CNN Avec peu de données

Chargement du code

In [ ]:

```
%load_ext autoreload
%autoreload 2
from TP3_utils import load_dataset
import numpy as np
from matplotlib import pyplot as plt

print("un")
DATA_PATH = 'Data'
CATEGORIES = ["accordion", "anchor", "barrel", "binocular"]
print("deux")
# Chargement du dataset
(X_train, Y_train), (X_test, Y_test), num_classes = load_dataset(DA
print("trois")

# Vérification rapide
data_shape = X_train[0].shape
print("Training data shape:", data_shape)
print("Training labels shape:", Y_train.shape)
print("Test data shape:", X_test.shape)
print("Test labels shape:", Y_test.shape)
print("Number of classes:", num_classes)
```

Redimensionnement sur les images de X_train

pour y_train on passe du label brut au vecteur one-hot (classification multiclass)

In [ ]:

```
import time
import tensorflow as tf
from tensorflow import keras
from keras.optimizers import Adam
from TP3_utils import CNN, affiche, eval_classif
# from keras.callbacks import ReduceLROnPlateau, EarlyStopping

# lr_schedulerBis = ReduceLROnPlateau(monitor='val_loss', factor=0..
# earlStopBis = EarlyStopping(monitor='val_loss', min_delta=1e-3, p

lr=1e-4
batch_size=min([X_train.shape[0], 256])
epochs=16
ad= Adam(learning_rate=lr)

model = CNN(data_shape)
model.summary()

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

tps1 = time.time()
history =model.fit(
    X_train,
    Y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(X_test, Y_test)
#     callbacks=[earlStopBis, lr_schedulerBis],
)
tps2 = time.time()

# Evaluation
loss, accuracy = model.evaluate(X_test, Y_test)
print(f'Test loss: {loss}, Test accuracy: {accuracy}')

affiche(history)
preds = model.predict(X_test)
eval_classif(Y_test, preds)
print("Temps d'entraînement : {:.2f} secondes".format(tps2 - tps1))
```

Premier essaie:

- Clairement overfitting: loss value stagne vite, l'accuracy est de 1 en train, mais .85 en test

In [ ]:

```
from keras.layers import Input, Flatten, Dense
from TP3_utils import MLP_transfer

input_tensor = Input(shape=(224,224,3))
VGG = tf.keras.applications.VGG16(weights='imagenet',
    include_top=True,
    input_tensor=input_tensor
)
VGG.summary()

feature_train = VGG.predict(X_train)
feature_test = VGG.predict(X_test)
```

In [ ]:

```
# Transfer learning
input_tensor = Input(shape=(224,224,3))
VGG = tf.keras.applications.VGG16(weights='imagenet',
    include_top=False,
    input_tensor=input_tensor
)
for layer in VGG.layers:
    layer.trainable = False

model_transfer = keras.Sequential()
model_transfer.add(VGG)
model_transfer.add(MLP_transfer(VGG.output_shape[1:], num_classes))
model_transfer.summary()
model_transfer.compile(
    loss='categorical_crossentropy',
    optimizer=ad,
    metrics=['accuracy']
)

tps1 = time.time()
history =model_transfer.fit(
    X_train,
    Y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(X_test, Y_test)
#     callbacks=[earlStopBis, lr_schedulerBis],
)
tps2 = time.time()
loss, accuracy = model_transfer.evaluate(X_test, Y_test)
print(f'Test loss: {loss}, Test accuracy: {accuracy}')

affiche(history)
preds = model_transfer.predict(X_test)
eval_classif(Y_test, preds)
print("Temps d'entraînement : {:.2f} secondes".format(tps2 - tps1))
```

In [ ]:

```
# Fine-tuning
```

In [ ]:

```
from tensorflow.keras.datasets.mnist import load_data

(X_train, y_train), (X_test, y_test) = load_data()
# Preprocess input data
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_tra
X_test = X_test.reshape(X_test.shape[0], X_train.shape[1], X_train.
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

X_train_noise = X_train + 0.2 * np.random.normal(loc=0.0, scale=1.0
X_test_noise = X_test + 0.4 * np.random.normal(loc=0.0, scale=1.0,

X_train_noise = np.clip(X_train_noise, 0.0, 1.0)
X_test_noise = np.clip(X_test_noise, 0.0, 1.0)
# Display the train data and a version of it with added noise
for i in range(5):
    plt.subplot(2,5,i+1)
    plt.imshow(X_train[i,:].reshape([28,28]), cmap='gray')
    plt.axis('off')
    plt.subplot(2,5,i+6)
    plt.imshow(X_train_noise[i,:].reshape([28,28]), cmap='gray')
    plt.axis('off')
    plt.show()
```

In [ ]:

```
from TP3_utils import auto_encoder
```

```
model_ae = auto_encoder(data_shape)

model_ae.compile(
    loss="mean_square_error",
    optimizer=ad,
    metrics=['accuracy']
)

history =model_ae.fit(
    X_train,
    Y_train,
    batch_size=32,
    epochs=20,
    verbose=1,
    validation_data=(X_test, Y_test)
#     callbacks=[earlStopBis, lr_schedulerBis],
)

loss, accuracy = model_ae.evaluate(X_test, Y_test)
print(f'Test loss: {loss}, Test accuracy: {accuracy}')

affiche(history)
preds = model_ae.predict(X_test)
eval_classif(Y_test, preds)
print("Temps d'entraînement : {:.2f} secondes".format(tps2 - tps1))
```