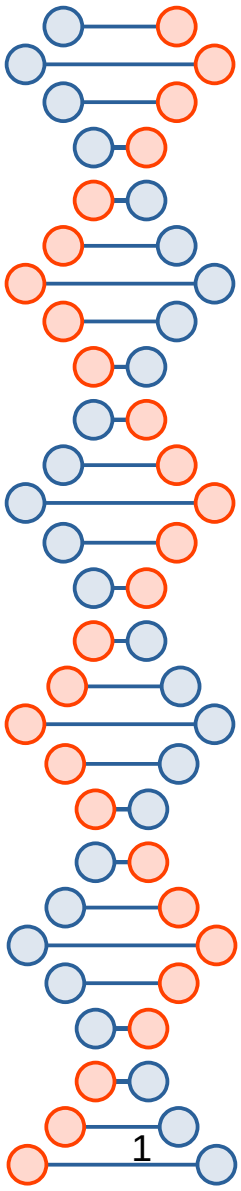
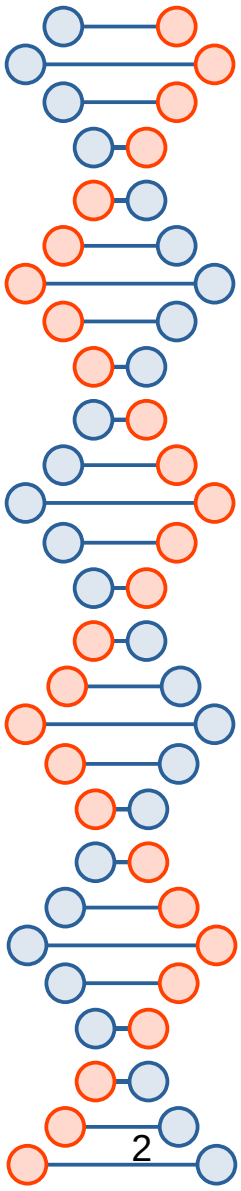
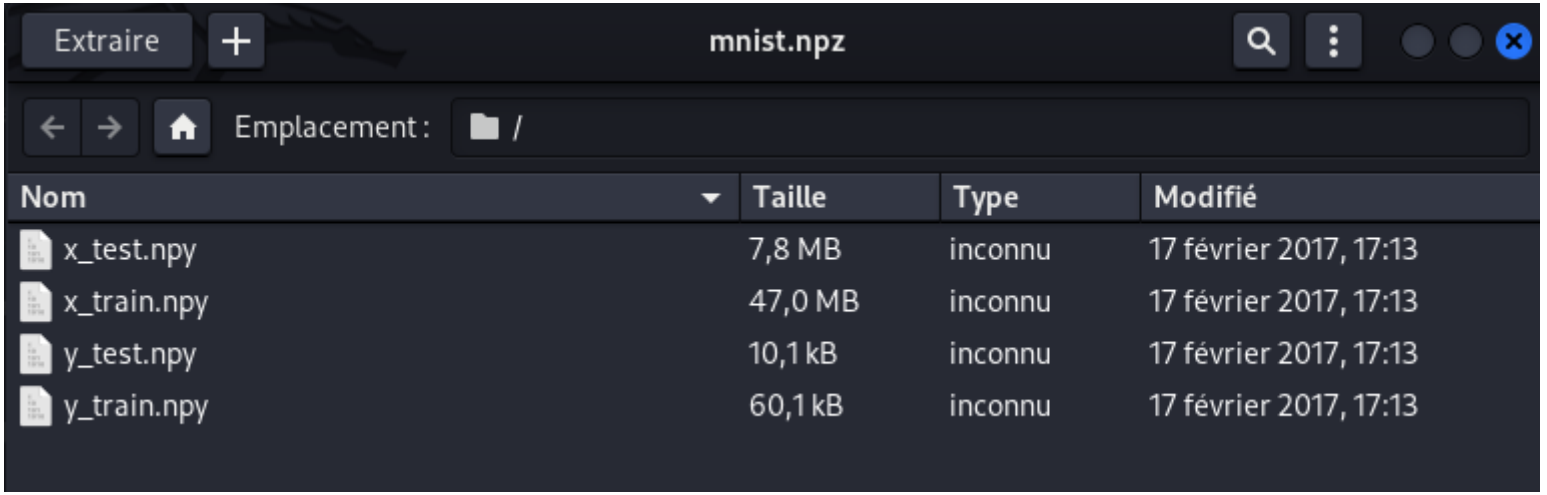


TP : Reconnaissance de chiffres avec un CNN

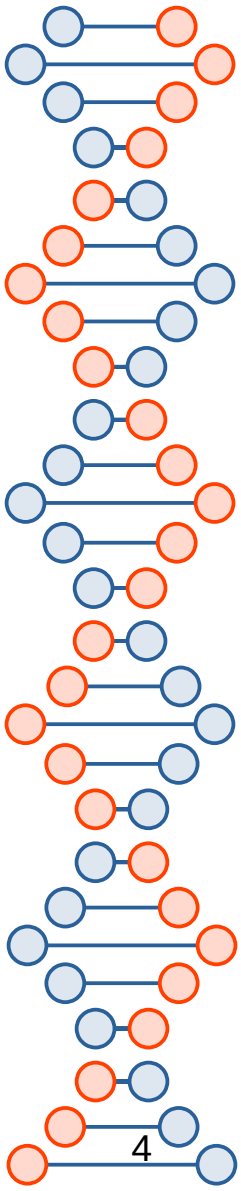


Principe

- Pour cet TP, nous allons créer un réseau de neurones convolutionnels capable de reconnaître les chiffres écrits à la main
- À partir d'une banque de 60000 images de chiffres écrits à la main (mnist de keras), nous allons entraîner 3 IA différentes et les comparer
- Ces IA seront différentes de par leur architecture CNN
- Le projet est trouvable sur https://github.com/Cirebla32/CNN_hand_written_numbers_recognition


Extraire
mnist.npz
Emplacement : /
Nom
Taille
Type
Modifié
x_test.npy
7,8 MB
inconnu
17 février 2017, 17:13
x_train.npy
47,0 MB
inconnu
17 février 2017, 17:13
y_test.npy
10,1 kB
inconnu
17 février 2017, 17:13
y_train.npy
60,1 kB
inconnu
17 février 2017, 17:13


Datasets
Machine Learning
1.jpg
2.jpg
3.jpg
4.jpg
5.jpg
6.jpg
7.jpg
8.jpg
9.jpg
10.jpg
11.jpg
12.jpg
13.jpg
14.jpg
15.jpg
16.jpg
17.jpg
18.jpg
19.jpg
20.jpg
21.jpg
22.jpg
23.jpg
24.jpg
25.jpg
26.jpg
27.jpg
28.jpg
29.jpg
30.jpg
31.jpg
32.jpg
33.jpg



Cirebla32 / **CNN_hand_written_numbers_recognition** Public

Code | Issues | Pull requests | Projects | Wiki | Security | Insights

master | Go to file | Add file | Code

Cirebla32 completion of v2 codes ... 6 minutes ago 19

__pycache__	add of v2 of models and prepareData_	18 hours ago
data_examples	init	4 years ago
drawings	init	4 years ago
save_model	init	4 years ago
save_model_v2	completion of v2 codes	8 minutes ago
CNN_presentation.pdf	Add of dataset nmist.npz & presentations	2 weeks ago
Dataset.zip	Upload of the 10,000 first pictures in Dataset.zip	2 weeks ago
Layers_explanation.pdf	Add of dataset nmist.npz & presentations	2 weeks ago
README.md	completion of v2 codes	6 minutes ago
TP-CNN-architecture-1-1...	add of v2 of models and prepareData	19 hours ago
cnnutils.py	add of v2 of models and prepareData	19 hours ago
cnnutils_v2.py	completion of v2 codes	8 minutes ago
large-cnn-tutorial-keras....	init	4 years ago
large-cnn-tutorial-keras_...	completion of v2 codes	8 minutes ago
medium-cnn-tutorial-ker	init	4 years ago

medium-cnn-tutorial-ker...	init	4 years ago
medium-cnn-tutorial-ker...	completion of v2 codes	8 minutes ago
mnist.npz	Add of dataset nmist.npz & presentations	2 weeks ago
no-preparation-small-cn...	init	4 years ago
no-preparation-small-cn...	completion of v2 codes	8 minutes ago
preparedata.py	add of v2 of models and prepareData	19 hours ago
preparedata_v2.py	add of v2 of models and prepareData	19 hours ago
small-cnn-tutorial-keras....	add of v2 of models and prepareData	19 hours ago
small-cnn-tutorial-keras_...	completion of v2 codes	8 minutes ago

README.md

CNN_hand_written_numbers_recognition by ADJOVI Albéric | CHITOU Kader | IGABOUY CHOBLI Hermine | SOTOHOU Aristide

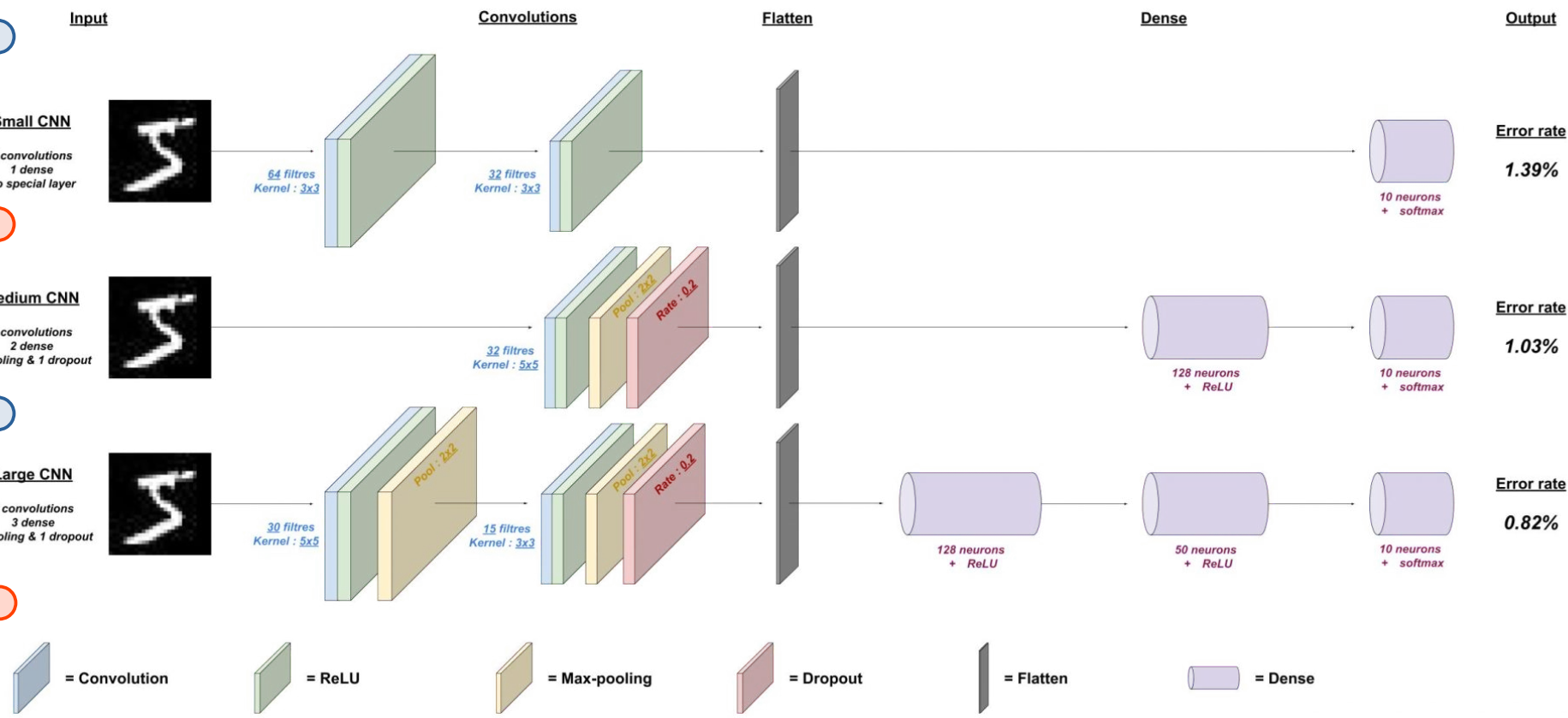
Command for running the project:

Pré-requis

- Python 3
- Pip 3
- Tensorflow
- Keras
- Matplotlib
- Pillow
- Numpy

```
git4@epac:~$ sudo apt install python3
git4@epac:~$ sudo apt install python3-pip
git4@epac:~$ pip3 install tensorflow
git4@epac:~$ pip3 install keras
git4@epac:~$ pip3 install matplotlib
git4@epac:~$ pip3 install pillow
git4@epac:~$ pip3 install numpy
```

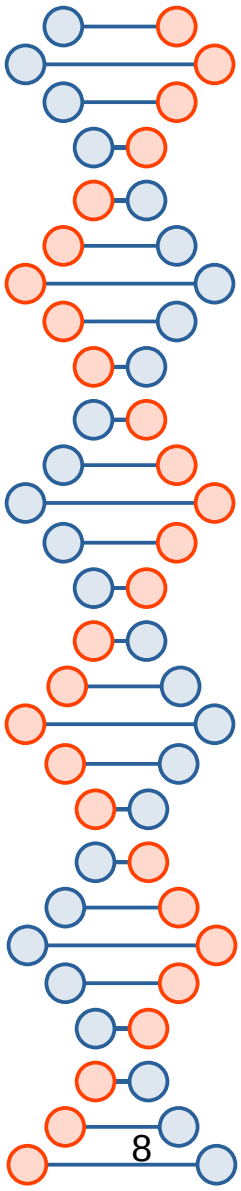
Architectures à réaliser



Organisation du code

Répertoires utiles

- **data_examples** : répertoire d'images d'apprentissage
- **drawings :** répertoire de chiffres tracés à la main pour tester le model
- **save_models_v2** : version fonctionnelle de save_models



Organisation du code

Fichiers utiles (.py)

- **cnnutils_v2** : fichier python contenant des fonctions utiles
- **preparedata_v2** : pré-traitement des données d'apprentissage
- **small-cnn-tutorial-keras_v2 :** le premier CNN implémenté et le plus simple des trois
- **medium-cnn-tutorial-keras_v2 :** le second CNN implémenté, avec l'ajout du pooling et du dropout
- **large-cnn-tutorial-keras_v2 :** le troisième CNN implémenté, plus complexe

preparedata_v2.py

- Importer les bibliothèques nécessaires

```
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
```

- Empêcher le hasard dans l'exécution afin de tous obtenir un même résultat (à faire uniquement dans un cadre pédagogique)

```
seed = 7
np.random.seed(seed)
```

- Créer la fonction *get_and_prepare_data_mnist()* qui normalise les pixels et renvoient les labels sous forme de probabilités d'appartenance à une catégories

```
def get_and_prepare_data_mnist():
    ...
    return ...
```

preparedata_v2.py

- Contenu de *get_and_prepare_data_mnist()*

```
def get_and_prepare_data_mnist():
    # load data [mnist.npz downloadable on https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz]
    # it will be automatically downloaded by doing <mnist.load_data()> instead of the following
    (X_train, y_train), (X_test, y_test) = mnist.load_data("/path/to/mnist.npz")
    # reshape to be [samples][width][height][pixels] <==> [60000][28][28][1]
    X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
    X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
    # normalize inputs from 0-255 to 0-1
    X_train = X_train / 255
    X_test = X_test / 255
    # one hot encode outputs Ex : 5 <==> [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]
    y_train = np_utils.to_categorical(y_train)
    y_test = np_utils.to_categorical(y_test)
    num_classes = y_test.shape[1]
    return (X_train, y_train), (X_test, y_test), num_classes
```

small-cnn-tutorial-keras_v2.py

- Importer les bibliothèques nécessaires

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import preparedata_v2 as pr
import cnnutils_v2 as cu
from os import  system
```

- Empêcher le hasard

```
seed = 7
np.random.seed(seed)
```

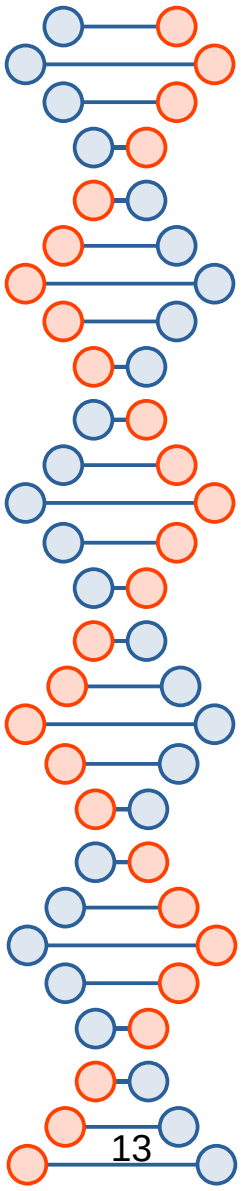
- Importer le Dataset

```
(X_train, y_train), (X_test, y_test), num_classes = pr.get_and_prepare_data_mnist()
```

small-cnn-tutorial-keras_v2.py

- Définir le modèle

```
def small_model():
    # create model
    model = Sequential()
    # build the model as shown on the image
    model.add(Conv2D(64, (3, 3), input_shape=(28, 28, 1), activation='relu'))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(Flatten())
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```



small-cnn-tutorial-keras_v2.py

- Construire, ajuster, évaluer et sauvegarder le modèle

```
# build the model
model = small_model()
# fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200)
# evaluate the model
cu.print_model_error_rate(model, X_test, y_test)
# save the model
cu.save_keras_model(model, "save_model_v2/small_model_v2_cnn")
```

small-cnn-tutorial-keras_v2.py

- Exécution…

```
git4@epac:~$ python3 small-cnn-tutorial-keras_v2.py
```

```
2022-12-26 21:04:49.472077: W tensorflow/tsl/framework/cpu_allocator_impl.cc:82] Allocation of 188160000 exceeds 10% of free system memory.
Epoch 1/10
300/300 [==============================] - 148s 490ms/step - loss: 0.2310 - accuracy: 0.9327 - val_loss: 0.0666 - val_accuracy: 0.9786
Epoch 2/10
300/300 [==============================] - 147s 490ms/step - loss: 0.0652 - accuracy: 0.9810 - val_loss: 0.0578 - val_accuracy: 0.9806
Epoch 3/10
300/300 [==============================] - 145s 485ms/step - loss: 0.0468 - accuracy: 0.9856 - val_loss: 0.0468 - val_accuracy: 0.9850
Epoch 4/10
300/300 [==============================] - 147s 490ms/step - loss: 0.0352 - accuracy: 0.9891 - val_loss: 0.0492 - val_accuracy: 0.9843
Epoch 5/10
300/300 [==============================] - 147s 491ms/step - loss: 0.0279 - accuracy: 0.9913 - val_loss: 0.0460 - val_accuracy: 0.9869
Epoch 6/10
300/300 [==============================] - 148s 493ms/step - loss: 0.0209 - accuracy: 0.9935 - val_loss: 0.0516 - val_accuracy: 0.9845
Epoch 7/10
300/300 [==============================] - 148s 492ms/step - loss: 0.0160 - accuracy: 0.9951 - val_loss: 0.0486 - val_accuracy: 0.9866
Epoch 8/10
300/300 [==============================] - 148s 495ms/step - loss: 0.0117 - accuracy: 0.9963 - val_loss: 0.0493 - val_accuracy: 0.9863
Epoch 9/10
300/300 [==============================] - 147s 492ms/step - loss: 0.0091 - accuracy: 0.9973 - val_loss: 0.0516 - val_accuracy: 0.9854
Epoch 10/10
300/300 [==============================] - 148s 494ms/step - loss: 0.0077 - accuracy: 0.9975 - val_loss: 0.0607 - val_accuracy: 0.9856
Model score : 98.56%
Model error rate : 1.44%
```

small-cnn-tutorial-keras_v2.py

- Exécution de la version sans préparation…

```
git4@epac:~$ python3 no_preparation_small_cnn_tutorial_keras_v2.py
```

```
Epoch 1/10
1875/1875 [==============================] - 165s 87ms/step - loss: 0.2345 - accuracy: 0.9517 - val_loss: 0.0928 - val_accuracy: 0.9730
Epoch 2/10
1875/1875 [==============================] - 162s 86ms/step - loss: 0.0688 - accuracy: 0.9793 - val_loss: 0.0891 - val_accuracy: 0.9737
Epoch 3/10
1875/1875 [==============================] - 151s 81ms/step - loss: 0.0456 - accuracy: 0.9859 - val_loss: 0.0731 - val_accuracy: 0.9795
Epoch 4/10
1875/1875 [==============================] - 152s 81ms/step - loss: 0.0342 - accuracy: 0.9890 - val_loss: 0.1026 - val_accuracy: 0.9768
Epoch 5/10
1875/1875 [==============================] - 164s 87ms/step - loss: 0.0262 - accuracy: 0.9915 - val_loss: 0.1397 - val_accuracy: 0.9727
Epoch 6/10
1875/1875 [==============================] - 164s 87ms/step - loss: 0.0233 - accuracy: 0.9928 - val_loss: 0.1424 - val_accuracy: 0.9746
Epoch 7/10
1875/1875 [==============================] - 151s 81ms/step - loss: 0.0201 - accuracy: 0.9941 - val_loss: 0.1234 - val_accuracy: 0.9790
Epoch 8/10
1875/1875 [==============================] - 157s 84ms/step - loss: 0.0174 - accuracy: 0.9952 - val_loss: 0.1199 - val_accuracy: 0.9799
Epoch 9/10
1875/1875 [==============================] - 164s 88ms/step - loss: 0.0168 - accuracy: 0.9955 - val_loss: 0.1501 - val_accuracy: 0.9788
Epoch 10/10
1875/1875 [==============================] - 163s 87ms/step - loss: 0.0152 - accuracy: 0.9960 - val_loss: 0.1539 - val_accuracy: 0.9802
Model score : 98.02%
Model error rate : 1.98%
```

medium-cnn-tutorial-keras_v2.py

- Définir le modèle

```
def medium_model():
    # create model
    model = Sequential()
    # build the model as shown on the image
    model.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

medium-cnn-tutorial-keras_v2.py

- Exécution…

git4@epac:~$ python3 medium-cnn-tutorial-keras_v2.py

```
2022-12-27 16:41:07.774297: W tensorflow/tsl/framework/cpu_allocator_impl.cc:82] Allocation of 188160000 exceeds 10% of free system memory.
Epoch 1/10
300/300 [==============================] - ETA: 0s - loss: 0.2436 - accuracy: 0.93012022-12-27 16:41:46.995427: W tensorflow/tsl/framework/cpu
cation of 31360000 exceeds 10% of free system memory.
300/300 [==============================] - 37s 107ms/step - loss: 0.2436 - accuracy: 0.9301 - val_loss: 0.0791 - val_accuracy: 0.9768
Epoch 2/10
300/300 [==============================] - 28s 92ms/step - loss: 0.0740 - accuracy: 0.9778 - val_loss: 0.0534 - val_accuracy: 0.9836
Epoch 3/10
300/300 [==============================] - 28s 94ms/step - loss: 0.0530 - accuracy: 0.9836 - val_loss: 0.0419 - val_accuracy: 0.9860
Epoch 4/10
300/300 [==============================] - 28s 93ms/step - loss: 0.0420 - accuracy: 0.9869 - val_loss: 0.0356 - val_accuracy: 0.9882
Epoch 5/10
300/300 [==============================] - 28s 92ms/step - loss: 0.0331 - accuracy: 0.9893 - val_loss: 0.0359 - val_accuracy: 0.9871
Epoch 6/10
300/300 [==============================] - 28s 92ms/step - loss: 0.0285 - accuracy: 0.9912 - val_loss: 0.0347 - val_accuracy: 0.9882
Epoch 7/10
300/300 [==============================] - 28s 92ms/step - loss: 0.0230 - accuracy: 0.9927 - val_loss: 0.0374 - val_accuracy: 0.9883
Epoch 8/10
300/300 [==============================] - 28s 93ms/step - loss: 0.0200 - accuracy: 0.9938 - val_loss: 0.0343 - val_accuracy: 0.9885
Epoch 9/10
300/300 [==============================] - 28s 93ms/step - loss: 0.0175 - accuracy: 0.9944 - val_loss: 0.0332 - val_accuracy: 0.9893
Epoch 10/10
300/300 [==============================] - 28s 93ms/step - loss: 0.0144 - accuracy: 0.9952 - val_loss: 0.0324 - val_accuracy: 0.9895
2022-12-27 16:45:59.199577: W tensorflow/tsl/framework/cpu_allocator_impl.cc:82] Allocation of 31360000 exceeds 10% of free system memory.
Model score : 98.95%
Model error rate : 1.05%
```

large-cnn-tutorial-keras_v2.py

- Définir le modèle

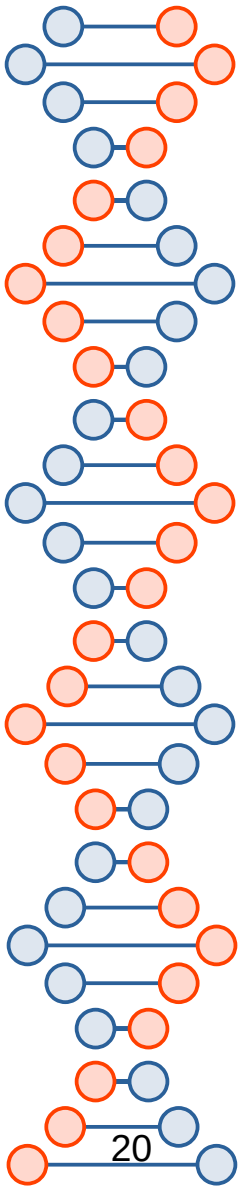
```
def large_model():
    # create model
    model = Sequential()
    # build the model as shown on the image
    model.add(Conv2D(30, (5, 5), input_shape=(28, 28, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(15, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

large-cnn-tutorial-keras_v2.py

- Exécution…

git4@epac:~$ python3 large-cnn-tutorial-keras_v2.py

```
2022-12-27 16:47:13.786310: W tensorflow/tsl/framework/cpu_allocator_impl.cc:82] Allocation of 188160000 exceeds 10% of free system memory
Epoch 1/10
300/300 [==============================] - 31s 98ms/step - loss: 0.3670 - accuracy: 0.8878 - val_loss: 0.0721 - val_accuracy: 0.9772
Epoch 2/10
300/300 [==============================] - 29s 96ms/step - loss: 0.0903 - accuracy: 0.9724 - val_loss: 0.0483 - val_accuracy: 0.9847
Epoch 3/10
300/300 [==============================] - 29s 95ms/step - loss: 0.0663 - accuracy: 0.9795 - val_loss: 0.0340 - val_accuracy: 0.9882
Epoch 4/10
300/300 [==============================] - 28s 95ms/step - loss: 0.0548 - accuracy: 0.9829 - val_loss: 0.0339 - val_accuracy: 0.9890
Epoch 5/10
300/300 [==============================] - 29s 95ms/step - loss: 0.0473 - accuracy: 0.9857 - val_loss: 0.0271 - val_accuracy: 0.9901
Epoch 6/10
300/300 [==============================] - 29s 95ms/step - loss: 0.0405 - accuracy: 0.9870 - val_loss: 0.0285 - val_accuracy: 0.9903
Epoch 7/10
300/300 [==============================] - 29s 95ms/step - loss: 0.0359 - accuracy: 0.9885 - val_loss: 0.0255 - val_accuracy: 0.9919
Epoch 8/10
300/300 [==============================] - 29s 95ms/step - loss: 0.0345 - accuracy: 0.9893 - val_loss: 0.0234 - val_accuracy: 0.9922
Epoch 9/10
300/300 [==============================] - 29s 97ms/step - loss: 0.0312 - accuracy: 0.9899 - val_loss: 0.0258 - val_accuracy: 0.9916
Epoch 10/10
300/300 [==============================] - 28s 95ms/step - loss: 0.0287 - accuracy: 0.9908 - val_loss: 0.0250 - val_accuracy: 0.9914
Model score : 99.14%
Model error rate : 0.86%
```



cnnutils_v2.py

Fonctions de cnnutils

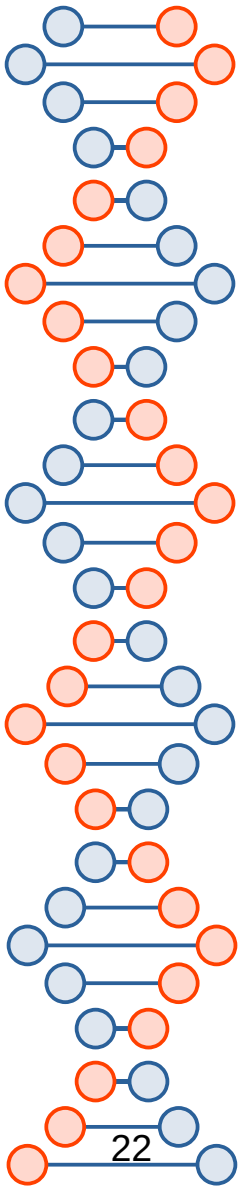
- def **save_keras_model**(model, filename)
- def **load_keras_model**(filename)
- def **print_model_error_rate**(model, X_test, y_test)
- def **export_image_from_dataset**(data, filename)
- def **plot_image_from_dataset**(data, filename)
- def **import_custom_image_to_dataset**(filename)

Testons le modèle

```
git4@epac:~$ python3
>>> import cnnutils_v2 as cu
>>> model = cu.load_keras_model("save_model_v2/large_model_v2_cnn")
>>> model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
>>> x = cu.import_custom_image_to_dataset("drawings/9.jpg")
>>> model.predict(x)
```

Remarques :

- Les images utilisées pour l'entraînement ont subit un prétraitement. Leur fond est noir
- Le modèle fournit donc de meilleurs résultats si l'image en entrée présente des caractéristiques similaires. Il est utile de pré-traiter les images avant toute prédiction.



Merci pour votre aimable attention…