Compression2

May 7, 2022

[ ]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
from tensorflow import keras
import tensorflow as tf
from keras import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Activation,␣
↪BatchNormalization, Flatten
from keras.utils.np_utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from numpy.lib.stride_tricks import as_strided
from natsort import natsorted
import cv2
import glob
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1 On s'intéresse à présent à la degradation des image au passage par un filtre gaussien 2D

[ ]:
```
import pathlib

data_loc = tf.keras.utils.get_file(
    "dataset.zip",
    "https://github.com/anajmedd/Image-Compression/blob/main/dataset.zip?
↪raw=true",
    extract=False)

import zipfile
with zipfile.ZipFile(data_loc, 'r') as zip_ref:
    zip_ref.extractall('/content/datasets')
```

```
data_loc = pathlib.Path('/content/datasets/datasets')
print(data_loc)
print(os.path.abspath(data_loc))
```

```
Downloading data from https://github.com/anajmedd/Image-
Compression/blob/main/dataset.zip?raw=true
26001408/25998756 [==============================] - 0s 0us/step
26009600/25998756 [==============================] - 0s 0us/step
/content/datasets/datasets
/content/datasets/datasets
```

Construction du dataset des image en noir et blanc

[ ]:
```
path = '/content/datasets/datasets/cat/flickr_cat_000056.jpg'
w = 11
sigma = 2
img = cv2.imread(path)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

f = plt.figure(figsize=(10,13))
f.add_subplot(121).imshow(img)
plt.axis('off')
f.add_subplot(122).imshow(img_gray, cmap='gray')
plt.axis('off')
```

[ ]:
```
(-0.5, 511.5, 511.5, -0.5)
```

[ ]:
```
fd = '/content/datasets/datasets/cat'
fd1 = '/content/datasets/datasets/dog'
```

[ ]:
```
dir = '/content/datasets/datasets_bw/cat'
#os.mkdir('/content/datasets/datasets_bw')
#os.mkdir('/content/datasets/datasets_bw/cat')
from PIL import Image
c=1
for filename in os.listdir(fd):
     if '.jpg' in filename:
       path = fd+'/'+filename
       img = cv2.imread(path)
       name = 'cat' + str(c)+ '_bw.jpg'
       c+=1
       img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
       cv2.imwrite(os.path.join(dir , name), img_gray)
       cv2.waitKey(0)
```

Construction de la réponse du filtre gaussien

[ ]:
```
def get_all_window(M, w):
    M = np.pad(M, w//2, 'symmetric')
    sub_shape = (w, w)
    view_shape = tuple(np.subtract(M.shape, sub_shape) + 1) + sub_shape
    arr_view = as_strided(M, view_shape, M.strides * 2)
    arr_view = arr_view.reshape((-1,) + sub_shape)
    return arr_view

def convolution_2d(im, K):
    w, _ = K.shape
    m,n = im.shape
    im_all_subw = get_all_window(im, w)
    X = np.sum(np.sum(im_all_subw * K, 1), 1)
    return X.reshape(m,n)
```

[ ]:
```
def filtre2D(w, sigma):
    w = w + (w % 2 == 0)
    F = np.zeros([w,w])
    mid = w//2
    k = np.arange(w) - mid
    for i in k:
        for j in k:
            par = (i**2 + j**2)/(2*sigma**2)
            F[i + mid,j + mid] = np.exp(-par)/(2*np.pi*sigma**2)
    return F
```

[ ]:
```
h = filtre2D(w, sigma)
im_gauss = convolution_2d(imgrey, h)

f = plt.figure(figsize=(10,13))
f.add_subplot(221, title='Originale').imshow(img_gray, cmap='gray')
```

```
plt.axis('off')
f.add_subplot(222, title='Resultat').imshow(im_gauss, cmap='gray')
plt.axis('off')
```

[ ]: (-0.5, 511.5, 511.5, -0.5)

Construction du dataset contenant les image filtrées

[ ]:
```
!rm -rf /content/datasets/data_filtered
os.mkdir('/content/datasets/data_filtered')
os.mkdir('/content/datasets/data_filtered/cat')
data_loc1 = '/content/datasets/datasets/cat'
data_loc3 = '/content/datasets/data_filtered/cat'
```

[ ]:
```
import os
c=1
for filename in os.listdir(data_loc1):
    if filename.endswith(".jpg"):
        path = data_loc1 +'/' + filename
        img = cv2.imread(path)
        name='cat'+str(c)+'_filtred.jpg'
        c+=1
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        im_gauss = convolution_2d(img_gray, h)
        cv2.imwrite(os.path.join(data_loc3 , name), im_gauss)
        cv2.waitKey(0)
    else:
         continue
```

[ ]:
```
def load_img(filepath):
    img = cv2.cvtColor(cv2.imread(filepath), cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32)
    img = img / 255.
    return img
```

[ ]:
```
from skimage.metrics import peak_signal_noise_ratio, structural_similarity

def print_arr_metrics(m, n):
    x = [peak_signal_noise_ratio(a, b) for a, b in zip(m, n)]
    y = [structural_similarity(a, b, multichannel=True, data_range=1) for a, b␣
 ↪in zip(m, n)]
    print(f"PSNR: {np.mean(x)}")
    print(f"SSIM: {np.mean(y)}")
#     return x, y
```

[ ]:
```
!rm -rf /content/datasets/folder
```

[ ]:
```
import shutil


target = '/content/datasets/folder'
pw1 = '/content/datasets/datasets_bw/cat'
pw2 = '/content/datasets/data_filtered/cat'
os.mkdir(target)
for filename in os.listdir(pw1):
  shutil.copyfile(pw1+'/'+filename,target+'/'+filename)
for filename in os.listdir(pw2):
  shutil.copyfile(pw2+'/'+filename,target+'/'+filename)
```

[ ]:
```
bw_files = natsorted(glob.glob('/content/datasets/folder/*_bw.jpg'))
filtred_files = natsorted(glob.glob('/content/datasets/folder/*_filtred.jpg'))
```

[ ]:
```
bw_images = [load_img(v) for v in bw_files]     ## Black and white images
filtred_images = [load_img(v) for v in filtred_files]
```

[ ]:
```
print(bw_files[0])
print(filtred_files[0])
```

```
/content/datasets/folder/cat1_bw.jpg
/content/datasets/folder/cat1_filtred.jpg
```

[ ]:
```
def print_arr_metrics(m, n):
    x = [peak_signal_noise_ratio(a, b) for a, b in zip(m, n)]
    y = [structural_similarity(a, b, multichannel=True, data_range=1) for a, b␣
 ↪in zip(m, n)]
    print(f"PSNR: {np.mean(x)}")
    print(f"SSIM: {np.mean(y)}")
```

```
#     return x, y
```

[ ]:
```
print_arr_metrics(filtred_images, bw_images)
```

```
PSNR: 26.99100209033164
SSIM: 0.743821117465575
```

2 Dégradation des images par la SVD

[82]:
```
!rm -rf /content/datasets/datasets_SVD
parent_dir = '/content/datasets/datasets_SVD'
cat_dir = '/content/datasets/datasets_SVD/cat'
os.mkdir(parent_dir)
os.mkdir(cat_dir)
```

[83]:
```
c=1
r=50   ## Valeur à changer pour visualiser l'effet sur le PSNR
data_loc1 = '/content/datasets/datasets/cat'
for filename in os.listdir(data_loc1):
    if filename.endswith(".jpg"):
        img = Image.open(data_loc1+'/'+filename)
        imggray = img.convert('LA')
        imgmat = np.array(list(imggray.getdata(band=0)),float)
        imgmat.shape= (imggray.size[1], imggray.size[0])
        imgmat = np.matrix(imgmat)
        U, sigma, V = np.linalg.svd(imgmat)
        name='cat' +str(c)+'_svd.jpg'
        c+=1
        reconsting = np.matrix(U[:, :r]) * np.diag(sigma[:r]) * np.matrix(V[:r,:
 ↪])
        im = Image.fromarray(reconsting)
        rgb_im = im.convert('RGB')
        rgb_im.save(cat_dir+'/'+name,'JPEG')
    else:
         continue
```

[84]:
```
import shutil

!rm -rf /content/datasets/folder2
target2 = '/content/datasets/folder2'
pwd1 = '/content/datasets/datasets_bw/cat'
pwd2 = '/content/datasets/datasets_SVD/cat'
os.mkdir(target2)
for filename in os.listdir(pwd1):
  shutil.copyfile(pwd1+'/'+filename,target2+'/'+filename)
for filename in os.listdir(pwd2):
  shutil.copyfile(pwd2+'/'+filename,target2+'/'+filename)
```

[85]:
```
bw_files = natsorted(glob.glob('/content/datasets/folder2/*_bw.jpg'))
svd_files = natsorted(glob.glob('/content/datasets/folder2/*_svd.jpg'))
```

[86]:
```
bw_images = [load_img(v) for v in bw_files]
svd_images = [load_img(v) for v in svd_files]
```

[89]:
```
print(bw_files[0])
print(svd_files[0])
```

```
/content/datasets/folder2/cat1_bw.jpg
/content/datasets/folder2/cat1_svd.jpg
```

[90]:
```
print_arr_metrics(svd_images, bw_images)
```

```
PSNR: 28.83465742408829
SSIM: 0.7838102175738757
```

Création du modèle des images dégradées par filrage

[ ]:
```
batch_size = 3
img_height = 200
img_width = 200

parent_dir1 = '/content/datasets/Compressed_data_gauss_filter'
train_data01 = tf.keras.preprocessing.image_dataset_from_directory(
  parent_dir1,
  validation_split=0.2,
  subset="training",
  seed=42,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_data01 = tf.keras.preprocessing.image_dataset_from_directory(
  parent_dir1,
  validation_split=0.2,
  subset="validation",
  seed=42,
  image_size=(img_height, img_width),
  batch_size=batch_size)

class_names1 = val_data01.class_names
print(class_names1)
```

[ ]:
```
from tensorflow.keras import layers
num_classes = 2

model1 = tf.keras.Sequential([
    layers.experimental.preprocessing.Rescaling(1./255),
    layers.Conv2D(128,4, activation='relu'),
```

```
    layers.MaxPooling2D(),
    layers.Conv2D(64,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(16,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

model1.compile(optimizer='adam', loss=tf.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'],)

logdir="logs"

tensorboard_callback1 = keras.callbacks.
 ↪TensorBoard(log_dir=logdir,histogram_freq=1, write_images=logdir,␣
 ↪embeddings_data=train_data02)

history1 = model1.fit(train_data01, validation_data=val_data01, epochs=25,␣
 ↪callbacks=[tensorboard_callback1])
```

Adaptative filter

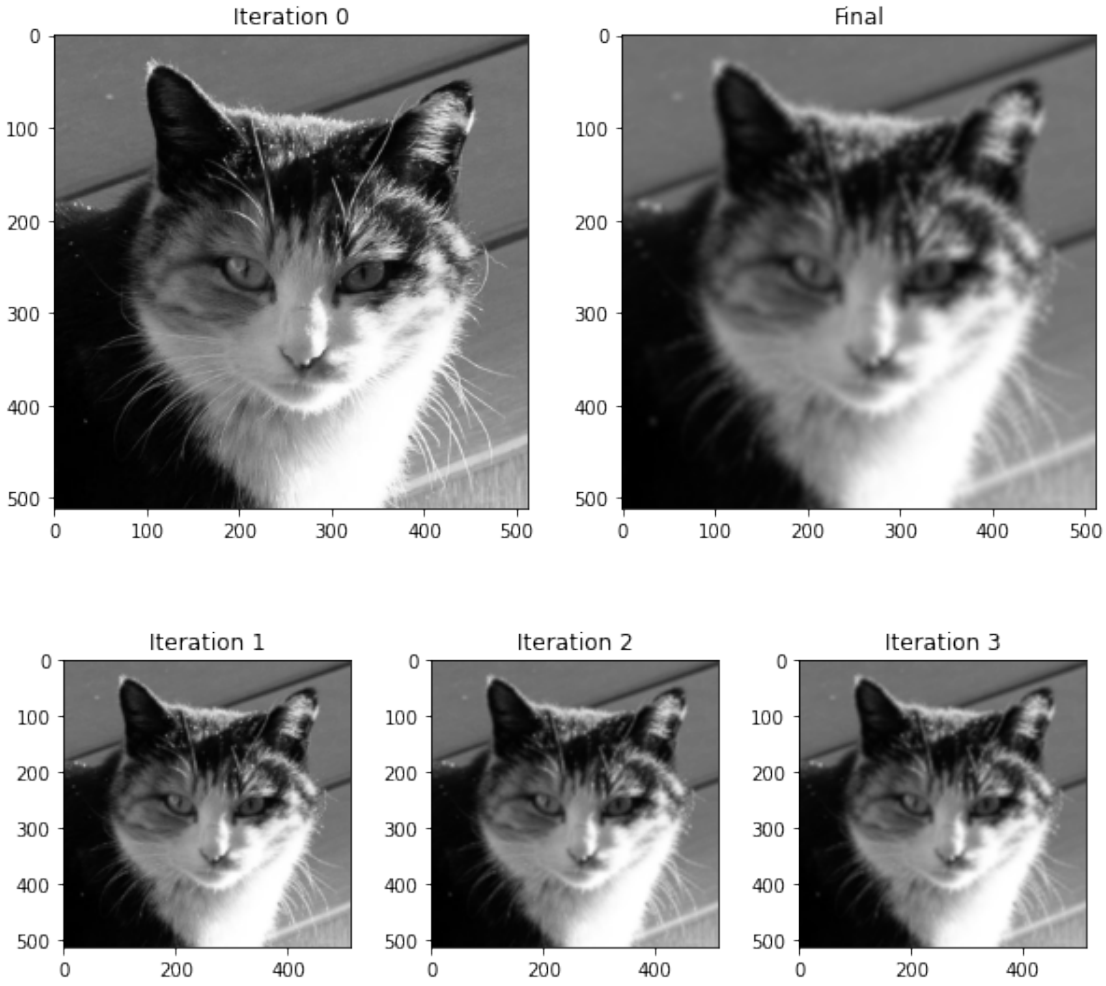
[ ]:
```
grad_operators = {
    'prewitt':(
        np.array([[-1,0,1],
                  [-1,0,2],
                  [-1,0,1]]),
        np.array([[1,1,1],
                  [0,0,0],
                  [-1,-1,-1]])
    ),
    'robertcross':(
        np.array([[1,0],
                  [0,1]]),
        np.array([[0,-1],
                  [-1,0]])
    ),
    'sobel':(
        np.array([[-1,0,1],
                  [-2,0,2],
                  [-1,0,1]]),
        np.array([[1,2,1],
                  [0,0,0],
```

```
                    [-1,-2,-1]])
    )
}

# Distance euclidienne
def euclidean(Gx, Gy):
    return np.sqrt((Gx**2)+(Gy**2))
```

[ ]:
```
def adaptive_weight(Gx, Gy, h):
    return np.exp(np.sqrt(euclidean(Gx, Gy) / (2 * (h ** 2))))

def adaptive_convolution_2d(im, weight):
    w = 3
    m,n = im.shape
    im_all_subw = get_all_window(im, w)
    weight_all_subw = get_all_window(weight, w)
    X = np.sum(np.sum(im_all_subw * weight_all_subw, 1), 1) / np.sum(np.
 ↪sum(weight_all_subw, 1), 1)
    return X.reshape(m,n)

def adaptive_filter(im, n=5, h=1.5, operator='sobel'):
    # 1. K = 1, fixer l'itération n et le coefficient de l'amplitude du edge h.
    #K = 1
    #h = 1.5
    weights = []
    iteration_result = [im]

    # loop
    for i in range(n):
        # 2. Calcul du gradient Gx et Gy
        op = grad_operators[operator]
        Gx = convolution_2d(im,op[0])
        Gy = convolution_2d(im,op[1])

        # 3. Calcul du poids
        weight = adaptive_weight(Gx, Gy, h)
        weights.append(weight)

        # 4. Convolution
        im = adaptive_convolution_2d(im, weight)
        iteration_result.append(im)

    return im, weights, iteration_result
```

[ ]:
```
N =5
im_adaptive, adaptive_weights, adaptive_iteration_result =␣
 ↪adaptive_filter(imgrey, N)
```

```
f = plt.figure(figsize=(10,13))
f.add_subplot(221,title='Iteration 0').imshow(imgrey, cmap='gray')
f.add_subplot(222,title='Final').imshow(im_adaptive, cmap='gray')
for i in range(1, N-1):
    f.add_subplot(4,N-2,i+(N-2)*2,title='Iteration ' + str(i)).
 ↪imshow(adaptive_iteration_result[i], cmap='gray')
```



[ ]:
```
!rm -rf /content/datasets/Compressed_data
#os.mkdir('/content/datasets/Compressed_data_adapted_filter')
#os.mkdir('/content/datasets/Compressed_data_adapted_filter/cat')
#os.mkdir('/content/datasets/Compressed_data_adapted_filter/dog')
parent_dir2 = '/content/datasets/Compressed_data_adapted_filter'
cat_dir = '/content/datasets/Compressed_data_adapted_filter/cat'
dog_dir = '/content/datasets/Compressed_data_adapted_filter/dog'
```

[ ]:
```
import os
data_loc1 = '/content/datasets/datasets/cat'
c=0
for filename in os.listdir(data_loc1):
    if filename.endswith(".jpg"):
        im = mpimg.imread(data_loc1+'/'+filename)
        imgrey = np.round(0.3 * im[:,:,0] + 0.59 * im[:,:,1] + 0.11 * im[:,:
 ↪,2]).astype(int)
        name='imgcat' +str(c)+'.jpg'
        im_adaptive, adaptive_weights, adaptive_iteration_result =␣
 ↪adaptive_filter(imgrey, N)
        img_arra = Image.fromarray(im_adaptive)
        rgb_im = img_arra.convert('RGB')
        rgb_im.save(cat_dir+'/'+name, 'JPEG')
        c=c+1
    else:
         continue
```

[ ]:
```
import os
data_loc2 = '/content/datasets/datasets/dog'
c=0
for filename in os.listdir(data_loc2):
    if filename.endswith(".jpg"):
        im = mpimg.imread(data_loc2+'/'+filename)
        imgrey = np.round(0.3 * im[:,:,0] + 0.59 * im[:,:,1] + 0.11 * im[:,:
 ↪,2]).astype(int)
        name='imgdog' +str(c)+'.jpg'
        im_adaptive, adaptive_weights, adaptive_iteration_result =␣
 ↪adaptive_filter(imgrey, N)
        img_arra = Image.fromarray(im_adaptive)
        rgb_im = img_arra.convert('RGB')
        rgb_im.save(dog_dir+'/'+name, 'JPEG')
        c=c+1
    else:
         continue
```

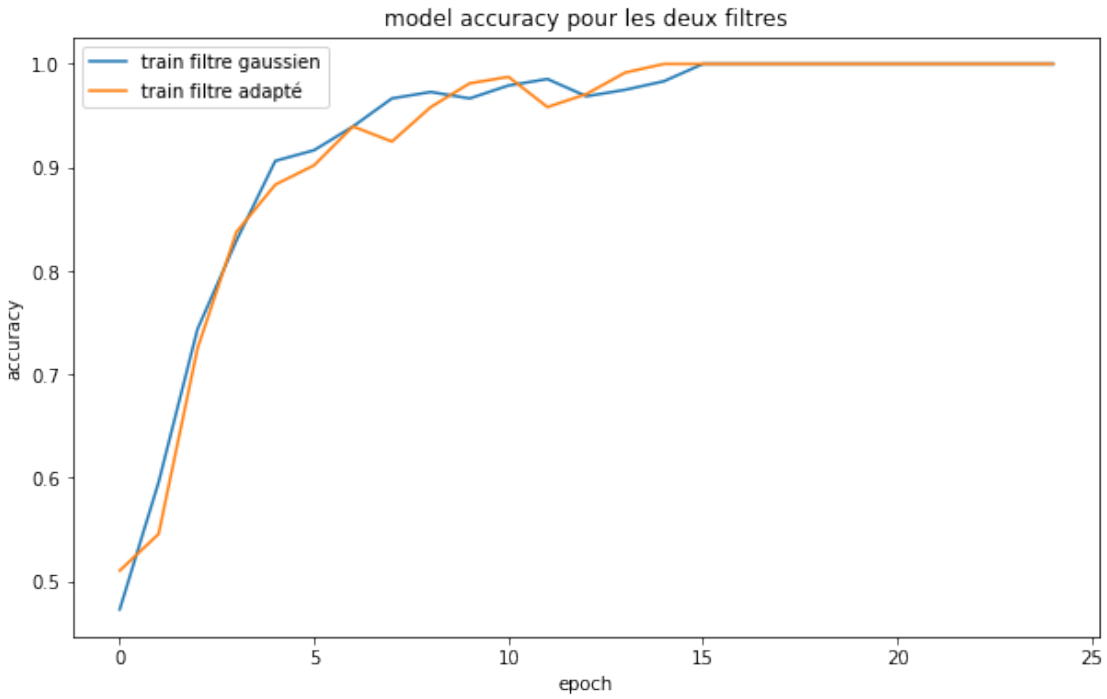
[ ]:
```
batch_size = 3
img_height = 200
img_width = 200

train_data02 = tf.keras.preprocessing.image_dataset_from_directory(
  parent_dir2,
  validation_split=0.2,
  subset="training",
  seed=42,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
val_data02 = tf.keras.preprocessing.image_dataset_from_directory(
  parent_dir2,
  validation_split=0.2,
  subset="validation",
  seed=42,
  image_size=(img_height, img_width),
  batch_size=batch_size)

class_names2 = val_data02.class_names
print(class_names2)
```

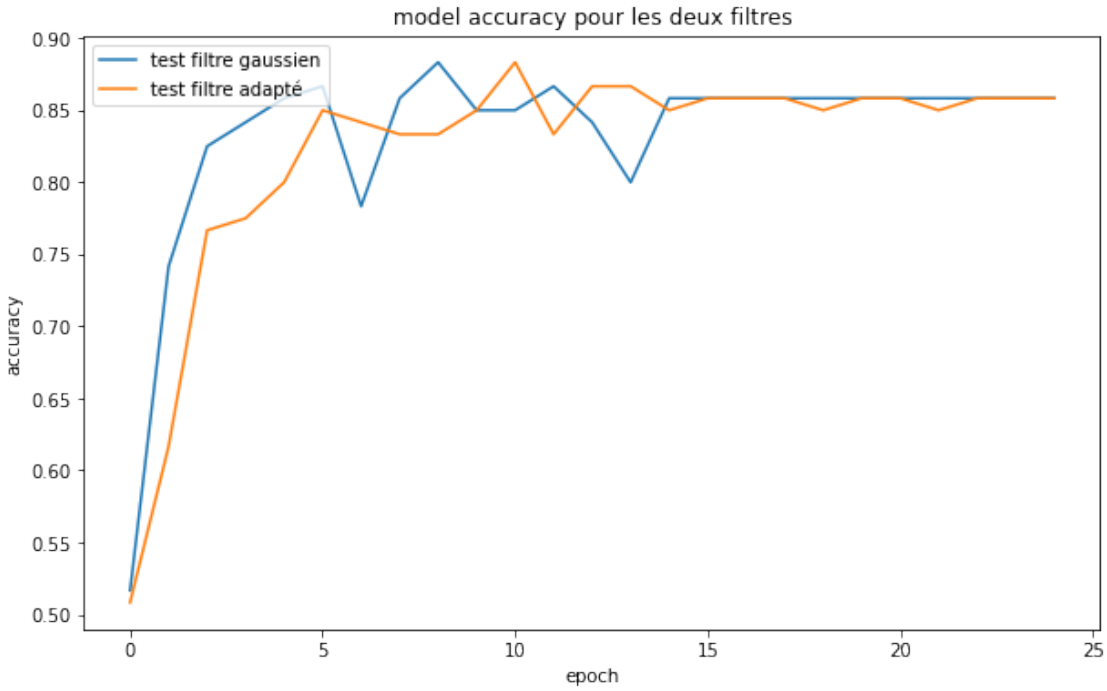
[ ]:
```
from tensorflow.keras import layers

num_classes = 2

model2 = tf.keras.Sequential([
    layers.experimental.preprocessing.Rescaling(1./255),
    layers.Conv2D(128,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(16,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

model2.compile(optimizer='adam', loss=tf.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'],)

logdir="logs"

tensorboard_callback2 = keras.callbacks.
 ↪TensorBoard(log_dir=logdir,histogram_freq=1, write_images=logdir,␣
 ↪embeddings_data=train_data01)

history2 = model2.fit(train_data02, validation_data=val_data02, epochs=25,␣
 ↪callbacks=[tensorboard_callback2])
```

[ ]:
```
figure(figsize=(10,6))
plt.plot(history1.history['accuracy'])
plt.plot(history2.history['accuracy'])
plt.title('model accuracy pour les deux filtres')
```

```
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train filtre gaussien', 'train filtre adapté'], loc='upper left')
plt.show()
```



[ ]:
```
figure(figsize=(10,6))
plt.plot(history1.history['val_accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('model accuracy pour les deux filtres')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['test filtre gaussien', 'test filtre adapté'], loc='upper left')
plt.show()
```


model accuracy pour les deux filtres
test filtre gaussien
test filtre adapté
accuracy
epoch
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
5
10
15
20
25