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

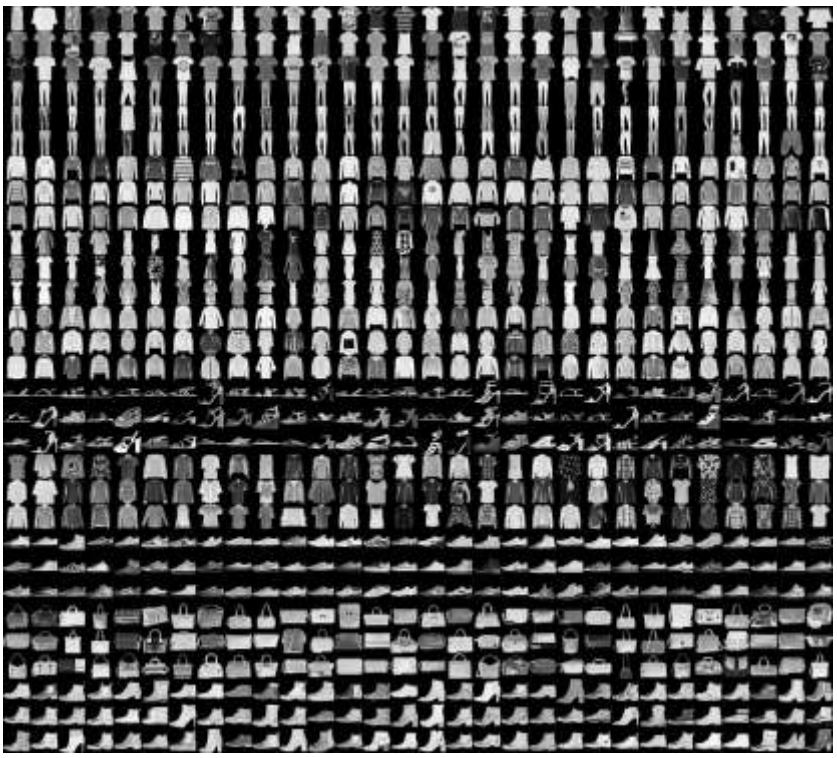
Table de matière

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

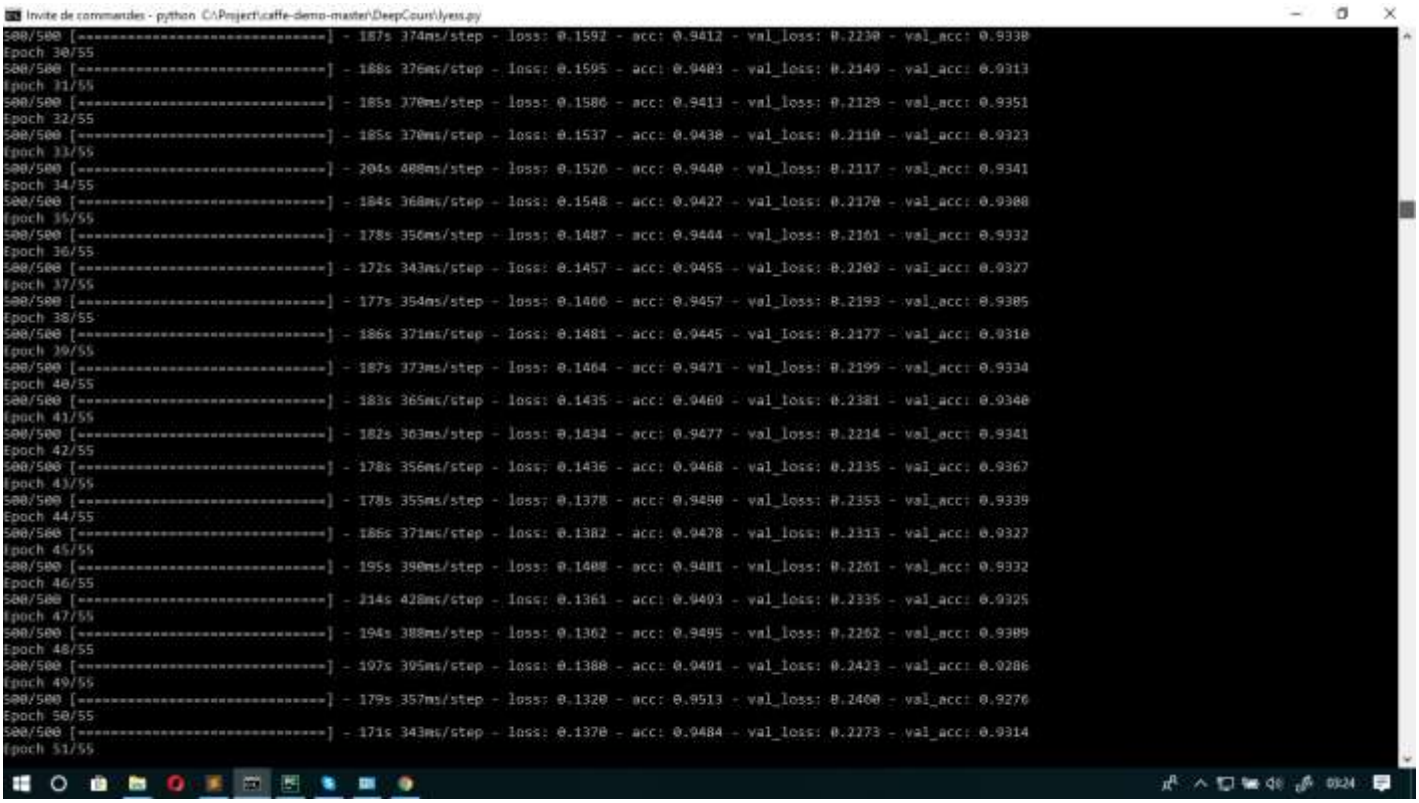
fit et fit-generator

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])
```

Meilleur résultat

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



Conclusion

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