

**École supérieure d'ingénieurs de recherche en matériaux et en  
infotronique**



## **Systèmes intelligents avancés**

# **Application de la CNN sur FASHION-Mnist à l'aide de Keras**

**BERKANE Soufiane**

**2018/2019**

## Table de matière

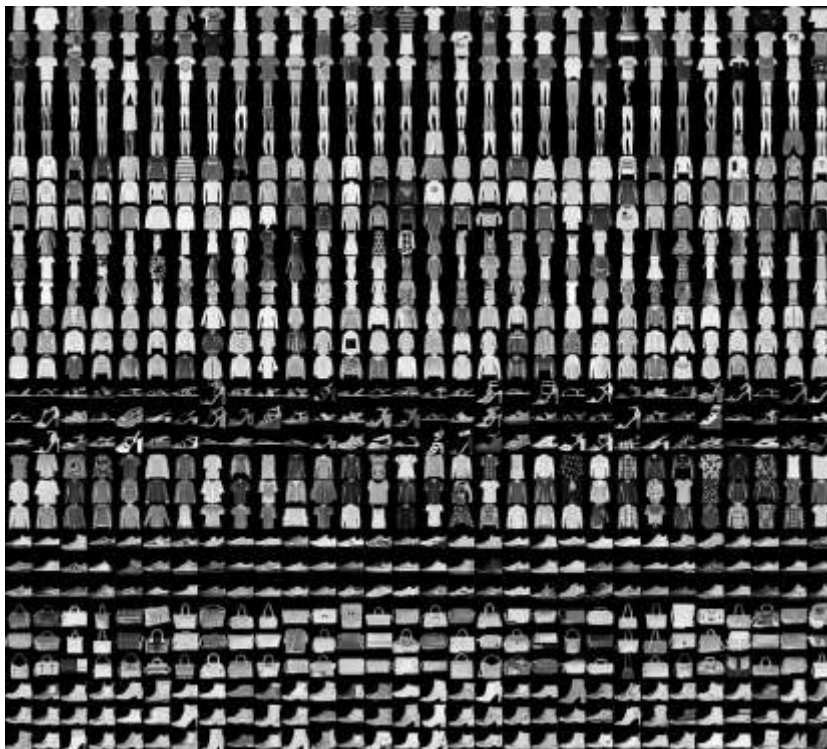
<b>Introduction.....</b>	<b>3</b>
<b>Dataset : .....</b>	<b>3</b>
<b>API Keras .....</b>	<b>3</b>
<b>Modèle utilisé .....</b>	<b>3</b>
Importation des outils : .....	4
Définition des paramètres d'apprentissage : .....	4
Chargement des ensembles de formation et de test FASHION MNIST .....	4
Manipulation des données.....	4
Convertir des vecteurs de classe en matrices de classe binaires.....	4
Model building et compilation de model .....	5
fit et fit-generator.....	5
Meilleur résultat.....	5
<b>Conclusion .....</b>	<b>6</b>

## Introduction

Ceci est un projet sur la classification du dataset Fashion-MNIST en utilisant keras, à l'aide d'une architecture CNN (Convolutional Neural Network). L'objectif c'est former un modèle capable de classer les images avec une précision de 95% tout en respectant plusieurs contraintes et en utilisant l'API Keras.

## Dataset

Fashion-MNIST est un ensemble de données d'images d'article de Zalando, composé d'un ensemble de formation de 60 000 exemples et un ensemble de test de 10 000 exemples. Chaque exemple est une image en niveaux de gris de 28x28, associée à une étiquette de 10 classes.



## API Keras



Keras est une API de réseaux de neurones de haut niveau, écrite en Python et capable de s'exécuter sur TensorFlow, CNTK ou Theano. Il a été développé pour permettre une expérimentation rapide. Pouvoir faire de la recherche de qualité est essentiel pour pouvoir passer d'une idée à l'autre dans les résultats.

## Modèle utilisé

## Importation des outils

Dans cette partie on a importé les outils avec lequel on va travailler.

```
from __future__ import print_function

import keras
from keras.datasets import fashion_mnist
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.callbacks import CSVLogger
from keras.preprocessing.image import ImageDataGenerator
import sys
```

## Définition des paramètres d'apprentissage

Taille d'un bloc de données (batch)

Nombre de classe à trouver

Nombre d'époques en phase d'apprentissage (entre 10 et 55)

```
batch_size = 128
num_classes = 10
epochs = 25
```

## Chargement des ensembles de formation et de test FASHION MNIST

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

## Manipulation des données

- De 28 x 28 images à des vecteurs compatibles MLP
- Conversion de données en float
- Normalisation des données vers [0 1]

```
24
25 if K.image_data_format() == 'channels_first':
26     x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
27     x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
28     input_shape = (1, img_rows, img_cols)
29 else:
30     x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
31     x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
32     input_shape = (img_rows, img_cols, 1)
33
34 x_train = x_train.astype('float32')
35 x_test = x_test.astype('float32')
36 x_train /= 255
37 x_test /= 255
38 print('x_train shape:', x_train.shape)
39 print(x_train.shape[0], 'train samples')
40 print(x_test.shape[0], 'test samples')
41
```

## Convertir des vecteurs de classe en matrices de classe binaires

```
42 y_train = keras.utils.to_categorical(y_train, num_classes)
43 y_test = keras.utils.to_categorical(y_test, num_classes)
44
```

## Model building et compilation de model

Dans cette partie j'ai effectué plusieurs tests, soit en changeant le nombre des couches, la fonction d'activation, et l'optimizer.

Après plusieurs tests j'ai constaté que c'est mieux de travailler avec une seule couche, une fonction d'activation linéaire (relu) et d'utiliser l'optimizer Adam().

```
59 model.add(Conv2D(32, kernel_size=(3, 3), activation = 'relu', input_shape = input_shape))
60 model.add(Conv2D(64, (3, 3), activation = 'relu'))
61 model.add(MaxPooling2D(pool_size=(2, 2)))
62 model.add(Dropout(0.25))
63 model.add(Flatten())
64 model.add(Dense(128, activation='relu'))
65 model.add(Dropout(0.25))
66 model.add(Dense(num_classes, activation='softmax'))
67
68
69 model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
```

## fit et fit-generator

Le fit permet d'entraîner le modèle pour un nombre donné d'époques. J'ai ajouté le callbacks comme paramètre, il contient la liste des callbacks à appliquer pendant le training.

Le fit-generator entraîne le modèle sur des données générées lot par lot par un générateur Python (ou une instance de séquence).

```
75
76 model.fit(x_train, y_train,
77         callbacks=[checkpointer, tensorb],
78         batch_size=batch_size,
79         epochs=epochs,
80         verbose=1,
81         validation_data=(x_test, y_test))
82 score = model.evaluate(x_test, y_test, verbose=0)
83
84 gen = ImageDataGenerator(
85     shear_range=0.2,
86     zoom_range=0.2,
87     horizontal_flip=True)
88 generator = gen.flow(x_train, y_train, batch_size)
89 model.fit_generator(
90     generator,
91     steps_per_epoch=60000/batch_size,
92     epochs=epochs,
93     validation_data=(x_test, y_test),
94     validation_steps=10000/batch_size,
95     use_multiprocessing=False,
96     shuffle=True
97 )
98 score = model.evaluate(x_test, y_test, verbose = 0)
99 print('Test loss:', score[0])
00 print('Test accuracy:', score[1])
01
```

## Meilleur résultat

Après 42 epochs, Je suis arrivé à 93,67 et une valeur de loss de 0.23 ce qui montre que le model peut être amélioré après autres epochs.

```
Invite de commandes - python C:\Projet\cafe-demo-master\DeepCourse\lyess.py
Epoch 30/55 [=====] - 187s 374ms/step - loss: 0.1592 - acc: 0.9412 - val_loss: 0.2230 - val_acc: 0.9339
Epoch 30/55 [=====] - 188s 376ms/step - loss: 0.1595 - acc: 0.9403 - val_loss: 0.2149 - val_acc: 0.9313
Epoch 31/55 [=====] - 185s 378ms/step - loss: 0.1586 - acc: 0.9413 - val_loss: 0.2129 - val_acc: 0.9351
Epoch 32/55 [=====] - 185s 378ms/step - loss: 0.1537 - acc: 0.9438 - val_loss: 0.2110 - val_acc: 0.9323
Epoch 33/55 [=====] - 204s 488ms/step - loss: 0.1526 - acc: 0.9440 - val_loss: 0.2117 - val_acc: 0.9341
Epoch 34/55 [=====] - 184s 368ms/step - loss: 0.1548 - acc: 0.9427 - val_loss: 0.2170 - val_acc: 0.9388
Epoch 35/55 [=====] - 178s 356ms/step - loss: 0.1487 - acc: 0.9444 - val_loss: 0.2161 - val_acc: 0.9332
Epoch 36/55 [=====] - 172s 343ms/step - loss: 0.1457 - acc: 0.9455 - val_loss: 0.2202 - val_acc: 0.9327
Epoch 37/55 [=====] - 177s 354ms/step - loss: 0.1466 - acc: 0.9457 - val_loss: 0.2193 - val_acc: 0.9385
Epoch 38/55 [=====] - 186s 371ms/step - loss: 0.1481 - acc: 0.9445 - val_loss: 0.2177 - val_acc: 0.9318
Epoch 39/55 [=====] - 187s 373ms/step - loss: 0.1464 - acc: 0.9471 - val_loss: 0.2199 - val_acc: 0.9334
Epoch 40/55 [=====] - 183s 365ms/step - loss: 0.1435 - acc: 0.9460 - val_loss: 0.2381 - val_acc: 0.9348
Epoch 41/55 [=====] - 182s 363ms/step - loss: 0.1434 - acc: 0.9477 - val_loss: 0.2214 - val_acc: 0.9341
Epoch 42/55 [=====] - 178s 356ms/step - loss: 0.1436 - acc: 0.9468 - val_loss: 0.2135 - val_acc: 0.9367
Epoch 43/55 [=====] - 178s 355ms/step - loss: 0.1378 - acc: 0.9498 - val_loss: 0.2153 - val_acc: 0.9339
Epoch 44/55 [=====] - 186s 371ms/step - loss: 0.1382 - acc: 0.9478 - val_loss: 0.2113 - val_acc: 0.9327
Epoch 45/55 [=====] - 195s 398ms/step - loss: 0.1488 - acc: 0.9481 - val_loss: 0.2261 - val_acc: 0.9332
Epoch 46/55 [=====] - 214s 428ms/step - loss: 0.1361 - acc: 0.9493 - val_loss: 0.2335 - val_acc: 0.9325
Epoch 47/55 [=====] - 194s 388ms/step - loss: 0.1362 - acc: 0.9495 - val_loss: 0.2262 - val_acc: 0.9389
Epoch 48/55 [=====] - 197s 395ms/step - loss: 0.1388 - acc: 0.9491 - val_loss: 0.2423 - val_acc: 0.9286
Epoch 49/55 [=====] - 179s 357ms/step - loss: 0.1320 - acc: 0.9513 - val_loss: 0.2400 - val_acc: 0.9276
Epoch 50/55 [=====] - 171s 343ms/step - loss: 0.1378 - acc: 0.9484 - val_loss: 0.2273 - val_acc: 0.9314
Epoch 51/55 [=====]
```

## Conclusion

Le but de ce projet était de se familiariser avec l'utilisation des réseaux de neurones pour résoudre des tâches de vision par ordinateur. Nous allons manipuler plusieurs types de réseaux de neurones ainsi que plusieurs jeux de données courants. Le travail réalisé permet d'utiliser un réseau de neurones pour classer les habits présentés sur une image de manipuler les images et modifier leurs formats pour s'adapter à Keras, et enfin changer les paramètres et le modèle pour arriver à une précision de 95 %. Pour faire ça nous avons été inspirés de publications ou de tutoriels existants.