Apprenant : LEPICAUT Didier
Objet : Learning par étude de cas
Période : Centre de formation
Jury Académique : Serge GRATTON - Co-Directeur Ms Valdom / Enseeiht
Intervenant : Pierre BOUDIER - Lecturer of Machine Learning / Enseeiht
Date : 29/01/18

Mots clefs : Pytorch, TF.Keras, CPU, GPU, Deep Learning, Computer Vision

CADRAGE	
date	29 novembre 2018
version	code Pytorch, TF.Keras
source des données	Fichier *.jpg, photos de mains
nombre de classes	k = 3
nombre d'enregistrements	297 (training) + 55 (test) = 352
périmètre	Jeu CHIFOUMI > Figures (Pierre / Feuille / Ciseaux)
problème	Classifieur multi-classes en traitement d'images (ie Computer Vision)
type d'analyse	Méthode d'apprentissage machine supervisée
langage de programmation	Langage Python – Pytorch & TensorFlow - Keras
Note destinée au lecteur	résultats clefs

- SOMMAIRE -

1. Résumé

```
# ====================
# Executive summary : classification multi-classes supervisee pour k = 3 (dataset = images)
# ====================

. Accuracy en validation = 94.23%
. calcul avec Keras-GPU
. les parametres epochs (10) et batch_size (25) mis a la valeur max pour tenir in-memory Cuda (gtx1060M - 6Go)

. architecture reseau : architecture Cnn habituellement utilisee sur le dataset Mnist
=> ie 3x Convolution 2D (travaille sur image 2 dim) + Maxpooling + dropout + sortie softmax + optimizer Adam

. la performance a ete realisee par une definition maximale de 600 x 600 pour les data-images en entrees
. le reseau, a ete entraine en moins de 8 minutes

Nota : resultat a confirme avec un nb d Epochs = 100, ie Accuracy Validation < Accuracy Train pour garantir que le
gradient a bien converge sur un minimum global.
```

2. Problème

Demande: « *… entrainer et spécifier une machine qui produit des étiquettes …* »

Apprentissage statistique

Mesurer les erreurs d'etiquettage

des entrées

Modèle mathématique -> CLASSIFIEUR qui appartient à une famille de méthodes mathématiques et codée sous forme d'un algorithme informatique

des sorties

Performance du modèle -> On calcule un taux d erreur qui mesure la performance de notre modèle a bien classer les images

Taux d Erreur = (vecteur étiquettes réelles – vecteur des étiquettes calculées) / longueur vecteur

Taux de Précision = 1 – Tx Erreur (ie Accuracy)

Des images format *.jpg
Taille : 600 x 600

des étiquettes réelles

Famille de méthodes -> classifieur supervisé sur des données particulières, des images, petite dimension du problème, petite taille des données

des etiquettes calculees

3. Données

> Type : image au format *.jpg, en couleur

> Taille des images : plusieurs tailles identifiées, mais en entrées avec dimension unique 600 x 600 [max utilisable]

> Taille dataset training = 297 images

> Taille dataset test = 55 images

> Taille dataset total = 297 + 55 = 352 images

> Procédure apprentissage simple (train vs test) = Train (84.37 %) / Test (15.63 %) de la Taille dataset total

> Qualité des données : pas de « Fake image » et/ou « image de mauvaise qualité », toutes vérifiées

> Pas de « preprocessing du type augmentation des données », utilisation brute des images

> Pour garantir la convergence des calculs numériques, une simple normalisation « 1/255 » appliquée aux entrées (normalisation de la valeur de nos entrées 0-255 en 0-1)

4. Outils

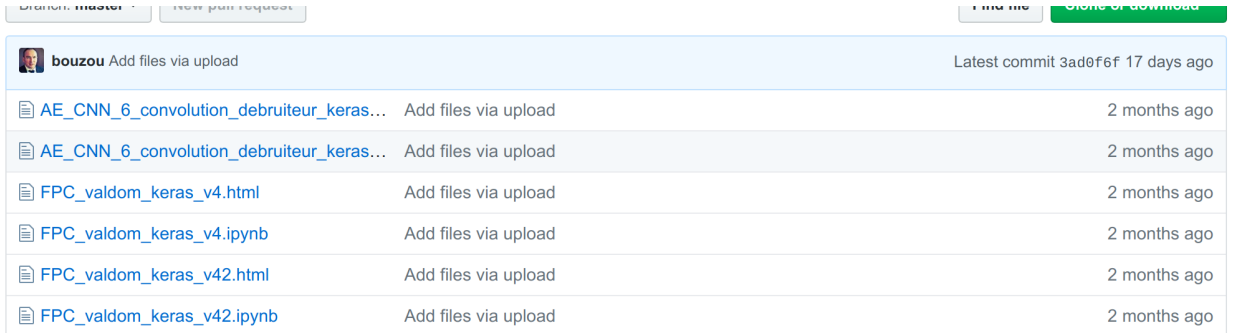
HARDWARE	
Processeur	Intel Core i7-8750 H@ 2.20GHz × 12
Carte graphique	NVIDIA GeForce GTX 1060 Max-Q 6Go
Mémoire	2×8 Go, DDR4, DDR4-2400 (jusqu'à 32 Go)
Stockage	SSD 512Go M.2 + HDD 1 To @ 5400 RPM)
SOFTWARE	
Ubuntu	Lts 18.04 Bionic Beaver
Anaconda	1.9.6
Jupyter Notebook	5.7.4
Nvidia	NVIDIA-SMI 396.54
	cudatoolkit 9.2
	cudnn 7.2.1
Python	version 3.6
Keras	version 2.2.4
Tensorflow-GPU	version 1.12.0

> Stack technique : code du modèle exécuté en mode local
(pas d utilisation Google colab, Google Platform … AWS)

> Unité de calcul : apprentissage du modèle en mode GPU uniquement

> Stack logiciel : mobilisation du framework TF-Keras apprentissage profond
(permet de faire converger le modèle CNN avec 3x couches convolution 2D)

> Github : le notebook Python Ms-Valdom est visible
https://github.com/bouzou/valdom_chifoumi



5. Modèle

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 598, 598, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 598, 598, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 299, 299, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 297, 297, 32)      9248
_________________________________________________________________
activation_2 (Activation)    (None, 297, 297, 32)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 148, 148, 32)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 146, 146, 64)      18496
_________________________________________________________________
activation_3 (Activation)    (None, 146, 146, 64)      0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 73, 73, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 341056)            0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                21827648
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 195
_________________________________________________________________
activation_5 (Activation)    (None, 3)                 0
=================================================================
Total params: 21,856,483
Trainable params: 21,856,483
Non-trainable params: 0
```

> Choix architecture du modèle apprentissage :

4 critères mobilisés pour sélectionner l'architecture adaptée a notre problème

. état de l'art des modèles d'apprentissage en computer vision

. vérification empirique modèle ranking top3 en compétition Kaggle

. utiliser un modèle qui passe l' échelle sur classif image multiclasses

. utiliser un CNN activable empiriquement sur des datasets de petites tailles

Modèle retenu = CNN a 3 couches convolutionnelles 2D [performeur sur Mnist]

> Typage des paramètres du CNN :

. fct activation Relu (Unité de Rectification Linéaire) dans les Conv2D

. MaxPooling2D 2 x 2 et un pas de 2 (couches d'agrégation spatiale, afin de permettre une invariance aux translations locales)

. dans la couche « fully connected »

- une fonction activation de sortie softmax (k=3)

. dropout = 0.5 technique de régularisation pour réduire l'over-fitting

. les métriques de performance du modèle

- fct de perte [loss] = categorical_crossentropy
- optimiseur = adam (adaptative moment estimation – Published as a conference paper at ICLR 2015)
- métrique = accuracy

> Mise en œuvre technique :

. prise en compte de la taille de nos entrées et déclinaison dans les couches

. tous les autres paramètres du modèle inchangés

. pas de travail de fine-tuning des paramètres par procédure de validation croisée

. pas d' utilisation de model de transfert learning VGG16 ou Inception V3

. epochs = 10 & batch_size = 25 [valeur max pour ce stack technique et de calculs]

. Code Py : introduction, d' un compteur de temps pour mesurer l' exécution des Epochs

6. Résultats

```
630
Epoch 10/10
11/11 [==============================] - 51s 5s/step - loss: 0.2857 - acc: 0.9105 - val_loss: 0.2060 - val_acc: 0.9
200
Learning TIme for 10 epochs : 477 seconds
```

```
##  Prédiction
```

```
ts = time.time()

score_conv_val = model_conv.evaluate_generator(validation_generator, test /batch_size, verbose=1)

score_conv_train = model_conv.evaluate_generator(train_generator, train / batch_size, verbose=1)

te = time.time()

t_prediction_conv_simple_model = te-ts
print('Train accuracy:', score_conv_train[1]*100)
print('Validation accuracy:', score_conv_val[1]*100)
print("Time Prediction: %.2f seconds" %t_prediction_conv_simple_model )
```

```
3/2 [==============================] - 6s 2s/step
12/11 [==============================] - 45s 4s/step
Train accuracy: 93.19727881019618
Validation accuracy: 94.2307690015206
Time Prediction: 50.79 seconds
```

A RETENIR :

Train accuracy	= 93.19 %	
Validation accuracy	= 94.23 %	
Temps d'apprentissage	= 477 sec ~ 8 minutes	
Taux Charge Max GPU	~ 99 %	[nvidia-smi]
Temp Max GPU	~ +65.0°C	[nvidia X server setting]
Temp Max CPU	~ +58.0°C	[sensors]

ANALYSE:

. Tx Acc. Validation CNN TF.Keras dans l'épure de cette métrique pour ce type de modèle
. Tx Acc. Val. CNN TF.Keras > Tx Acc Val. MLP Pytorch

Remarque : résultat à confirmer avec un nombre d'Epochs = 100,
C-a-d retrouver le résultat théorique nomimal Accuracy train > Accuracy Validation
pour garantir que le gradient a bien converge sur un minimum global.

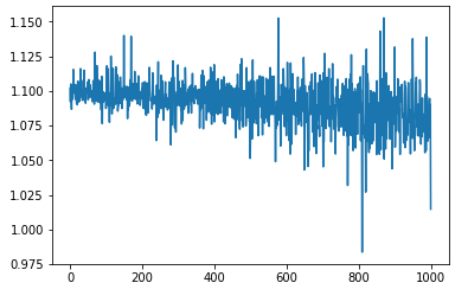
Implementation – Pytorch – version 3 MLP {Pytorch v1.0.0}

```
#== architecture du reseaux fully connected : Multi Layers Perceptron (a 3 couches)
#== + dropout
#== + 1 sortie log_softmax
```

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(32*32, 64)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(64, 32)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(32, 3)

    def forward(self, x):
        x = x.view(-1, 32*32)
        x = F.relu(self.fc1(x))
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)
        return F.log_softmax(self.fc3(x),dim=1)

model = Model()
if cuda:
    model.cuda()

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

print(model)
```

```
plt.plot(losses)
```

```
[<matplotlib.lines.Line2D at 0x7f6e804dedd8>]
```



Le perceptron multicouche (multilayer perceptron = MLP) est un type de réseau neuronal formel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement ; il s'agit donc d'un réseau à propagation directe (feedforward).

Entrée [32]:
```
model.eval()

output = model(evaluate_x)
pred = output.data.max(1)[1]
d = pred.eq(evaluate_y.data).cpu()

accuracy = d.sum().item()/d.size()[0]
print ('#=======================================')
print ('Performance de note classifieur en test a la 100e Epoch :')
print ('#=======================================')
print ('Accuracy = tx de Bien Classe', round((100*accuracy),2),'%')
TMC=round((1-accuracy),2)
print('Taux de mal Classe:', round((TMC*100),2),'%')
```

```
#=======================================
Performance de note classifieur en test a la 100e Epoch :
#=======================================
Accuracy = tx de Bien Classe 34.62 %
Taux de mal Classe: 65.0 %
```

Entrée [28]:
```
# == Analyse : sur le critere d accuracy, avec un small dataset, le MLP fait moins bien que le hasard (50%)
```

Entrée [29]:
```
# == 3  leviers d amelioration pour redresser l accuracy :
# levier 1 = faire de la "vrai data augmentation", en generant des images avec un reseau GAN
# levier 2 = utiliser une modele de transfert learning (ie inception v3)
# levier 3 = levier 2 + un mlp surcouche + une validation croisee kfolds (hyper parametre : nb de couches (?))
```

Entrée [30]:
```
# Affichage du temps d execution
print("Temps d execution : %s secondes ---" % round((time.time() - start_time),1))
```

```
Temps d execution : 1371.9 secondes ---
```

Entrée [31]:
```
# == Analyse : pour un GPU GTX1060M < 23 minutes, le temps de convergence est "nominal"
```

7. Annexes

Ms-Valdom « retrouver nos 16 livrables sur Github »

LISEZMOI.md https://github.com/bouzou/valdom_chifoumi

LE PROJET:

- date: 08.01.19
- auteur: didier lepicaut membre de l'équipe projet Valorisation des Donnees Massives
- objet: code pytorch d'algorithmes d'apprentissage statistique en Computer Vision
- version: mode projet, tests de codes, tests d'architectures

LES LIVRABLES:

suite à l'intervention de Pierre Boudier, expert de la société Nvidia, voici les codes de:

2018:

- Debruiteur RNN y = sin (x) - GRU - CPU - pytorque
- RNN Fit - y = sin (x) - CPU - pytorque
- RN Fit - y = sin (x) - Réseaux de neurones couche dense - GPU - keras ** mean_squared_error: 0.01% **
- CNN avec attaque-défense (GPU - pytorch)
- AE 6 couches de débruiteur d image - GPU - keras
- VAE CNN - GPU - pytorch
- AE - CPU - Pytorch
- AE - GPU - keras
- 1 CNN multi-classes k = 3, Chifouni - GPU - keras ** précisions en validation = 94.23% **

2019:

- GAN = Generative Adversarial Networks - GPU - Keras, 400 epochs, creer les chiffres 0 à 9 de MNIST