tif1cjb1t

March 30, 2025

[1]:
```
#!pip install kagglehub
#!pip install tensorflow
```

[2]:
```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("volodymyrpivoshenko/
 ↪brain-mri-scan-images-tumor-detection")

print("Path to dataset files:", path)
```

```
Path to dataset files:
/Users/markus/.cache/kagglehub/datasets/volodymyrpivoshenko/brain-mri-scan-
images-tumor-detection/versions/1
```

[3]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
from skimage import io, transform
from skimage.feature import hog
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
```

```
2025-03-30 21:15:35.874916: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

[4]:
```
# Chemin vers le dataset
dataset_root = "/Users/markus/.cache/kagglehub/datasets/volodymyrpivoshenko/
 ↪brain-mri-scan-images-tumor-detection/versions/1/brain_mri_scan_images"
```

```python
# Classes
classes = ["negative", "positive"]
```

[5]:
```python
# Charger et prétraiter les images
X = []
y = []
filenames = []

for class_name in classes:
    class_path = os.path.join(dataset_root, class_name)
    for filename in tqdm(os.listdir(class_path), desc=f"Loading {class_name}"):
        try:
            img_path = os.path.join(class_path, filename)
            img = io.imread(img_path, as_gray=True)

            img = transform.resize(img, (64, 64), mode='reflect',␣
↪anti_aliasing=True)

            img = img / 255.0 # Normalisation [0-1]

            # Ajout de la dimension de canal pour CNN (1 pour niveaux de gris)
            img = img.reshape(64, 64, 1)

            X.append(img)
            y.append(0 if class_name == 'negative' else 1) # 0 = Pas de tumeur,␣
↪1 = Tumeur
            filenames.append(filename)
        except Exception as e:
            print(f"Error loading {img_path}: {e}")
```

```
Loading negative: 100%|              | 98/98 [00:03<00:00, 28.05it/s]
Loading positive: 100%|              | 129/129 [00:03<00:00, 33.98it/s]
```

[6]:
```python
# Conversion numpy
X = np.array(X)
y = np.array(y)
filenames = np.array(filenames)
```

[7]:
```python
# Éliminer les doublons
# Train-Test split

unique_indices = np.unique(filenames, return_index=True)[1]
X = X[unique_indices]
y = y[unique_indices]
filenames = filenames[unique_indices]
```

```
X_train, X_test, y_train, y_test, filenames_train, filenames_test =␣
↪train_test_split(X, y, filenames, test_size=0.2, stratify=y)
```

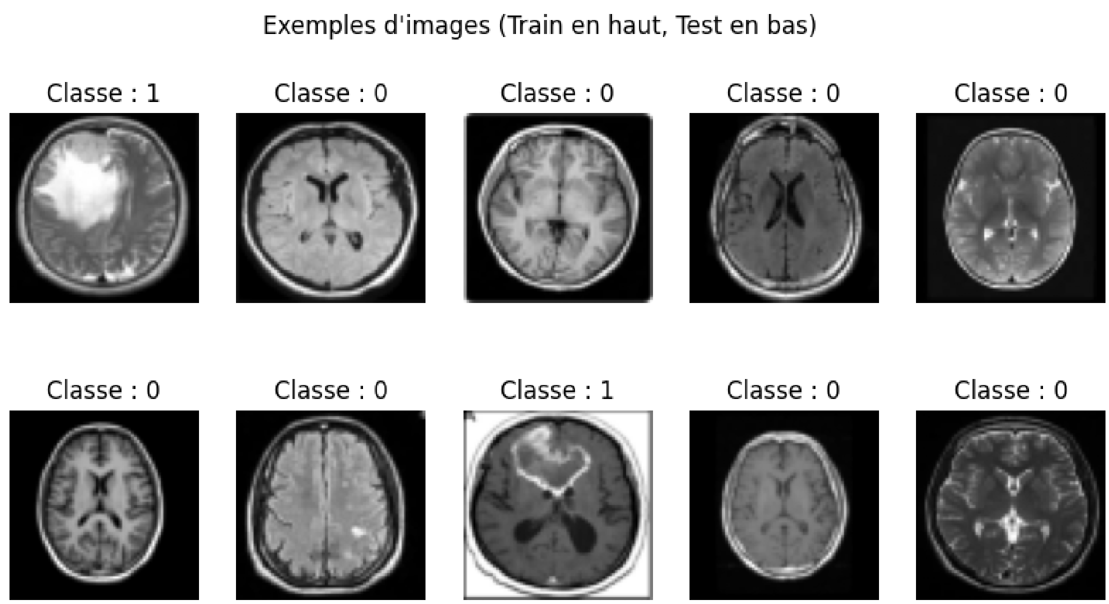
[8]:
```
# Vérification de la contamination des données de test
overlap = set(filenames_train) & set(filenames_test)
if overlap:
    print(f"{len(overlap)} fichiers sont présents dans les deux ensembles !")
else:
    print("Aucune observation partagée entre les ensembles d'entraînement et de␣
↪test.")

# Vérification de l'équilibre des classes
print("\nDistribution des classes dans l'ensemble d'entraînement :", np.
↪bincount(y_train))
print("Distribution des classes dans l'ensemble de test :", np.bincount(y_test))

# Vérification visuelle du prétraitement
plt.figure(figsize=(10, 5))
for i in range(5):
    plt.subplot(2, 5, i+1)
    plt.imshow(X_train[i].reshape(64, 64), cmap='gray')
    plt.title(f"Classe : {y_train[i]}")
    plt.axis('off')
for i in range(5):
    plt.subplot(2, 5, i+6)
    plt.imshow(X_test[i].reshape(64, 64), cmap='gray')
    plt.title(f"Classe : {y_test[i]}")
    plt.axis('off')
plt.suptitle("Exemples d'images (Train en haut, Test en bas)")
plt.show()
```

```
Aucune observation partagée entre les ensembles d'entraînement et de test.

Distribution des classes dans l'ensemble d'entraînement : [78 25]
Distribution des classes dans l'ensemble de test : [20  6]
```



[9]:

```python
# Construction du modèle CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

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

```
/Users/markus/miniconda3/lib/python3.12/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[10]:

```python
#Rééquilibrage des classes
from sklearn.utils.class_weight import compute_class_weight
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weight_dict = {i: weight for i, weight in enumerate(class_weights)}
```

```python
# Entraînement
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_data=(X_test, y_test),
    class_weight=class_weight_dict,
    verbose=0
)
```

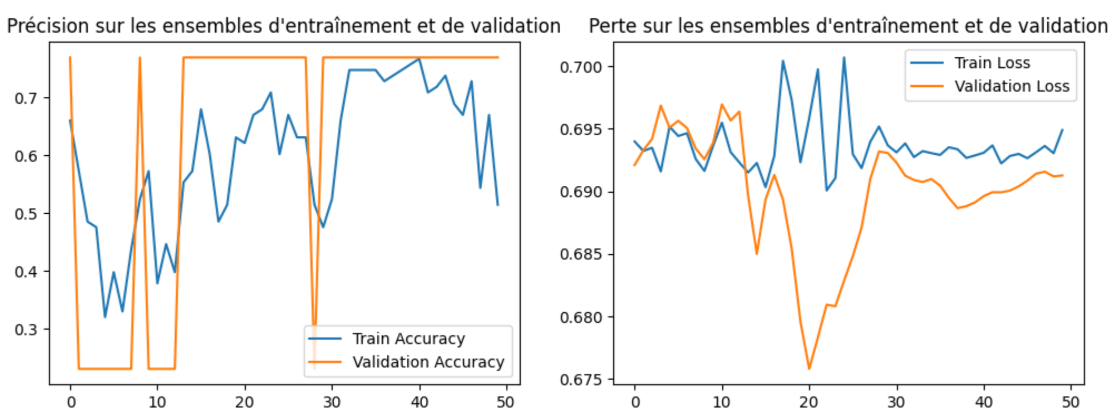
[11]:
```python
# Évaluation
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
1/1              0s 71ms/step -
accuracy: 0.7692 - loss: 0.6913
Test Accuracy: 76.92%
```

[12]:
```python
# Courbes d'apprentissage

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Précision sur les ensembles d\'entraînement et de validation')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Perte sur les ensembles d\'entraînement et de validation')
plt.show()
```



[13]:
```python
# Prédiction des probabilités pour l'ensemble de test
y_pred_proba = model.predict(X_test)
y_pred = (y_pred_proba > 0.5).astype(int)

from sklearn.metrics import precision_score, recall_score, f1_score,␣
↪roc_auc_score, confusion_matrix, roc_curve, auc

# Calcul des métriques
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred_proba)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
print(f"AUC: {auc_score:.2f}")
```

```
1/1            0s 117ms/step
Precision: 0.00
Recall: 0.00
F1-Score: 0.00
AUC: 0.50
```

```
/Users/markus/miniconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[14]:
```python
# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[20  0]
 [ 6  0]]
```

[15]:
```python
# Modèle biasé vers la classe négative
```

[16]:
```python
from imblearn.over_sampling import SMOTE

# Application de SMOTE sur les données d'entraînement
smote = SMOTE()
```

```
X_train_resampled, y_train_resampled = smote.fit_resample(X_train.reshape(-1,␣
 ↪64*64), y_train)
X_train_resampled = X_train_resampled.reshape(-1, 64, 64, 1)
```

[17]:
```
# Vérification de la répartition des classes après SMOTE
print("Distribution des classes après SMOTE :", np.bincount(y_train_resampled))
```

```
Distribution des classes après SMOTE : [78 78]
```

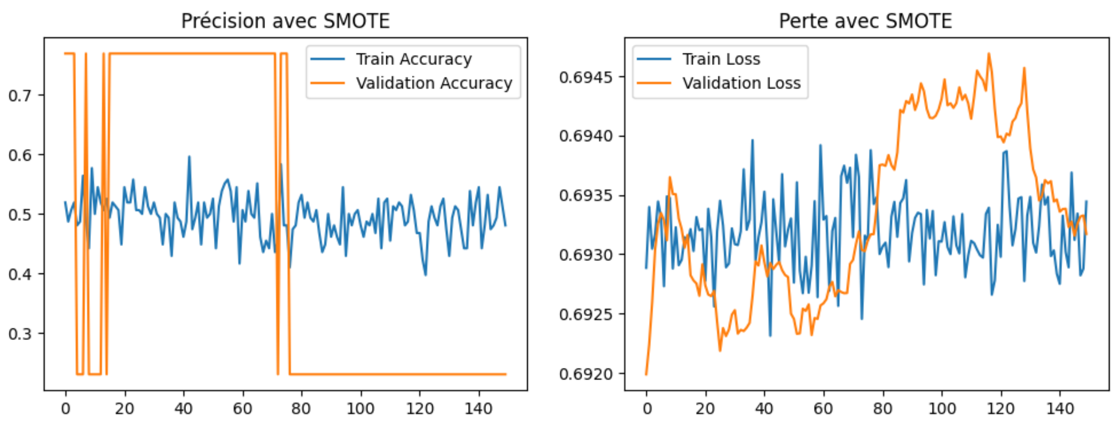
[18]:
```
# Entraînement avec SMOTE
history_smote = model.fit(
    X_train_resampled, y_train_resampled,
    epochs=150,
    batch_size=32,
    validation_data=(X_test, y_test),
    verbose=0
)
```

[19]:
```
# Évaluation avec SMOTE
loss_smote, accuracy_smote = model.evaluate(X_test, y_test)
print(f"Test Accuracy with SMOTE: {accuracy_smote * 100:.2f}%")
```

```
1/1            0s 78ms/step -
accuracy: 0.2308 - loss: 0.6932
Test Accuracy with SMOTE: 23.08%
```

[20]:
```
# Courbes d'apprentissage avec SMOTE
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_smote.history['accuracy'], label='Train Accuracy')
plt.plot(history_smote.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Précision avec SMOTE')

plt.subplot(1, 2, 2)
plt.plot(history_smote.history['loss'], label='Train Loss')
plt.plot(history_smote.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Perte avec SMOTE')
plt.show()
```



[21]:
```
# Prédiction des probabilités pour l'ensemble de test avec SMOTE
y_pred_proba_smote = model.predict(X_test)
y_pred_smote = (y_pred_proba_smote > 0.5).astype(int)

# Calcul des métriques avec SMOTE
precision_smote = precision_score(y_test, y_pred_smote)
recall_smote = recall_score(y_test, y_pred_smote)
f1_smote = f1_score(y_test, y_pred_smote)
auc_score_smote = roc_auc_score(y_test, y_pred_proba_smote)

print(f"Precision with SMOTE: {precision_smote:.2f}")
print(f"Recall with SMOTE: {recall_smote:.2f}")
print(f"F1-Score with SMOTE: {f1_smote:.2f}")
print(f"AUC with SMOTE: {auc_score_smote:.2f}")
```

```
1/1              0s 67ms/step
Precision with SMOTE: 0.23
Recall with SMOTE: 1.00
F1-Score with SMOTE: 0.38
AUC with SMOTE: 0.50
```

[22]:
```
# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred_smote)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[ 0 20]
 [ 0  6]]
```

[23]:
```
#Modèle biaisé vers la classe positive
```

Solutions :

1. Vérification des Données
2. Augmenter la Complexité du Modèle
3. Augmentation des Données
4. Entraînement Plus Long