

Titre du projet

Détection de chiffres manuscrits en utilisant les réseaux de neurones convolutifs.

---

Base de données : MNIST handwritten digits.

---

Domaine : Deep Learning

---

Réalisé par :

Bihani Mohamed Amine

## Table de matières

Titre du projet.....	1
Remerciements.....	<b>Erreur ! Signet non défini.</b>
Résumé .....	3
Présentation de la base de données Mnist .....	4
Environnement du projet .....	4
Bibliothèques utilisées .....	4
Keras API .....	4
Algorithmes utilisés .....	4
K fold cross validation .....	4
ReLU activation .....	4
SoftMax .....	5
Stochastic gradient descent.....	5
Modèle adopté .....	6
Convolutional Neural Networks .....	6
Architecture.....	6
Conception.....	7
Input : .....	7
Etapes : .....	7
Schéma général : .....	9
Réalisation .....	10
Test du modèle .....	14
Conclusion .....	16
Webographie .....	16

## Résumé

L'apprentissage profond (deep learning) est une technique d'apprentissage automatique (machine learning) qui a considérablement amélioré les résultats dans de nombreux domaines tels que la vision par ordinateur, la reconnaissance de la parole et la traduction automatique. Les techniques d'apprentissage profond permettent, à l'aide de données, de résoudre de nombreux problèmes dans de nombreux domaines de l'économie tels que la santé, le transport, le commerce, la finance ainsi que l'énergie.

On appelle réseau neuronal convolutif, ou réseau de neurones à convolution, (CNN pour Convolutional Neural Network) un type de réseau de neurones artificiels utilisé dans la reconnaissance et le traitement des images, et spécialement conçu pour l'analyse des pixels.

Contrairement aux techniques d'apprentissage supervisé, les réseaux de neurones convolutifs apprennent les features de chaque image. C'est là que réside leur force : les réseaux font tout le boulot d'extraction de features automatiquement, contrairement aux techniques d'apprentissage .

Dans ce projet, nous avons choisi de travailler avec la base de données MNIST de chiffres manuscrits, comprend un ensemble de formation de 60 000 exemples et un ensemble de test de 10 000 exemples. Il s'agit d'un sous-ensemble d'un ensemble plus grand disponible auprès du NIST. Les chiffres ont été normalisés en fonction de la taille et centrés dans une image de taille fixe.

C'est une bonne base de données pour les personnes qui souhaitent en savoir plus sur les différentes méthodes de reconnaissance des formes pour les données du monde réel tout en dépensant un minimum d'efforts sur le prétraitement et le formatage.

Le but donc de ce projet est de pouvoir reconnaître les entrées étant des chiffres manuscrits, puis les classifier.

## Présentation de la base de données Mnist

La base de données MNIST (base de données modifiées de l'Institut national des normes et de la technologie) est une grande base de données de chiffres manuscrits utilisés pour la formation de divers systèmes de traitement d'images. La base de données est également largement utilisée pour la formation et les tests dans le domaine de l'apprentissage automatique.

## Environnement du projet

### Bibliothèques utilisées

#### Keras API

Keras est une API de réseaux de neurones de haut niveau, écrite en Python et interfaçable avec TensorFlow. Elle a été développée avec pour objectif de permettre des expérimentations rapides. Être capable d'aller de l'idée au résultat avec le plus faible délai possible étant la clef d'une recherche efficace.

On utilisera la fonction `model.evaluate` qui est présente dans l'API Keras, celle-ci intègre des algorithmes permettant de mieux définir le modèle. Ceci sera détaillé dans la section suivante.

### Algorithmes utilisés

#### K fold cross validation

La validation croisée est une méthode statistique utilisée pour estimer la compétence des modèles d'apprentissage automatique.

Elle est couramment utilisée dans l'apprentissage automatique appliqué pour comparer et sélectionner un modèle pour un problème de modélisation prédictive donné, car il est facile à comprendre, à mettre en œuvre et donne des estimations de compétences qui ont généralement un biais plus faible que les autres méthodes.

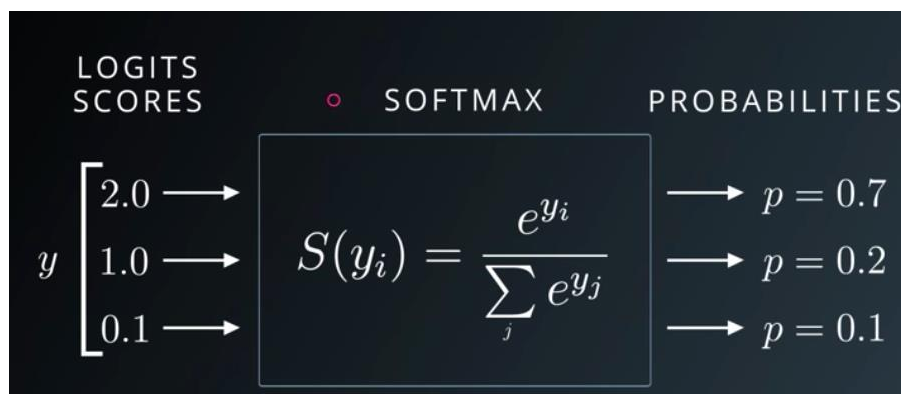
#### ReLU activation

Dans un réseau de neurones, la fonction d'activation est responsable de la transformation de l'entrée pondérée sommée du nœud en l'activation du nœud ou de la sortie pour cette entrée.

La fonction d'activation linéaire rectifiée est une fonction linéaire par morceaux qui sortira l'entrée directement si elle est positive, sinon, elle sortira zéro. Il est devenu la fonction d'activation par défaut pour de nombreux types de réseaux de neurones car un modèle qui l'utilise est plus facile à entraîner et obtient souvent de meilleures performances.

## SoftMax

C'est une fonction d'activation qui transforme les nombres aka logits en probabilités qui se résument à un. La fonction Softmax génère un vecteur qui représente les distributions de probabilité d'une liste de résultats potentiels.



## Stochastic gradient descent

### Gradient descent

La descente de gradient peut être décrite comme une méthode itérative utilisée pour trouver les valeurs des paramètres d'une fonction qui minimise autant que possible la fonction de coût. Les paramètres sont initialement définis une valeur particulière et à partir de cela, Gradient Descent est exécuté de manière itérative pour trouver les valeurs optimales des paramètres, en utilisant le calcul, pour trouver la valeur minimale possible de la fonction de coût donnée.

### Stochastic gradient descent

Le mot «stochastique» signifie un système ou un processus lié à une probabilité aléatoire. Par conséquent, dans la descente de gradient stochastique, quelques échantillons sont sélectionnés au hasard au lieu de l'ensemble de données pour chaque itération. Dans Gradient Descent, il existe un terme appelé «batch» qui

désigne le nombre total d'échantillons d'un ensemble de données qui est utilisé pour calculer le gradient pour chaque itération.

**SGD algorithm:**

$$\text{for } i \text{ in range } (m) : \\ \theta_j = \theta_j - \alpha (\hat{y}^i - y^i) x_j^i$$

## Modèle adopté

### Convolutional Neural Networks

La principale caractéristique structurelle de RegularNets est que tous les neurones sont connectés les uns aux autres. Par exemple, lorsque nous avons des images de 28 x 28 pixels avec uniquement des niveaux de gris, nous finirons par avoir 784 (28 x 28 x 1) neurones dans une couche qui semble gérable. Cependant, la plupart des images ont beaucoup plus de pixels et elles ne sont pas à échelle de gris. Par conséquent, en supposant que nous ayons un ensemble d'images couleur en 4K Ultra HD, nous aurons 26 542 080 (4096 x 2160 x 3) neurones différents connectés les uns aux autres dans la première couche, ce qui n'est pas vraiment gérable. Par conséquent, nous pouvons dire que les RegularNets ne sont pas évolutifs pour la classification des images. Cependant, surtout en ce qui concerne les images, il semble y avoir peu de corrélation ou de relation entre deux pixels individuels, à moins qu'ils ne soient proches l'un de l'autre. Cela conduit à l'idée de couches convolutionnelles et de couches de mise en commun.

### Architecture

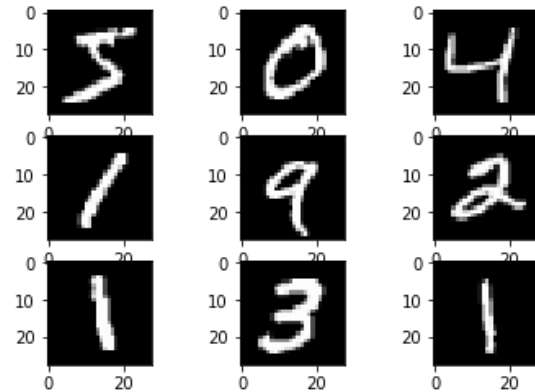
Celle-ci est plus détaillée dans la partie conception.

## Conception

**Input :** Images qui contiennent des chiffres

manuscrits

⑨ en forme matricielle de niveaux de gris

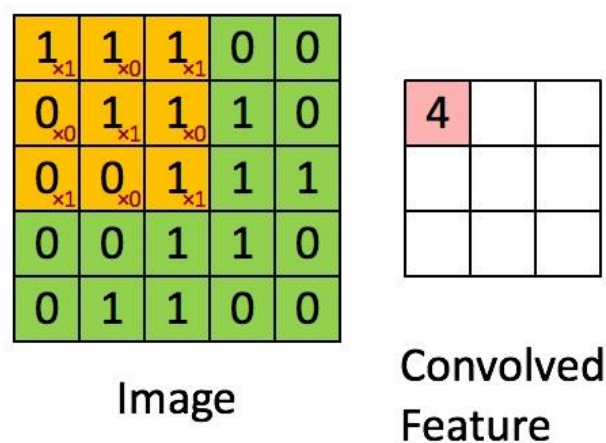


### Etapes :

D'abord, la convolution :

La convolution dans ce cas vise à la traduction des champs récepteurs locaux à travers une image pour créer une carte d'entités de la couche d'entrée vers les neurones de la couche cachée.

Ceci donnera lieu à des matrices convolutées suivant le principe suivant :



On ajoute une couche de Pooling pour réduire aussi l'espace alloué aux matrices, et ca permet aussi d'extraire les valeurs dominantes. On utilisera Max Pooling parcequ' il offre une meilleure performance en ajustant les valeurs dominantes :

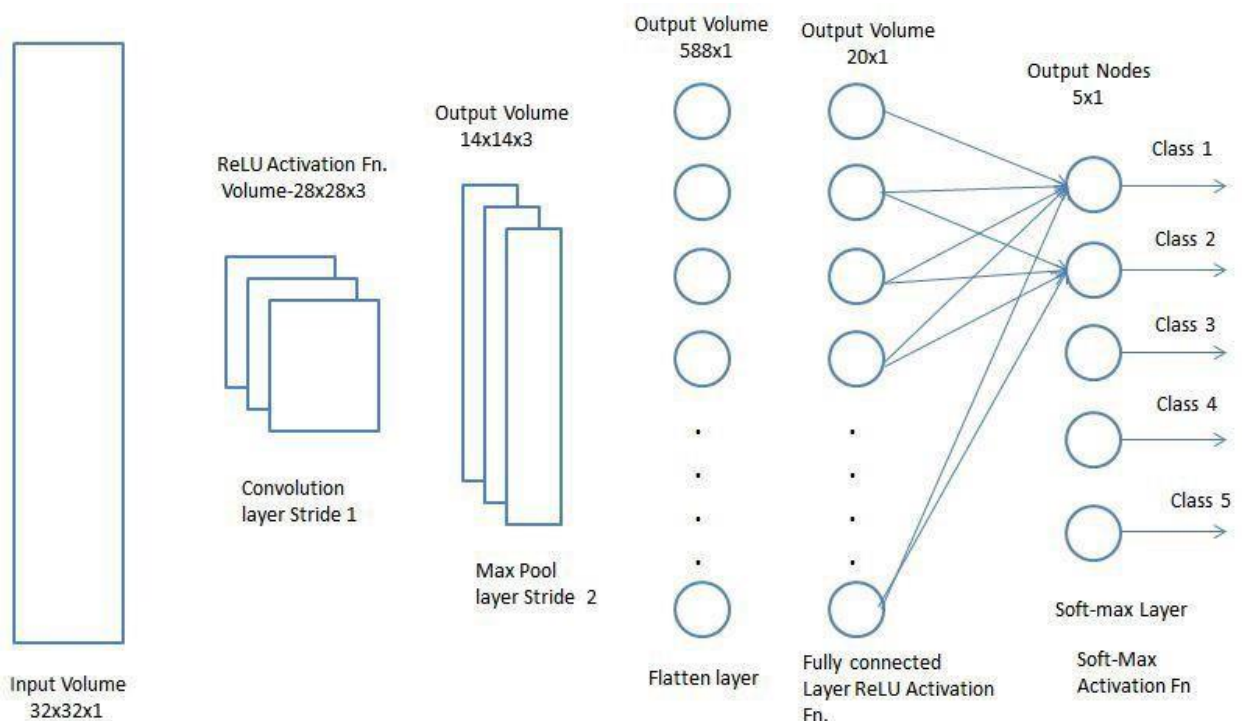
3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

Cela permet de simplifier les couches suivantes et de réduire le nombre de paramètres que le modèle doit apprendre.

A la sortie de cette étape, on a la configuration des couches du CNN.





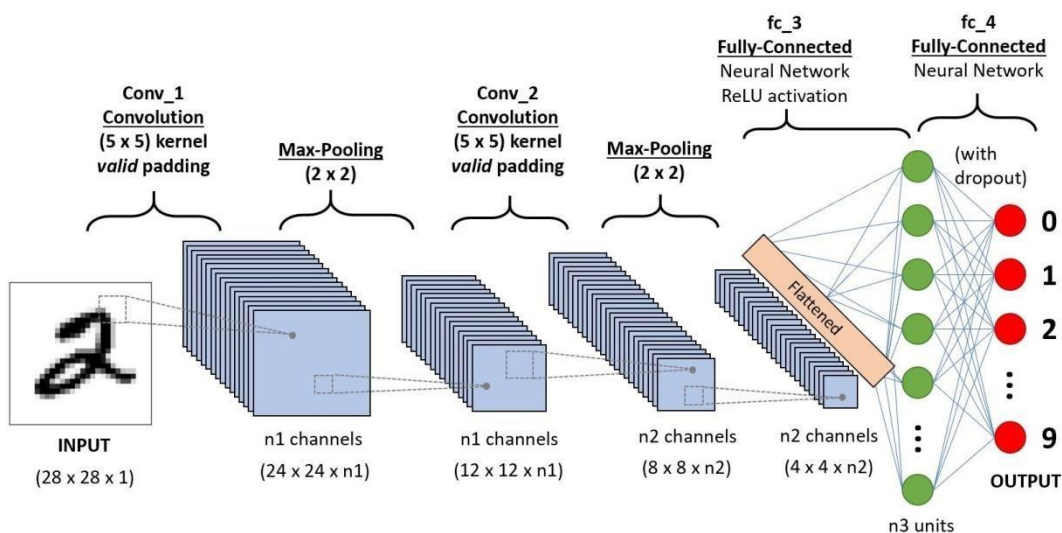
Ensuite, l'activation.

L'étape de l'activation consiste à appliquer des transformations sur chaque output de chaque neurone à l'aide d'une fonction d'activation : ici on opte pour la fonction ReLu : Rectified Linear Unit.

En gros, cette fonction agit come suit : elle prend la sortie du neurone et la mappe à la valeur la plus élevée ; ou si la sortie est négative, elle la mappe à zéro.

### Schéma général :

En général, notre conception aura le même principe que le schéma ci-dessous :



A CNN  
sequence to  
classify

handwritten  
digits

Source :


<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neuralnetworks-the-eli5-way-3bd2b1164a53>

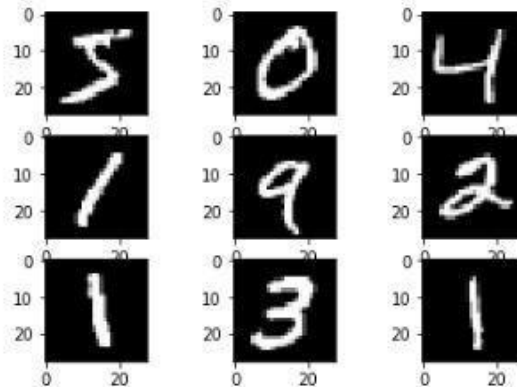
## Réalisation

```

from keras.layers import Conv2D
[ ] from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
# load dataset
(trainX, trainy), (testX, testy) = mnist.load_data()
# summarize loaded dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
# plot first few images
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # plot raw pixel data
    pyplot.imshow(trainX[i], cmap=pyplot.get_cmap('gray'))
# show the figure
pyplot.show()

```

 Train: X=(60000, 28, 28), y=(60000,)
   
 Test: X=(10000, 28, 28), y=(10000,)



```

def load_dataset():
    # load dataset
    (trainX, trainy), (testX, testy) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainy = to_categorical(trainy)
    testy = to_categorical(testy)
    return trainX, trainy, testX, testy

```

```

# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm

```

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
# evaluate a model using k-fold cross-validation
def evaluate_model(model, dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # select rows for train and test
        trainX, trainy, testX, testy = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainy, epochs=10, batch_size=32, validation_data=(testX, testy), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testy, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

```
# plot diagnostic learning curves
def summarize_diagnostics(histories):
    for i in range(len(histories)):
        # plot loss
        pyplot.subplot(211)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(histories[i].history['loss'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_loss'], color='orange', label='test')
        # plot accuracy
        pyplot.subplot(212)
        pyplot.title('Classification Accuracy')
        pyplot.plot(histories[i].history['acc'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_acc'], color='orange', label='test')
    pyplot.show()
```

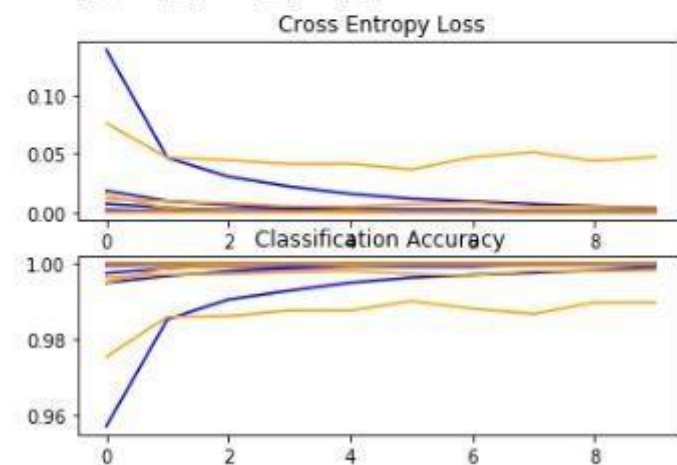
```
# summarize model performance
def summarize_performance(scores):
    # print summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)))
    # box and whisker plots of results
    pyplot.boxplot(scores)
    pyplot.show()
```

```
# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainy, testX, testy = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # define model
    model = define_model()
    # evaluate model
    scores, histories = evaluate_model(model, trainX, trainy)
    # learning curves
    summarize_diagnostics(histories)
    # summarize estimated performance
    summarize_performance(scores)
    # fit model
    model.fit(trainX, trainy, epochs=10, batch_size=32, verbose=0)
    # save model
    model.save('final_model.h5')
```

```
# entry point, run the test harness
run_test_harness()
```

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op\_def\_library.py:263:

```
> 98.983
> 99.825
> 99.975
> 100.000
> 100.000
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1:
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1:
  if __name__ == '__main__':
```



Accuracy: mean=99.757 std=0.392, n=5



## Test du modèle

```
# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

```
# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm
```

```
# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # load model
    model = load_model('final_model.h5')
    # evaluate model on test dataset
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('> %.3f' % (acc * 100.0))
```

```
# entry point, run the test harness
run_test_harness()
```

```
> 99.250
```

```
## Testing the prediction on actual data

# make a prediction for a new image.
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
# load and prepare the image
def load_image(filename):
    # load the image
    img = load_img(filename, grayscale=True, target_size=(28, 28))
    # convert to array
    img = img_to_array(img)
    # reshape into a single sample with 1 channel
    img = img.reshape(1, 28, 28, 1)
    # prepare pixel data
    img = img.astype('float32')
    img = img / 255.0
    return img
```

Input : image suivante



```
# load an image and predict the class
def run_example():
    # load the image
    img = load_image('ex4.PNG')
    # load model
    model = load_model('final_model.h5')
    # predict the class
    digit = model.predict_classes(img)
    print(digit[0])

# entry point, run the example
run_example()
```

```
/usr/local/lib/python3.6/dist-packages/keras_preprocessi
warnings.warn('grayscale is deprecated. Please use '
4
```

## Conclusion

« La technologie du deep learning apprend à représenter le monde. C'est-à-dire comment la machine va représenter la parole ou l'image par exemple », pose Yann LeCun, considéré par ses pairs comme un des chercheurs les plus influents dans le domaine. « Avant, il fallait le faire à la main, expliquer à l'outil comment transformer une image afin de la classifier. Avec le deep learning, la machine apprend à le faire elle-même. Et elle le fait beaucoup mieux que les ingénieurs, c'est presque humiliant ! »

Ces propos illustrent parfaitement le rôle immense que le deep learning joue afin d'enrichir le domaine de l'intelligence artificielle .

## Webographie

<https://www.louisbachelier.org/wp-content/uploads/2017/07/170620-ilbpresentation-ludo-denoyer.pdf>

<https://towardsdatascience.com/mnist-handwritten-digits-classification-using-aconvolutional-neural-network-cnn-af5fafbc35e9>

[https://fr.wikipedia.org/wiki/R%C3%A9seau\\_neuronal\\_convolutif](https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif)

<https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-desdonnees-visuelles/5082166-quest-ce-quun-reseau-de-neurones-convolutif-ou-cnn>

<https://towardsdatascience.com/mnist-handwritten-digits-classification-using-aconvolutional-neural-network-cnn-af5fafbc35e9>

<https://www.actuia.com/keras/#>

<https://machinelearningmastery.com/rectified-linear-activation-function-for-deeplearning-neural-networks/> <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/>

<https://www.mathworks.com/videos/introduction-to-deep-learning-what-areconvolutional-neural-networks--1489512765771.html>

<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neuralnetworks-the-eli5-way-3bd2b1164a53>