eegmxkcd5

March 15, 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
```

```
2025-03-05 13:33:15.181669: 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"
```

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

[5]:
```
# 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 [00:02<00:00, 48.20it/s]
Loading positive: 100%|             | 129/129 [00:02<00:00, 48.38it/s]
```

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

[7]:
```
# 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]
```

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

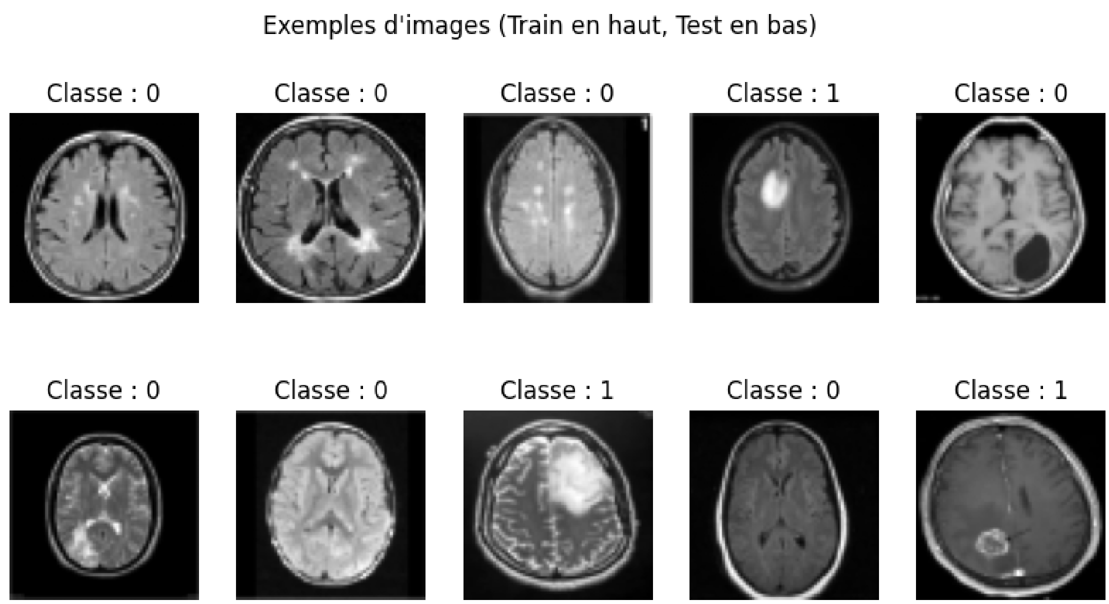
[8]:
```python
# Vérification de la contamination des données de test
overlap = set(filenames_train) & set(filenames_test)
if overlap:
    print(f"  Erreur : {len(overlap)} fichiers sont présents dans les deux ensembles !")
else:
    print("  Aucune contamination détectée entre les ensembles d'entraînement 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 contamination détectée 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]:
```
# 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]:
```
#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=1
)
```

```
Epoch 1/50
4/4             2s 185ms/step -
accuracy: 0.2596 - loss: 0.7332 - val_accuracy: 0.7692 - val_loss: 0.6912
Epoch 2/50
4/4             1s 120ms/step -
accuracy: 0.6511 - loss: 0.7010 - val_accuracy: 0.7692 - val_loss: 0.6886
Epoch 3/50
4/4             0s 115ms/step -
accuracy: 0.6365 - loss: 0.7190 - val_accuracy: 0.7692 - val_loss: 0.6894
Epoch 4/50
4/4             1s 118ms/step -
accuracy: 0.5464 - loss: 0.7110 - val_accuracy: 0.7692 - val_loss: 0.6928
Epoch 5/50
4/4             0s 114ms/step -
accuracy: 0.5215 - loss: 0.6999 - val_accuracy: 0.2308 - val_loss: 0.6958
Epoch 6/50
4/4             0s 114ms/step -
accuracy: 0.3396 - loss: 0.6970 - val_accuracy: 0.2308 - val_loss: 0.6957
Epoch 7/50
4/4             1s 133ms/step -
accuracy: 0.3469 - loss: 0.7034 - val_accuracy: 0.2308 - val_loss: 0.6945
Epoch 8/50
4/4             1s 118ms/step -
accuracy: 0.3195 - loss: 0.6871 - val_accuracy: 0.2308 - val_loss: 0.6936
Epoch 9/50
4/4             0s 113ms/step -
accuracy: 0.4228 - loss: 0.7043 - val_accuracy: 0.2308 - val_loss: 0.6939
Epoch 10/50
4/4             1s 155ms/step -
accuracy: 0.3167 - loss: 0.6872 - val_accuracy: 0.2308 - val_loss: 0.6943
Epoch 11/50
4/4             1s 135ms/step -
accuracy: 0.2483 - loss: 0.6723 - val_accuracy: 0.2308 - val_loss: 0.6947
Epoch 12/50
4/4             1s 129ms/step -
accuracy: 0.2513 - loss: 0.6862 - val_accuracy: 0.2308 - val_loss: 0.6954
```

```
Epoch 13/50
4/4           1s 133ms/step -
accuracy: 0.2651 - loss: 0.7230 - val_accuracy: 0.2308 - val_loss: 0.6960
Epoch 14/50
4/4           0s 111ms/step -
accuracy: 0.2395 - loss: 0.6841 - val_accuracy: 0.2308 - val_loss: 0.6957
Epoch 15/50
4/4           1s 138ms/step -
accuracy: 0.2528 - loss: 0.6824 - val_accuracy: 0.2308 - val_loss: 0.6959
Epoch 16/50
4/4           1s 131ms/step -
accuracy: 0.2471 - loss: 0.6910 - val_accuracy: 0.2308 - val_loss: 0.6958
Epoch 17/50
4/4           0s 122ms/step -
accuracy: 0.2315 - loss: 0.6886 - val_accuracy: 0.2308 - val_loss: 0.6956
Epoch 18/50
4/4           1s 158ms/step -
accuracy: 0.2809 - loss: 0.7066 - val_accuracy: 0.2308 - val_loss: 0.6961
Epoch 19/50
4/4           1s 143ms/step -
accuracy: 0.2905 - loss: 0.7014 - val_accuracy: 0.2308 - val_loss: 0.6961
Epoch 20/50
4/4           0s 115ms/step -
accuracy: 0.2575 - loss: 0.7120 - val_accuracy: 0.2308 - val_loss: 0.6962
Epoch 21/50
4/4           0s 115ms/step -
accuracy: 0.2353 - loss: 0.6756 - val_accuracy: 0.2308 - val_loss: 0.6959
Epoch 22/50
4/4           0s 114ms/step -
accuracy: 0.2975 - loss: 0.6870 - val_accuracy: 0.2308 - val_loss: 0.6962
Epoch 23/50
4/4           0s 113ms/step -
accuracy: 0.2851 - loss: 0.7130 - val_accuracy: 0.2308 - val_loss: 0.6970
Epoch 24/50
4/4           0s 116ms/step -
accuracy: 0.2697 - loss: 0.7192 - val_accuracy: 0.2308 - val_loss: 0.6977
Epoch 25/50
4/4           1s 126ms/step -
accuracy: 0.2398 - loss: 0.6906 - val_accuracy: 0.2308 - val_loss: 0.6980
Epoch 26/50
4/4           1s 112ms/step -
accuracy: 0.2710 - loss: 0.7198 - val_accuracy: 0.2308 - val_loss: 0.6976
Epoch 27/50
4/4           1s 158ms/step -
accuracy: 0.2601 - loss: 0.6867 - val_accuracy: 0.2308 - val_loss: 0.6969
Epoch 28/50
4/4           1s 112ms/step -
accuracy: 0.2620 - loss: 0.7208 - val_accuracy: 0.2308 - val_loss: 0.6965
```

```
Epoch 29/50
4/4            1s 112ms/step -
accuracy: 0.2403 - loss: 0.6693 - val_accuracy: 0.2308 - val_loss: 0.6958
Epoch 30/50
4/4            1s 129ms/step -
accuracy: 0.2855 - loss: 0.6871 - val_accuracy: 0.2308 - val_loss: 0.6957
Epoch 31/50
4/4            0s 116ms/step -
accuracy: 0.3130 - loss: 0.6776 - val_accuracy: 0.2308 - val_loss: 0.6952
Epoch 32/50
4/4            0s 114ms/step -
accuracy: 0.3690 - loss: 0.6983 - val_accuracy: 0.2308 - val_loss: 0.6948
Epoch 33/50
4/4            0s 113ms/step -
accuracy: 0.4341 - loss: 0.7207 - val_accuracy: 0.2308 - val_loss: 0.6942
Epoch 34/50
4/4            1s 129ms/step -
accuracy: 0.4604 - loss: 0.6657 - val_accuracy: 0.2308 - val_loss: 0.6935
Epoch 35/50
4/4            0s 118ms/step -
accuracy: 0.5402 - loss: 0.6932 - val_accuracy: 0.2308 - val_loss: 0.6935
Epoch 36/50
4/4            0s 114ms/step -
accuracy: 0.4897 - loss: 0.6860 - val_accuracy: 0.2308 - val_loss: 0.6938
Epoch 37/50
4/4            0s 119ms/step -
accuracy: 0.4703 - loss: 0.7130 - val_accuracy: 0.2308 - val_loss: 0.6941
Epoch 38/50
4/4            0s 117ms/step -
accuracy: 0.4087 - loss: 0.6863 - val_accuracy: 0.2308 - val_loss: 0.6943
Epoch 39/50
4/4            0s 121ms/step -
accuracy: 0.5177 - loss: 0.6924 - val_accuracy: 0.2308 - val_loss: 0.6943
Epoch 40/50
4/4            0s 118ms/step -
accuracy: 0.4802 - loss: 0.6943 - val_accuracy: 0.2308 - val_loss: 0.6940
Epoch 41/50
4/4            0s 113ms/step -
accuracy: 0.4687 - loss: 0.7118 - val_accuracy: 0.2308 - val_loss: 0.6938
Epoch 42/50
4/4            0s 114ms/step -
accuracy: 0.4118 - loss: 0.7093 - val_accuracy: 0.2308 - val_loss: 0.6935
Epoch 43/50
4/4            0s 121ms/step -
accuracy: 0.5011 - loss: 0.7061 - val_accuracy: 0.7692 - val_loss: 0.6929
Epoch 44/50
4/4            0s 112ms/step -
accuracy: 0.4388 - loss: 0.6940 - val_accuracy: 0.7692 - val_loss: 0.6925
```

```
Epoch 45/50
4/4             0s 117ms/step -
accuracy: 0.5246 - loss: 0.6961 - val_accuracy: 0.7692 - val_loss: 0.6925
Epoch 46/50
4/4             0s 114ms/step -
accuracy: 0.5651 - loss: 0.6826 - val_accuracy: 0.7692 - val_loss: 0.6925
Epoch 47/50
4/4             1s 115ms/step -
accuracy: 0.5856 - loss: 0.6684 - val_accuracy: 0.7692 - val_loss: 0.6927
Epoch 48/50
4/4             0s 114ms/step -
accuracy: 0.5298 - loss: 0.6740 - val_accuracy: 0.7692 - val_loss: 0.6931
Epoch 49/50
4/4             0s 113ms/step -
accuracy: 0.5875 - loss: 0.7159 - val_accuracy: 0.2308 - val_loss: 0.6935
Epoch 50/50
4/4             0s 118ms/step -
accuracy: 0.3765 - loss: 0.6630 - val_accuracy: 0.2308 - val_loss: 0.6937
```

[11]:

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

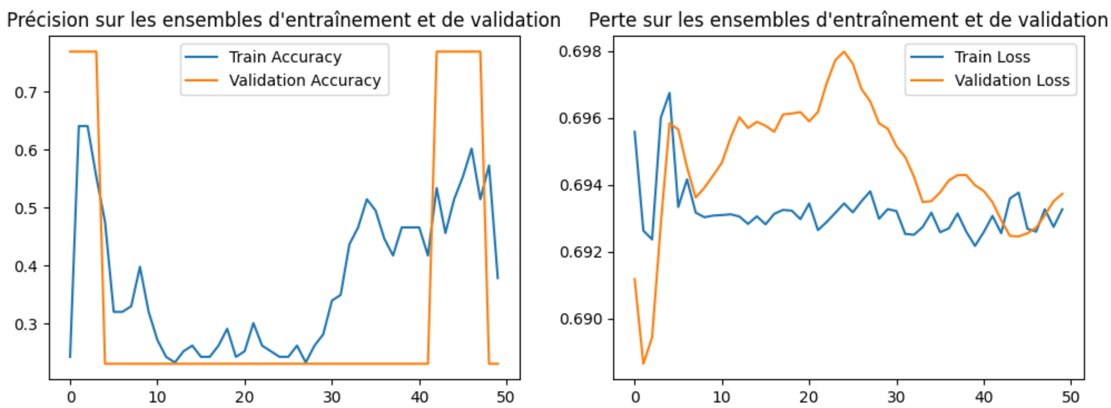
```
1/1             0s 74ms/step -
accuracy: 0.2308 - loss: 0.6937
Test Accuracy: 23.08%
```

[12]:

```
# 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}")
```

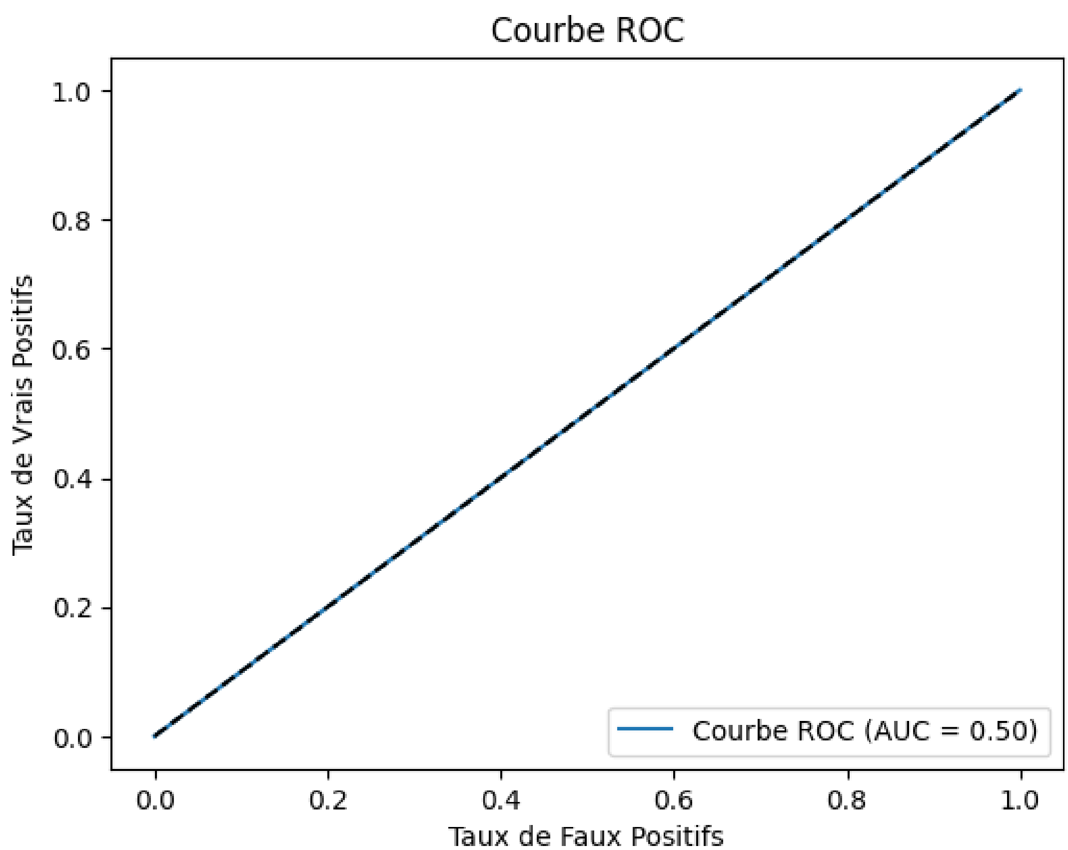
```
1/1              0s 122ms/step
Precision: 0.23
Recall: 1.00
F1-Score: 0.38
AUC: 0.50
```

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

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

[15]:
```python
# Courbe ROC
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f'Courbe ROC (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('Taux de Faux Positifs')
plt.ylabel('Taux de Vrais Positifs')
plt.title('Courbe ROC')
plt.legend(loc='lower right')
plt.show()
```



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

# Application de SMOTE sur les données d'entraînement
smote = SMOTE(random_state=42)
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]:
```python
# 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]
```

[23]:
```python
# 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
)
```

[24]:
```python
# 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 83ms/step -
accuracy: 0.7692 - loss: 0.6914
Test Accuracy with SMOTE: 76.92%
```

[25]:
```python
# 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 71ms/step
Precision with SMOTE: 0.00
Recall with SMOTE: 0.00
F1-Score with SMOTE: 0.00
AUC with SMOTE: 0.45
```

```
/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))
```

[26]:

```python
# 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 sur les ensembles d\'entraînement et de validation 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 sur les ensembles d\'entraînement et de validation avec SMOTE')
plt.show()
```