

DEEP LEARNING AVEC KERAS

APPLICATION À LA RECONNAISSANCE D'IMAGE

Brendan Guillouet

19 Décembre 2017

Institut National des Sciences Appliquées

INTRODUCTION

POURQUOI LE DEEP LEARNING ?

Le Boom du DEEP LEARNING :

- A partir de 2010, deux avancées académiques :
 - Utilisation de Relu contre le "vanishing gradient",
 - Drop out.
- Avancées technologiques :
 - Données disponibles en masse,
 - API simples à utiliser (*TensorFlow Keras*),
 - Capacités de calcul accrues (*cluster, GPU*).

Application au TRAITEMENT D'IMAGES :

- Une architecture adaptée : **Réseau de neurones à convolution**,
- Une grande variété d'applications,
- Des modèles déjà disponibles.

PLAN DE PRÉSENTATION

Introduction

De Nouveaux Framework

CPU VS GPU

La reconnaissance d'images

Classification d'image avec Keras

DE NOUVEAUX FRAMEWORK

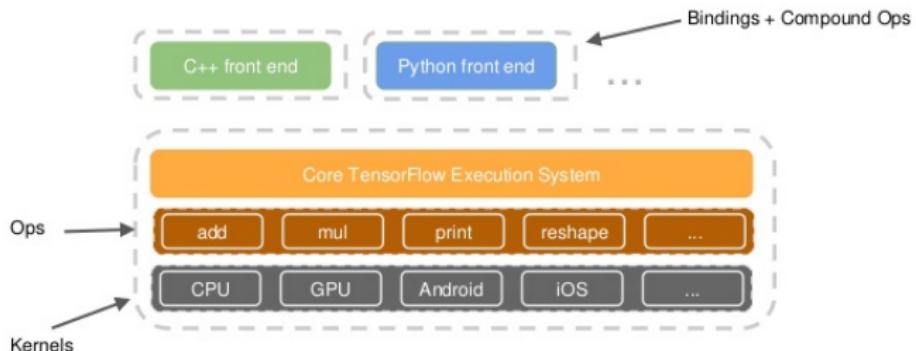
TENSORFLOW

Un framework open source développé par Google Brain (2015).

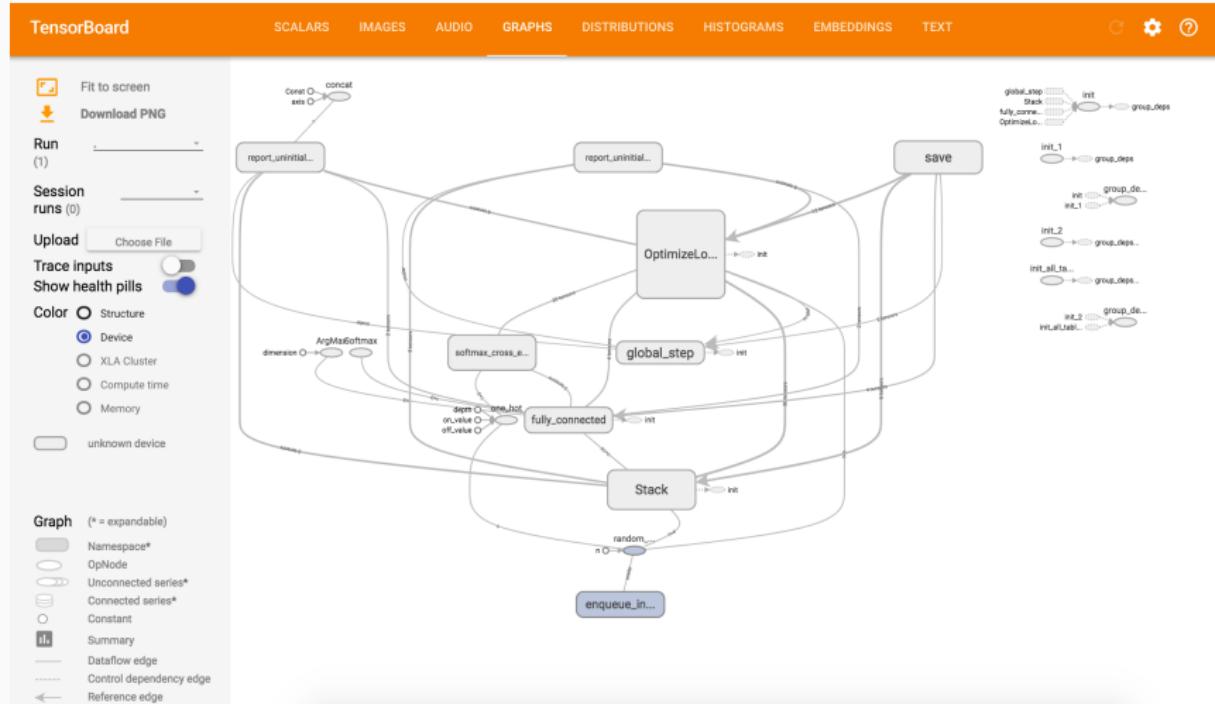
- Implémentation du noyau en C++/CUDA
- Différentes API (Python, Java, C++, Go)
- Différentes API de "Haut niveau" (Keras)



TensorFlow Architecture



TENSORBOARD



KERAS - POURQUOI UNE TELLE API?

Pour plus de simplicité!

Définition d'une couche de convolution en :



- **TENSORFLOW**

```
kernel = tf.Variable(tf.truncated_normal([3, 3, 64, 64], type=tf.float32, stddev=1e-1),
name='weights')
conv = tf.nn.conv2d(self.conv1_1, kernel, [1, 1, 1, 1], padding='SAME')
biases = tf.Variable(tf.constant(0.0, shape=[64], dtype=tf.float32), trainable=True,
name='biases')
out = tf.nn.bias_add(conv, biases)
self.conv1_2 = tf.nn.relu(out, name='block1_conv2')
```

- **KERAS**

```
x = Convolution2D(64, 3, 3, activation='relu', border_mode='same', name='block1_conv2')(x)
```

KERAS - AVANTAGES

- Librairie Python facile à utiliser
- Facile à combiner avec d'autres librairies python (Pandas, Numpy)
- Nombreuses possibilités
 - Convolution
 - LSTM
 - Pre-trained model
 - ...
- Englobe d'autre Frameworks (CNTK, Theano)
- Officiellement supporter par Google

Github : <https://github.com/fchollet/keras>

KERAS - MODELS

Il existent deux façon de construire des modèles de réseaux de neurones avec KERAS :

- SEQUENTIAL :

Pour construire des modèles personnalisés.

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(10, activation='softmax'))
...  
...
```

- MODEL API :

Pour utiliser des modèles pré-existant.

```
a = Input(shape=(32,))
b = Dense(32)(a)
model = Model(inputs=a, outputs=b)
```

KERAS - MODELS

Il existent deux façon de construire des modèles de réseaux de neurones avec KERAS :

- **SEQUENTIAL** : Utiliser pour les TP
Pour construire des modèles personnalisés.

```
model = Sequential()  
model.add(Dense(32, input_shape=(500,)))  
model.add(Dense(10, activation='softmax'))  
...
```

- MODEL API :
Pour utiliser des modèles pré-existant.

```
a = Input(shape=(32,))  
b = Dense(32)(a)  
model = Model(inputs=a, outputs=b)
```

KERAS - LAYERS

Les fonctions LAYERS permettent d'ajouter différentes couches à un modèle SEQUENTIAL, parmi lesquelles :

- DENSE
 - *Fully connected layer*
- ACTIVATION
 - *Relu, sigmoïd, tanh*
- DROPOUT
- FLATTEN, RESHAPE
- CONV2D
 - *Réseau de convolution*
- MAXPOOLING2D
- LSTM
 - *Long Short-Term Memory*

...

KERAS - APPLICATIONS

Keras permet d'utiliser plusieurs modèles de classification d'images entraînés sur la base *ImageNet*.

- XCEPTION
- VGG16
- VGG19
- RESNET50
- INCEPTIONV3
- INCEPTIONRESNETV2
- MOBILENET

KERAS - EXEMPLE

```
# Définition du réseau
model = km.Sequential()
model.add(kl.Dense(512, activation='relu', input_shape=(784,)))
model.add(kl.Dropout(0.2))
model.add(kl.Dense(N_classes, activation='softmax'))

# Compilation
model.compile(loss='categorical_crossentropy',
              optimizer=ko.RMSprop(),
              metrics=['accuracy'])

# Apprentissage
history = model.fit(X_train, Y_train_cat,
                      batch_size=batch_size,
                      epochs=epochs,
                      verbose=1,
                      validation_data=(X_test, Y_test_cat))

# Evaluation
score_mpl = model.evaluate(X_test, Y_test_cat, verbose=0)
```

BIEN D'AUTRES FRAMEWORKS

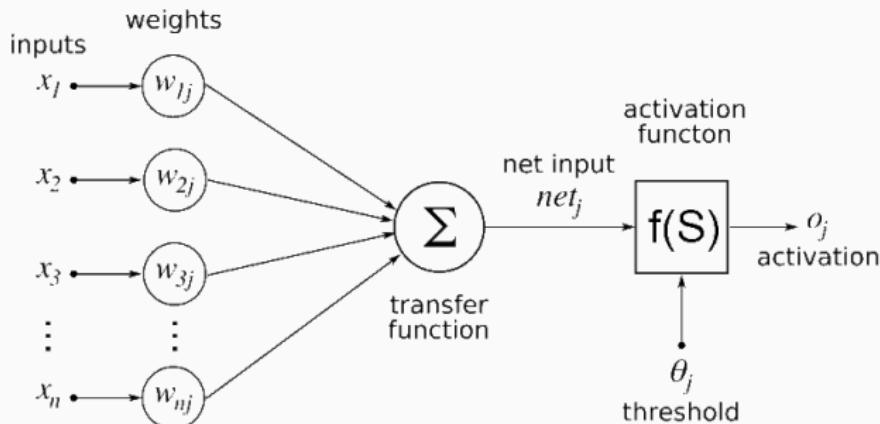
- Caffe, *Berkeley Vision and Learning Center (BVLC)*, 2013
- Theano, *University of Montréal*, 2009
- Lasagne, (*High level API of Theano*)
- CNTK *Microsoft*, 2016
- Torch, Pytorch
- ...

CPU VS GPU

DEEP LEARNING ET APPRENTISSAGE

L'apprentissage d'un réseau de neurone est composé de deux opérations principales :

- **Forward Pass** : Les *input* passent à travers le réseau entier jusqu'à obtenir une valeur en sortie.
- **Backward Pass** Les poids de chaque neurone sont mis à jour à partir de l'erreur obtenue pendant l'étape forward.



⇒ Essentiellement des multiplications de matrices.

Essentiellement des multiplications de matrice :

column vectors

row vectors

$$\begin{array}{c} \mathbf{a} \xrightarrow{\text{row vectors}} \left(\begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \right) \\ \mathbf{b} \xrightarrow{\text{row vectors}} \left(\begin{array}{ccc} b_1 & b_2 & b_3 \end{array} \right) \\ \mathbf{c} \xrightarrow{\text{row vectors}} \left(\begin{array}{ccc} c_1 & c_2 & c_3 \end{array} \right) \end{array} \left| \begin{array}{ccc} \mathbf{x} & \mathbf{y} & \mathbf{z} \\ \downarrow & \downarrow & \downarrow \\ \begin{array}{ccc} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{array} \end{array} \right.$$

PROBLÈME : Calcul simple mais en très grand nombre.

Modèle VGG : 138,357,544 paramètres à optimiser à chaque itération !

SOLUTION : Le calcul GPU.

CPU VS GPU

CPU :

- Peu de cœur....
- .. mais très complexe et rapide,
- mémoire partagée avec le système.

⇒ Taches séquentielles.

GPU :

- Beaucoup de cœur...
- ...mais peu complexe et peu rapide,
- mémoire séparée du système.

⇒ Taches en parallèles.

A L'INSA - GEI 103

CPU : INTEL XEON E5-1620 v4		GPU : GEFORCE GTX 1080
Cores	4 (8 threads)	2560
Fréquence	3,5/3,8Ghz	1,6/1,73Ghz

REMARQUES

- Pour les autres algorithmes d'apprentissage le GPU n'est pas forcément nécessaire.
 - *Complexité moindre*
- Le temps de chargement des données vers le GPU peut-être coûteux!
 - *GPU n'est utile que si le temps de calcul est supérieur au temps de chargement.*
 - *i.e. utile pour des modèle complexe*

LA RECONNAISSANCE D'IMAGES

APPLICATION - CLASSIFICATION SIMPLE

Your specialized huggable recommendation

Go for it! Hug it out. With a score of **0.695779**, we're somewhat sure.



Your specialized huggable recommendation

Don't hug that. Please. With a score of **0.999875**, we're pretty sure.



Your specialized huggable recommendation

Don't hug that. Please. With a score of **0.778686**, we're pretty sure.



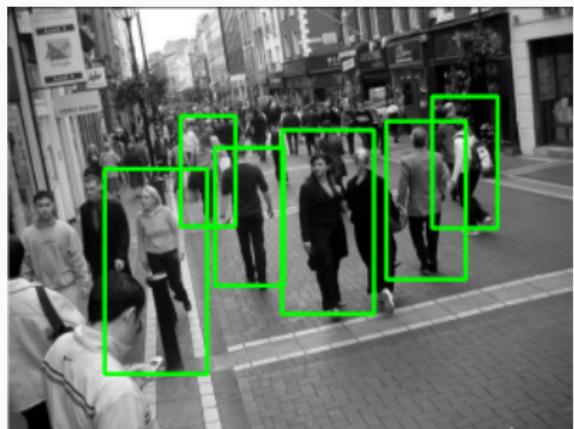
Your specialized huggable recommendation

Go for it! Hug it out. With a score of **0.958104**, we're pretty sure.

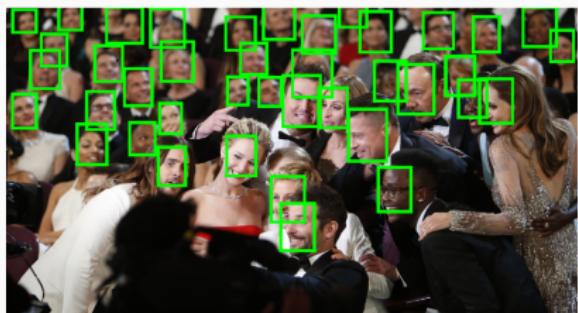


APPLICATION - RECONNAISSANCE DE PATTERNS

Caméra de sécurité



Facebook

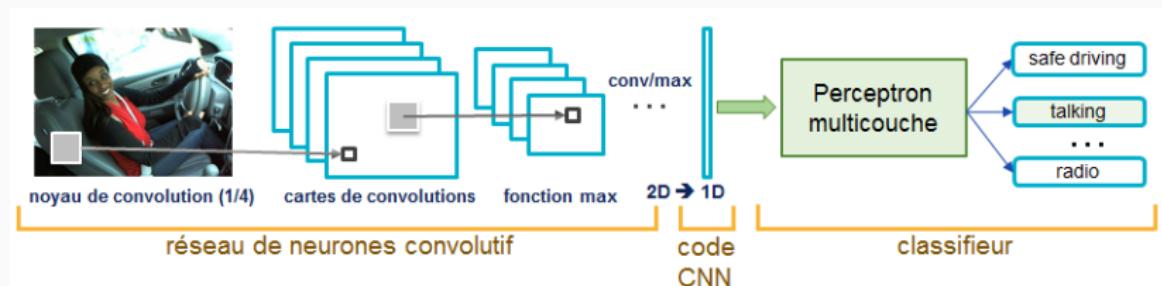


RÉSEAU DE NEURONES CONVOLUTIFS

Les réseaux de neurones classiques, *perceptron multicouche*, ne sont pas adaptés aux traitement d'images :

- Appliqués sur des vecteurs
- Perte d'information

SOLUTION : Réseau de neurones convolutifs.



RÉSEAU DE NEURONES CONVOLUTIFS - EXEMPLE

PROBLÈME : La complexité explose rapidement.

EXEMPLE : Modèle simple avec keras.

DÉFINITION

```
model_conv = km.Sequential()
# Convolutional part
model_conv.add(kl.Conv2D(32, (3, 3), input_shape=(150, 150, 3), data_format="channels_last"))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.MaxPooling2D(pool_size=(2, 2)))

# 2D to 1D
model_conv.add(kl.Flatten())

# Classifier part
model_conv.add(kl.Dense(32))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.Dropout(0.5))
model_conv.add(kl.Dense(1))
model_conv.add(kl.Activation('sigmoid'))
```

RÉSEAU DE NEURONES CONVOLUTIFS - EXEMPLE

EXEMPLE : Modèle simple avec keras.

SUMMARY

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	401920
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130
Total params: 669,706		

RÉSEAU DE NEURONES CONVOLUTIFS - EXEMPLE

EXEMPLE : Modèle simple avec keras.

SUMMARY

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	401920
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130
=====		
Total params: 669,706		

⇒ 5,608,385 de paramètres!

RÉSEAU DE NEURONES CONVOLUTIFS - REMARQUE

PROBLÉMATIQUE pour des applications industrielles :

- réflexion sur l'architecture du réseau,
- nécessite beaucoup de données,
- réseau plus complexe,
- temps d'apprentissage très long.

SOLUTION : Le *transfert learning*.

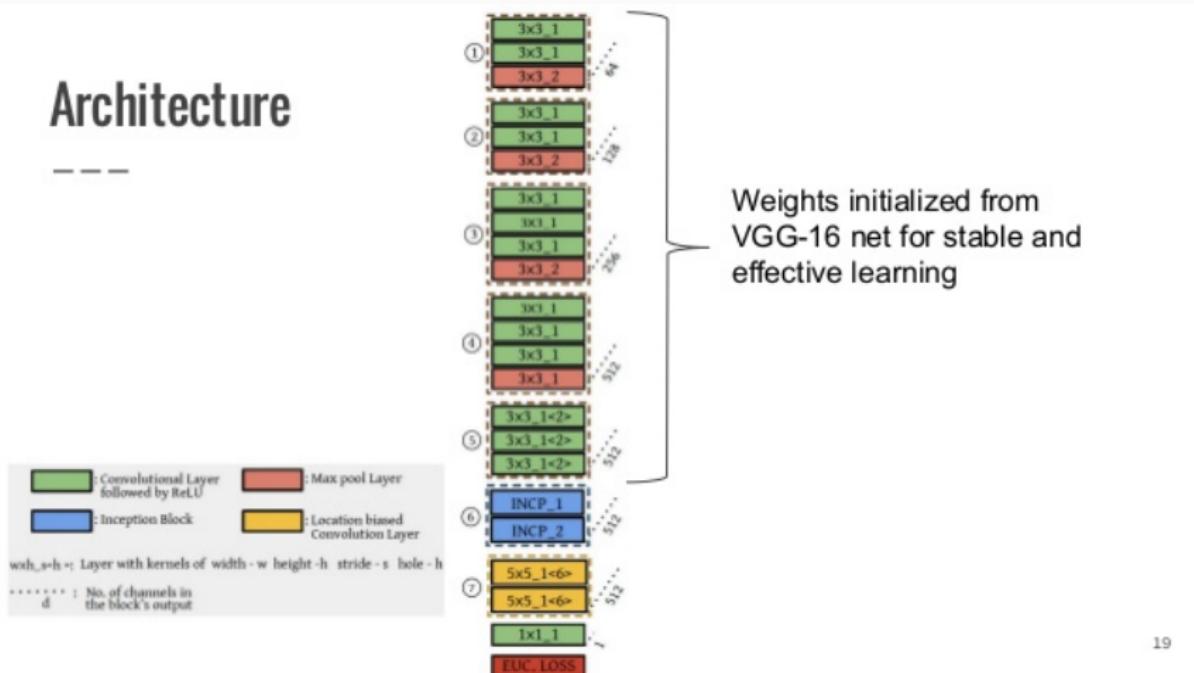
Utiliser des réseaux de convolutions déjà entraînés.

Appris sur de très gros jeux d'apprentissage.

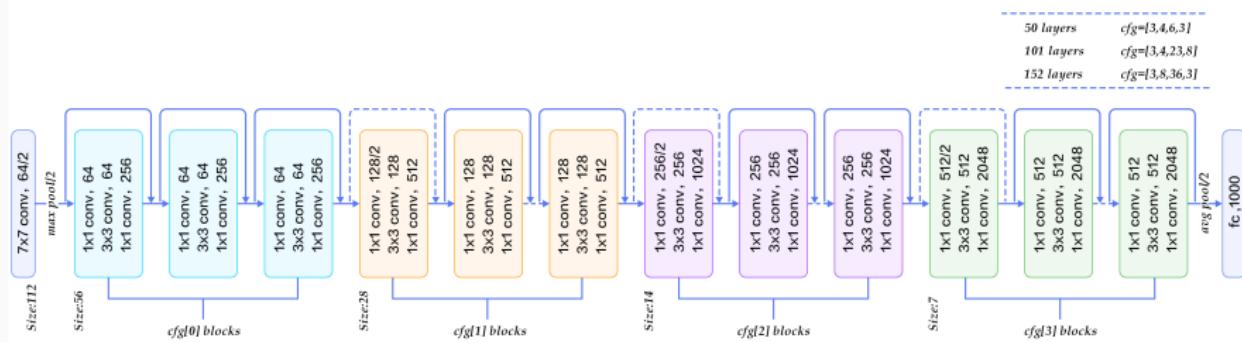


- 1000 classes
 - 1,2 M d'images pour l'apprentissage
 - 100k d'image pour la validation

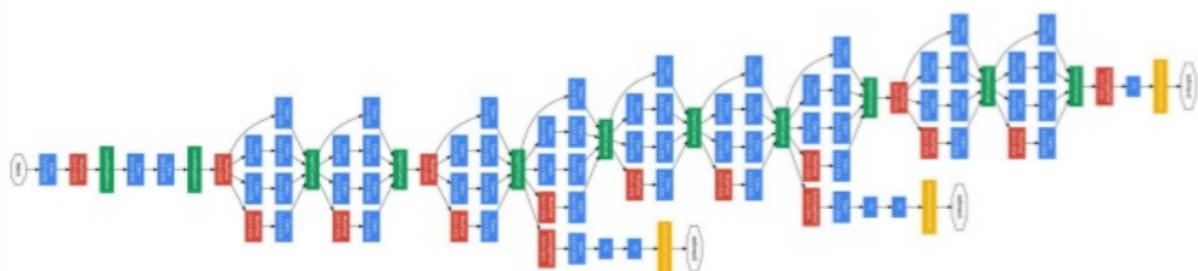
Architecture



RÉSEAU PRÉ-ENTRAÎNÉ : RESNET



The Inception Architecture (GoogLeNet, 2014)



Going Deeper with Convolutions

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov,
Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich

ArXiv 2014, CVPR 2015



CLASSIFICATION D'IMAGE AVEC KERAS

INTRODUCTION

Deux cas d'études :

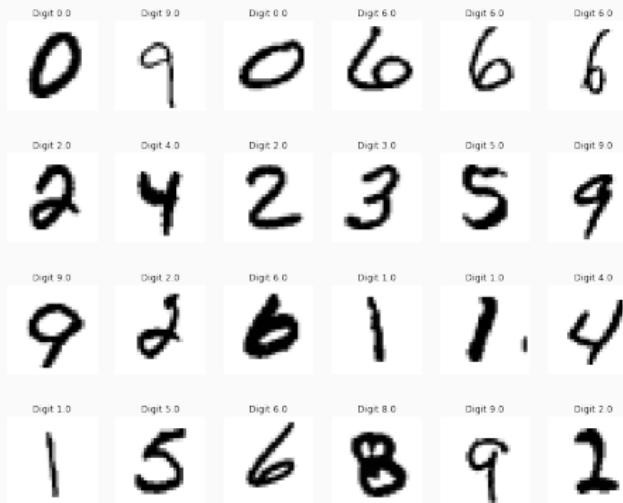
- Problème *simple* : **MNIST**.
 - Comparaison avec les autres algorithmes étudiés (*random forest*).
 - Performance du GPU.
- Problème plus *complexe* : **Cats VS Dogs**.
 - Utilisation des modèles pré-entraînés.
 - Performance du GPU.

CLASSIFICATION D'IMAGE AVEC KERAS

MNIST

MNIST

- $28 \times 28 = 784$ pixels.
- 1 niveau de couleur : gris.
- Apprentissage = 60.000, Test = 10.000.



MNIST - PERCEPTRON MULTICOUCHE

INPUT

```
model = km.Sequential()
model.add(kl.Dense(512, activation='relu', input_shape=(784,)))
model.add(kl.Dropout(0.2))
model.add(kl.Dense(512, activation='relu'))
model.add(kl.Dropout(0.2))
model.add(kl.Dense(N_classes, activation='softmax'))
```

input_shape = (nombre de paramètre,)

OUTPUT

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	401920
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130

669,706 paramètres

MNIST - RÉSEAU DE CONVOLUTION

INPUT

```
# Convolution
model.add(kl.Conv2D(32, kernel_size=(3, 3),
                    activation='relu',
                    input_shape=(28,28, 1), data_format="channels_last"))
model.add(kl.Conv2D(64, (3, 3), activation='relu'))
model.add(kl.MaxPooling2D(pool_size=(2, 2)))
model.add(kl.Dropout(0.25))

# 2D to 1D
model.add(kl.Flatten())

# Classifier
model.add(kl.Dense(128, activation='relu'))
model.add(kl.Dropout(0.5))
model.add(kl.Dense(N_classes, activation='softmax'))
```

input_shape = (Nombre de pixel x, Nombre de pixel y, nombre de niveaux de couleurs)
data_format = "Où est le niveau de couleur"

MNIST - RÉSEAU DE CONVOLUTION

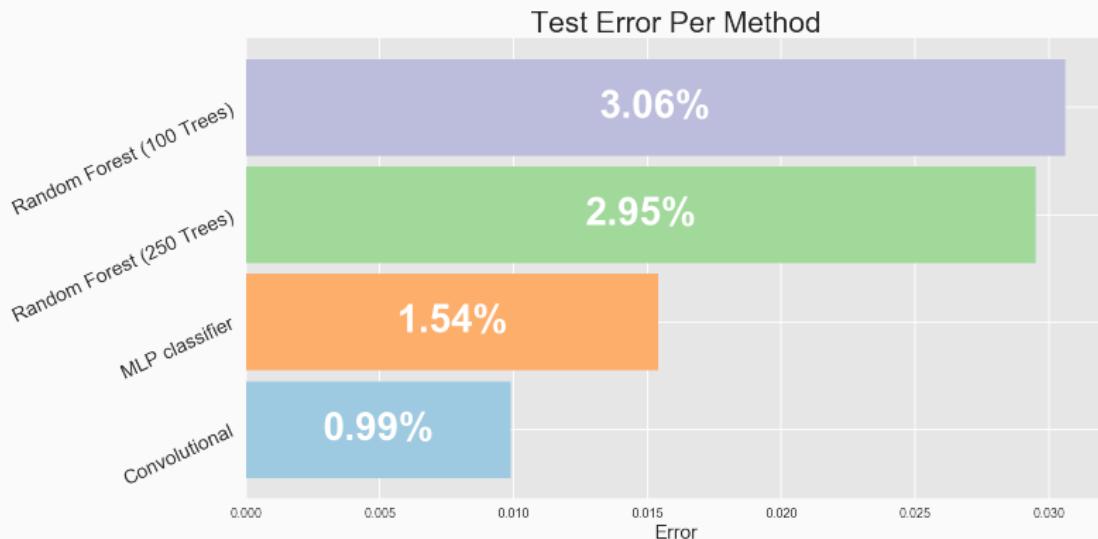
OUTPUT

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_5 (Dropout)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_7 (Dense)	(None, 128)	1179776
dropout_6 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 10)	1290
Total params: 1,199,882		

1,199,882 paramètres

MNIST - ERREUR DE CLASSIFICATION - TEST

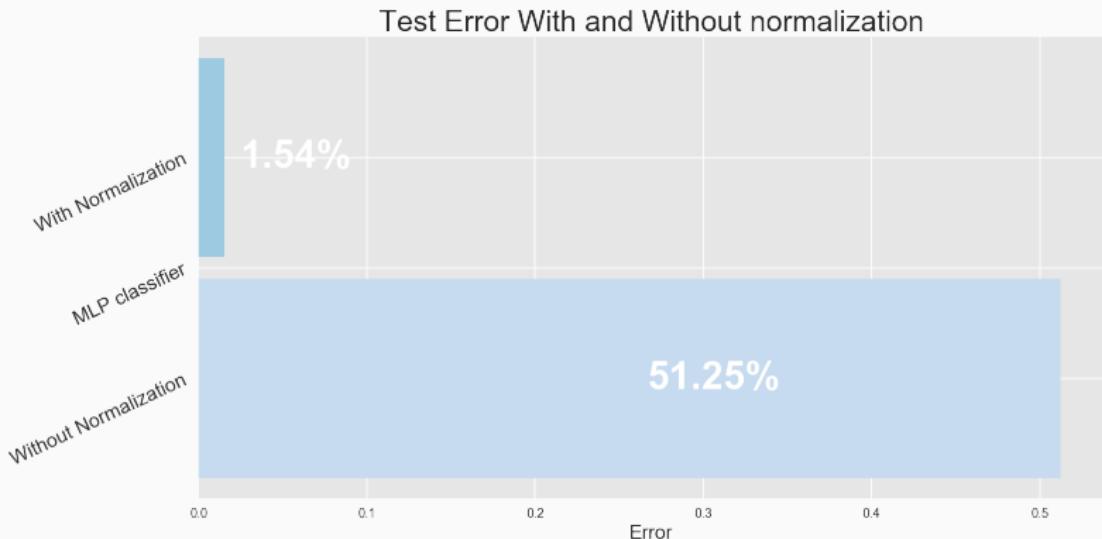
batch_size = 128, epochs = 20



Amélioration assez nette des résultats.

MNIST - ERREUR DE CLASSIFICATION - TEST - NORMALISATION

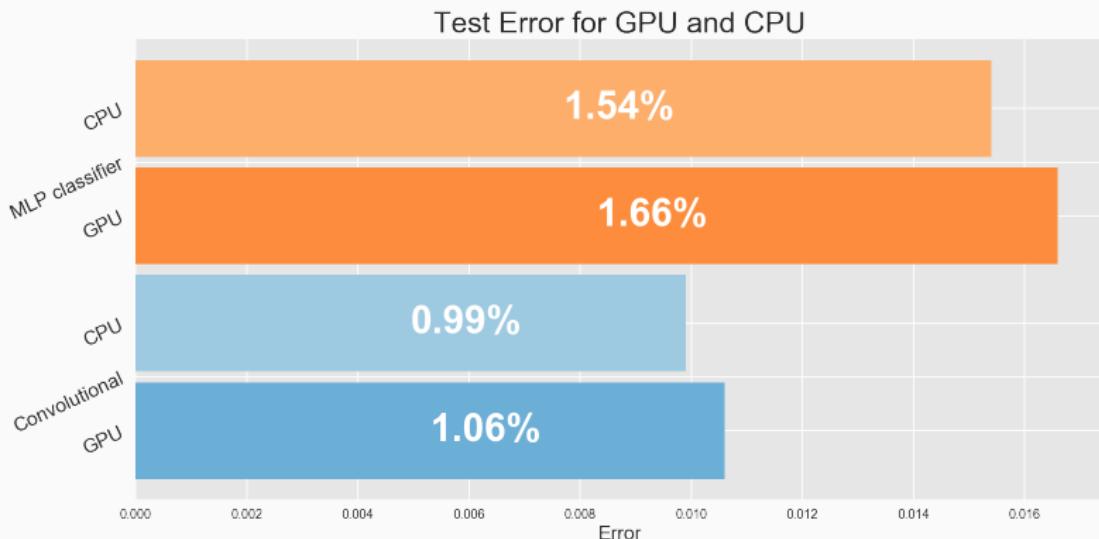
batch_size = 128, epochs = 20



Très grande influence de la normalisation des images sur le résultat.

MNIST - ERREUR DE CLASSIFICATION - TEST - CPU/GPU

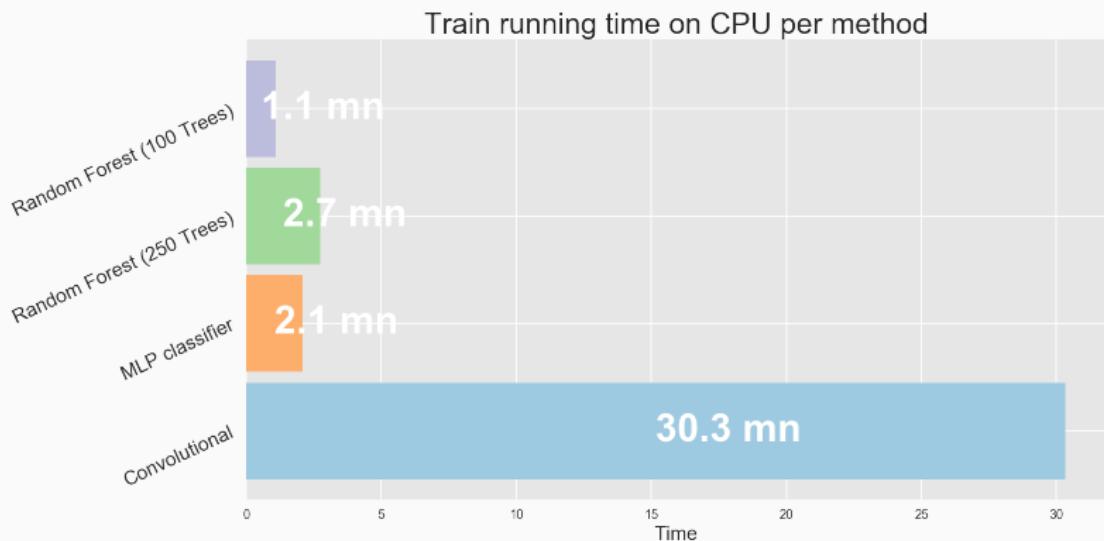
batch_size = 128, epochs = 20



Des résultats stables entre le CPU et le GPU.

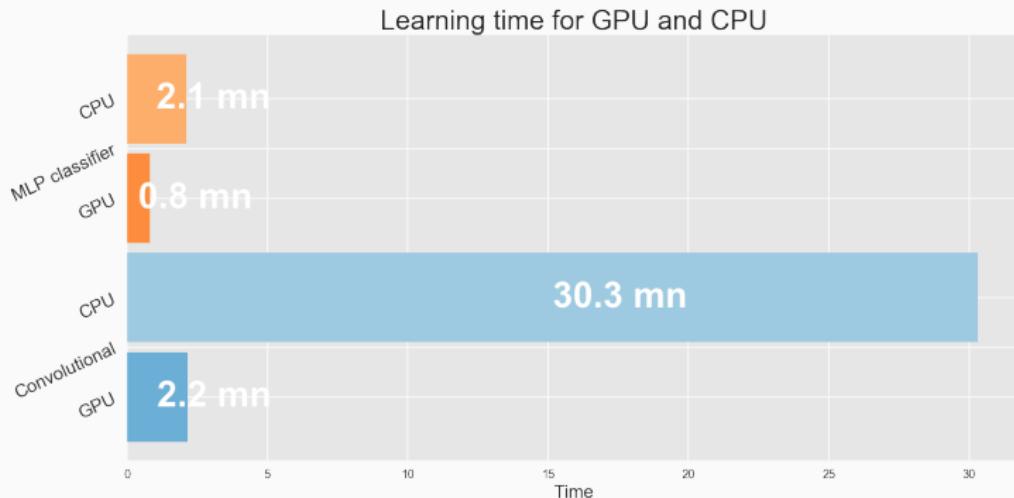
MNIST - TEMPS D'APPRENTISSAGE - CPU

batch_size = 128, epochs = 20 - Machine GEI-103



MNIST - TEMPS D'APPRENTISSAGE - CPU/GPU

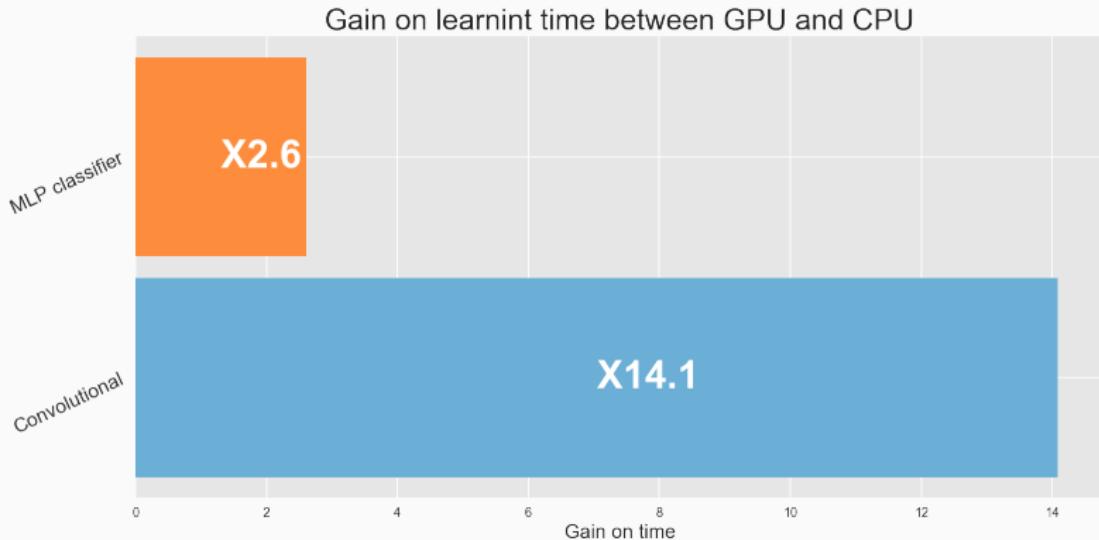
batch_size = 128, epochs = 20 - Machine GEI-103



Le GPU permet un très net gain sur le temps d'apprentissage.

MNIST - TEMPS D'APPRENTISSAGE - CPU/GPU

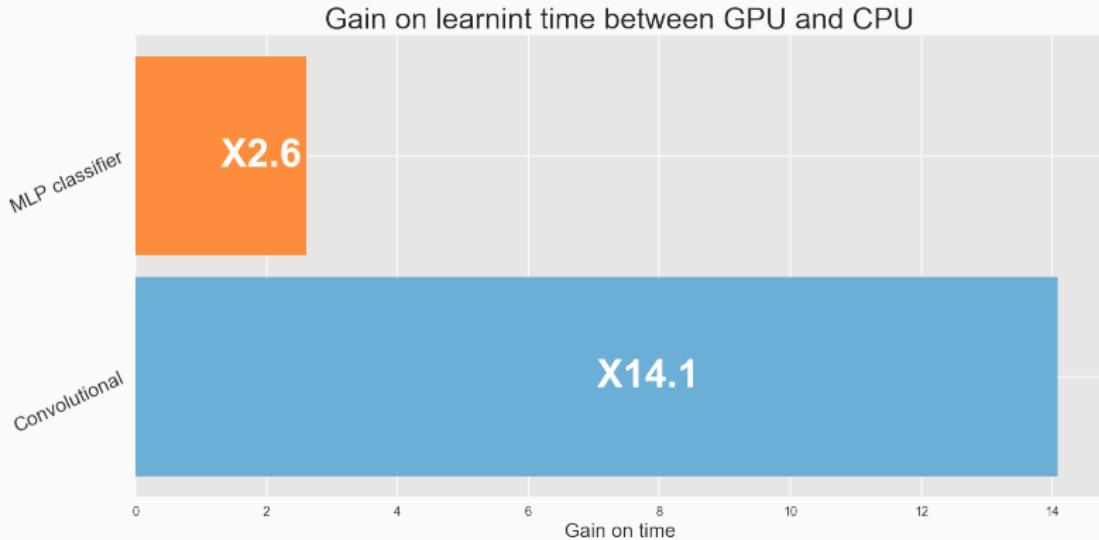
batch_size = 128, epochs = 20 - Machine GEI-103



Comment expliquer cette différence ?

MNIST - TEMPS D'APPRENTISSAGE - CPU/GPU

batch_size = 128, epochs = 20 - Machine GEI-103



Comment expliquer cette différence ?

- Réseau de convolution plus complexe..
- ... Coût du transfert des données plus vite atténué.

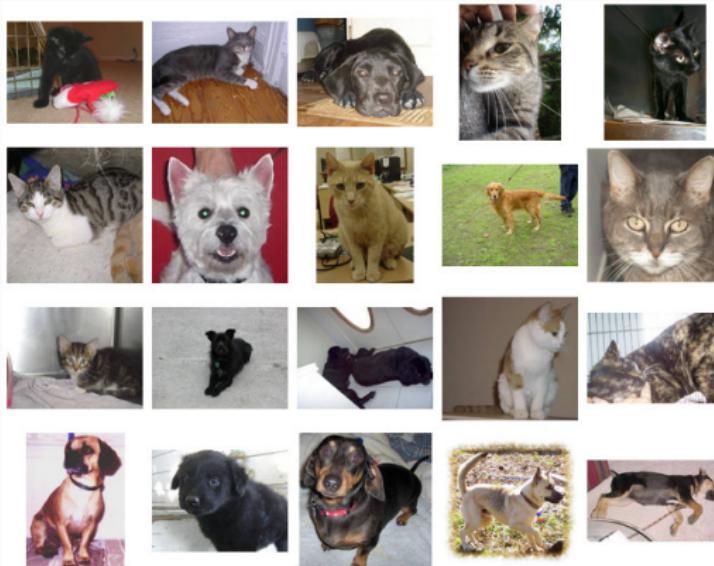
CLASSIFICATION D'IMAGE AVEC KERAS

CATS Vs DOGS

CATS VS DOGS

Un jeu de donnée plus complexe :

- Images de tailles différentes.
- 3 niveaux de couleurs : bleu, rouge, vert.
- Jeu de données 25.000 images.



CATS VS DOGS - AUGMENTATION DES DONNÉES

1/ Déterminer les transformations à appliquer :

```
datagen = kpi.ImageDataGenerator(  
    rotation_range=40,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True,  
    fill_mode='nearest')
```

- Rotation.
- Translation.
- Standardisation.
- ZCA Whitening.

2/ Générer des transformations aléatoires depuis un dossier d'images

```
train_generator = train_datagen.flow_from_directory(  
    data_dir_sub+"/train/",  
    target_size=(img_width, img_height),  
    batch_size=batch_size,  
    class_mode='binary')
```

- Fonction Lazy.
- Image de même dimension.
- Empêche le sur-apprentissage.

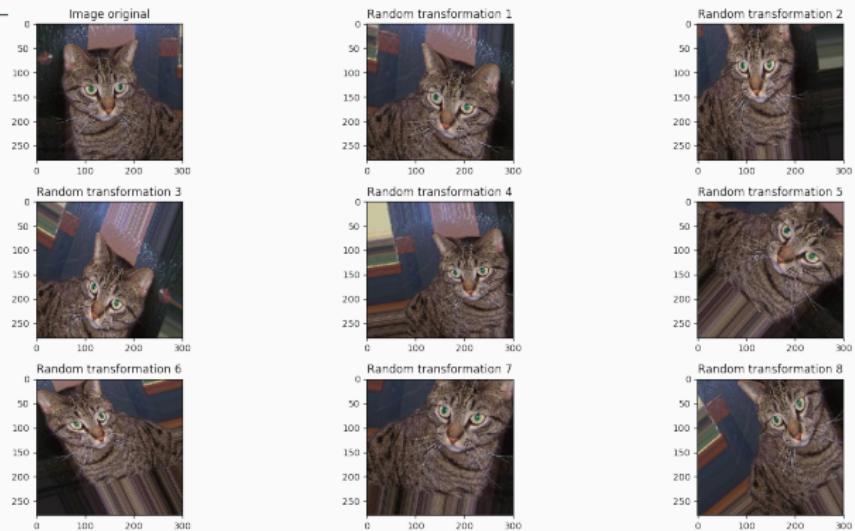
CATS VS DOGS - AUGMENTATION DES DONNÉES

3/ Apprentissage sur les données augmentées

```
model_conv.fit_generator(train_generator, steps_per_epoch=N_train // batch_size, epochs=epochs,  
validation_data=validation_generator, validation_steps=N_val // batch_size)
```

steps_per_epoch = Nb individu / batch_size

Exemple de transformation :



CATSVSDGOGS - RÉSEAU DE CONVOLUTION

DEFINITION

```
model_conv = km.Sequential()
model_conv.add(kl.Conv2D(32, (3, 3), input_shape=(img_width, img_height, 3),
                      data_format="channels_last"))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.MaxPooling2D(pool_size=(2, 2)))

model_conv.add(kl.Conv2D(32, (3, 3)))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.MaxPooling2D(pool_size=(2, 2)))

model_conv.add(kl.Conv2D(64, (3, 3)))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.MaxPooling2D(pool_size=(2, 2)))

model_conv.add(kl.Flatten()) # this converts our 3D feature maps to 1D feature vectors
model_conv.add(kl.Dense(64))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.Dropout(0.5))
model_conv.add(kl.Dense(1))
model_conv.add(kl.Activation('sigmoid'))
```

input_shape = (Nombre de pixel x, Nombre de pixel y, nombre de niveau de couleur)
data_format = "Où est le niveau de couleur"

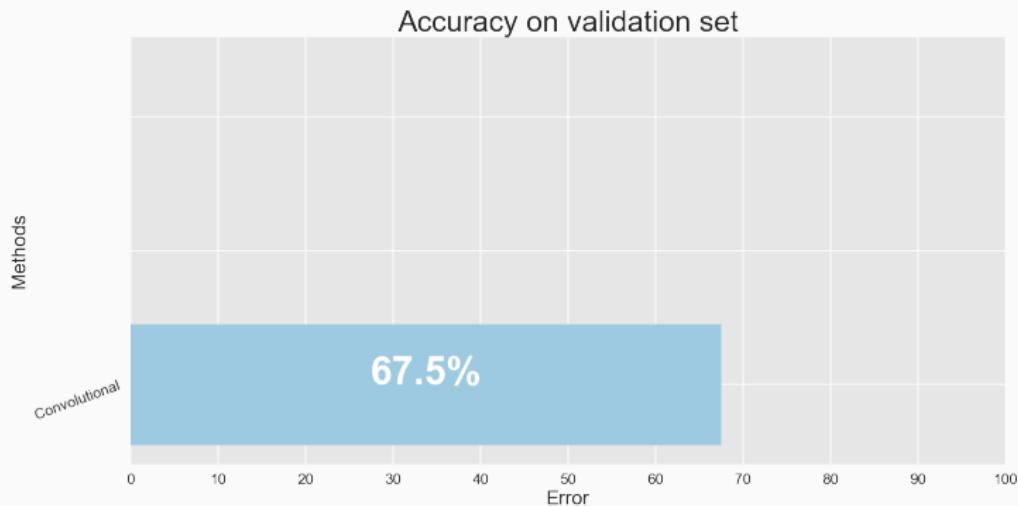
CATSVSDGOGS - RÉSEAU DE CONVOLUTION

SUMMARY (1,212,513 paramètres)

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 32)	896
activation_1 (Activation)	(None, 148, 148, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 32)	9248
activation_2 (Activation)	(None, 72, 72, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 32)	0
conv2d_3 (Conv2D)	(None, 34, 34, 64)	18496
activation_3 (Activation)	(None, 34, 34, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 17, 17, 64)	0
flatten_1 (Flatten)	(None, 18496)	0
dense_1 (Dense)	(None, 64)	1183808
activation_4 (Activation)	(None, 64)	0
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65
activation_5 (Activation)	(None, 1)	0

CATSVSDGOGS - VALIDATION SCORE

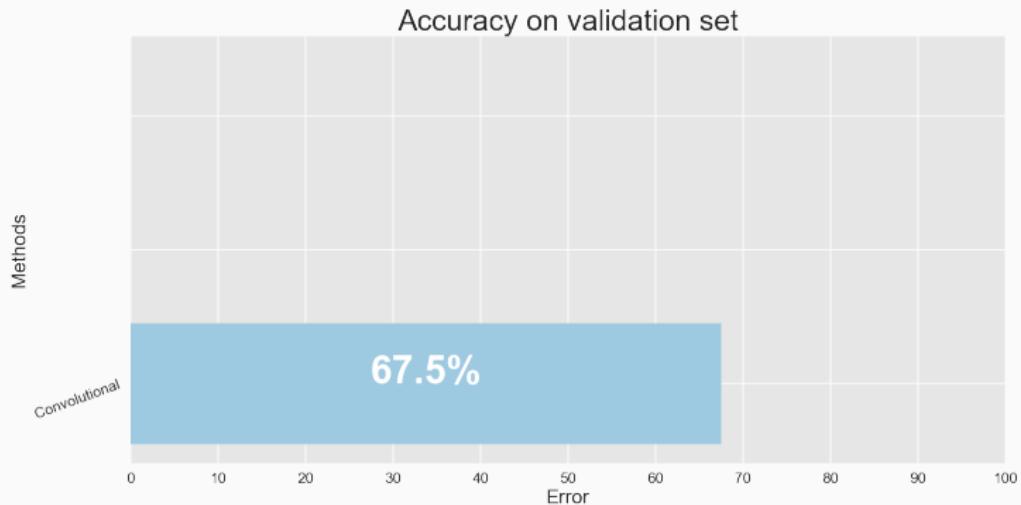
batch_size = 200, epochs = 15



Résultats assez faible.

CATSVSDGOGS - VALIDATION SCORE

batch_size = 200, epochs = 15



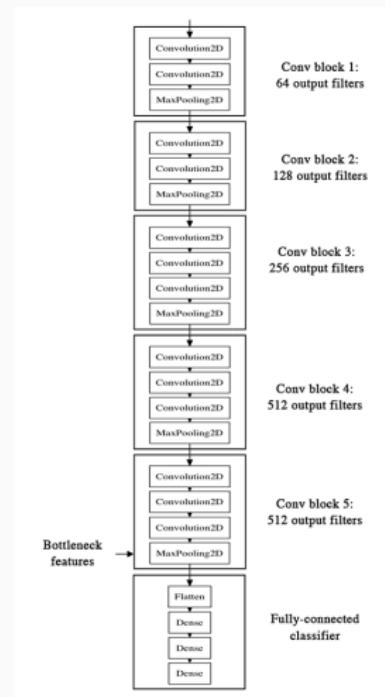
Résultats assez faible.

⇒ Utilisation d'un modèle pré-entraîné.

CATSVSDGOGS - RÉSEAU PRÉ-ENTRAÎNÉ

Exemple : Réseau VGG-16 : 138,357,544 paramètres.

Utilisation en deux étapes :

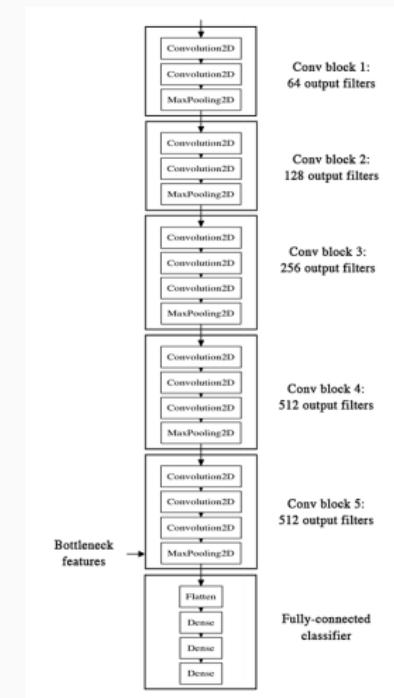


CATSVSDGOGS - RÉSEAU PRÉ-ENTRAINÉ

Exemple : Réseau VGG-16 : 138,357,544 paramètres.

Utilisation en deux étapes :

- **Extracteur de features** : On utilise les blocs de convolutions pour extraire des features

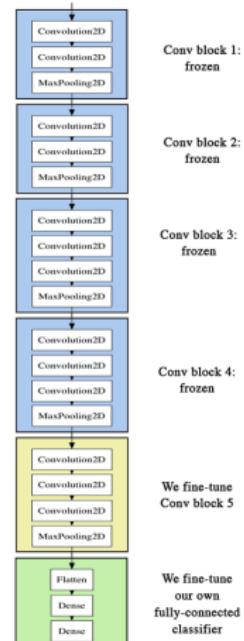


CATSVSDGOGS - RÉSEAU PRÉ-ENTRAINÉ

Exemple : Réseau VGG-16 : 138,357,544 paramètres.

Utilisation en deux étapes :

- **Extracteur de features** : On utilise les blocs de convolutions pour extraire des features
- **Fine Tuning** : On re-apprends les paramètres des deux derniers blocs



CATSVSDGOGS - VGG-16 - 1/ EXTRACTEUR DE FEATURES

```
# On charge le modèle sans le dernier bloc de classification :
model_VGG16_without_top = ka.VGG16(include_top=False, weights='imagenet')

# Model summary
Layer (type)          Output Shape         Param #
=====
input_1 (InputLayer)   (None, None, None, 3)  0
-----
block1_conv1 (Conv2D)  (None, None, None, 64)  1792
...
-----
block5_pool (MaxPooling2D) (None, None, None, 512)  0
=====

# Features extraction

datagen = kpi.ImageDataGenerator(rescale=1. / 255)
generator = datagen.flow_from_directory(
    data_dir_sub+"/train",
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)
features_train = model_VGG16_without_top.predict_generator(generator, N_train//batch_size, verbose=1)
```

- Pas d'augmentation !
- features_train de dimension (4, 4, 512)

CATSVSDGOGS - VGG-16 - 1/ EXTRACTEUR DE FEATURES

On définit un classifieur simple prenant en entrée les données extraites du modèle VGG-16 :

```
model_VGG_fcm = km.Sequential()
# La première couche "aplatis" les données
model_VGG_fcm.add(kl.Flatten(input_shape=features_train.shape[1:]))
model_VGG_fcm.add(kl.Dense(64, activation='relu'))
model_VGG_fcm.add(kl.Dropout(0.5))
model_VGG_fcm.add(kl.Dense(1, activation='sigmoid'))
```

- input_shape de dimension (4, 4, 512)

Layer (type)	Output Shape	Param #
=====		
flatten_2 (Flatten)	(None, 8192)	0

dense_3 (Dense)	(None, 64)	524352

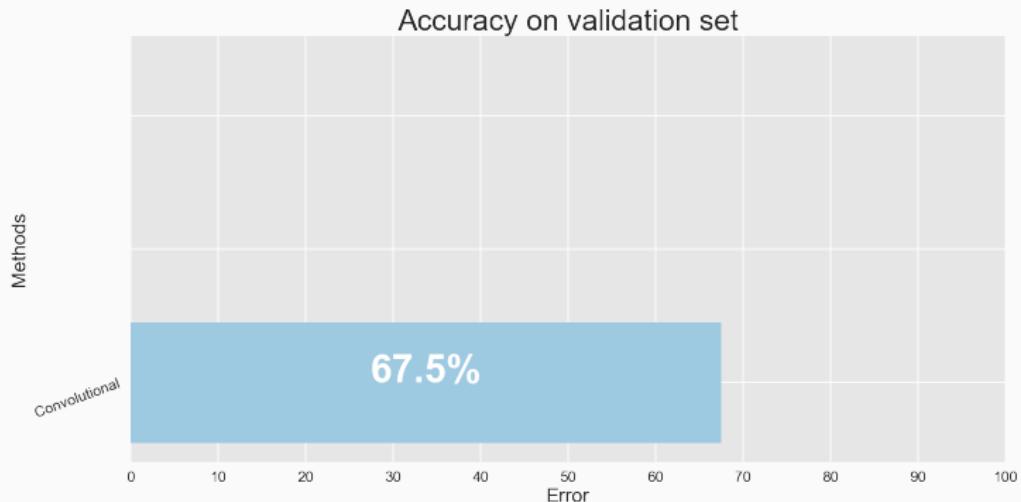
dropout_2 (Dropout)	(None, 64)	0

dense_4 (Dense)	(None, 1)	65
=====		

Modèle très simple utilisé sur des features issues d'un modèle très élaboré.

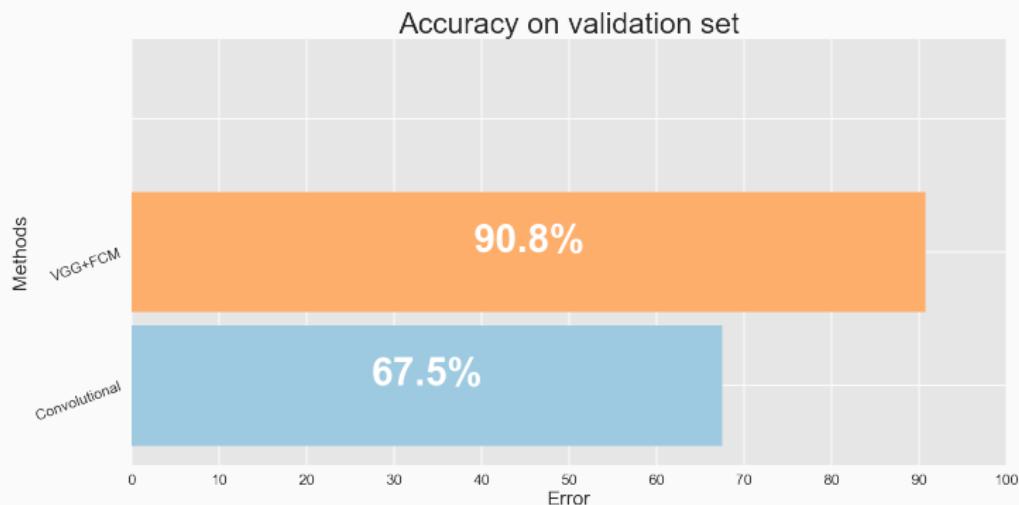
CATSVSDGOGS - VALIDATION SCORE

batch_size = 200, epochs = 15



CATSVSDGOGS - VALIDATION SCORE

batch_size = 200, epochs = 15



Nette amélioration!

CATSVSDGOGS - VGG-16 - 2/ FINE TUNING

On règle les poids des deux derniers blocs du modèle :

```
# Même modèle que précédemment
top_model = km.Sequential()
top_model.add(kl.Flatten(input_shape=model_VGG16_without_top.output_shape[1:]))
top_model.add(kl.Dense(64, activation='relu'))
top_model.add(kl.Dropout(0.5))
top_model.add(kl.Dense(1, activation='sigmoid'))

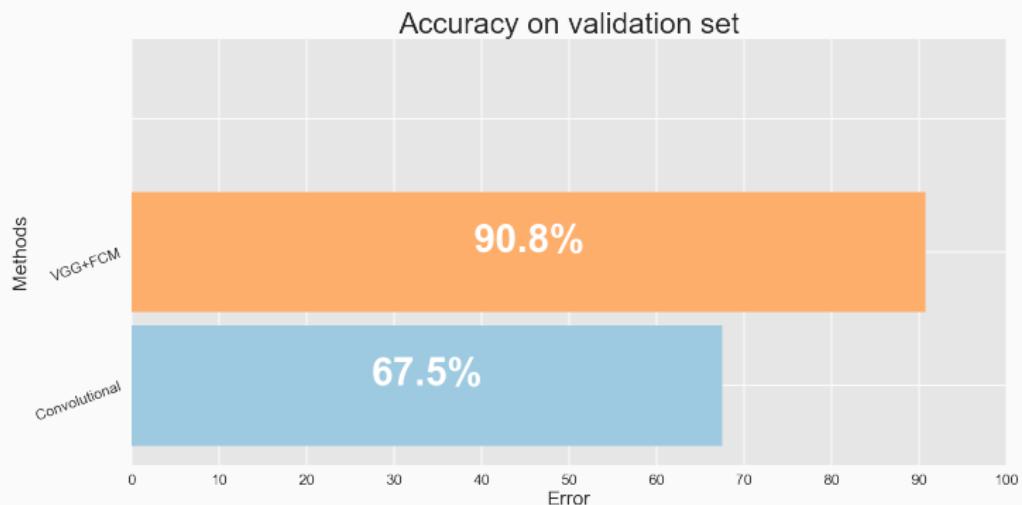
# On charge les poids du modèle évalué précédemment
top_model.load_weights(data_dir_sub+'/' + MODE + '_weights_model_VGG_fully_connected_model_'

# Add the classifier model on top of the convolutional base
model_VGG_LastConv_fcm = km.Model(inputs=model_VGG16_without_top.input,
outputs=top_model(model_VGG16_without_top.output))

#Summary
-----
Layer (type)          Output Shape         Param #
=====
input_3 (InputLayer)   (None, 150, 150, 3)      0
block1_conv1 (Conv2D)    (None, 150, 150, 64)     1792
...
block5_pool (MaxPooling2D) (None, 4, 4, 512)      0
sequential_5 (Sequential) (None, 1)                524417
=====
Total params: 15,239,105
```

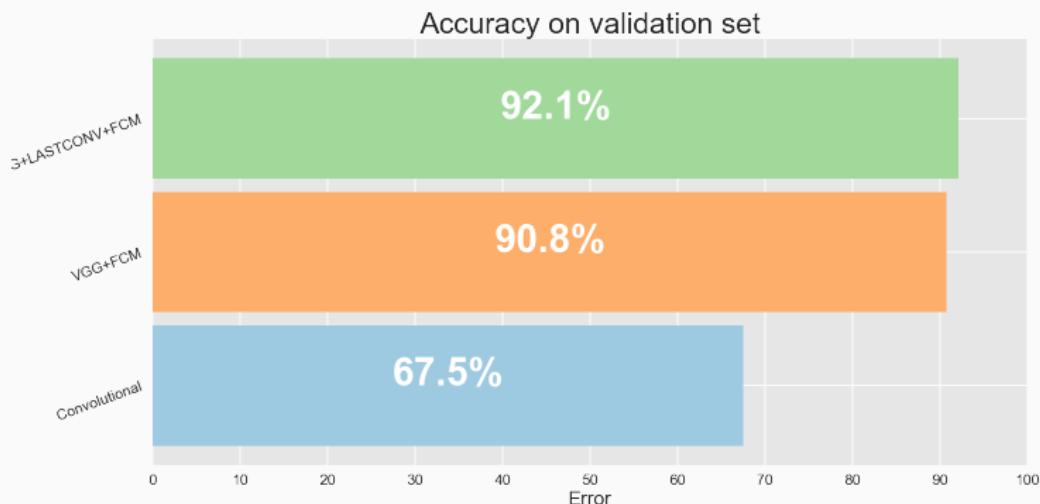
CATSVSDGOGS - VALIDATION SCORE

batch_size = 200, epochs = 15



CATSVSDGOGS - VALIDATION SCORE

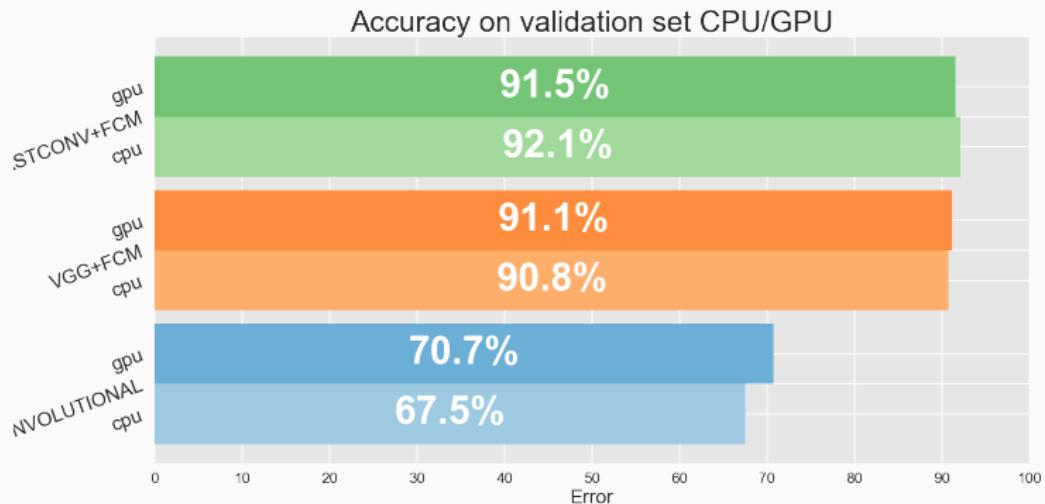
batch_size = 200, epochs = 15



Résultat encore amélioré

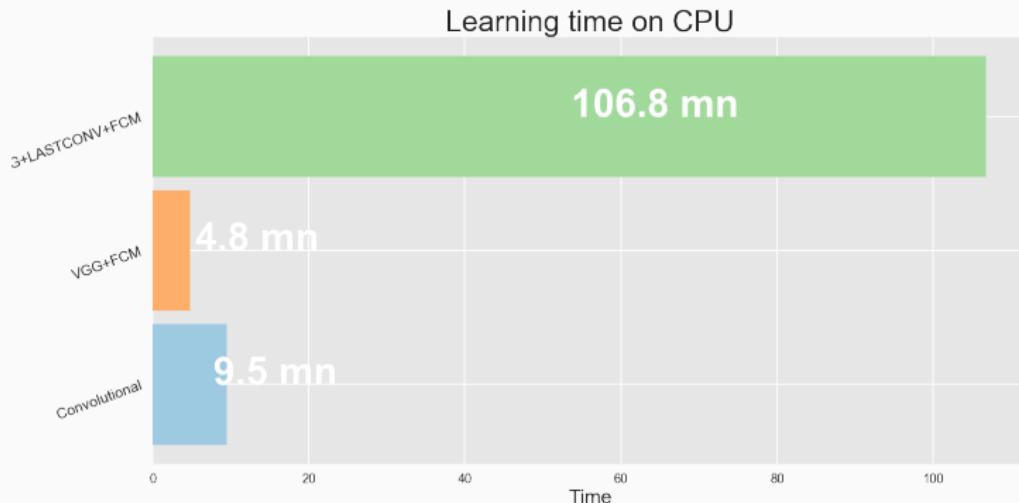
CATSVSDGOGS - VALIDATION SCORE - CPU/GPU

batch_size = 200, epochs = 15



CATSVSDGOGS - TEMPS D'APPRENTISSAGE - CPU

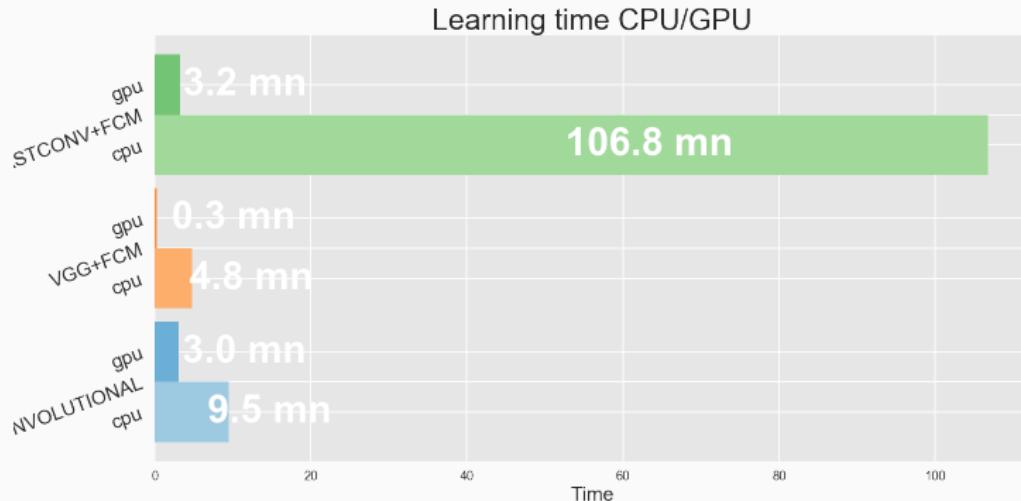
batch_size = 200, epochs = 15



L'apprentissage sur le modèle entier est très coûteux : pourquoi ?

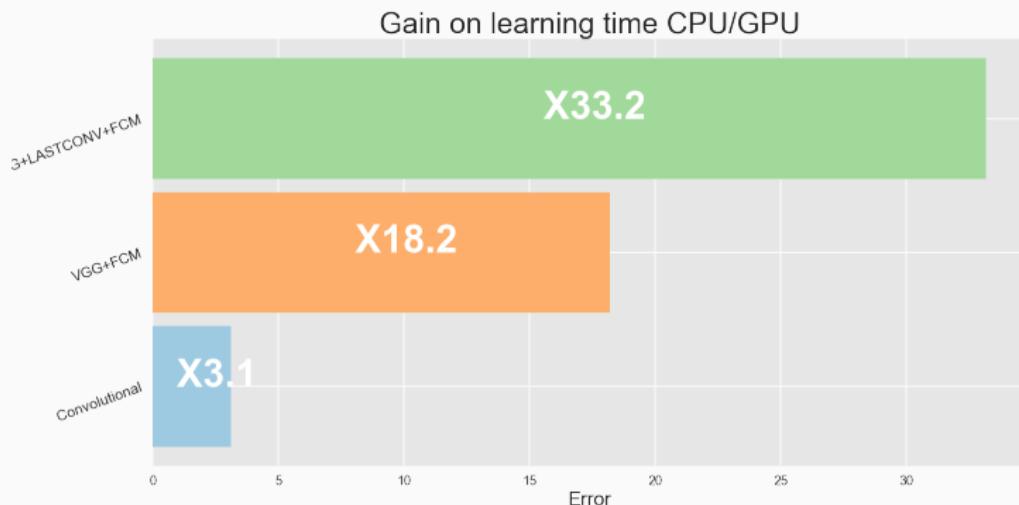
CATSVSDGOGS - TEMPS D'APPRENTISSAGE - CPU/GPU

batch_size = 200, epochs = 15



CATSVSDGOGS - TEMPS D'APPRENTISSAGE - CPU/GPU

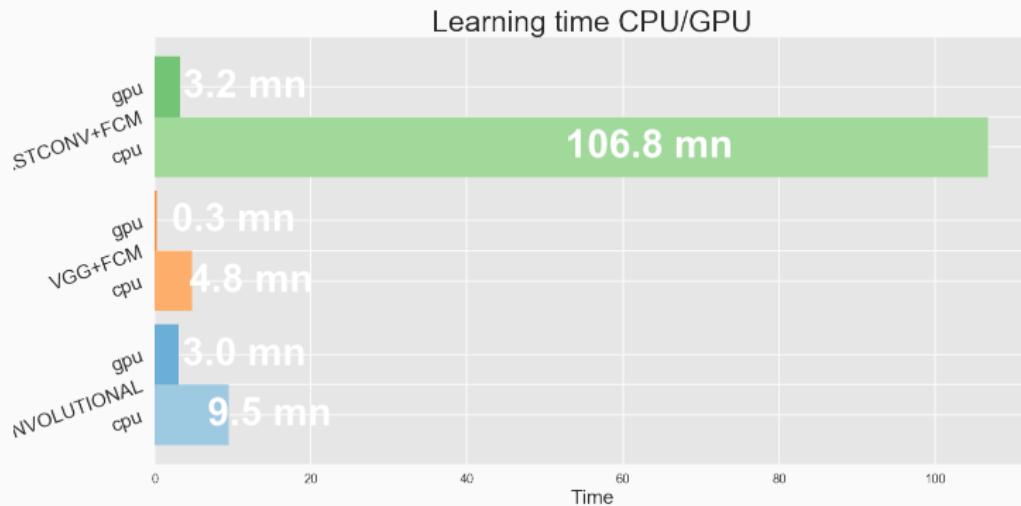
batch_size = 200, epochs = 15



Comment expliquer cette différence de gain ?

CATSVSDGOGS - TEMPS D'APPRENTISSAGE - CPU/GPU

batch_size = 200, epochs = 15

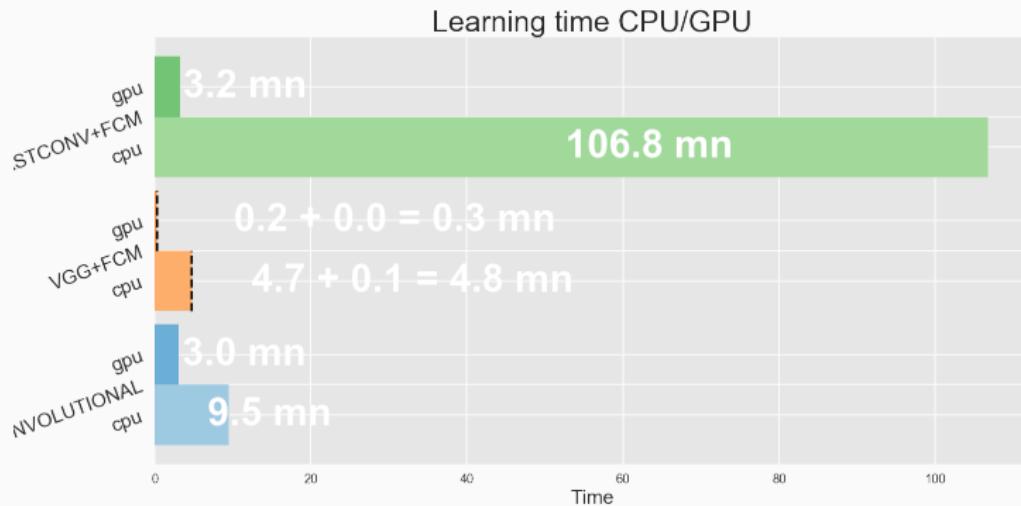


Dans la partie "Extraction de Features", c'est le calcul des features qui prend du temps!

Le modèle, très simple, ne prend pas beaucoup de temps sur le CPU. 53/53

CATSVSDGOGS - TEMPS D'APPRENTISSAGE - CPU/GPU

batch_size = 200, epochs = 15



Dans la partie "Extraction de Features", c'est le calcul des features qui prend du temps!

Le modèle, très simple, ne prend pas beaucoup de temps sur le CPU. 53/53

REFERENCES I

RÉFÉRENCES
