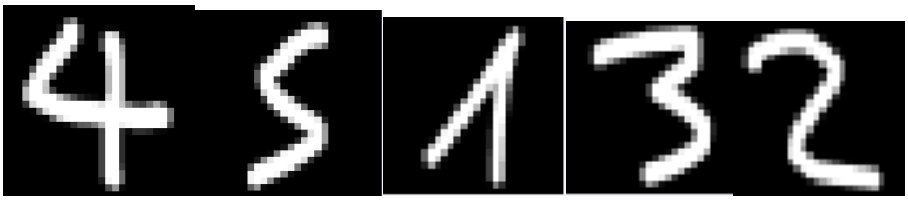


image findcontours

Sortie preprocess :



Exemple avec une vrai image :

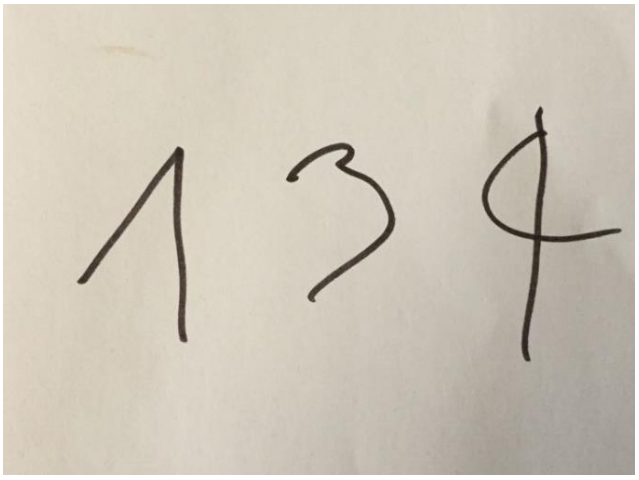


Image normale

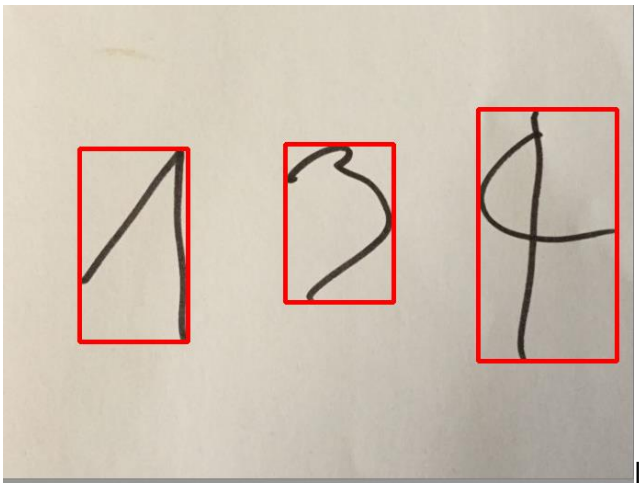
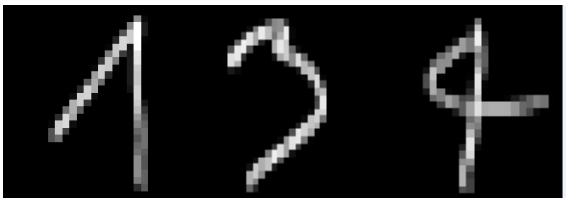


Image contour

Sortie preprocess :



Neural Network

Fait main

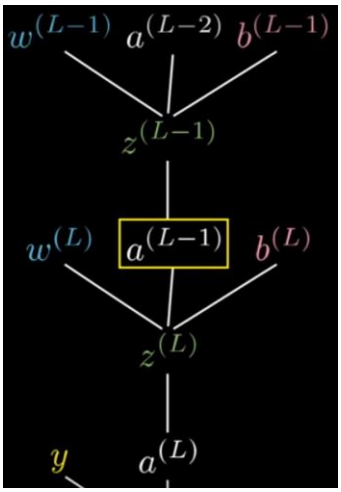
Nous avon commencé à coder un Deep Neural Network fait main. le système de neurones est fonctionnel et peut, être utilisé. L'apprentissage se fait avec de la back propagation. la backpropagation n'ayant pas pu être finir à temps, nous nous sommes mis à développer le système de neurones à l'aide de scikit learn.

DNN

Le dnn permet de créer des layers, chaque layer peut utiliser une fonction d'activation différente. Les layers sont composés de neurones, ces neurones contiennent le bias. En plus de tout cela, une table de poids est ajouté au dnn afin de pouvoir simuler les synapses

Backpropagation

La back propagation est basée sur le modèle de dérivées partielles suivant: $\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$



- *Co* étant l'erreure *Co = (Y - Solution)^2*
- *a* étant la fonction d'activation (choix entre sigmoid, tanh, ...)
- z étant la valeur du neurone AVANT activation
- et w, b, a-1 étant respectivement, le poid ou le biat ou le neurone précédent. L'on remplace la dérivée partielle de gauche en fonction de ce que l'on veut calculer.

Scikit-learn

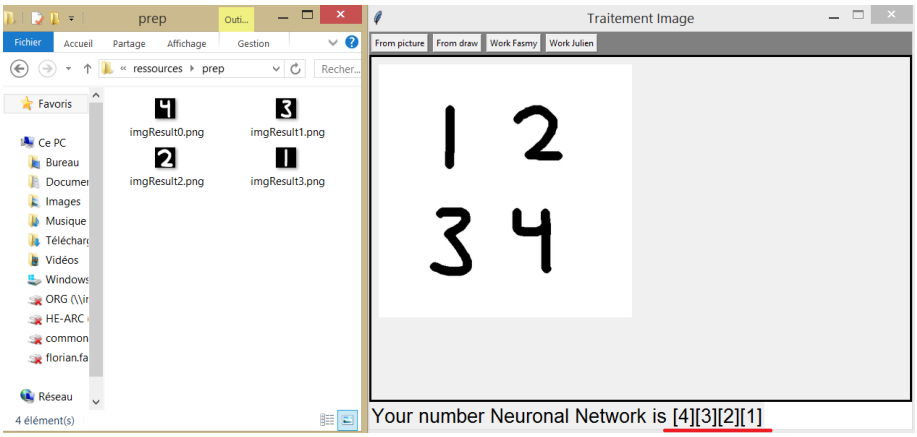
N'ayant pas eu le temps de terminer notre algorithme principale, nous avons terminé notre logiciel avec un dnn fait à l'aide de Scikit-learn.

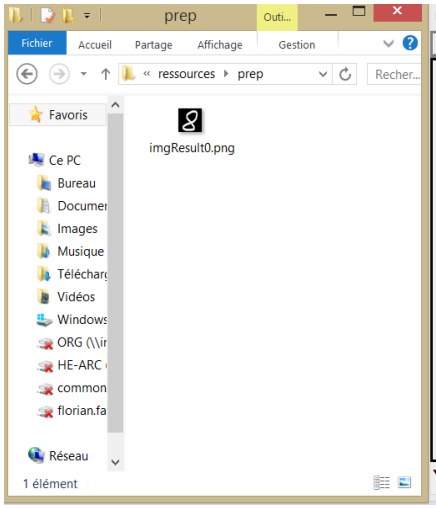
Le code à été réalisé sur le jupyter notebook, permettant de tester les méthodes en live sans pour autant avoir à charger le code ayant été exécuté précédemment.

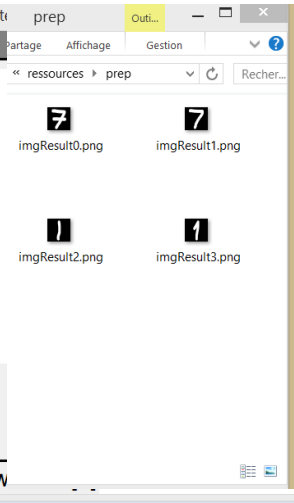
Le code à été séparé en deux parties: Celle où l'on crée, entraîne et sauvegarde l'algorithme et celle ou l'on charge et utilise l'algorithme.

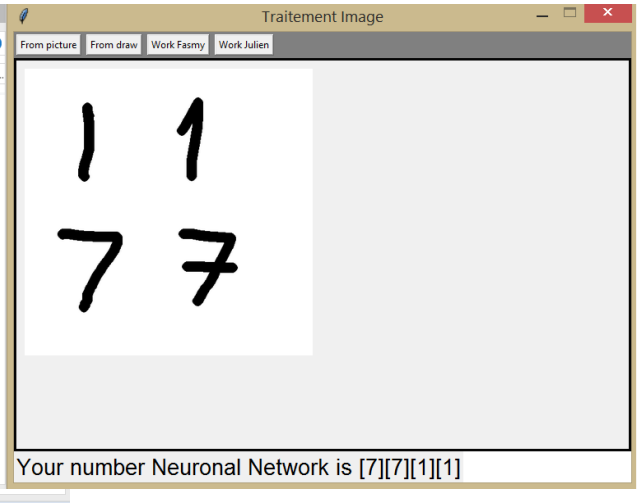
Les résultats n'étant pas très probants, nous avons utilisé un Support Vector Clustering au lieu du DNN.

Les résultats se sont révélées effectives et permettent de prédire avec une bonne efficacité la valeur affichée par les images. *Note: Dans les résultat suivants, nous utilisons un SVC, pas un Neural Network comme indiqué*.









Méthodes des plus proches voisins (KNN)

Le datamining emploie des techniques et des algorithmes issus de disciplines scientifiques divers telles que les statistiques, l'intelligence artificielle ou l'informatique pour construire des modèles à partir des données.

Parmi les techniques utilisées, il y a la méthode de k plus proche voisin (ou **KNN** de l'anglais K-*Nearest Neighbor*). L'objectif de ce dernier est de classer les exemples non étiquetés sur la base de leur similarité avec les exemples de la base d'apprentissage.

Pour faire simple, il faut entraîner l'algorithme afin qu'il puisse prédire.

Apprentissage

Afin d'utiliser au mieux l'algorithme de KNN, on a décidé d'entraîner la méthode avec **MNIST** qui est une énorme base de donnée de chiffres écrits à la main. C'est un jeu de données très utilisé en apprentissage automatique.

Bibliothèque

La bibliothèque **sklearn** est utilisé afin de faciliter l'importation de MNIST et de simplifier l'utilisation de l'algorithme.

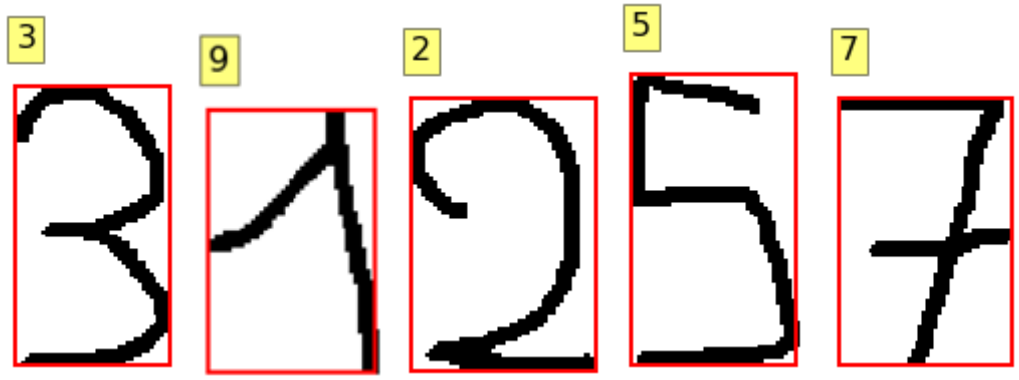
Utilisation

Lors de la première utilisation, notre algorithme va télécharger les données MNIST puis va apprendre directement et en faire un fichier «**.sav** ».

Cette opération peut prendre quelques minutes puisqu'il doit créer le fichier « **.sav** ».

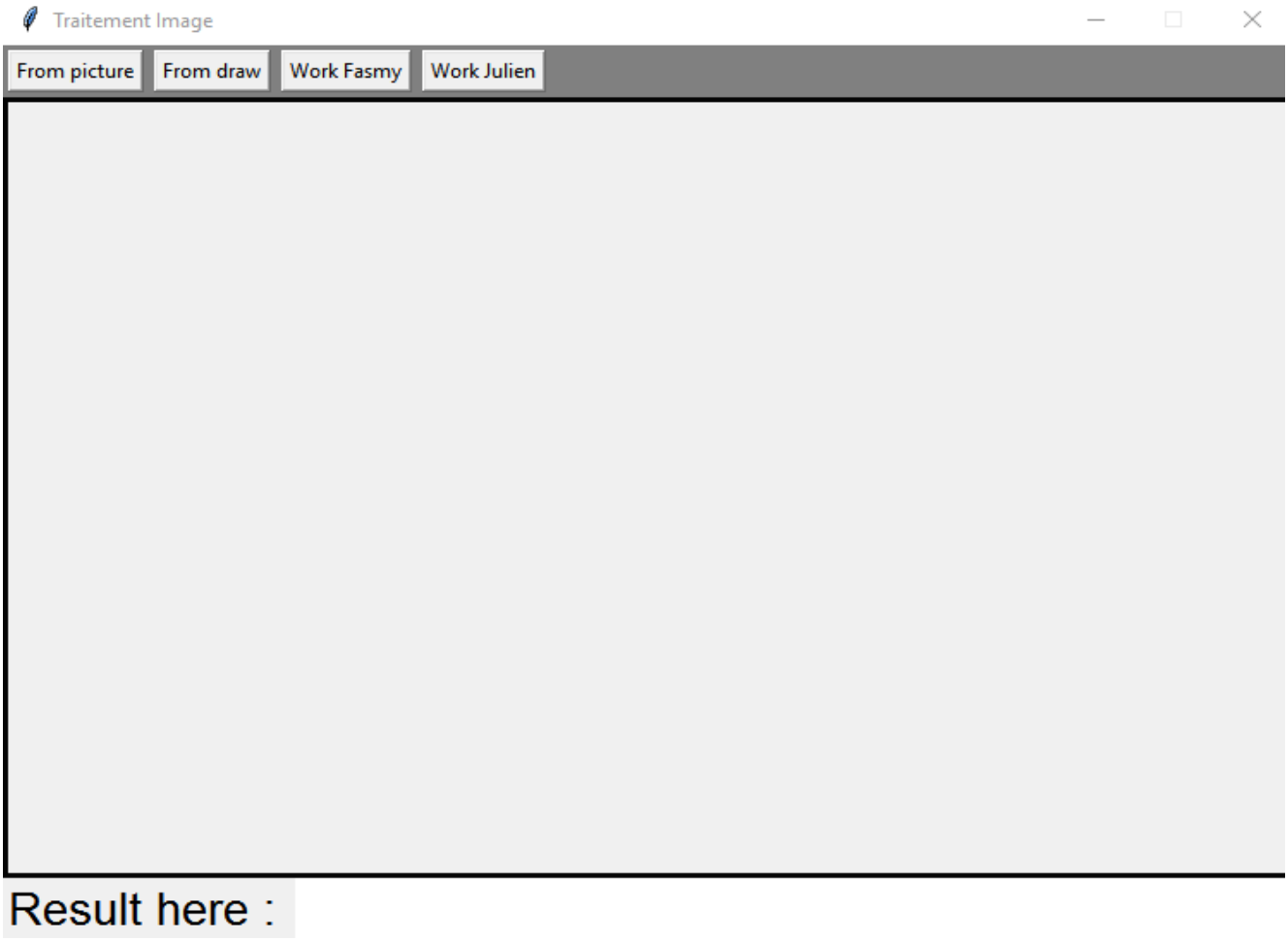
Résultat

L'algorithme n'est fiable qu'à environ 90%, ce qui peut donner des loupés dans la prédiction du chiffre dessiner.



Interface graphique

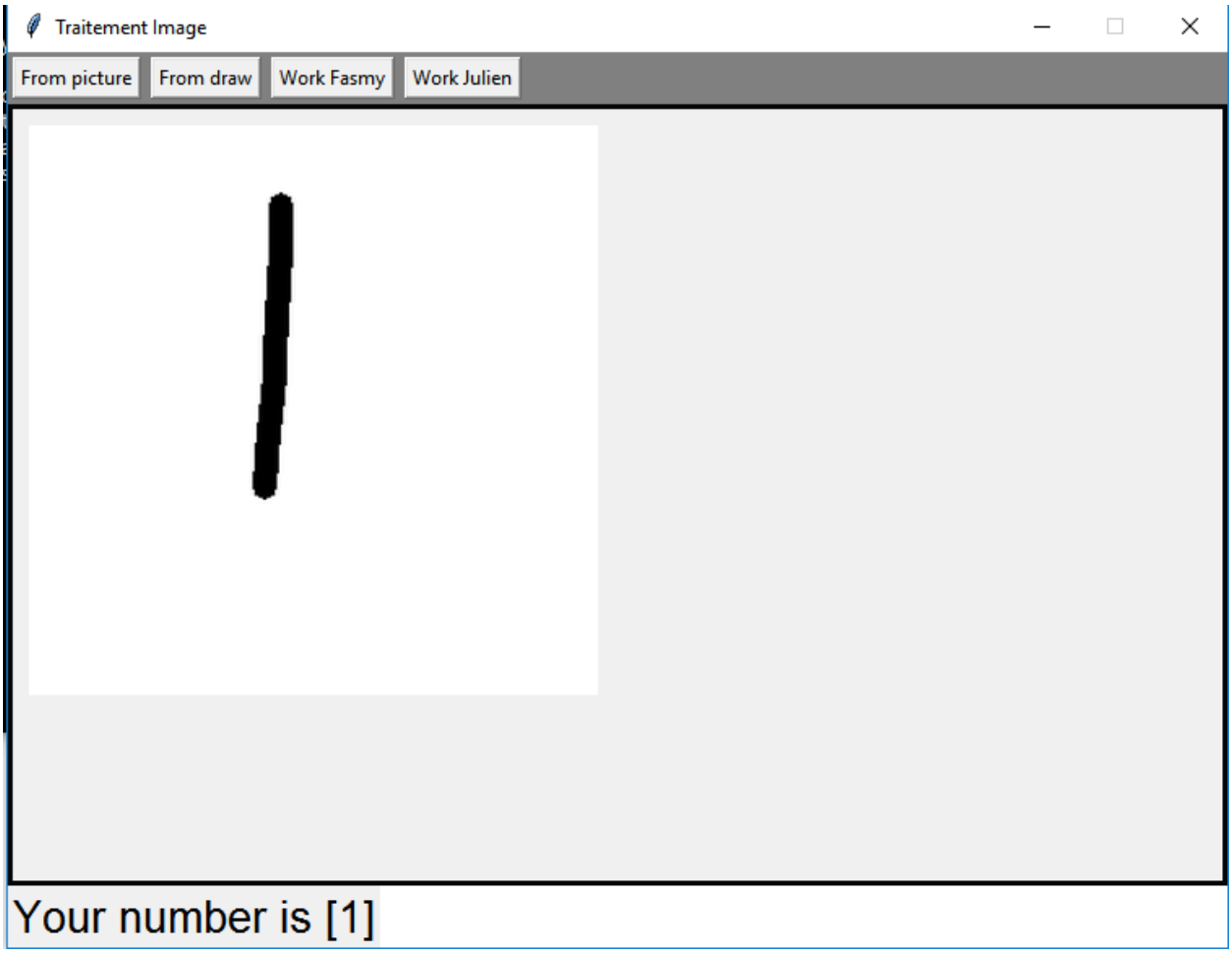
L'interface graphique a été faite avec le module TKinter. Cette dernière se présente comme



ceci :

Il y a deux façons de commencer. Soit en cliquant sur le premier bouton *from picture*, ce qui va vous laisser choisir un fichier image à traiter, soit en cliquant sur *from draw*. Ce dernier bouton ouvrira une fenêtre *PyGame* 800x800 sur laquelle vous pourrez dessiner votre nombre. Appuyez sur **espace** pour enregistrer le dessin ou **echap** pour réinitialiser la fenêtre *PyGame*.

Une fois le dessin enregistré ou l'image choisie, elle sera affichée le canvas de gauche de l'UI.



La seconde étape est de choisir entre le traitement *Fasmy* ou le traitement *Julien*. Dans les deux cas, la fonction *prepFiles()* va effacer les dossiers *prep* et *knn* préparer les images via l'algorithme de Luca. Il en résulte les images traitées dans le dossier *ressources/prep*. Ces images seront utilisées pour les deux algorithmes suivants.

Le clic sur le bouton *work Fasmy* va lancer le traitement et la reconnaissance via l'algorithme de réseau de neurones. Dans le second cas, le traitement via le bouton *Work Julien* utilisera son algorithme KNN pour reconnaître le ou les chiffres de l'image.

Le résultat est donnée sous forme de tableau et sera affiché dans le label en bas de l'interface graphique. Ce résultat est à prendre avec des pincettes dû à la difficulté de reconnaissance et de traitement des images.

Le plus gros défaut de cette interface est l'affichage du nombre traitée sur le canvas de droite. Comme il n'y a qu'un seul canvas à droite, l'affichage de plusieurs images sur ce dernier est malheureusement impossible. En peaufinant les algorithmes, nous n'avons pas eu le temps de remédier à ce problème. Cependant les images concernées par ce problèmes sont dans le dossier */ressources/prep.*

Conclusion

Malgré un certain manque de temps vers la fin du projet, nous avons réussit à créer une application répondant à tous les objectifs du projet.

L'application peut lire une image contenant plusieur nombres en désordre et non alignés. Les résultats obtenus sont relativement fiables et précis.

Bien que la partie DNN n'ait pas été utilisée, la backpropagation étant relativement lente à coder et à mettre en place, le fonctionnement de l'algorithme est compris.

Une marge est ajoutée autour des nombres afin que les données envoyée correspondent aux données de test. Tout à été mis en place pour que les données envoyés à nos algorithmes ressemblent le plus possible aux données du MNIST.

Il est à déplorer le fait que le nom de notre projet ait été victime d'une faute de lecture, étant ainsi nommé *MINST of love* au lieu de *MNIST of love*. Mais ceci n'a heureusement pas d'impacte sur l'efficacité de notre programme.