

# *Classification d'images*

## **Une comparaison entre R et Python**

*Conférence Machine Learning*



Clément Melina, Le Moan Delalande Riwal, Mathieu Anna

21-11-25

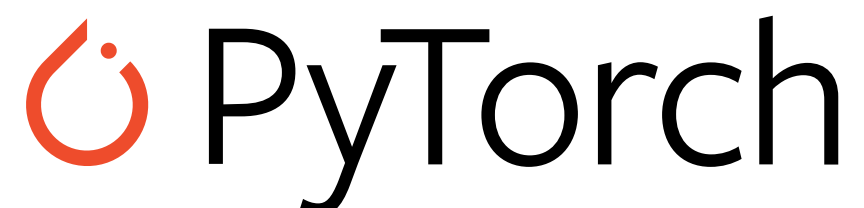


# *Sommaire*

- |             |  |
|-------------|--|
| <b>. 01</b> | Introduction                                 |
| <b>. 02</b> | Qu'est-ce qu'une API ?                       |
| <b>. 03</b> | Comment fonctionne Keras sur R et sur Python |
| <b>. 04</b> | Méthodologie                                 |
| <b>. 05</b> | Comparaison des méthodes                     |
| <b>. 06</b> | Conclusion                                   |

# *Introduction* *Plusieurs approches pour le deep learning*

Le deep learning peut être abordé à travers plusieurs outils et langages

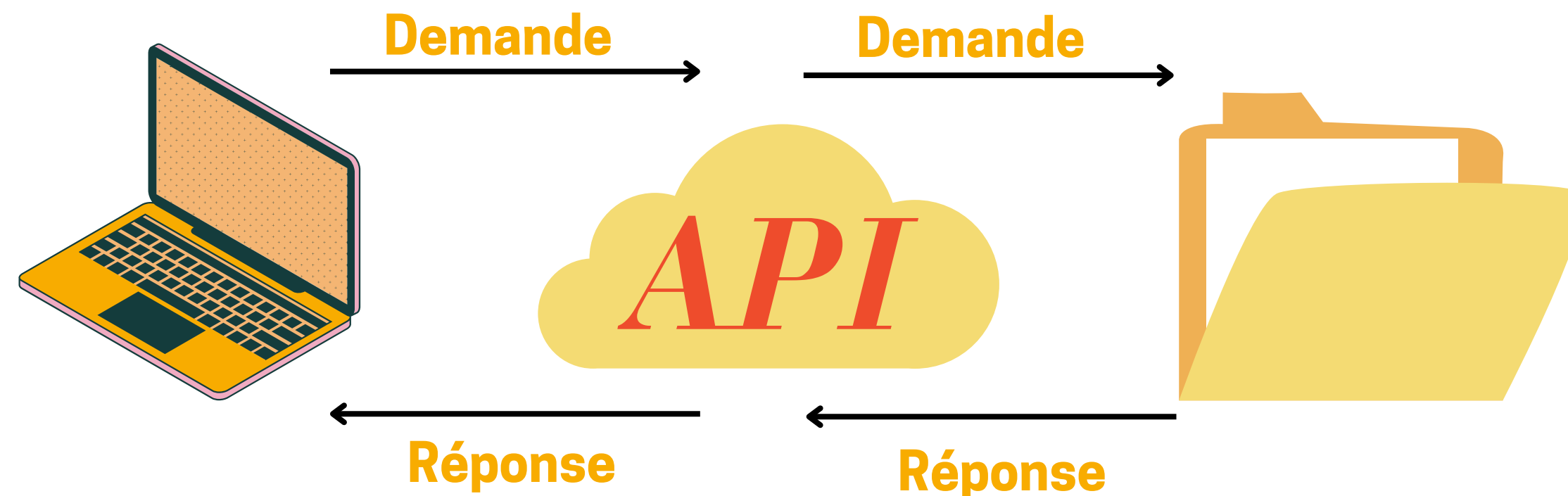


**Comment un même projet de deep learning peut-il être abordé différemment selon l'outil utilisé ? Quels sont les avantages et limites de R Keras, Python Keras et PyTorch pour construire, entraîner et exploiter un modèle ?**

# Qu'est ce qu'une API

Application **P**rogramming **I**nterface

- Connecte 2 logiciels/services ensemble
- Permet échange de données et de fonctionnalités



Exemples d'API :

- site web utilise **l'API Google Calendar** pour créer automatiquement un événement dans le calendrier du client.
- **OpenWeatherMap API** : Fournit des données météorologiques en temps réel (affichage de la météo dans une app ou sur un site)

# *Keras*



## Objectifs :

Orienté humain

Vise à simplifier la façon de coder

## Framework :

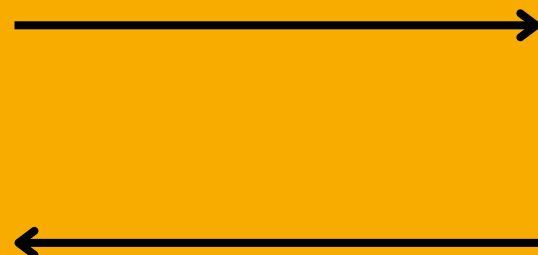
Permet d'utiliser plusieurs

Framework de la même façon :  
JAX, TensorFlow, et PyTorch.

## Utilisation :

2 modes d'utilisation principaux :

- Séquentiel (Empilement de couches, CNN)
- Fonctionnel (Modèles plus complexes, non linéaires)



 PyTorch

 TensorFlow



# *Sur R*

Package R permettant de faire le lien entre python et R

```
reticulate::install_python(version = "3.9")
```

Permet d'installer une version de python directement depuis R

Reticulate permet de traduire les objets R en objets python



## Keras

```
library(keras)
```

```
library(keras3)
```

interface R vers l'API Keras Python

Facilite la création de CNN dans une environnement R

Similaire à la structure Keras python

2 version du package coexistent

## TensorFlow

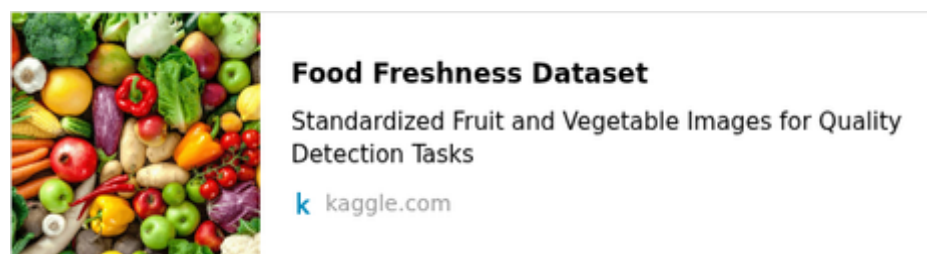
```
library(TensoFlow)
```

Base pour l'utilisation du package keras

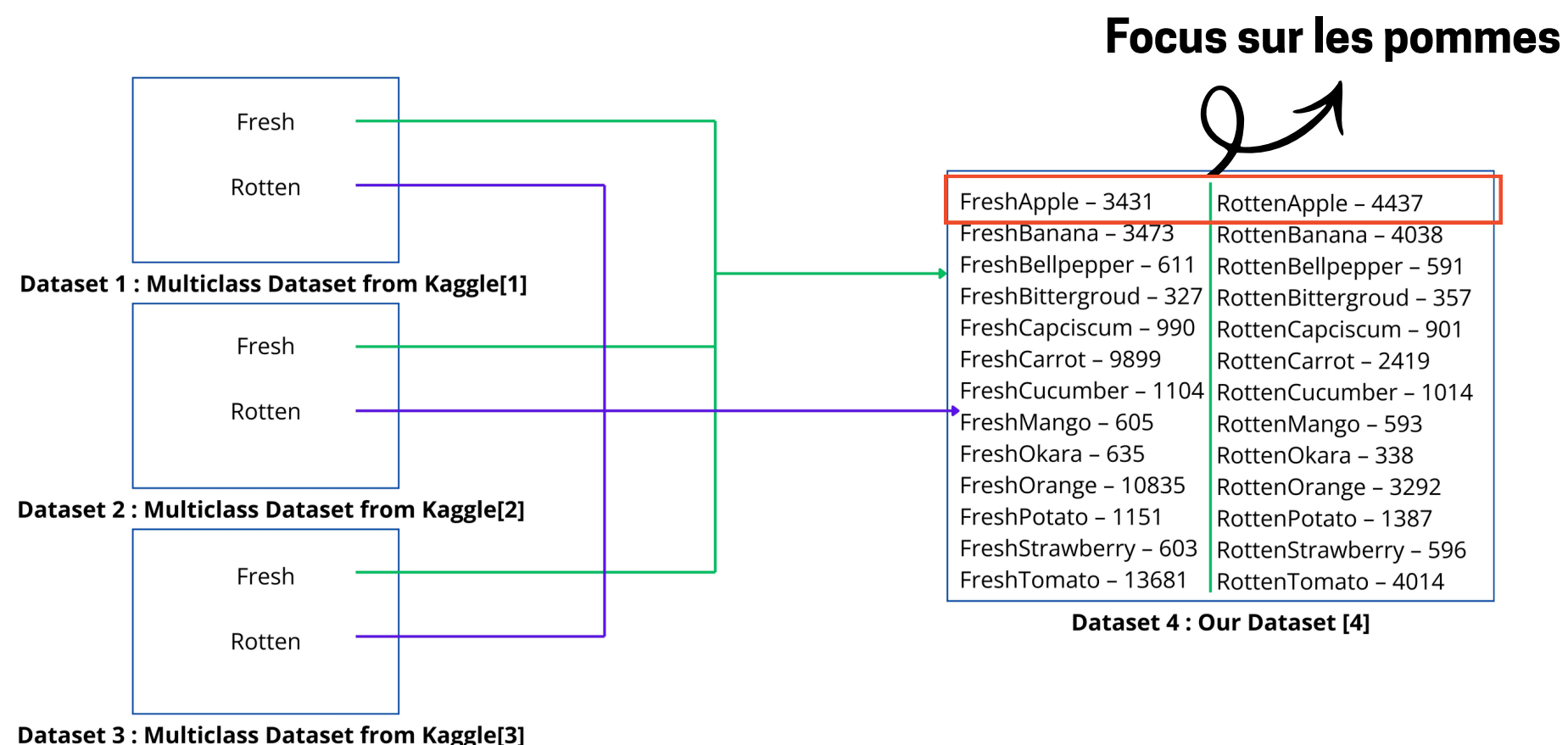
Permet la manipulation des Tenseurs

# Nos données : classification d'image

Source :



Le jeu de données : issu de la compilation de trois jeux de données de classification fruits frais/pourris



## Composition

- > 70 000 images
  - Fruits et légumes frais
  - Fruits et légumes pourris
- 13 catégories de fruits et légumes
- taille : 6.4 GB

Fresh apple



Rotten apple



## Obtention des données

- Preprocessing et data augmentation
- Standardisation en 128x128 pixels et transformations
  - Conversion en niveaux de gris
  - détection des contours
  - réglage de la luminosité



# Méthodologie



## 0 Préparation des fichiers

```
classes ← c("Mures", "Pourris")
train_dir ← file.path(base_dir, "train")
test_dir ← file.path(base_dir, "test")

dir.create(train_dir, recursive = TRUE, showWarnings = FALSE)
dir.create(test_dir, recursive = TRUE, showWarnings = FALSE)

for (class in classes) {
  cat("Traitement de la classe:", class, "\n")

  source_path ← file.path(base_dir, class)
  train_class_path ← file.path(train_dir, class)
  test_class_path ← file.path(test_dir, class)

  dir.create(train_class_path, showWarnings = FALSE)
  dir.create(test_class_path, showWarnings = FALSE)

  all_files ← list.files(source_path, full.names = TRUE)
  num_files ← length(all_files)

  if (num_files == 0) {
    cat(" !!! Aucun fichier trouvé dans", source_path, "!!!\n")
    next
  }

  train_indices ← sample(seq_len(num_files), size = floor(split_ratio * num_files))
  train_files ← all_files[train_indices]
  test_files ← all_files[-train_indices]

  file.copy(from = train_files, to = file.path(train_class_path, basename(train_files)))
  file.copy(from = test_files, to = file.path(test_class_path, basename(test_files)))

  cat(" →", length(train_files), "fichiers déplacés vers 'train/', class, "'\n", sep="")
  cat(" →", length(test_files), "fichiers déplacés vers 'test/', class, "'\n", sep="")
}
```

## 1 Préparation des données

```
image_size ← c(128, 128)

train_datagen ← image_data_generator(
  rescale = 1/255,
  shear_range = 0.2,
  zoom_range = 0.2,
  horizontal_flip = TRUE
)

test_datagen ← image_data_generator(
  rescale = 1/255
)

train_generator ← flow_images_from_directory(
  train_dir,
  generator = train_datagen,
  target_size = image_size,
  batch_size = 100,
  class_mode = 'binary'
)

test_generator ← flow_images_from_directory(
  test_dir,
  generator = test_datagen,
  target_size = image_size,
  batch_size = 100,
  class_mode = 'binary'
)
```

## 2 Construction du modèle

```
model ← keras_model_sequential() %>%
  layer_conv_2d(filters = 16, kernel_size = c(3, 3),
    activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3),
    activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')
```



# Méthodologie



Keras



TensorFlow

## ③ Compilation et entraînement

```
model %>% compile(
  optimizer = optimizer_adam(),
  loss = 'binary_crossentropy',
  metrics = c('accuracy')
)
```

```
EPOCHS ← 10
```

```
options(keras.view_metrics = FALSE)
```

```
history ← model %>% fit(
  train_generator,
  steps_per_epoch = ceiling(train_generator$n / train_generator$batch_size),
  epochs = EPOCHS,
  validation_data = test_generator,
  validation_steps = ceiling(test_generator$n / test_generator$batch_size),
  view_metrics = FALSE |
)
```

## ④ Validation

```
evaluation ← model %>% evaluate(
  test_generator,
  steps = ceiling(test_generator$n / test_generator$batch_size)
)
cat('Test loss:', evaluation$loss, '\n')
cat('Test accuracy:', evaluation$accuracy, '\n')
```

# Méthodologie



Keras



TensorFlow

①

Préparation des données

```
train_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(128, 128),
    batch_size=32,
    label_mode="binary"
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(128, 128),
    batch_size=32,
    label_mode="binary"
)

train_ds = train_ds.prefetch(tf.data.AUTOTUNE)
val_ds = val_ds.prefetch(tf.data.AUTOTUNE)
```

②

Construction du modèle

```
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomZoom(0.2),
    layers.RandomRotation(0.1),
])

model = Sequential([
    keras.Input(shape=(128, 128, 3)),
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

③

Compilation et entraînement

```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10
)
```

# Méthodologie

## ① Préparation des données

```
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor()
])

full_dataset = datasets.ImageFolder(root=path, transform=transform)

val_size = int(0.2 * len(full_dataset))
train_size = len(full_dataset) - val_size
train_dataset, val_dataset = random_split(
    full_dataset,
    [train_size, val_size],
    generator=torch.Generator().manual_seed(123)
)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
```



## ② Construction du modèle

```
class AppleCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(32 * 30 * 30, 64)
        self.fc2 = nn.Linear(64, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        return x

model = AppleCNN().to(device)

criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

train_acc_list, val_acc_list = [], []
train_loss_list, val_loss_list = [], []
```

# Méthodologie

## ③ Compilation et entraînement

```
for epoch in range(EPOCHS):
    # ----- TRAIN -----
    model.train()
    running_loss = 0.0
    running_correct = 0
    total_train = 0

    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.float().view(-1, 1).to(device) #

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)
        preds = (outputs >= 0.5).float()
        running_correct += (preds == labels).sum().item()
        total_train += labels.size(0)

    epoch_loss = running_loss / total_train
    epoch_acc = running_correct / total_train
```



## ④ Validation

```
# ----- VALIDATION -----
model.eval()
val_loss = 0.0
val_correct = 0
total_val = 0

with torch.no_grad():
    for images, labels in val_loader:
        images = images.to(device)
        labels = labels.float().view(-1, 1).to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)

        val_loss += loss.item() * images.size(0)
        preds = (outputs >= 0.5).float()
        val_correct += (preds == labels).sum().item()
        total_val += labels.size(0)

val_loss /= total_val
val_acc = val_correct / total_val

train_loss_list.append(epoch_loss)
train_acc_list.append(epoch_acc)
val_loss_list.append(val_loss)
val_acc_list.append(val_acc)
```



# Gestion des données



Données locales uniquement (pas de chargement direct depuis le web)

Split manuel en dossiers train/ et test/  
=> double stockage des données

Durée dépendante de la quantité de données :

- < 1 min | 1 Go
- ≈ 7 min | 2 Go
- ≈ 15 min | 3.5 Go .....

Chargement avec `image_data_generator()`

Pipeline simple mais moins flexible



Charge facilement :



Compatible datasets distants :



Chargement très automatisé :

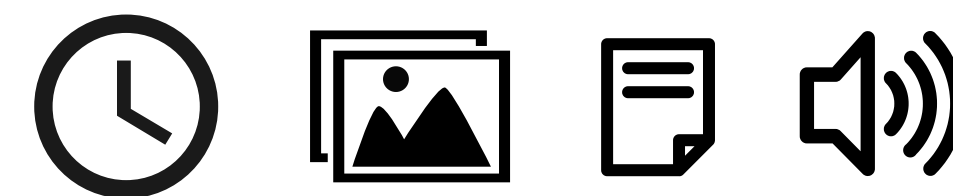
```
tf.keras.utils.image_dataset_from_directory()
```

Split intégré :

```
validation_split=0.2,
```



Charge facilement :



Compatible datasets distants :



Système de Dataset et de DataLoader :

```
datasets.ImageFolder()  
DataLoader()
```

Split facilité :

```
torch.utils.data.random_split()
```

Transformations de données avancées

# Construction du modèle



La construction du modèle via `keras_model_sequential()` avant l'ajout des couches avec des fonctions `layer_ ... ()`

```
model <- keras_model_sequential()
model %>%
  ## layers....
```

Ensuite compilation du modèle :

```
model %>% compile(
  optimizer = 'rmsprop',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)
```

Et formation du modèle :

```
model %>% fit(x_train, y_train, epochs = 20, batch_size = 128)
```

**Facile, rapide et intuitif**

Différences uniquement liées  
au langage de programmation



La construction du modèle via `Sequential()` avec une facilité d'ajouter des couches

```
model = Sequential([
  ## layers....
])
```

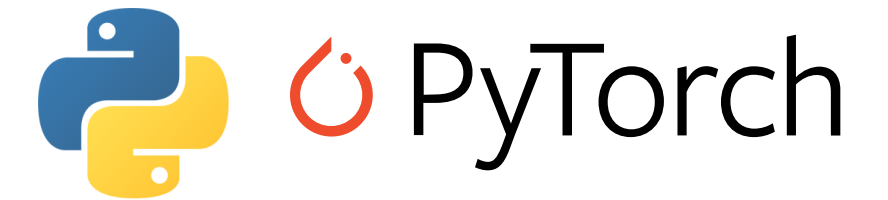
Ensuite compilation du modèle :

```
model.compile(
  optimizer='adam',
  loss='binary_crossentropy',
  metrics=['accuracy']
)
```

Et formation du modèle :

```
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=10
)
```

**Facile, rapide et intuitif**



Les modèles sont définis sous forme de classes (`nn.module`)

```
class AppleCNN(nn.Module):
```

La construction de la structure du réseau :

```
def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 32, kernel_size=3)
    ...
    def forward(self, x):
```

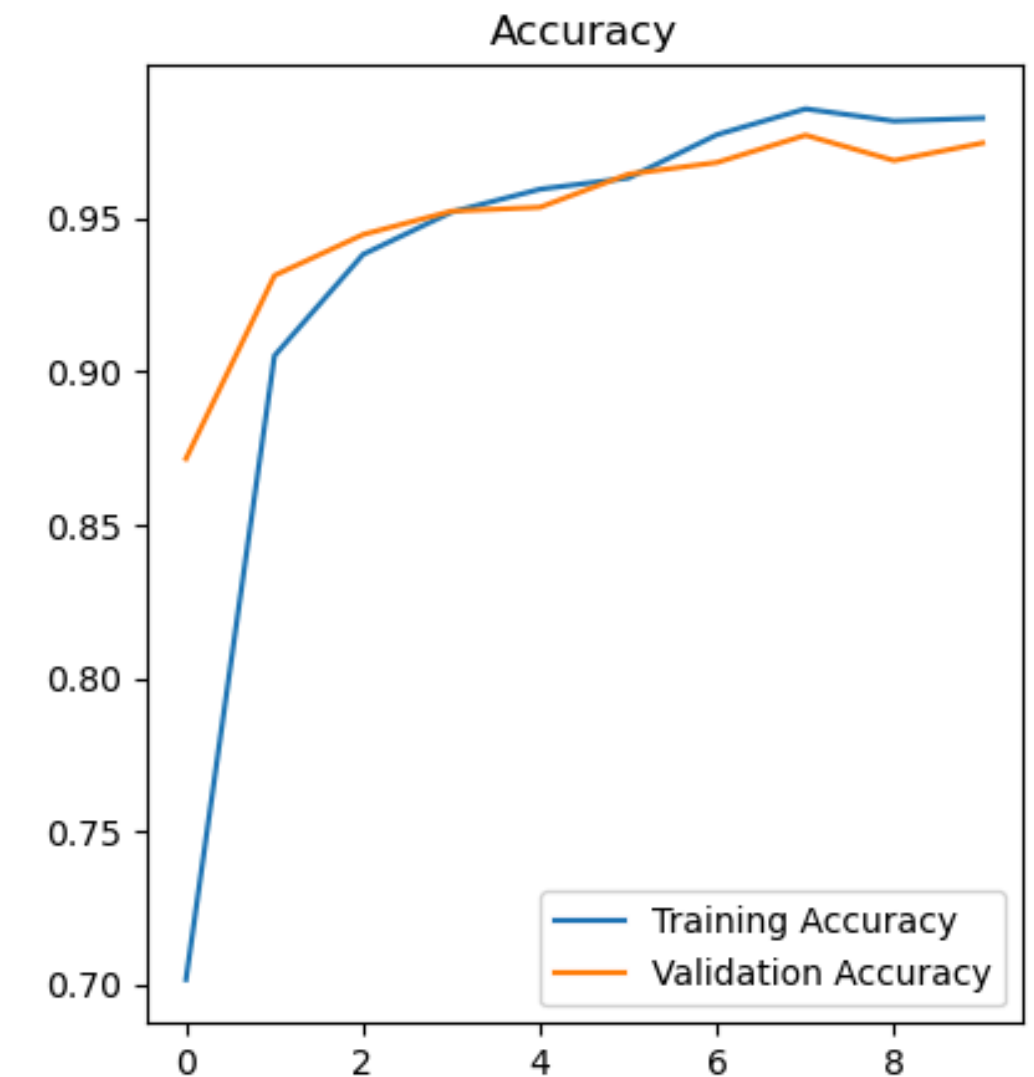
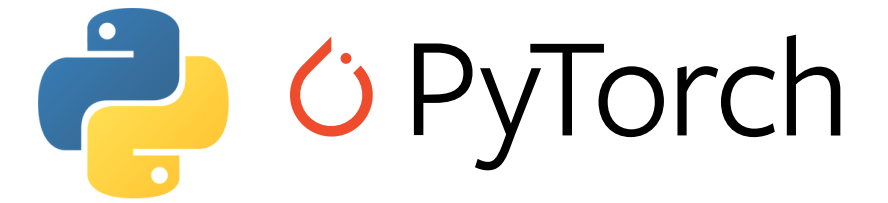
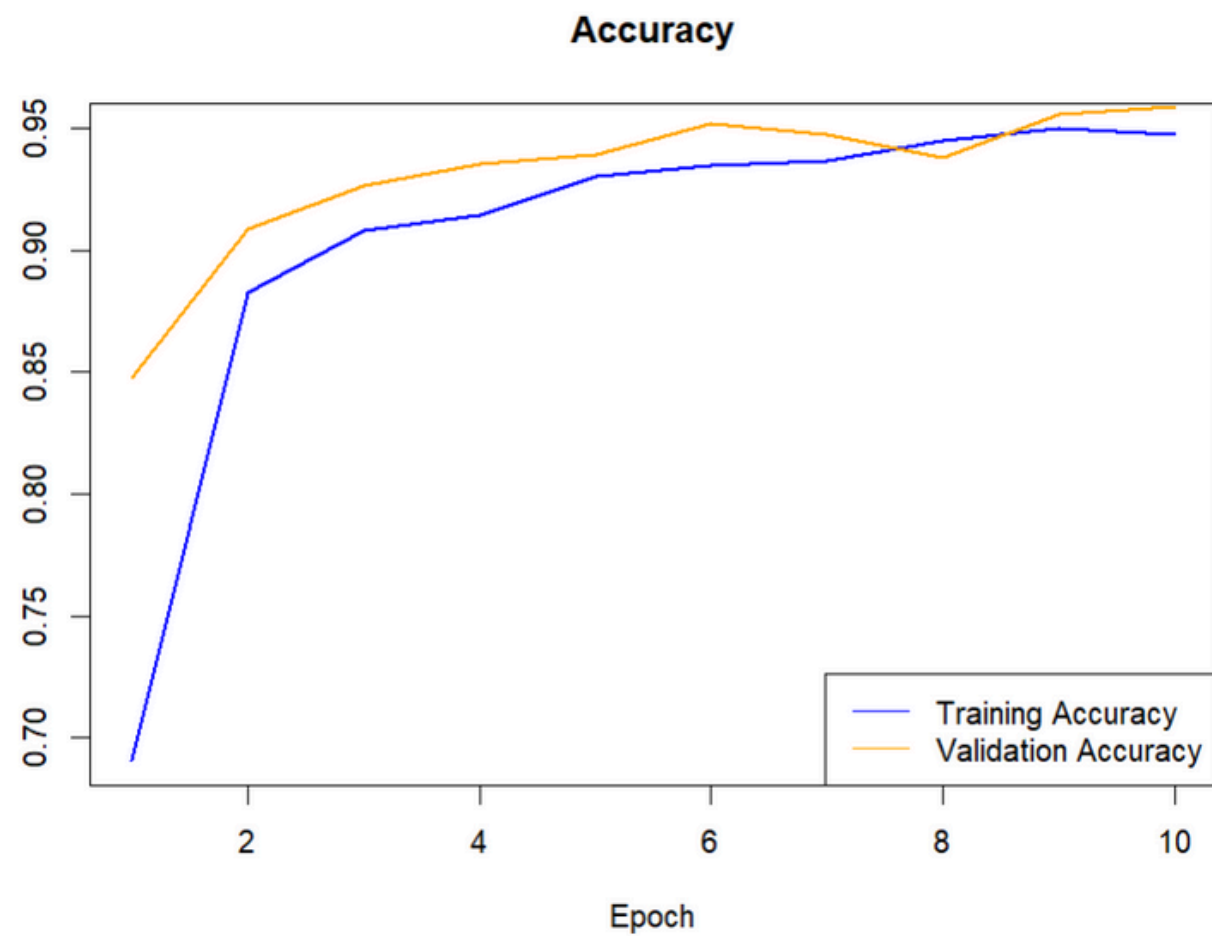
Logique de compréhension des tenseurs

Le modèle d'apprentissage se fait via des boucles explicites manuelles

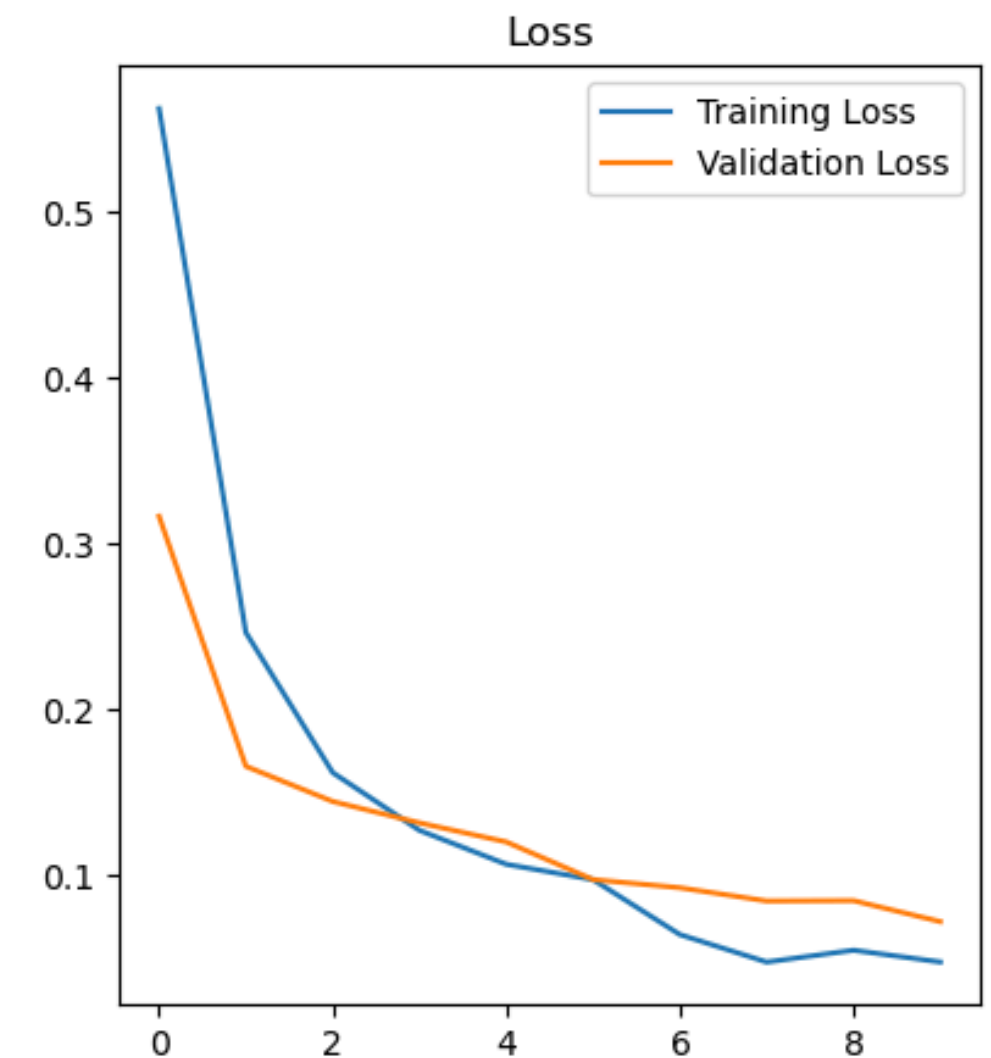
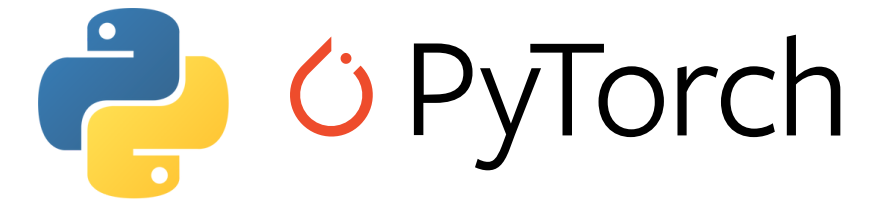
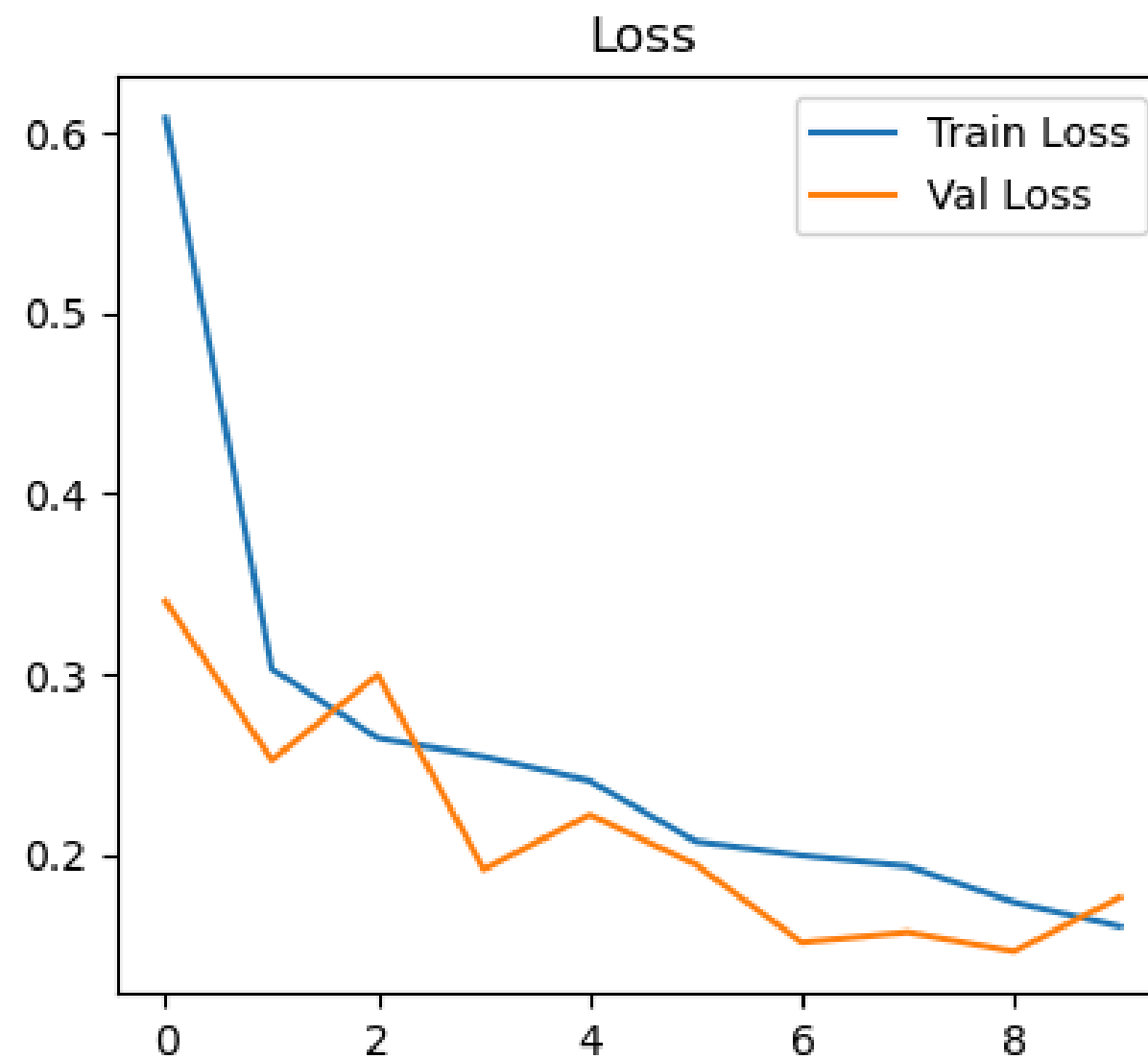
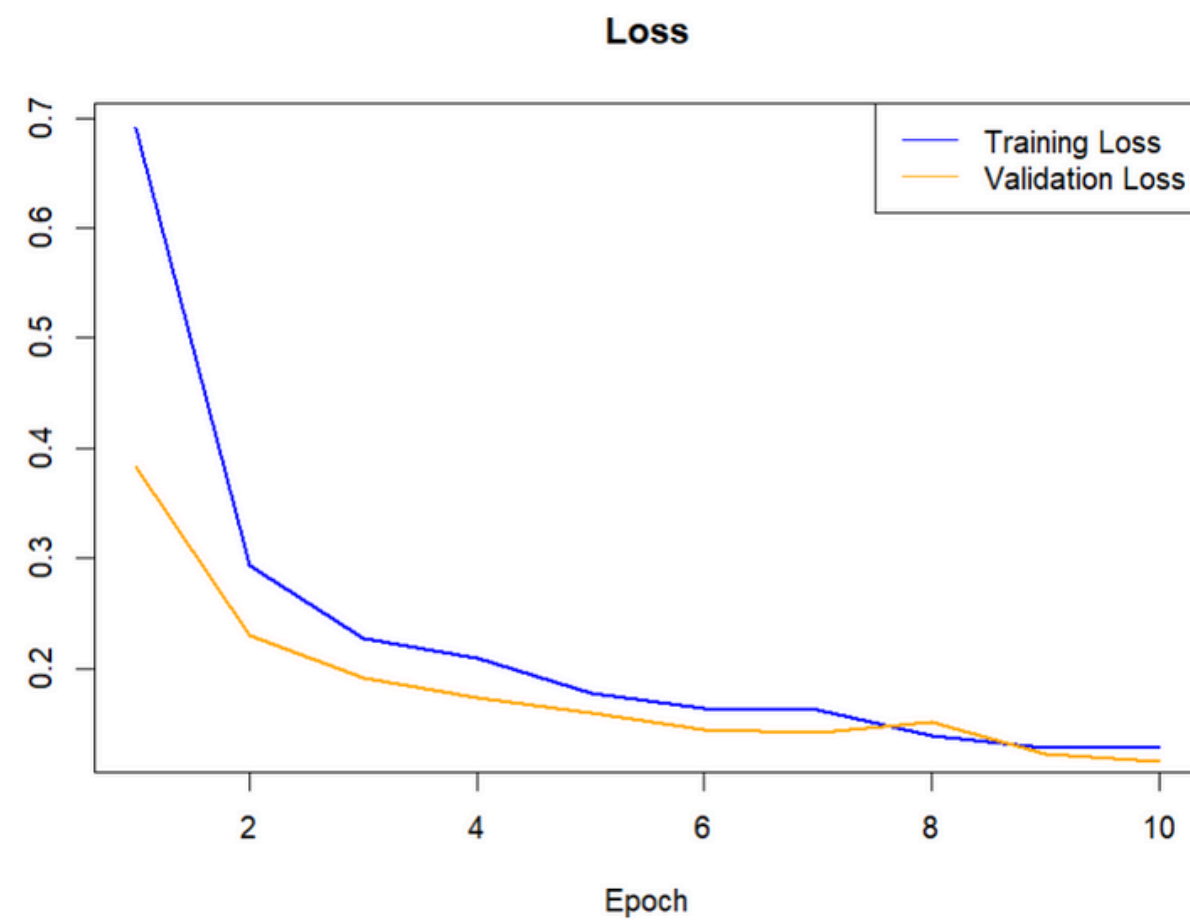
**Plus flexible, plus de code mais plus de maîtrise pour la recherche ou tester de nouvelles architectures**



# Validation du modèle



# Validation du modèle



# *Temps d'entraînement*



**16 minutes**

Performance dépendantes de reticulate

R Studio utilise par défaut un seul coeur



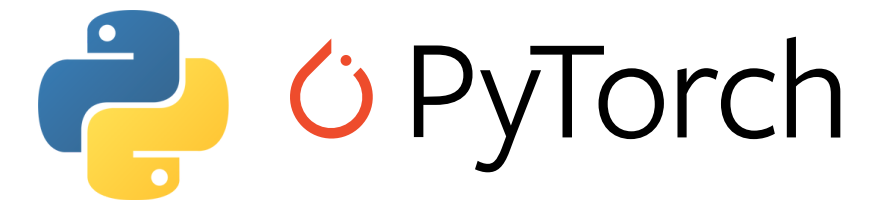
**Très rapide : 1 min 30**

Model.fit() → Boucle d'entraînement compilée (C++/CUDA)

Keras active automatiquement :

- cache (les images sont gardées en mémoire quand possible)
- prefetch (chargement du batch suivant pendant que le GPU travaille)

Les transformations intégrées sont optimisées → s'exécutent dans la pipeline TensorFlow (pas sur python)



**Un peu plus long : 8 min**

Boucle d'entraînement manuelle en Python → plus de surcoût à chaque batch

Le DataLoader n'est pas automatiquement optimisé → il faut une configuration manuelle

Il y a des possibilités d'optimiser le temps de calcul :

- num\_workers
- pin\_memory
- cuDNN benchmark

# Ressources disponibles



Documentation officielle

Documentation officielle

Documentation officielle

+ CRAN

### Getting Started with Keras

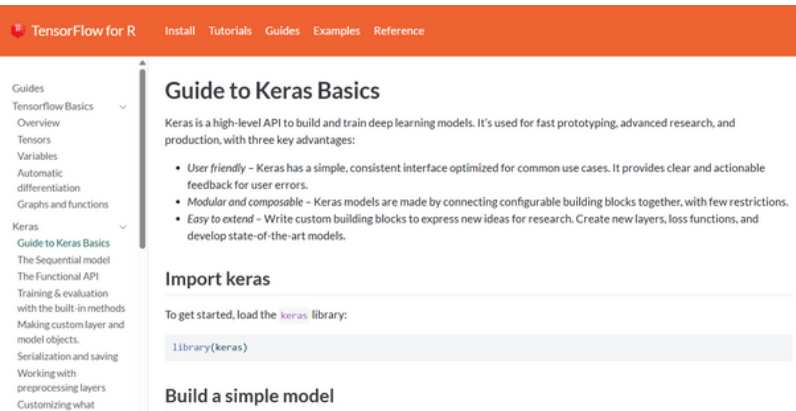
#### Overview

[Keras](#) is a high-level neural networks API developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.* Keras has the following key features:

- Allows the same code to run on CPU or on GPU, seamlessly.
- User-friendly API which makes it easy to quickly prototype deep learning models.
- Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.

This website provides documentation for the R interface to Keras. See the main Keras website at <https://keras.io> for additional information on the project.

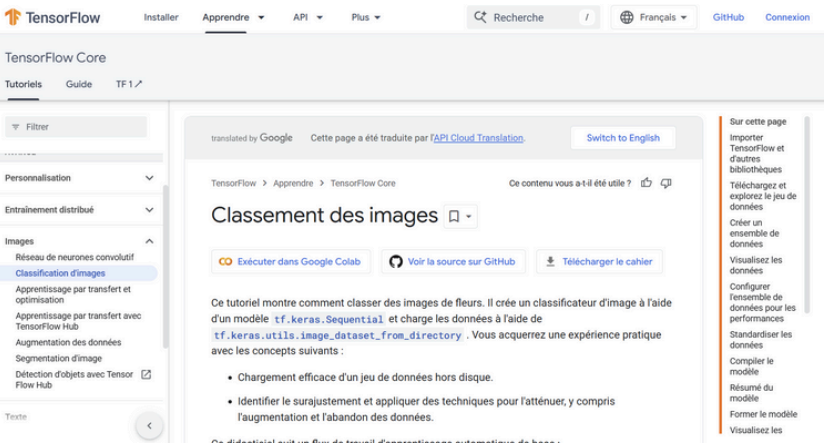
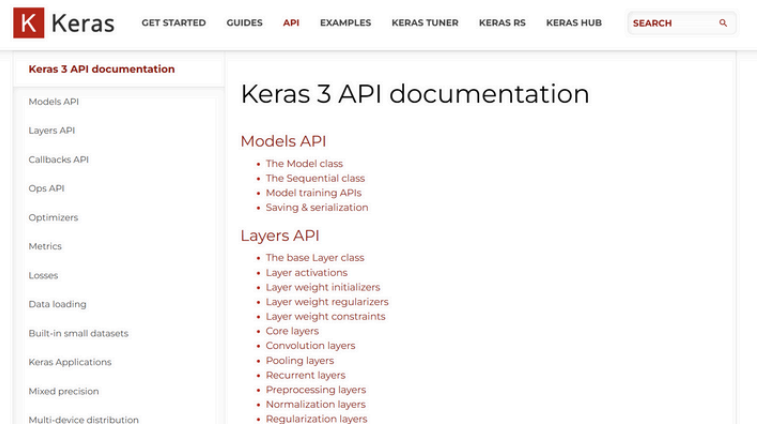
## Nombreux guides



### StackOverFlow

‘keras’ + ‘r’ → **> 600** questions

‘keras’ + ‘CNN’ + ‘r’ → **> 30** questions

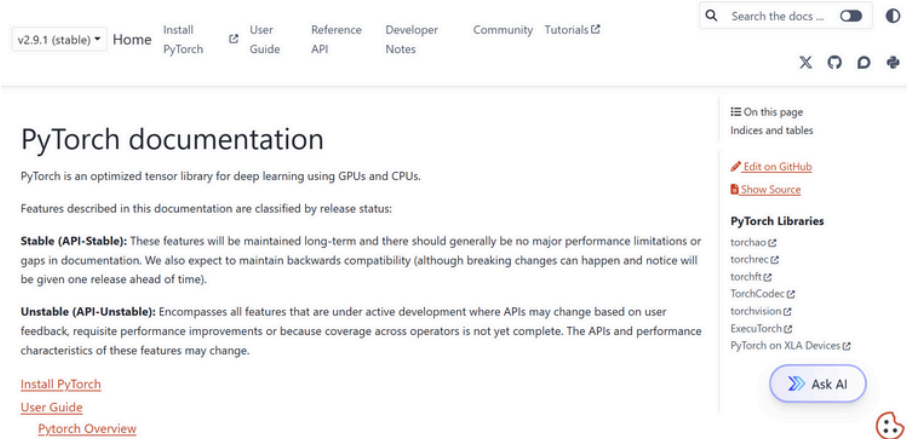


## Immense communauté

### StackOverFlow

‘keras’ + ‘python’ → **> 26 000** questions

‘keras’ + ‘CNN’ + ‘python’ → **> 2 000** questions



## Immense communauté

### StackOverFlow

‘Pytorch’ → **≈ 24 000** questions

‘Pytorch’ + ‘CNN’ → **> 1 400** questions

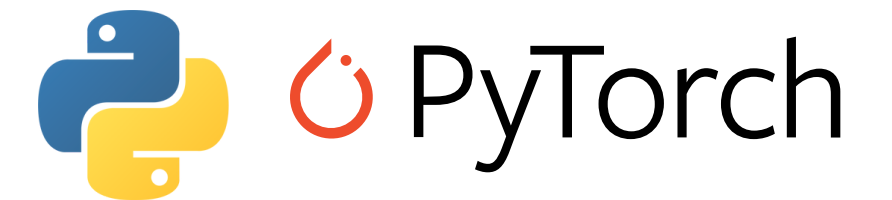
# Conclusion



Installation compliquée  
Gestion des données lourde  
Risque de confusion entre keras et keras3  
Ressources importante  
Syntaxe simple  
  
+ rapide  
- performant



Syntaxe simple  
Optimisation déjà faite  
Ressources importante  
  
+ rapide  
+ performant

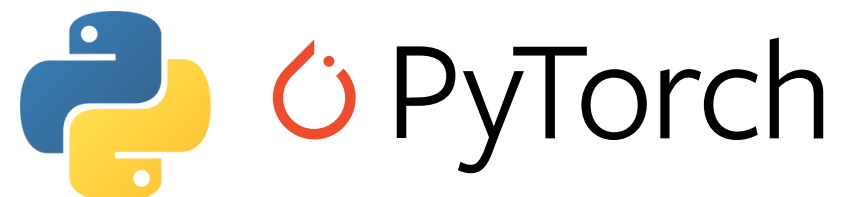


Syntaxe plus complète  
Difficile d'optimiser les calculs  
Meilleur contrôle  
  
- rapide  
+ performant

# Sources



- R Interface to Keras - <https://keras.rstudio.com/>
- CRAN *Getting started with keras* <https://cran.r-project.org/web/packages/keras/vignettes/>
- Karlijn Willems, « keras : Deep Learning in R », Juin 2017





# *Merci*

Avez-vous des questions ?