Machine learning introduction

Part II – Deep Neural Networks

3 - Libraries for Deep Neural Networks : TensorFlow and Keras

Simon Gay

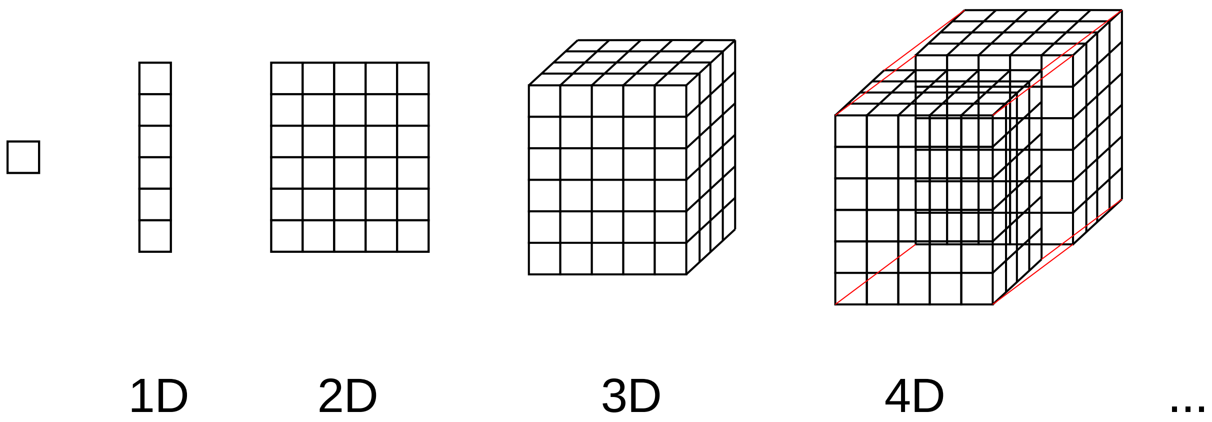
Github : https://github.com/gaysimon/ARTISAN2022

Part II-3 : Libraries for deep neural networks

- **Libraries for deep learning model development:**
  - TensorFlow : developed by Google, open-source since 2015, derived from *Disbelief* project (2011)
    - Since 2017, a Lite version was created for embedded systems
  - MXNet : Developed by Apache, can use many languages
  - Caffe
  - Theanos : since 2008
  - Microsoft Cognitive Toolkit : since 2016
  - PyTorch : developed by Facebook, based on Torch
  - Keras : librairy of high level function to easily develop a deep network. Uses Tensorflow or Theanos libraries

Part II-3 : Libraries for deep neural networks

- **TensorFlow**
  - Library dedicated to the manipulation of *tensors*
    - A tensor is a matrix with a given number of dimensions



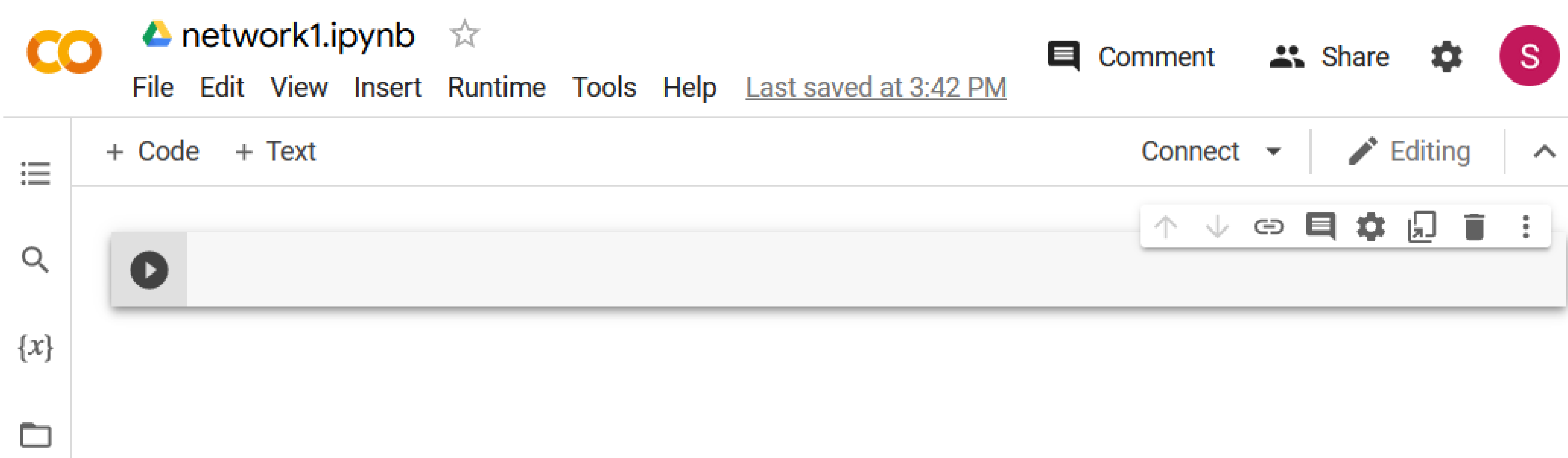
  - Optimized functions
  - Possibility to parallelize on GPU !

Part II-3 : Libraries for deep neural networks

- **Keras**
  - Library of functions exploiting Tensorflow or Theanos
  - Function to:
    - Load a ‘common’ dataset (ie used for benchmarking)
    - Define the architecture of a network
    - Define learning parameters and loss function
    - Train the network
    - Evaluate the network
    - Exploit the network on new data

Part II-3 : Libraries for deep neural networks

- **Let’s start with Keras : implementing a convolutional network**
  - Open your session in Google Colab
  - Create a new notebook



Part II-3 : Libraries for deep neural networks

- **Let's start with Keras : implementing a convolutional network**
  - In the first cell, import the required libraries

```
import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt
import numpy as np
```

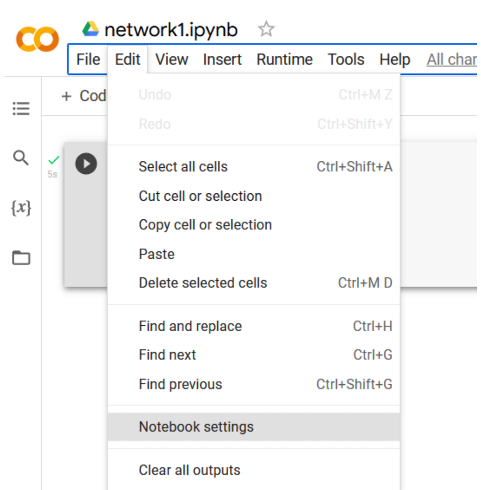

```
import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt
import numpy as np
```

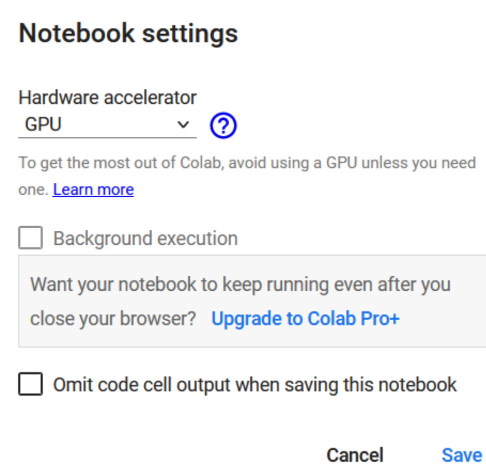

  - Tensorflow and Keras for the neural network
  - Pyplot and numpy for image display

- Run the cell to check the imports

Part II-3 : Libraries for deep neural networks

- **Using a GPU**
  - Training a deep neural network requires a huge computational power, and can take several hours to train
  - Hopefully, Tensorflow can parallelize calculations on a GPU
  - Hopefully, Google Colab can provide GPUs
  - Edit → notebook settings, then select ‘GPU’





Part II-3 : Libraries for deep neural networks

Keras

- **Using a GPU**
  - DO NOT FORGET to turn off the GPU and stop the session after using it !
    - Runtime → manage sessions

Active sessions

Title		Last execution	RAM used	
network1.ipynb Current session	GPU	0 minutes ago	0.89 GB	TERMINATE

  - Otherwise, Google will reduce your access to GPUs for next sessions !

Part II-3 : Libraries for deep neural networks

Keras

- **Using a GPU**
  - Then, copy this part of code and paste it just after importing Tensorflow

```
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')


print('Found GPU at: {}'.format(device_name))
```

/!\ Python takes indentation into account

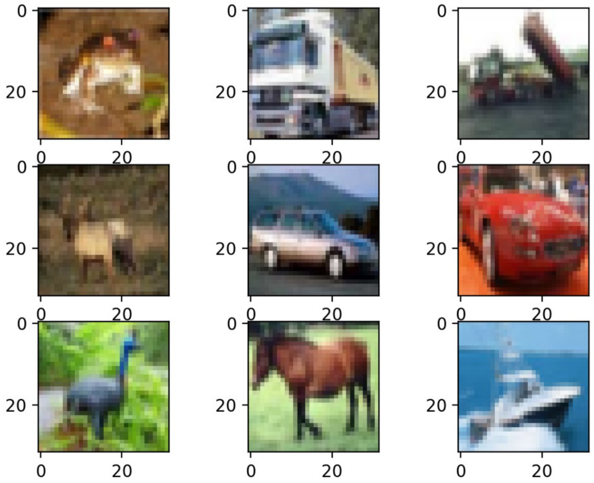
- If the message ‘found GPU’ appears, the GPU is ready

```
import tensorflow as tf

device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))

from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
```

```
Found GPU at: /device:GPU:0
```

Part II-3 : Libraries for deep neural networks

- **Importing a dataset**
  - Keras proposes different common datasets of different types :
    - Boston housing prices (values)
      → *boston_housing*
    - CIFAR-10 (color images of size 32x32)
      → *cifar10*
    - CIFAR-100 (same but with 100 classes)
      → *cifar100*
    - Fashion MNIST (images B&W 28x28)
      → *fashion_mnist*
    - IMDB movie reviews (texts) → *imdb*
    - MNIST (images B&W 28x28) → *mnist*
    - Articles Reuters (texts) → *reuters*



CIFAR-10
(50 000 images)

Part II-3 : Libraries for deep neural networks

- **Importing a dataset**
  - We will use the MNIST dataset
    - Add a new cell in your Colab (‘+code’)
    - import dataset library
    - Load the MNIST dataset into four matrices

```
import tensorflow_datasets as tfds

dataset = keras.datasets.mnist
(img_train, label_train),(img_test,label_test) = dataset.load_data()
```


```
import tensorflow_datasets as tfds

dataset = keras.datasets.mnist
(img_train, label_train),(img_test,label_test) = dataset.load_data()
```

Part II-3 : Libraries for deep neural networks

- **Importing a dataset**
  - The matrices must be converted
    - Images from integer [0,255] to float [0,1]
    - Labels must be converted from a number value to a vector of 10 results (e.g. 3 → [0, 0, 0, 1, 0, 0, 0, 0, 0, 0])
      - Keras proposes a function for that : *to_categorical*

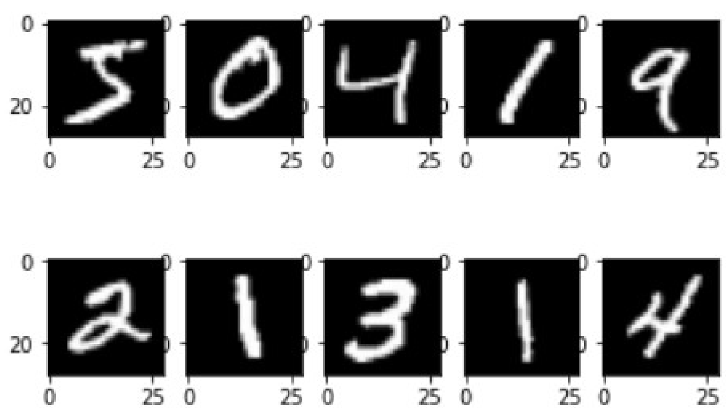
```
import tensorflow_datasets as tfds
from tensorflow.keras.utils import to_categorical

dataset = keras.datasets.mnist
(img_train, label_train),(img_test,label_test) = dataset.load_data()

# convert images into float
img_train=img_train/255.0
img_test=img_test/255.0

output_train=keras.utils.to_categorical(label_train, num_classes=10)
output_test=keras.utils.to_categorical(label_test, num_classes=10)

print(img_train.shape)
print(img_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

Part II-3 : Libraries for deep neural networks

- **Importing a dataset**
  - Let’s display a sample of our dataset
    - Add a new colab cell
    - Copy the following code :

```
for i in range(10):
  plt.subplot(2,5,i+1)
  plt.imshow(img_train[i], cmap='gray')

plt.show()
```



Part II-3 : Libraries for deep neural networks

- **Importing a dataset**
  - Problem : the images have 2 dimensions, but our network requires images with a depths !
  - We add a dimension with *numpy's expand* function
    - Add a new colab cell
    - Add these lines :

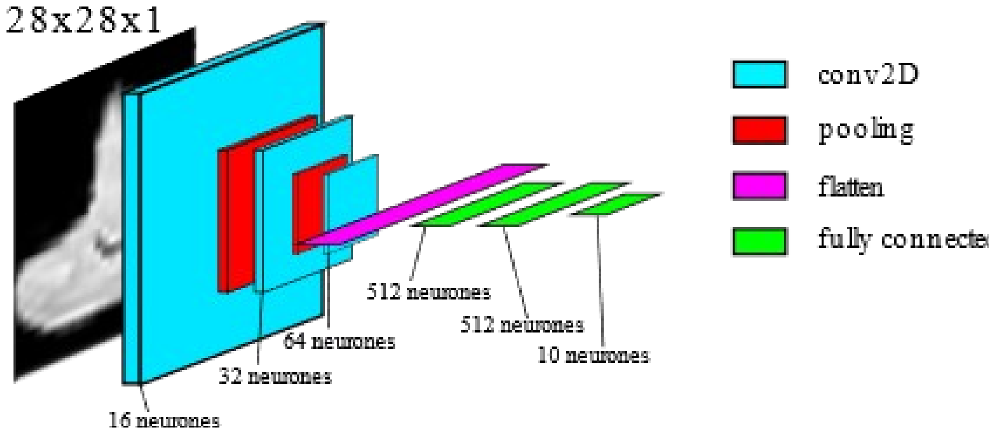
```
img_train = np.expand_dims(img_train,axis=-1)
img_test  = np.expand_dims(img_test, axis=-1)

print(img_train.shape)
print(img_test.shape)
```

```
(60000, 28, 28)        →   (60000, 28, 28, 1)
(10000, 28, 28)            (10000, 28, 28, 1)
```

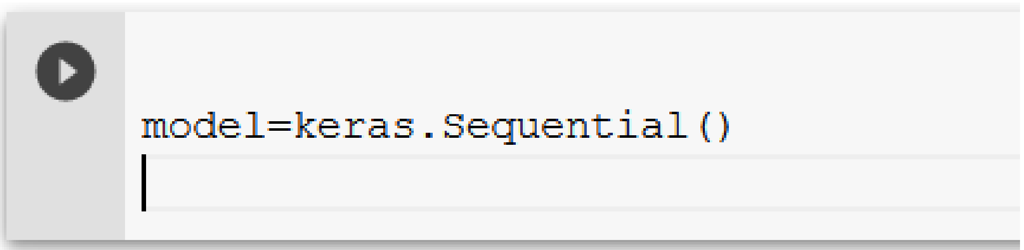
    - Now, our images have a depth

Part II-3 : Libraries for deep neural networks

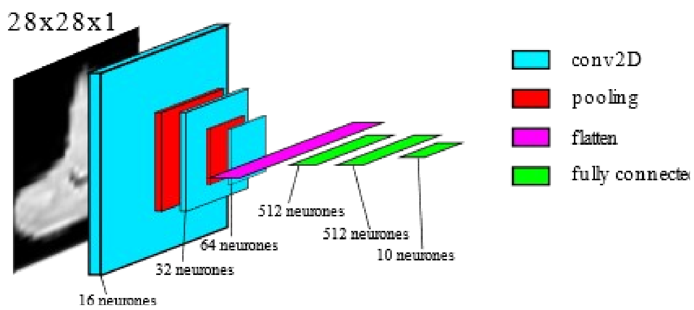
- **Implementing a convolutional network**
  - We will implement this simple network :



    - Create a new Colab cell
    - We start by declaring a Sequential network



Part II-3 : Libraries for deep neural networks



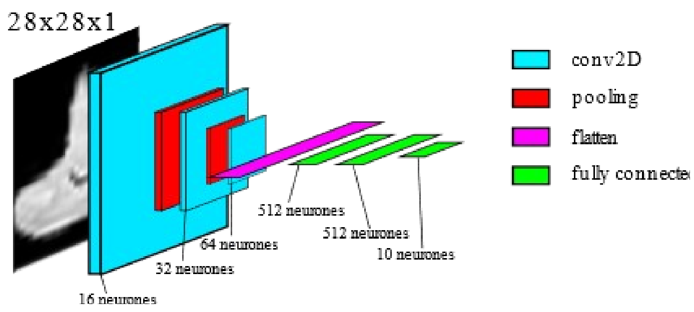
- **Implementing a convolutional network**
  - We then add layers :
    - A first layer of convolutional neurons
      - Input images of size 28x28
      - 16 neurons
      - Convolution kernel of size 3x3
      - Padding
      - A ReLU activation function
    - A max pooling layer with groups of size 2x2

```
model=keras.Sequential()

model.add(keras.layers.Conv2D(input_shape=(28,28,1), filters=16, kernel_size=(3,3), padding="same", activation="relu"))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))
```


```
model=keras.Sequential()

model.add(keras.layers.Conv2D(input_shape=(28,28,1), filters=16, kernel_size=(3,3), padding="same", activation="relu"))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))
```

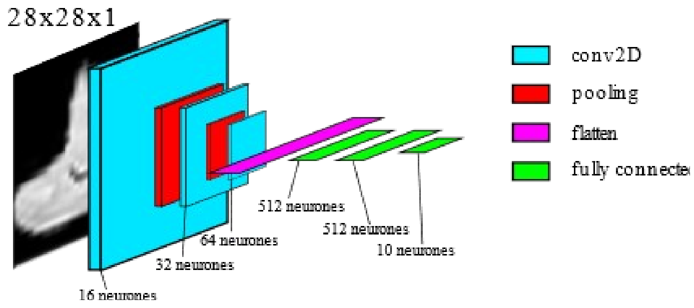
Part II-3 : Libraries for deep neural networks

- **Implementing a convolutional network**
  - We then add layers :
    - A new convolutional layer (32 neurons)
    - A new max pooling
    - A last convolutional layer (64 neurons)




```
model=keras.Sequential()

model.add(keras.layers.Conv2D(input_shape=(28,28,1), filters=16, kernel_size=(3,3), padding="same", activation="relu"))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))

model.add(keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding="same", activation="relu"))
model.add(keras.layers.MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu"))
```

Part II-3 : Libraries for deep neural networks

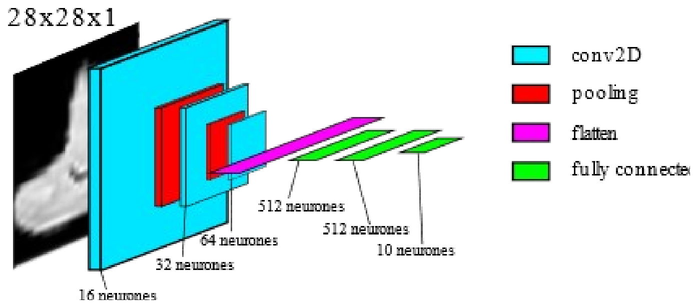


- **Implementing a convolutional network**
  - We then add the fully connected part :
    - A flatten layer
    - 2 fully connected layers of 512 neurons
    - An output fully connected layer of 10 neurons (softmax function)

```
[ ]
model=keras.Sequential()

model.add(keras.layers.Conv2D(input_shape=(28,28,1), filters=16, kernel_size=(3,3), padding="same", activation="relu"))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding="same", activation="relu"))
model.add(keras.layers.MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu"))


model.add(keras.layers.Flatten( ))

model.add(keras.layers.Dense(units=512,activation="relu"))
model.add(keras.layers.Dense(units=512,activation="relu"))

model.add(keras.layers.Dense(units=10, activation="softmax"))
```


```
model.add(keras.layers.Flatten( ))

model.add(keras.layers.Dense(units=512,activation="relu"))
model.add(keras.layers.Dense(units=512,activation="relu"))

model.add(keras.layers.Dense(units=10, activation="softmax"))
```

Part II-3 : Libraries for deep neural networks



- **Implementing a convolutional network**
  - The model architecture can be displayed with

    model.summary()

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        160

 max_pooling2d (MaxPooling2D  (None, 14, 14, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 64)          18496

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 512)               1606144

 dense_1 (Dense)             (None, 512)               262656

 dense_2 (Dense)             (None, 10)                5130

=================================================================
Total params: 1,897,226
Trainable params: 1,897,226
Non-trainable params: 0
_________________________________________________________________
```

Part II-3 : Libraries for deep neural networks

- **Keras :** some important types of layers (with <u>some</u> important parameters) :

  - **tf.keras.layers.Dense**(
    units, → number of neurons in the layer
    activation=None, → activation function
    use_bias=True, → use a bias or not
)

  - **tf.keras.layers.Conv2D**(note : versions 1D and 3D also exist
    filters, → number of neurons
    kernel_size, → size of kernel
    strides=(1, 1), → displacement (steps) of kernel
    padding="valid", → padding when *"same"*
    activation=None, → activation function
    use_bias=True, → use a bias or not
)

  – Some activation functions: relu, sigmoid, softmax, tanh, ...

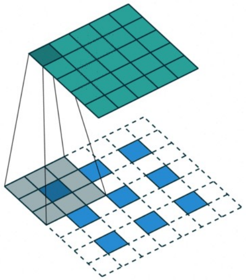
Part II-3 : Libraries for deep neural networks

- **Keras :** Some important types of layers :

  - **tf.keras.layers.MaxPooling2D(**
    pool_size=(2, 2), → pooling reduction
    strides=None, → ‘movements’
    padding="valid"
)

  - **tf.keras.layers.UpSampling2D(**
    size=(2, 2), → unpooling ‘inflation’
    interpolation="nearest" → "nearest" or "bilinear"
)

  - **tf.keras.layers.Conv2DTranspose(**
    filters, → number of output images (depth)
    kernel_size, → kernel size
    strides=(1, 1), → adds 0 between pixels
    equivalent to a ‘bed of nails
    padding="valid",
    activation=None,
    use_bias=True,
)



  – Complete list of 100+ types of layers on: https://keras.io/api/layers/

Part II-3 : Libraries for deep neural networks

- **Keras :** construction of a network model
  - A sequential model can also be defined as a vector of layers:

```
model = Sequential( [
        Conv2D(input_shape=(224,224,3), filters=64, kernel_size=(3,3), padding="same", activation="relu"),
        Conv2D(filters=64,kernel_size=(3,3), padding="same", activation="relu"),
        MaxPool2D(pool_size=(2,2), strides=(2,2)), …
] )
```

  - Another method: functionnal model
    - Layers are created separately and connected

```
input_layer = Input(shape=(28,28,1))
conv1 = Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32,32,1))(input_layer)
conv2 = Conv2D(64, (3, 3), activation='relu', input_shape=(32,32,1))(input_layer)
merge = concatenate([conv1, conv2])
flatten = Flatten( )(merge)
output_layer = Dense(num_classes, activation='softmax')(flatten)
model = Model(inputs=input_layer, outputs=output_layer)
```

    - Allows defining non sequential models (fusion, split...)

Part II-3 : Libraries for deep neural networks

- **Learning parameters**
  - After constructing the network, we have to define
    - The loss function (error measure to optimize)
    - The optimization function (weight update algorithm)

  - With a softmax activation function, we will use the CategoricalCrossentropy loss function !

    Other loss functions :
    - *MeanSquaredError :* $E = (y - y)^2$
    - … and dozens of other available functions : https://keras.io/api/losses

  - In a new Colab cell, add :

```
loss=keras.losses.CategoricalCrossentropy();
```

loss=keras.losses.CategoricalCrossentropy();

Part II-3 : Libraries for deep neural networks

- **Learning parameters**
  - The optimization function is the function that reinforce neurons' weights
    - The SGD is the gradient descent presented previously
    - Other more optimized function adapt the learning rate to reduce learning time
      - Adam, Adadelta, Nadam, Ftrl… (see https://keras.io/api/optimizers/)

    - We select Adam optimizer, with a learning rate of 0.001 :

```
loss=keras.losses.CategoricalCrossentropy();
optim=keras.optimizers.Adam(learning_rate=0.001)
```

```
optim=keras.optimizers.Adam(learning_rate=0.001)
```

Part II-3 : Libraries for deep neural networks

- **Finalizing the network**
  - Finally, the network is compiled : the neuron reinforcement functions are defined according to the network architecture and selected loss and optimization functions :

```
loss=keras.losses.CategoricalCrossentropy();
optim=keras.optimizers.Adam(learning_rate=0.001)

model.compile(loss=loss, optimizer=optim, metrics=["accuracy"])
```

```
model.compile(loss=loss, optimizer=optim, metrics=["accuracy"])
```

  - Run the cell : if no error message appears, the network is ready !

Part II-3 : Libraries for deep neural networks

Keras

- **Training the network**
  - Keras takes care of everything with function *fit*
    - Parameters:
      - The training data and label
      - The size of batches (number of data learned simultaneously)
      - The number of epoches
      - The level of details to display
    - In a new cell, write :

```
model.fit(img_train, output_train, batch_size=16, epochs=10, verbose=2)
```

```
Epoch 1/10
3750/3750 - 13s - loss: 0.1175 - accuracy: 0.9638 - 13s/epoch - 3ms/step
Epoch 2/10
3750/3750 - 10s - loss: 0.0502 - accuracy: 0.9848 - 10s/epoch - 3ms/step
```

- The learning process can takes several minutes...

Part II-3 : Libraries for deep neural networks

Keras

- **Evaluating/Exploiting the network**
  - Keras takes care of everything (again) :
    - Function evaluate (works in a same way than fit)

```
model.evaluate(img_test, output_test, batch_size=16, verbose=2)
```

```
625/625 - 1s - loss: 0.0703 - accuracy: 0.9884 - 1s/epoch - 2ms/step
[0.07027573138475418, 0.9883999824523926]
```

< 0.12 %

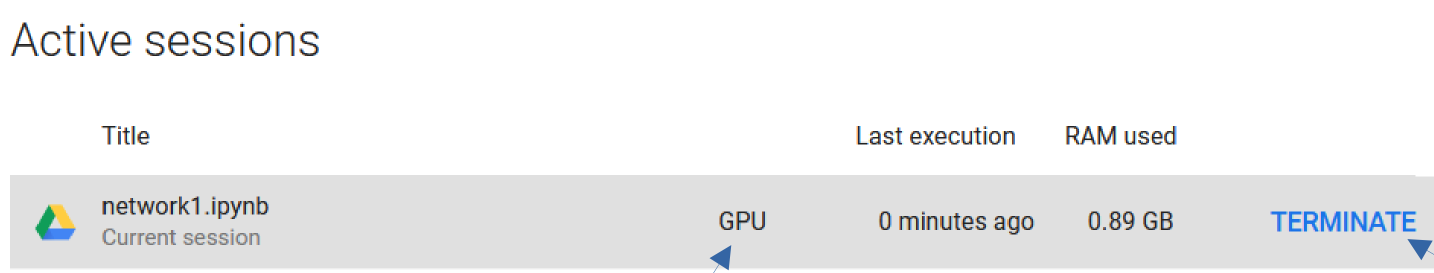
- An image can be used as parameter to get a prediction :

```
print(label_test[0:1])

model(img_test[0:1,:,:])
```

```
[7]
<tf.Tensor: shape=(1, 10), dtype=float32, numpy=
array([[1.1164599e-26, 2.6863731e-18, 2.3253600e-17, 1.2962786e-17,
        2.5873384e-16, 5.7846084e-20, 1.6555734e-36, 1.0000000e+00,
        1.5967833e-22, 9.6893570e-12]], dtype=float32)>
```

Part II-3 : Libraries for deep neural networks

- **Next network**
  - Save your colab
  - **Do not forget to disable the GPU and stop the session !**

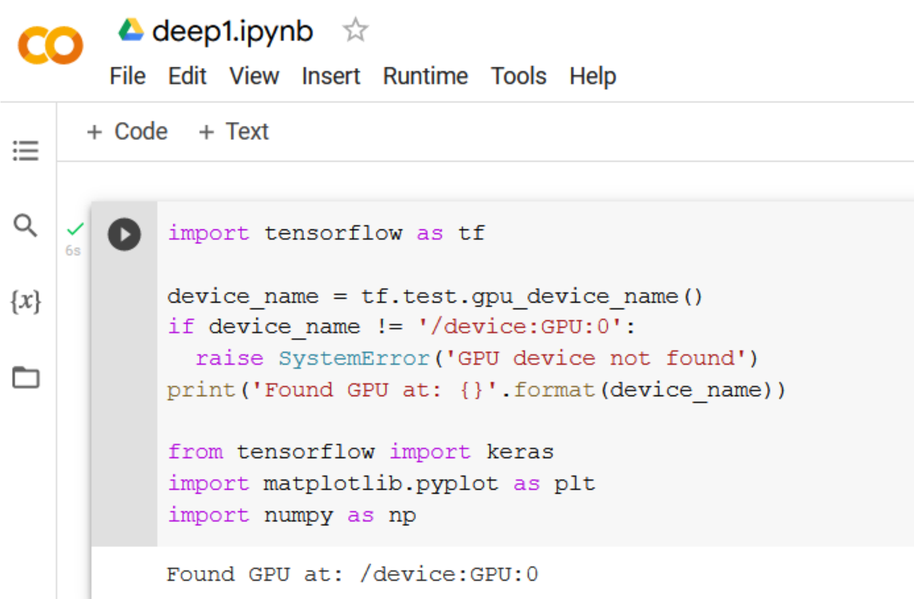


Edit → notebook settings, then select ‘none’

- **Note to go further :** there is a dataset similar to MNIST :
  - dataset = keras.datasets.**fashion_mnist**
  - You can try to improve your model to get the best accuracy

Part II-3 : Libraries for deep neural networks

- **A true deep neural network !**
  - New way to build a dataset
  - Transfert learning from an existing network
  - Goal : recognize a cat from a dog
  - Create a new Colab Notebook, enable GPU, and copy the imports




```
import tensorflow as tf

device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))

from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
```

Part II-3 : Libraries for deep neural networks

- **The dataset**
  - We will download a set of images and unzip them in different folders
  - Copy and paste these code lines in two different colab cells, and execute them :

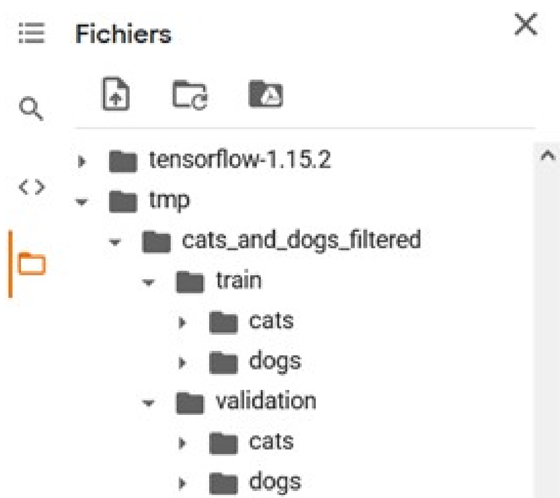
```
!wget --no-check-certificate \
    https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
    -O /tmp/cats_and_dogs_filtered.zip
```

```
import os
import zipfile

local_zip = '/tmp/cats_and_dogs_filtered.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()

train_dir = '/tmp/cats_and_dogs_filtered/train'
test_dir = '/tmp/cats_and_dogs_filtered/validation'
```

Part II-3 : Libraries for deep neural networks

Keras

- **The dataset**
  - The images are stored in a specific folder tree



  - Keras proposes an object that can browse this kind of folder tree to feed a neural network for training or evaluating it :
    - The ***ImageDataGenerator***

Part II-3 : Libraries for deep neural networks

- **The dataset**
  - In a new Colab cell, we write the image generator :

```
trainDataGenerator = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
trainData = trainDataGenerator.flow_from_directory(directory=train_dir, target_size=(224,224))

testDataGenerator = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
testData = testDataGenerator.flow_from_directory(directory=test_dir, target_size=(224,224))
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

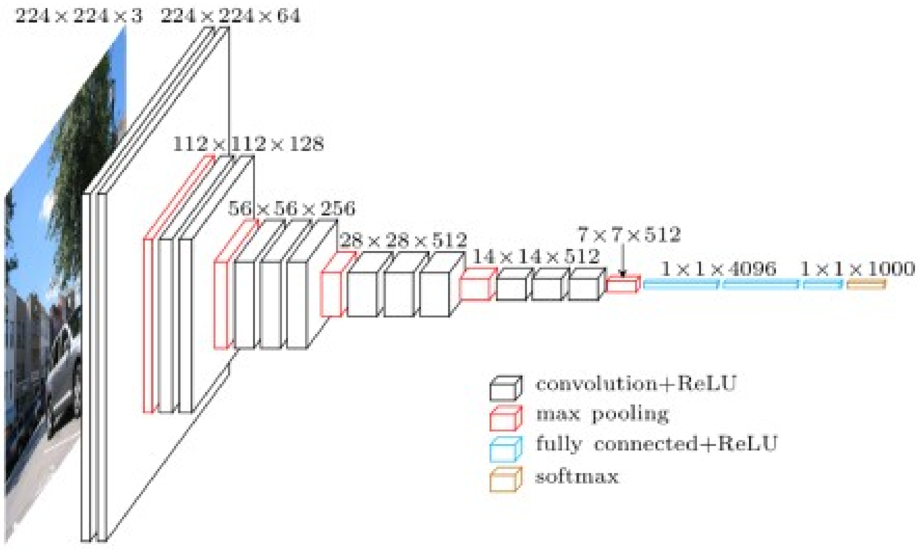
  - Then, we load the VGG16 network

```
VGG16= keras.applications.VGG16(weights="imagenet")

VGG16.summary()
```

Part II-3 : Libraries for deep neural networks

- **The VGG16 network**
  - We cannot train a network with so many parameters

```
==============================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```



  - Especially with such a small dataset !
  - Hopefully, the VGG16 network was trained with multiple animals (including cats ans dogs) : the lowest layers may recognize pertinent features
- Note : VGG16 was trained during multiple days on high-end GPUs with *imageNet* dataset (>1M labeled images)

Part II-3 : Libraries for deep neural networks

- **The VGG16 network**
  - The VGG16 network, as proposed by Keras, can be loaded without the fully connected part
    - Parameter include_top can be set to False
    - Parameter weights='imagenet' allows loading weight obtained with imagenet dataset (10M+ images)
  - Add a new Colab cell and paste this :

```
base_VGG16= keras.applications.VGG16(weights="imagenet", include_top=False, input_shape=(224,224,3))
base_VGG16.summary()
```

  - Observe where the network is cut :
    - As the flatten layer is not included, other convolutional layers can be added

Part II-3 : Libraries for deep neural networks

- **The VGG16 network**
  - We freeze the weights of the network (in a new Colab cell) :

```
base_VGG16.trainable=False

base_VGG16.summary()
```

```
==========================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
__________________________________________
```

Part II-3 : Libraries for deep neural networks

- **The VGG16 network**
  - The VGG16 part is then used as a layer for our network
  - We then add the fully connected part using a flatten layer, a fully connected layer with 50 neurons and an output flatten layer with 2 neurons

```
model=keras.models.Sequential()
model.add(base_VGG16)

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(50, activation="relu"))
model.add(keras.layers.Dense(2, activation="softmax"))

model.summary()
```

```
==============================================
Total params: 15,969,240
Trainable params: 1,254,552
Non-trainable params: 14,714,688
______________________________________________
```

Part II-3 : Libraries for deep neural networks

Keras

- **The VGG16 network**
  - The network is compiled with Categorical_Crossentropy and Adam functions

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

  - And trained using the data generator that provides both data and labels

```
model.fit(trainData, epochs=5, batch_size=32)
```

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

model.fit(trainData, epochs=5, batch_size=32)
```

```
Epoch 1/5
19/63 [========>.....................] - ETA: 6s - loss: 0.8847 - accuracy: 0.7105
```

Part II-3 : Libraries for deep neural networks

Keras

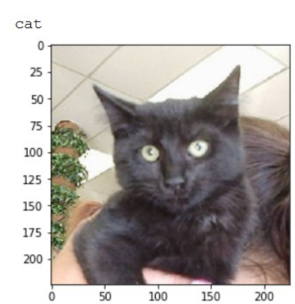
- **The VGG16 network**
  - The network can then be evaluated :

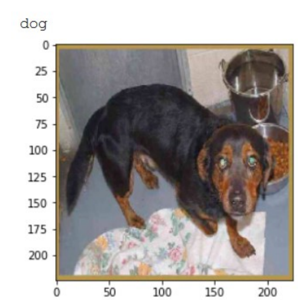
```
model.evaluate(testData, batch_size=32, verbose=2)
```

```
32/32 - 6s - loss: 0.2513 - accuracy: 0.9080 - 6s/epoch - 186ms/step
[0.2513466775417328, 0.9079999923706055]
```

  - The following code allows testing an image :

```
test_cats_files = os.listdir(test_dir+"/cats")
test_dogs_files = os.listdir(test_dir+"/dogs")

img = keras.preprocessing.image.load_img(test_dir+"/cats/"+test_cats_files[8],target_size=(224,224))
#img = keras.preprocessing.image.load_img(test_dir+"/dogs/"+test_dogs_files[8],target_size=(224,224))
img = np.asarray(img)
plt.imshow(img)
img = np.expand_dims(img, axis=0)

prediction=model.predict(img)

if prediction.argmax()==0: print("cat")
else : print("dog")
```





Conclusion

- Deep learning is a recent domain...
- … but already shows impressive results and achievements
- Deep learning can still be improved on many aspects, and there is still a great margin for improvements

- A neuronal network however remains a classifier algorithm, unable to understand or interpret data that it generates, and cannot generate more than what was in training dataset.

- There are other forms of AI, such as reinforcement learning and developmental robotics/learning, that try to interact with an environment to overcome these limitations…
  - … But this is another story !