Images

```
library(keras)
library(jpeg)
```

Les données

Ce problème est un problème de classification. Chaque image contient un objet qui est sa classe. Nous avons trois classes: voitures, chats, et fleurs. Il y a au total 1596 images.

Chargement des images

```
filenames_car <- array(list.files(path='data/images_train/car', full.names = TRUE, pattern='*.jpg'))
filenames_cat <- array(list.files(path='data/images_train/cat', full.names = TRUE, pattern='*.jpg'))
filenames_flower <- array(list.files(path='data/images_train/flower', full.names = TRUE, pattern='*.jpg
```

Nous obtenons la répartition suivante selon les classes. Il y a à peu près autant d'individus dans chaque classe.

	Nombre d'images
Car	485
Cat	590
Flower	521

Création des étiquettes

Nous créons un vecteur contenant les étiquettes des données: 0 pour les voitures, 1 pour les chats, 2 pour les fleurs.

```
y <- c(rep(0, length(filenames_car)), rep(1, length(filenames_cat)), rep(2, length(filenames_flower)))
length(y)
```

```
## [1] 1596
```

Séparation des données d'entrainement, de validation et de test

Nous séparons les données en trois ensembles: entrainement valdiation et test. L'ensemble de validation va nous servir à faire un arrêt précoce: lorsque le réseau de neurones n'améliorera plus la précision sur l'ensemble de validation l'entrainement s'arrêtera. Cela permet de limiter le surentrainement. L'ensemble de test serivara à caluler une estimation de l'erreur de classification non biaisée.

```
set.seed(123)
filenames <- c(filenames_car, filenames_cat, filenames_flower)
size_training <- floor(0.6*length(filenames))
size_validation <- floor(0.2*length(filenames))
size_test <- length(filenames) - size_training - size_validation
sample_train <- sample(1:length(filenames), size = size_training)
filenames_train <- filenames[sample_train]
```

```
y_train <- y[sample_train]
sample_val <- sample((1:length(filenames))[-sample_train], size = size_validation)
filenames_val <- filenames[sample_val]
y_val <- y[sample_val]
c(length(filenames_train), length(y_train), length(filenames_val), length(y_val))
```

```
## [1] 957 957 319 319
```

Transformation des données

Nous utilisons les pixels comme variables d'entrée et donc nous n'avons pas besoin d'extraire des caractéristiques de l'image. Cependant toutes les images n'ont pas la même taille nous réalisons donc une interpolation bilinéaire pour que toutes les images aient la même taille. Nous avons choisi une taille de 100x100.

```
read_images <- function(list_files, dim_images){
  images <- array(0, dim=c(length(list_files), dim_images[1], dim_images[2], 3))
  for (i in 1:length(list_files)){
    images[i,,,] <- image_to_array(image_load(path=list_files[i], target_size=dim_images, interpolation
  }
  return(images)
}
```

```
dim_images <- c(100, 100)
images_train <- read_images(filenames_train, dim_images)
images_val <- read_images(filenames_val, dim_images)
dim(images_train)
```

```
## [1] 957 100 100   3
```

```
dim(images_val)
```

```
## [1] 319 100 100   3
```

Modèle

Notre modèle est un réseau de neurones convolutif. Ce type de modèle est adapté aux images puisque de part sa construction il prend en compte l'invariance aux transaltions des images.

Les images en entrées ont les dimensions: 100x100x3 (Hauteur x Largeur x Nombre de couleurs). Ensuite nous appliquons des convolutions jusqu'à obtenir un vecteur de taille 2x2x64 que nous "applatissons" en un vecteur de taille 256. Chaque convolution a un filtre de taille 3x3 (pour chaque carte de caractéristique en entrée et en sortie). Nous choisissons de réduire la hauteur et la largeur des cartes de caractéristiques uniquement avec le pas des convolution (donc nous n'utilisons pas de pooling et mettons systématiquement du padding). Chaque convolution est suivi d'une activation 'relu' pour mettre de la non-linéarité. Le nombre de carte de caractéristiques est augmenté progressivement de 3 cartes (pour les 3 couleurs) jusqu'à 64 de façpn à extraire de plus en plus d'informations au fur et à mesure.

Ensuite nous passons le vecteur de taille 256 dans 2 couches entièrement connectées. La dernière couche entièrement connectée est suivie d'un softmax. Le modèle est entrainé par minimisation de l'entropie croisée ce qui revient à la maximisation du maximum de vraisemblance de notre modèle.

Afin d'éviter le surentrainement nous utilisonsdu dropout avec une probabilité de 0.5 (chaque neurone a une chance sur deux de renvoyer zéro). De plus nous effectuons un arrêt précoce lorsque le CNN n'a pas progressé lors de 10 epochs.

Notre modèle contient 185 000 paramètres et atteint 92% de précision sur l'ensemble de validation.

```
model <- keras_model_sequential()

model %>%
   layer_conv_2d(
    filter = 16, kernel_size = c(3,3), padding = "same", strides = c(2, 2), input_shape = c(dim_images[
  ) %>%
  layer_conv_2d(filter = 16, kernel_size = c(3,3), padding = "same", activation = "relu") %>%

  layer_conv_2d(filter = 16, kernel_size = c(3,3), padding = "same", strides = c(2, 2), activation = "re
  layer_conv_2d(filter = 16, kernel_size = c(3,3), padding = "same", activation = "relu") %>%
  layer_dropout(0.5) %>%

  layer_conv_2d(filter = 32, kernel_size = c(3,3), padding = "same", strides = c(2, 2), activation = "re
  layer_conv_2d(filter = 32, kernel_size = c(3,3), padding = "same", activation = "relu") %>%

  layer_conv_2d(filter = 32, kernel_size = c(3,3), padding = "same", strides = c(2, 2), activation = "re
  layer_conv_2d(filter = 32, kernel_size = c(3,3), padding = "same", activation = "relu") %>%
  layer_dropout(0.5) %>%

  layer_conv_2d(filter = 64, kernel_size = c(3,3), padding = "same", strides = c(2, 2), activation = "re
  layer_conv_2d(filter = 64, kernel_size = c(3,3), padding = "same", activation = "relu") %>%

  layer_conv_2d(filter = 64, kernel_size = c(3,3), padding = "same", strides = c(2, 2), activation = "re
  layer_conv_2d(filter = 64, kernel_size = c(3,3), padding = "same", activation = "relu") %>%

  layer_flatten() %>%
  layer_dropout(0.5) %>%

  layer_dense(64, activation="relu") %>%
  layer_dropout(0.5) %>%

  layer_dense(3, activation="softmax")

opt <- optimizer_adam(lr = 0.001)

model %>% compile(
  loss = "sparse_categorical_crossentropy",
  optimizer = opt,
  metrics = "accuracy"
)
summary(model)
```

```
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## conv2d_1 (Conv2D)            (None, 50, 50, 16)        448
## _________________________________________________________________________
## conv2d_2 (Conv2D)            (None, 50, 50, 16)        2320
## _________________________________________________________________________
## conv2d_3 (Conv2D)            (None, 25, 25, 16)        2320
## _________________________________________________________________________
## conv2d_4 (Conv2D)            (None, 25, 25, 16)        2320
## _________________________________________________________________________
```

```
## dropout_1 (Dropout)              (None, 25, 25, 16)            0
## ________________________________________________________________________
## conv2d_5 (Conv2D)                (None, 13, 13, 32)            4640
## ________________________________________________________________________
## conv2d_6 (Conv2D)                (None, 13, 13, 32)            9248
## ________________________________________________________________________
## conv2d_7 (Conv2D)                (None, 7, 7, 32)              9248
## ________________________________________________________________________
## conv2d_8 (Conv2D)                (None, 7, 7, 32)              9248
## ________________________________________________________________________
## dropout_2 (Dropout)              (None, 7, 7, 32)              0
## ________________________________________________________________________
## conv2d_9 (Conv2D)                (None, 4, 4, 64)              18496
## ________________________________________________________________________
## conv2d_10 (Conv2D)               (None, 4, 4, 64)              36928
## ________________________________________________________________________
## conv2d_11 (Conv2D)               (None, 2, 2, 64)              36928
## ________________________________________________________________________
## conv2d_12 (Conv2D)               (None, 2, 2, 64)              36928
## ________________________________________________________________________
## flatten_1 (Flatten)              (None, 256)                   0
## ________________________________________________________________________
## dropout_3 (Dropout)              (None, 256)                   0
## ________________________________________________________________________
## dense_1 (Dense)                  (None, 64)                    16448
## ________________________________________________________________________
## dropout_4 (Dropout)              (None, 64)                    0
## ________________________________________________________________________
## dense_2 (Dense)                  (None, 3)                     195
## ========================================================================
## Total params: 185,715
## Trainable params: 185,715
## Non-trainable params: 0
## ________________________________________________________________________
```

Entrainement

Nous entrainons notre modèle avec des lots de 30.

```
early_stopping <- callback_early_stopping(monitor = 'val_acc', patience = 10)
history <- model %>% fit(
  images_train, y_train,
  batch_size = 30,
  epochs = 60,
  validation_data = list(images_val, y_val),
  shuffle = TRUE,
  callbacks = c(early_stopping)
)
```

```
plot(history)
```