
Hands-on AI - Rapport - Defi 1

Geoffrey Picron, Michael Rombaix

Afin de construire un classifieur capable de distinguer des images en trois classes (départ de feu de forêt, feu de forêt et pas de feu), nous avons à notre disposition 4 ensembles de fichiers image (small, medium, big, huge).

Dans un premier temps, nous avons pris comme stratégie d'utiliser une architecture de réseau de neurones CNN pré-entraîné pour extraire les features à laquelle nous ajouterions une ou plusieurs Dense et Dropout.

Afin d'évaluer la performance de différentes architectures, nous avons utilisé la méthode de validation croisée. Néanmoins, les images mise à disposition étant issue de séquence video, les images successive sont très semblables et les résultats d'une évaluation par la méthode de validation croisée risquent d'être trop optimiste et les paramètres optimum inférés de conduire à un overfitting. Afin de limiter cette possibilité, nous avons introduit des variations (rotation, rognage, flip, contraste, luminosité) sur les images reçues.

Après quelques essais, nous avons développé un Helper qui automatise l'exploration des architecture et paramètres les plus performants. Ce Helper utilise un algorithme d'optimisation Bayésien (TPE, tree of Parzen estimators) pour optimiser le choix des paramètres.

Les premières analyses semblent montrer que des architectures pré-entraînées d'extractions de feature les plus "simples" fournissent les meilleurs résultats.

En effet, si l'on considère le problème proposé, intuitivement, il s'agit essentiellement de détecter une zone de couleurs "feu" dans une zone de couleurs "forêt" et/ou une zone de "fumée" issue d'une zone de couleurs "forêt". Et d'éviter de confondre avec le "soleil" et les "nuages".

Le modèle finalement choisi a été entraîné avec les images du dataset "huge" augmentées et testé avec les images du dataset "images_test" augmentées.



Le code source se trouve principalement dans notebook colab ci-joint. Néanmoins, nous avons créé un module

python contenant les routines que nous prévoyons de pouvoir réutiliser pour les prochains défis. Le module est publié sur GitHub à l'adresse https://github.com/gpicron/ia_tools

1. Preparation des données




Pour la partie exploration des architectures possibles, nous avons exploité le dataset "big". Pour l'entraînement, de l'architecture que nous avons finalement choisie, nous avons utilisé le dataset "huge"

Nous avons utilisé la technique de "Data augmentation" avec la librairie *Augmentor*[\[Augmentor\]](#) pour améliorer de nos dataset. La configuration utilisée applique un certain nombre de modifications aléatoirement sur les images d'origine:

- miroir horizontal
- rotation entre -20° et 20°
- zoom de 1.1 à 1.5 fois
- extraction d'une zone de 70% de la surface totale l'image
- modification du contraste
- modification de la luminosité

Et finalement, le plus grand "carré" au centre de l'image et extrait de l'image et redimensionné à la taille d'input de notre architecture.

Tableau 1. Exemples d'images d'entraînement augmentées

fire	start_fire	no_fire
		

2. Architecture paramétrique du classifieur

Nous avons conçu une fonction python qui nous permet de construire différentes architectures pour notre classifieur. Nos classifieurs sont composés:

- des couches de convolution d'un modèle pré-entraîné de la librairie *Keras*[\[Keras_apps\]](#)
- d'une couche GlobalAveragePooling2D
- de une ou plusieurs couches Dense (activation relu) intercalée optionnelle de couche Dropout
- d'une couche Dense à 3 neurones pour la sortie en activation softmax

Tableau 2. Paramètres de la fonction de construction d'un classifieur

Paramètre	Description
keras_app	le nom de modèle pré-entraîné pour les couches de convolution
layers	le nombre de couches Dense après les couches de convolution et avant la couche de sortie
units	le nombre de neurones par couche Dense est déterminé par la formule $units * units_alpha^n$, avec n le numéro de la couche commençant par 0
units_alpha	voir units
dropout	le taux de drop de la couche intercalaire Dropout déterminé par la formule $dropout * dropout_alpha^n$, avec n le numéro de la couche commençant par 0
dropout_alpha	voir dropout

3. Exploration des variations d'architecture du classifieur

Pour explorer les différentes variations sur base de notre architecture paramétrique dans colab, nous avons développé un Helper basé sur la librairie *hyperopt*[\[hyperopt\]](#). Cette librairie utilise un algorithme d'optimisation Bayésien (TPE, tree of Parzen estimators) pour diriger l'exploration des combinaisons de paramètres dans un espace de recherche donné en cherchant à minimiser une fonction objective.

Pour notre exploration, la fonction objective consiste à entraîner une architecture selon les paramètres choisis par *hyperopt* et à évaluer sa performance par rapport au dataset "test" et en validation croisée.

Pour la première exploration, nous avons ouvert l'espace de recherche à de nombreuses possibilités.



La mémoire du GPU dans colab est limité à 12GB, cela a posé des problèmes pour entraîner des architectures à base de Xception, ResNet50 et Inception.

3.1. Exploration large

Tableau 3. Espace des paramètres, exploration large

Paramètre	Espace des valeurs possibles
classifier_base	'MobileNet', 'VGG16', 'VGG19'
epochs	[2,4,8,16,32,64]
batch_size	de 4 à 24 par pas de 4
layers	de 1 à 4
units	de 128 à 1024 par pas de 128
units_alpha	entre 0.5 et 2.0
dropout	entre 0.25 et 1.0
dropout_alpha	entre 0.5 et 2.0

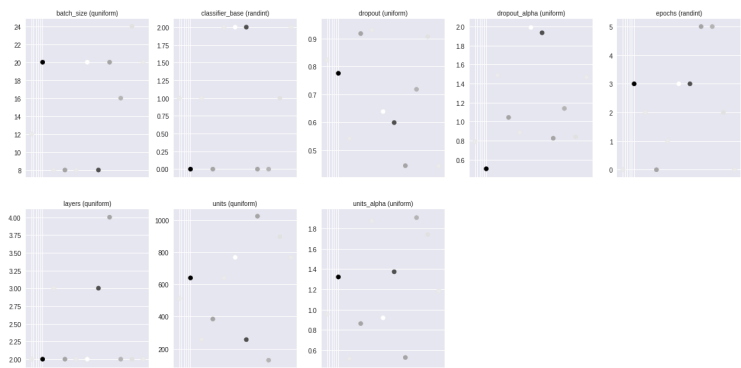


Figure 1. Analyse des paramètres, exploration large



Dans le graphique ci-dessus, en abscisse, nous avons l'itération de recherche, en ordonnée la valeur prise par le paramètre pour cette itération, et le point est d'autant plus foncé que l'accuracy de l'architecture testée pour le dataset "test"

Après quelques itérations, il nous a semblé que les architectures à base de MobileNet était la plus prometteuse. Par ailleurs, lors de précédentes exécutions de la recherche nous avons constaté que les entraînements de moins de 16 époques ne donnaient jamais de bons résultats. Il est probable que les architectures à base de VGG16 et VGG19, qui ont beaucoup plus de paramètres, donneraient de meilleurs résultats avec plus d'époques d'entrainement. Pour la suite, nous avons limité l'espace de recherche à MobileNet avec un minimum de 16 époques.

3.2. Exploration affinée

Tableau 4. Espace des paramètres, exploration affinée

Paramètre	Espace des valeurs possibles
classifier_base	'MobileNet'
epochs	[16,32,64]
batch_size	de 4 à 24 par pas de 4
layers	de 1 à 4
units	de 128 à 1024 par pas de 128
units_alpha	entre 0.5 et 2.0
dropout	entre 0.25 et 1.0
dropout_alpha	entre 0.5 et 2.0

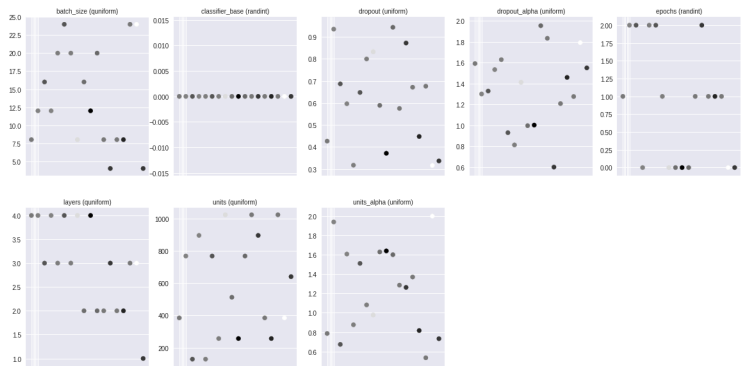


Figure 2. Analyse des paramètres, exploration affinée

Avec le nombre d'itérations effectuées, il n'est pas possible de dégager de tendance claire. Il faudrait continuer les itérations mais nous manquons de temps. Nous utiliserons le meilleur paramétrage trouvé pour la suite.

4. Architecture sélectionnée & résultats

Nous avons construit le modèle avec les meilleurs paramètres trouvés durant l'exploration.

Tableau 5. Paramètres choisis

Paramètre	Valeur
classifier_base	'MobileNet'
batch_size	12
layers	4
units	256
units_alpha	1.639789202190635
dropout	0.37109677567186583
dropout_alpha	1.0027477957252686

Description du modèle.

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, 224, 224, 3)	0
conv1_pad (ZeroPadding2D)	(None, 226, 226, 3)	0

conv1 (Conv2D)	(None, 112, 112, 32)	864
conv1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv1_relu (Activation)	(None, 112, 112, 32)	0
conv_pad_1 (ZeroPadding2D)	(None, 114, 114, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_dw_1_bn (BatchNormaliza	(None, 112, 112, 32)	128
conv_dw_1_relu (Activation)	(None, 112, 112, 32)	0
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2048
conv_pw_1_bn (BatchNormaliza	(None, 112, 112, 64)	256
conv_pw_1_relu (Activation)	(None, 112, 112, 64)	0
conv_pad_2 (ZeroPadding2D)	(None, 114, 114, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	576
conv_dw_2_bn (BatchNormaliza	(None, 56, 56, 64)	256
conv_dw_2_relu (Activation)	(None, 56, 56, 64)	0
conv_pw_2 (Conv2D)	(None, 56, 56, 128)	8192
conv_pw_2_bn (BatchNormaliza	(None, 56, 56, 128)	512
conv_pw_2_relu (Activation)	(None, 56, 56, 128)	0
conv_pad_3 (ZeroPadding2D)	(None, 58, 58, 128)	0
conv_dw_3 (DepthwiseConv2D)	(None, 56, 56, 128)	1152
conv_dw_3_bn (BatchNormaliza	(None, 56, 56, 128)	512
conv_dw_3_relu (Activation)	(None, 56, 56, 128)	0
conv_pw_3 (Conv2D)	(None, 56, 56, 128)	16384
conv_pw_3_bn (BatchNormaliza	(None, 56, 56, 128)	512

conv_pw_3_relu (Activation)	(None, 56, 56, 128)	0
conv_pad_4 (ZeroPadding2D)	(None, 58, 58, 128)	0
conv_dw_4 (DepthwiseConv2D)	(None, 28, 28, 128)	1152
conv_dw_4_bn (BatchNormaliza	(None, 28, 28, 128)	512
conv_dw_4_relu (Activation)	(None, 28, 28, 128)	0
conv_pw_4 (Conv2D)	(None, 28, 28, 256)	32768
conv_pw_4_bn (BatchNormaliza	(None, 28, 28, 256)	1024
conv_pw_4_relu (Activation)	(None, 28, 28, 256)	0
conv_pad_5 (ZeroPadding2D)	(None, 30, 30, 256)	0
conv_dw_5 (DepthwiseConv2D)	(None, 28, 28, 256)	2304
conv_dw_5_bn (BatchNormaliza	(None, 28, 28, 256)	1024
conv_dw_5_relu (Activation)	(None, 28, 28, 256)	0
conv_pw_5 (Conv2D)	(None, 28, 28, 256)	65536
conv_pw_5_bn (BatchNormaliza	(None, 28, 28, 256)	1024
conv_pw_5_relu (Activation)	(None, 28, 28, 256)	0
conv_pad_6 (ZeroPadding2D)	(None, 30, 30, 256)	0
conv_dw_6 (DepthwiseConv2D)	(None, 14, 14, 256)	2304
conv_dw_6_bn (BatchNormaliza	(None, 14, 14, 256)	1024
conv_dw_6_relu (Activation)	(None, 14, 14, 256)	0
conv_pw_6 (Conv2D)	(None, 14, 14, 512)	131072
conv_pw_6_bn (BatchNormaliza	(None, 14, 14, 512)	2048
conv_pw_6_relu (Activation)	(None, 14, 14, 512)	0
conv_pad_7 (ZeroPadding2D)	(None, 16, 16, 512)	0

conv_dw_7 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_dw_7_bn (BatchNormaliza	(None, 14, 14, 512)	2048
conv_dw_7_relu (Activation)	(None, 14, 14, 512)	0
conv_pw_7 (Conv2D)	(None, 14, 14, 512)	262144
conv_pw_7_bn (BatchNormaliza	(None, 14, 14, 512)	2048
conv_pw_7_relu (Activation)	(None, 14, 14, 512)	0
conv_pad_8 (ZeroPadding2D)	(None, 16, 16, 512)	0
conv_dw_8 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_dw_8_bn (BatchNormaliza	(None, 14, 14, 512)	2048
conv_dw_8_relu (Activation)	(None, 14, 14, 512)	0
conv_pw_8 (Conv2D)	(None, 14, 14, 512)	262144
conv_pw_8_bn (BatchNormaliza	(None, 14, 14, 512)	2048
conv_pw_8_relu (Activation)	(None, 14, 14, 512)	0
conv_pad_9 (ZeroPadding2D)	(None, 16, 16, 512)	0
conv_dw_9 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_dw_9_bn (BatchNormaliza	(None, 14, 14, 512)	2048
conv_dw_9_relu (Activation)	(None, 14, 14, 512)	0
conv_pw_9 (Conv2D)	(None, 14, 14, 512)	262144
conv_pw_9_bn (BatchNormaliza	(None, 14, 14, 512)	2048
conv_pw_9_relu (Activation)	(None, 14, 14, 512)	0
conv_pad_10 (ZeroPadding2D)	(None, 16, 16, 512)	0
conv_dw_10 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_dw_10_bn (BatchNormaliz	(None, 14, 14, 512)	2048

conv_dw_10_relu (Activation) (None, 14, 14, 512)	0
conv_pw_10 (Conv2D) (None, 14, 14, 512)	262144
conv_pw_10_bn (BatchNormaliz (None, 14, 14, 512)	2048
conv_pw_10_relu (Activation) (None, 14, 14, 512)	0
conv_pad_11 (ZeroPadding2D) (None, 16, 16, 512)	0
conv_dw_11 (DepthwiseConv2D) (None, 14, 14, 512)	4608
conv_dw_11_bn (BatchNormaliz (None, 14, 14, 512)	2048
conv_dw_11_relu (Activation) (None, 14, 14, 512)	0
conv_pw_11 (Conv2D) (None, 14, 14, 512)	262144
conv_pw_11_bn (BatchNormaliz (None, 14, 14, 512)	2048
conv_pw_11_relu (Activation) (None, 14, 14, 512)	0
conv_pad_12 (ZeroPadding2D) (None, 16, 16, 512)	0
conv_dw_12 (DepthwiseConv2D) (None, 7, 7, 512)	4608
conv_dw_12_bn (BatchNormaliz (None, 7, 7, 512)	2048
conv_dw_12_relu (Activation) (None, 7, 7, 512)	0
conv_pw_12 (Conv2D) (None, 7, 7, 1024)	524288
conv_pw_12_bn (BatchNormaliz (None, 7, 7, 1024)	4096
conv_pw_12_relu (Activation) (None, 7, 7, 1024)	0
conv_pad_13 (ZeroPadding2D) (None, 9, 9, 1024)	0
conv_dw_13 (DepthwiseConv2D) (None, 7, 7, 1024)	9216
conv_dw_13_bn (BatchNormaliz (None, 7, 7, 1024)	4096
conv_dw_13_relu (Activation) (None, 7, 7, 1024)	0
conv_pw_13 (Conv2D) (None, 7, 7, 1024)	1048576

conv_pw_13_bn (BatchNormaliz	(None, 7, 7, 1024)	4096
conv_pw_13_relu (Activation)	(None, 7, 7, 1024)	0
global_average_pooling2d_3 ((None, 1024)	0
dropout_11 (Dropout)	(None, 1024)	0
dense_11 (Dense)	(None, 256)	262400
dropout_12 (Dropout)	(None, 256)	0
dense_12 (Dense)	(None, 420)	107940
dropout_13 (Dropout)	(None, 420)	0
dense_13 (Dense)	(None, 689)	290069
dropout_14 (Dropout)	(None, 689)	0
dense_14 (Dense)	(None, 1130)	779700
dropout_15 (Dropout)	(None, 1130)	0
dense_15 (Dense)	(None, 3)	3393
=====		
Total params: 4,672,366		
Trainable params: 4,650,478		
Non-trainable params: 21,888		

Nous avons ensuite entraîné ce modèle avec les images augmentées du dataset 'huge' et analysé sa capacité à classer les images tests

Nous avons entraîné le modèle durant 16 époques, et analyser les résultat du modèle final et du modèle qui a donné la meilleure accuracy pendant l'entraînement.

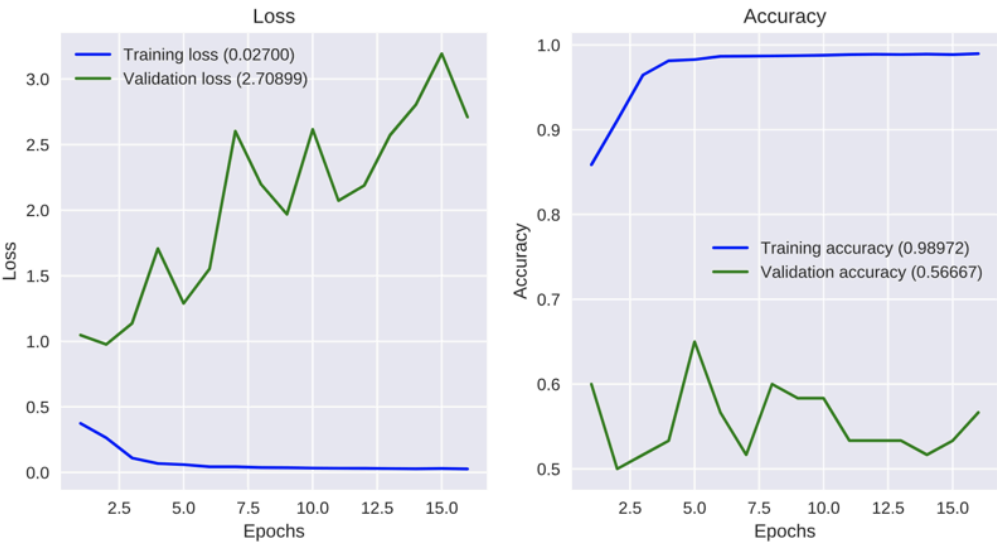


Figure 3. Historique de l'entraînement

Le résultat est une accuracy sur les images tests augmentées d'environ 51%, mais la matrice de confusion montre que le modèle classe les images de test 'no_fire' comme 'fire' dans 99% des cas, ce qui n'est pas acceptable.

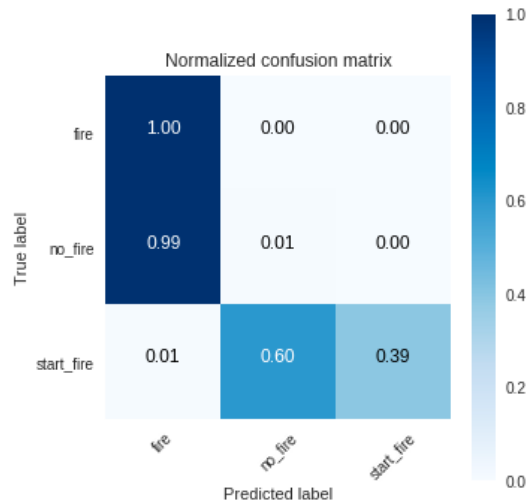


Figure 4. Matrice de confusion (model final entraîné et test dataset)

Si l'on reprend les poids du modèle à la 5 époques d'entraînement, ce qui semble être avant que le modèle n'overfit. L'accuracy est de 58% mais la matrice

de confusion montre le classsifieur fait la même erreur. Dans une utilisation pratique réelle, cela ne serait pas acceptable.

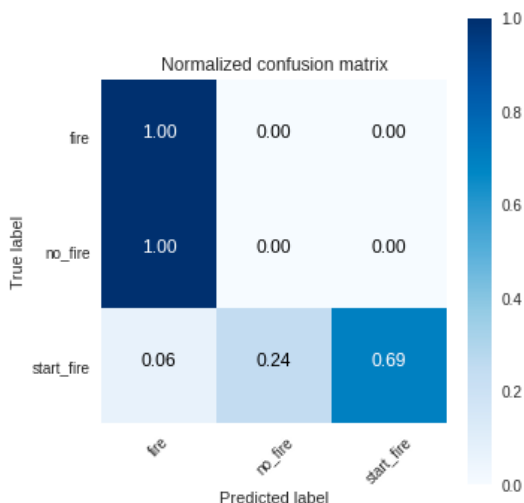


Figure 5. Matrice de confusion (model entraîné intermédiaire aillant le meilleur score et test dataset)

5. Conclusion

Pour ce défi, nous avons exploré différentes possibilités de réduire la tendance à l'overfit lorsque les données images d'entraînement est limité. Sur base de ces données augmentées, nous avons automatisé l'exploration de diverses architectures.

Nous nous sommes focalisés sur des variations autour d'une architecture basée sur un réseau convolutionnel pré-entraîné de la librairie Keras. Notre "explorateur" automatique n'a pas tourné suffisamment longtemps pour nous donner une idée très précise de l'influence des paramètres sur l'architecture paramétrique proposée. Néanmoins, nous pensons que la stratégie est bonne et qu'il nous sera utile pour les prochains défis.

A l'analyse des résultats, la tendance à l'overfit reste flagrante malgré l'augmentation et si notre exploration s'était limitée à la validation croisée sur le dataset d'entraînement "big" sans mesurer l'accuracy du modèle sur le dataset "test" il aurait été impossible d'identifier une architecture valable. L'importance de qualité et de la quantité des données d'entraînement est pré-dominante.

6. Références

Modules python

[Augmentor] <https://github.com/mdbloice/Augmentor>.

[Keras_apps] <https://keras.io/applications/>

[hyperopt] <https://github.com/hyperopt/hyperopt>