Introduction

Ce notebook a pour objectif de tester et d'évaluer les fonctionnalités de la toolbox **ADAPT** sur divers jeux de données, illustrant ainsi sa flexibilité et son efficacité dans des scénarios variés d'adaptation de domaine. Pour ce faire, nous avons sélectionné deux types d'exemples :

1. **Exemples Synthétiques (Small Examples)** : Ces exemples utilisent des données générées artificiellement pour simuler des scénarios d'adaptation de domaine. Ils permettent de contrôler précisément les distributions des données source et cible, facilitant ainsi l'analyse des performances des différentes méthodes d'adaptation sans les complexités inhérentes aux données réelles. Ces exemples servent de base pour comprendre les mécanismes fondamentaux des algorithmes d'adaptation de domaine.
2. **Exemples du Monde Réel (Real-World Examples)** : Dans cette section, nous appliquons les techniques d'adaptation de domaine de la toolbox **ADAPT** à des jeux de données réels, tels que **Office-31** et **Mini-VisDA2017**. Ces exemples illustrent comment la toolbox peut être utilisée pour traiter des problématiques concrètes où les distributions des domaines source et cible diffèrent de manière significative. Ils mettent en lumière les défis et les solutions apportées par les méthodes d'adaptation de domaine dans des contextes applicatifs variés.

Structure du Notebook

1. **Exemples Synthétiques** :
   - **Classification Binaire avec** `make_classification_da`, `make_blobs` ... : Création de jeux de données synthétiques et application de différentes méthodes d'adaptation de domaine telles que **DANN** (Domain-Adversarial Neural Network) et **KMM** (Kernel Mean Matching).
   - **Régression avec** `make_regression_da` : Simulations de scénarios de régression unidimensionnelle pour tester les approches d'adaptation de domaine.
2. **Exemples du Monde Réel** :
   - **Gestion du Biais d'Échantillonnage avec le Jeu de Données Diabetes et Office-31** : Introduction de biais dans les jeux de données et utilisation des techniques de **KLIEP** et **KMM** pour corriger ces biais.
   - **flowers dataset et intel classification** : Application des méthodes de fine-tuning et d'adaptation de domaine avancées pour améliorer les performances des modèles sur des jeux de données complexes.
   - **Fine-Tuning avanacé avec office-31 et Mini-VisDA201** : Application des méthodes d'adaptation de domaine avancées
3. **Visualisation et Animation** : Présentation des résultats à travers des visualisations graphiques et des animations, permettant de suivre l'évolution des performances des modèles au fil des époques d'entraînement.

[ ] ↳ *1 cellule masquée*

Domain Adaptation Small Examples

Classification

Dans cette section, nous avons testé des méthodes de classification dans le cadre de l'adaptation de domaine. Dans le code original, les ensembles de données utilisés étaient `make_classification_da` et `make_moons`. Afin d'évaluer la robustesse et la généralisation des approches d'adaptation de domaine, nous avons également étendu ces expériences en utilisant deux autres méthodes :

- **make_blobs** : Génère des données regroupées en plusieurs blobs, introduisant une séparation entre les classes qui peut être linéaire ou non linéaire selon les paramètres utilisés.
- **make_circles** : Génère des données en forme de cercles concentriques, introduisant une séparation non linéaire entre les classes.

Ces configurations de données variées permettent de mieux comprendre comment chaque méthode d'adaptation de domaine se comporte face à différentes structures de données et divergences entre les domaines source et cible.

Les sections suivantes détaillent les résultats obtenus avec chaque ensemble de données et les différentes méthodes d'adaptation de domaine testées.

Adaptation de Domaine en Classification Binaire avec `make_classification_da`

Dans ce notebook, nous explorons l'application des méthodes d'adaptation de domaine (DA) fournies par le package **ADAPT** sur un problème de classification binaire en deux dimensions. L'objectif est de démontrer comment différentes techniques de DA peuvent améliorer la performance d'un modèle de classification lorsqu'il est confronté à des données provenant de domaines distincts.

Les méthodes abordées incluent :

- **Source Only** : Entraînement sur les données source sans adaptation.
- **DANN (Domain-Adversarial Neural Network)** : Apprentissage de représentations invariantes au domaine via des techniques adversariales.
- **KMM (Kernel Mean Matching)** : Réajustement des poids des instances source pour aligner les distributions des domaines source et cible.

```
# === Importation des bibliothèques ===

# Standard libraries
import numpy as np

# Visualisation et animations avec Matplotlib
import matplotlib.pyplot as plt
import matplotlib
import matplotlib.animation as animation
from matplotlib import rc

# Deep Learning avec TensorFlow
```

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, Reshape
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers.legacy import Adam as LegacyAdam
from tensorflow.keras.callbacks import Callback

# Adaptation de domaine avec ADAPT
from adapt.utils import make_classification_da
from adapt.feature_based import DANN
from adapt.instance_based import KMM

# Machine Learning
from sklearn.metrics import accuracy_score
from sklearn.gaussian_process.kernels import RBF

# Configuration de Matplotlib
rc('animation', html='jshtml')

```

Configuration Expérimentale

Nous définissons un problème de classification synthétique en deux dimensions en utilisant la fonction `make_classification_da` du module `adapt.utils`. Cette fonction génère des ensembles de données source et cible avec des distributions distinctes pour simuler un scénario d'adaptation de domaine non supervisée.

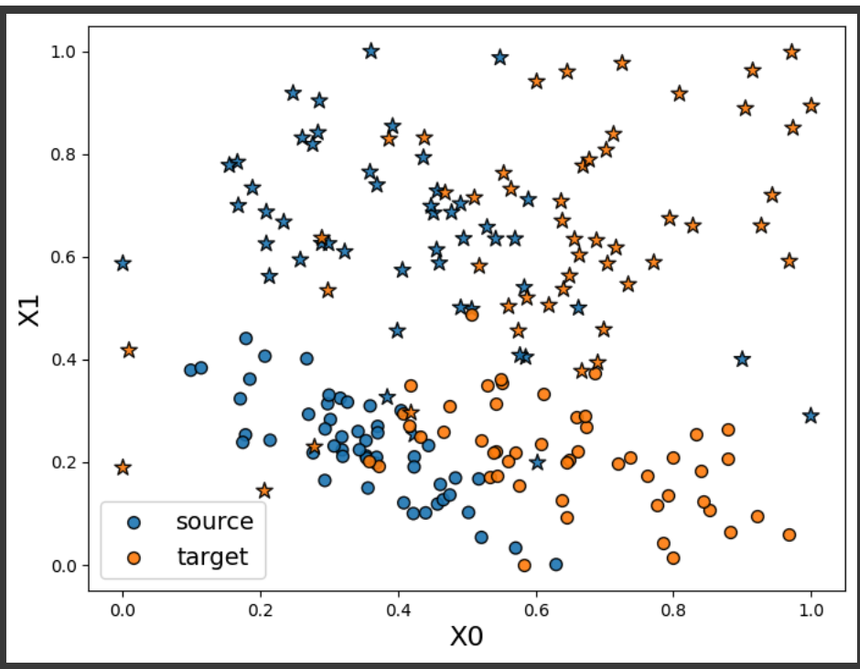
```
Xs, ys, Xt, yt = make_classification_da()

x_grid, y_grid = np.meshgrid(np.linspace(-0.1, 1.1, 100),
                             np.linspace(-0.1, 1.1, 100))
X_grid = np.stack([x_grid.ravel(), y_grid.ravel()], -1)
```

Fonction de Visualisation

La fonction `show` permet de visualiser les performances des algorithmes sur le problème de classification. Elle affiche les données source et cible, les frontières de décision apprises par le modèle, ainsi que les zones discriminées par les méthodes adversariales.

```
def show(ax, yp_grid=None, yp_t=None, x_grid=x_grid, y_grid=y_grid, Xs=Xs, Xt=Xt,
         weights_src=50*np.ones(100), disc_grid=None):
    cm = matplotlib.colors.ListedColormap(['w', 'r', 'w'])
    # ax = plt.gca()
    if yp_grid is not None:
        ax.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=1.)
        ax.plot([Xs[0, 0]], [Xs[0, 1]], c="red", label="class separation")
    if disc_grid is not None:
        cm_disc = matplotlib.colors.ListedColormap([(1,1,1,0), 'g', (1,1,1,0)])
        ax.contourf(x_grid, y_grid, disc_grid, cmap=cm_disc, alpha=0.5)
        ax.plot([Xs[0, 0]], [Xs[0, 1]], c="green", label="disc separation")
    if yp_t is not None:
        score = accuracy_score(yt.ravel(), yp_t.ravel())
        score = " - Acc=%.2f"%score
    else:
        score = ""
    ax.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k',
               c="C0", s=weights_src[ys==0], marker="o", alpha=0.9)
    ax.scatter(Xs[ys==1, 0], Xs[ys==1, 1], edgecolors='k',
               c="C0", s=2*weights_src[ys==1], marker="*", alpha=0.9)
    ax.scatter(Xt[yt==0, 0], Xt[yt==0, 1], label="target"+score, edgecolors='k',
               c="C1", s=50, marker="o", alpha=0.9)
    ax.scatter(Xt[yt==1, 0], Xt[yt==1, 1], edgecolors='k',
               c="C1", s=100, marker="*", alpha=0.9)
    ax.legend(fontsize=14, loc="lower left")
    ax.set_xlabel("X0", fontsize=16)
    ax.set_ylabel("X1", fontsize=16)
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
show(ax)
plt.show()
```



Définition du Modèle de Base

Nous définissons un réseau de neurones simple avec deux couches cachées pour apprendre la tâche de classification. De plus, nous créons un callback `SavePrediction` pour enregistrer les prédictions du modèle à chaque époque, ce qui nous permettra de visualiser l'évolution des performances au fil de l'entraînement.

```
def get_model(input_shape=(2,)):
    model = Sequential()
    model.add(Dense(100, activation='elu',
                    input_shape=input_shape))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(1, activation="sigmoid"))
    model.compile(optimizer=Adam(0.01), loss='binary_crossentropy')
    return model
```

```
class SavePrediction(Callback):
    """
    Callbacks which stores predicted
    labels in history at each epoch.
    """
    def __init__(self, X_grid_=X_grid, Xt_=Xt):
        self.X_grid = X_grid_
        self.Xt = Xt_
        self.custom_history_grid_ = []
        self.custom_history_ = []
        super().__init__()

    def on_epoch_end(self, batch, logs={}):
        """Applied at the end of each epoch"""
        predictions = self.model.predict_on_batch(self.X_grid).reshape(100, 100)
        self.custom_history_grid_.append(predictions)
        predictions = self.model.predict_on_batch(self.Xt).ravel()
        self.custom_history_.append(predictions)
```

Méthode Source Only

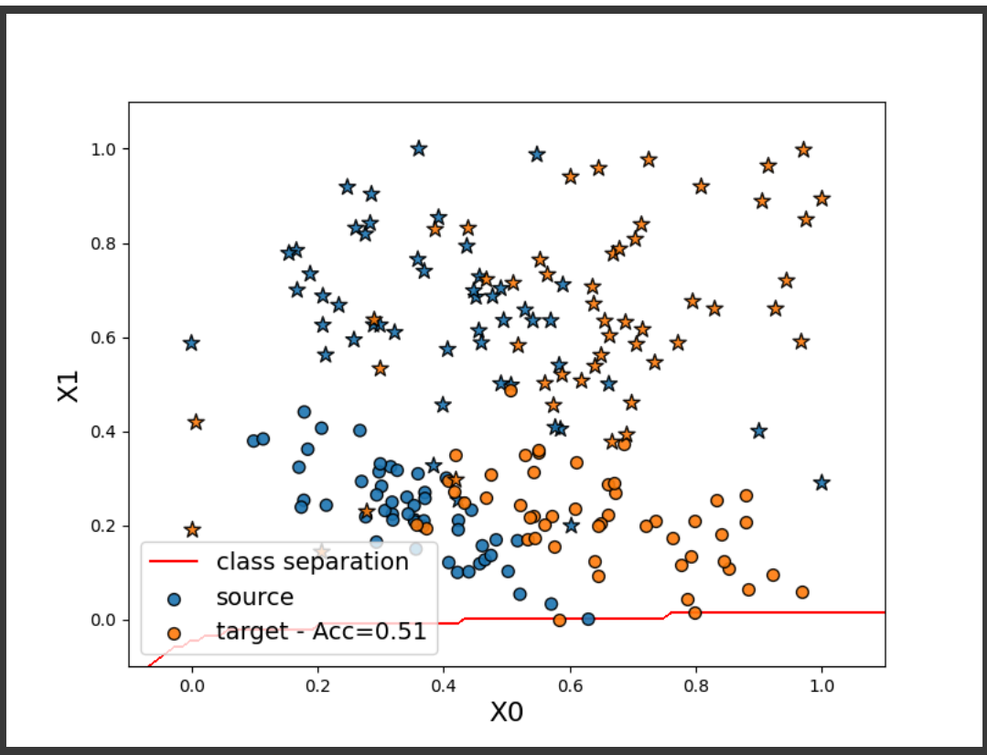
Dans cette approche, nous entraînons le réseau de neurones uniquement sur les données source sans aucune adaptation de domaine. Cette méthode sert de référence pour évaluer l'impact des techniques d'adaptation de domaine. Comme attendu, le modèle peut mal classer les données cibles en raison de la différence entre les distributions source et cible.

```
np.random.seed(0)
tf.random.set_seed(0)

model = get_model()
save_preds = SavePrediction()
model.fit(Xs, ys, callbacks=[save_preds], epochs=100, batch_size=100, verbose=0);
```

```
def animate(i):
    ax.clear()
    yp_grid = (save_preds.custom_history_grid_[i]>0.5).astype(int)
    yp_t = save_preds.custom_history_[i]>0.5
    show(ax, yp_grid, yp_t)
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 6));
ani = animation.FuncAnimation(fig, animate, frames=100, interval=60, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```



Once Loop Reflect

DANN (Domain-Adversarial Neural Network)

La méthode DANN vise à apprendre une représentation des données où les domaines source et cible sont indiscernables par un réseau discriminateur. Cela est réalisé en entraînant simultanément un encodeur, un classificateur de tâche et un discriminateur de domaine de manière adversariale. L'objectif est de minimiser la perte de classification tout en maximisant la confusion du discriminateur entre les domaines.

```
def get_encoder(input_shape=(2,)):
    model = Sequential()
    model.add(Dense(100, activation='elu',
                    input_shape=input_shape))
    model.add(Dense(2, activation="sigmoid"))
    model.compile(optimizer=Adam(0.01), loss='mse')
    return model

def get_task(input_shape=(2,)):
    model = Sequential()
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1, activation="sigmoid"))
    model.compile(optimizer=Adam(0.01), loss='mse')
    return model

def get_discriminator(input_shape=(2,)):
    model = Sequential()
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1, activation="sigmoid"))
    model.compile(optimizer=Adam(0.01), loss='mse')
    return model
```

Callback `SavePredictionDann`

Ce callback enregistre non seulement les prédictions de la tâche principale, mais également les représentations encodées des données source, cible et de la grille de visualisation, ainsi que les prédictions du discriminateur. Cela facilite la visualisation de l'évolution de l'adaptation de domaine au fil des époques.

```

class SavePredictionDann(Callback):
    """
    Callbacks which stores predicted
    labels in history at each epoch.
    """
    def __init__(self, X_grid_=X_grid, Xt_=Xt, Xs_=Xs):
        self.X_grid = X_grid_
        self.Xt = Xt_
        self.Xs = Xs_
        self.custom_history_grid_ = []
        self.custom_history_ = []
        self.custom_history_enc_s = []
        self.custom_history_enc_t = []
        self.custom_history_enc_grid = []
        self.custom_history_disc = []
        super().__init__()

    def on_epoch_end(self, batch, logs={}):
        """Applied at the end of each epoch"""
        predictions = model.task_.predict_on_batch(
```

```
            model.encoder_.predict_on_batch(self.X_grid)).reshape(100, 100)
        self.custom_history_grid_.append(predictions)
        predictions = model.task_.predict_on_batch(
            model.encoder_.predict_on_batch(self.Xt)).ravel()
        self.custom_history_.append(predictions)
        predictions = model.encoder_.predict_on_batch(self.Xs)
        self.custom_history_enc_s.append(predictions)
        predictions = model.encoder_.predict_on_batch(self.Xt)
        self.custom_history_enc_t.append(predictions)
        predictions = model.encoder_.predict_on_batch(self.X_grid)
        self.custom_history_enc_grid.append(predictions)
        predictions = model.discriminator_.predict_on_batch(
            model.encoder_.predict_on_batch(self.X_grid)).reshape(100, 100)
        self.custom_history_disc.append(predictions)
```

```

save_preds = SavePredictionDann()

model = DANN(
    get_encoder(),
    get_task(),
    get_discriminator(),
    lambda_=1.0,
    optimizer=Adam(0.001),
    random_state=0
)


model.fit(Xs, ys, Xt,
          callbacks=[save_preds],
          epochs=500, batch_size=100, verbose=0);
```

```
WARNING:tensorflow:5 out of the last 203 calls to <function Model.make_predict_function.<locals>.predict_function at 0x79d6f26cc550> tri
WARNING:tensorflow:6 out of the last 208 calls to <function Model.make_predict_function.<locals>.predict_function at 0x79d6f26cc430> tri
```

Animation de l'Évolution des Représentations Encodées

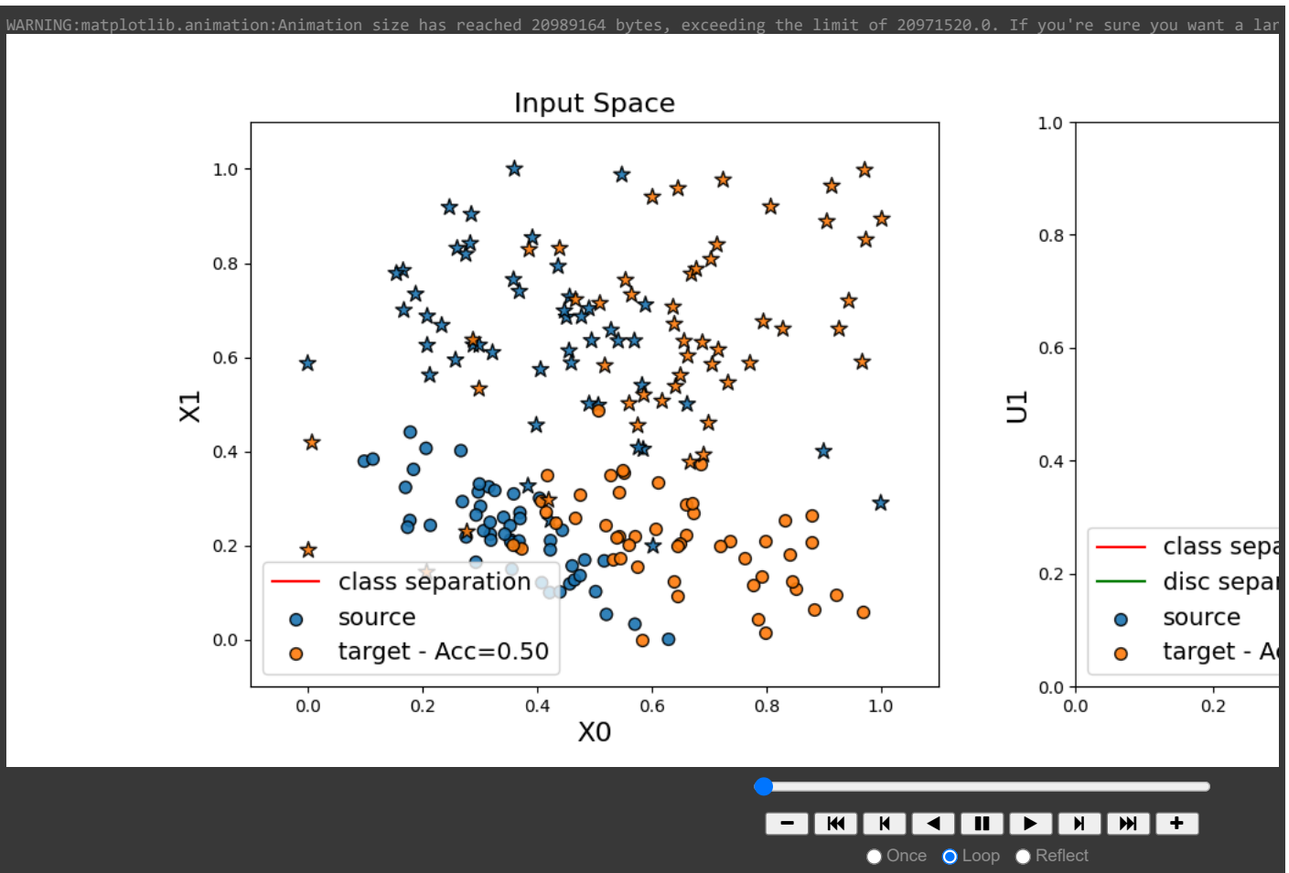
Nous visualisons à la fois l'espace d'entrée et l'espace encodé pour observer comment le modèle DANN aligne les distributions source et cible au fil des époques.

```
enc_s = np.concatenate(save_preds.custom_history_enc_s)
enc_t = np.concatenate(save_preds.custom_history_enc_t)
enc = np.concatenate((enc_s, enc_t))
x_min, y_min = enc.min(0)
x_max, y_max = enc.max(0)

x_min, y_min = (0., 0.)
x_max, y_max = (1., 1.)

def animate_dann(i):
    i *= 3
    yp_grid = (save_preds.custom_history_grid_[i]>0.5).astype(int)
    yp_t = save_preds.custom_history_[i]>0.5
    ax1.clear()
    ax2.clear()
    ax1.set_title("Input Space", fontsize=16)
    show(ax1, yp_grid, yp_t)
    ax2.set_title("Encoded Space", fontsize=16)
    Xs_enc = save_preds.custom_history_enc_s[i]
    Xt_enc = save_preds.custom_history_enc_t[i]
    X_grid_enc = save_preds.custom_history_enc_grid[i]
    x_grid_enc = X_grid_enc[:, 0].reshape(100,100)
    y_grid_enc = X_grid_enc[:, 1].reshape(100,100)
    disc_grid = (save_preds.custom_history_disc[i]>0.5).astype(int)
    show(ax2, yp_grid, yp_t,
         x_grid=x_grid_enc, y_grid=y_grid_enc,
         Xs=Xs_enc, Xt=Xt_enc, disc_grid=disc_grid)
    ax2.set_xlabel("U0", fontsize=16)
    ax2.set_ylabel("U1", fontsize=16)
    ax2.set_xlim(x_min, x_max)
    ax2.set_ylim(y_min, y_max)
```

```
fig, (ax1 , ax2) = plt.subplots(1, 2, figsize=(16, 6))
ani = animation.FuncAnimation(fig, animate_dann, frames=150, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```

```
WARNING:matplotlib.animation:Animation size has reached 20989164 bytes, exceeding the limit of 20971520.0. If you're sure you want a lar
```




```
ani.save('dann.gif', writer="imagemagick")
```

```
WARNING:matplotlib.animation:MovieWriter imagemagick unavailable; using Pillow instead.
```

KMM (Kernel Mean Matching)

La méthode KMM est une approche basée sur les instances qui réajuste les poids des données source afin de minimiser la distance MMD (Maximum Mean Discrepancy) entre les distributions source et cible. En ajustant ces poids, KMM permet au classificateur d'apprendre à partir des données source rééquilibrées, améliorant ainsi la généralisation sur le domaine cible.

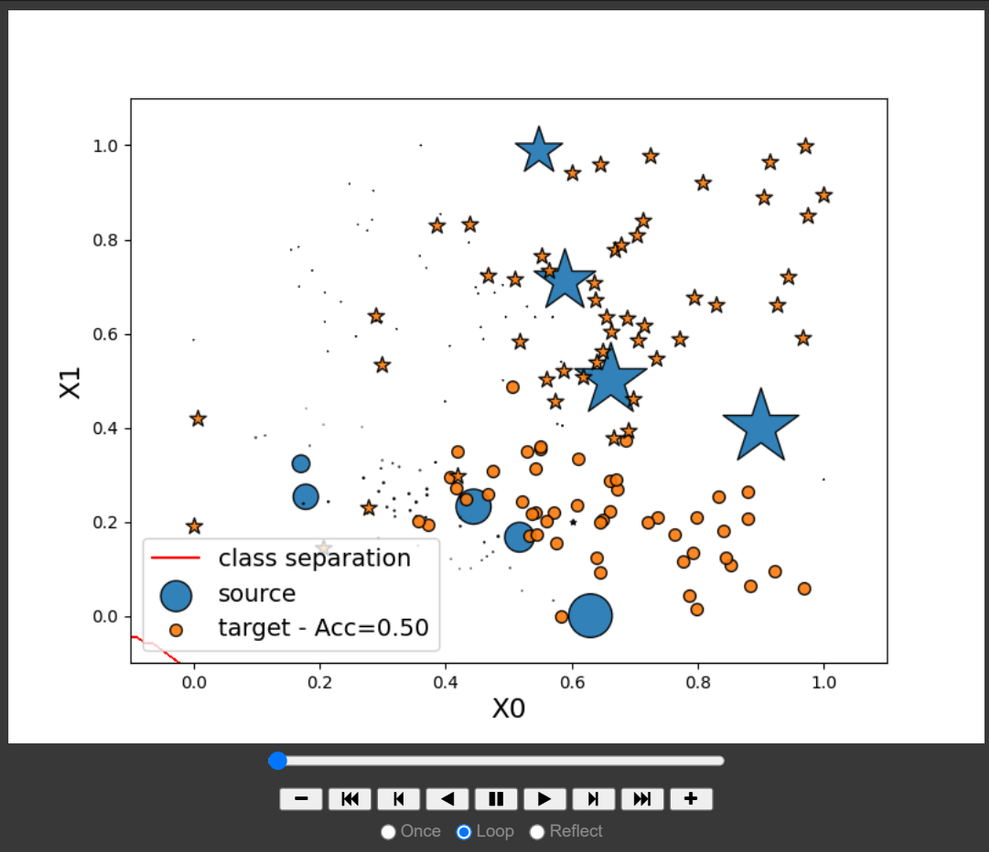
```
save_preds = SavePrediction()

model = KMM(
    get_model(),
    kernel="rbf",  # Utiliser 'rbf' pour le noyau gaussien
    gamma=1,       # Hyperparamètre du noyau
    random_state=0
)


model.fit(Xs, ys, Xt,
          callbacks=[save_preds],
          epochs=100, batch_size=100, verbose=0);
```

```
Fit weights...
        pcost       dcost       gap    pres   dres
 0:  4.1412e+04 -1.3491e+06  3e+07  4e-01  2e-15
 1:  1.8736e+02 -2.9533e+05  4e+05  2e-03  1e-12
 2:  2.0702e+02 -3.6581e+04  4e+04  2e-05  7e-14
 3:  8.2217e+01 -1.6809e+04  2e+04  7e-06  3e-14
 4: -3.5699e+03 -2.6162e+04  2e+04  7e-06  2e-14
 5: -3.6501e+03 -7.6959e+03  4e+03  1e-06  4e-15
 6: -3.8524e+03 -8.5199e+03  5e+03  3e-16  4e-16
 7: -4.0411e+03 -4.6607e+03  6e+02  2e-16  4e-16
 8: -4.0654e+03 -4.4933e+03  4e+02  2e-16  3e-16
 9: -4.0776e+03 -4.1640e+03  9e+01  2e-16  3e-16
10: -4.0853e+03 -4.1556e+03  7e+01  2e-16  3e-16
11: -4.0894e+03 -4.0973e+03  8e+00  2e-16  4e-16
12: -4.0903e+03 -4.0934e+03  3e+00  1e-16  5e-16
13: -4.0906e+03 -4.0912e+03  6e-01  2e-16  4e-16
14: -4.0906e+03 -4.0911e+03  4e-01  2e-16  3e-16
15: -4.0907e+03 -4.0908e+03  1e-01  2e-16  4e-16
16: -4.0907e+03 -4.0908e+03  5e-02  2e-16  4e-16
17: -4.0908e+03 -4.0908e+03  2e-02  2e-16  4e-16
18: -4.0908e+03 -4.0908e+03  3e-03  2e-16  4e-16
Optimal solution found.
Fit Estimator...
```

```
def animate_kmm(i):
    ax.clear()
    yp_grid = (save_preds.custom_history_grid_[i]>0.5).astype(int)
    yp_t = save_preds.custom_history_[i]>0.5
    weights_src = model.predict_weights().ravel() * 50
    show(ax, yp_grid, yp_t, weights_src=weights_src)
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ani = animation.FuncAnimation(fig, animate_kmm, frames=100, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```




```
Commencez à coder ou à générer avec l'IA.
```

```
Commencez à coder ou à générer avec l'IA.
```

Adaptation de Domaine en Classification Binaire avec `make_blobs`

Dans cette section, nous explorons l'application des méthodes d'adaptation de domaine (DA) fournies par le package **ADAPT** sur un problème de classification binaire en deux dimensions. Contrairement à l'utilisation de `make_classification_da`, nous générons les données synthétiques en utilisant la fonction `make_blobs` de `sklearn.datasets`. Cela nous permet de contrôler explicitement les centres des clusters pour simuler un scénario d'adaptation de domaine non supervisée.

```
# === Importation des bibliothèques ===

# Standard libraries
import numpy as np

# Visualisation et animations avec Matplotlib
import matplotlib.pyplot as plt
import matplotlib
import matplotlib.animation as animation
from matplotlib import rc

# Machine Learning
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_blobs
from sklearn.gaussian_process.kernels import RBF

# Deep Learning avec TensorFlow
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, Reshape
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers.legacy import Adam as LegacyAdam
from tensorflow.keras.callbacks import Callback

# Adaptation de domaine avec ADAPT
from adapt.feature_based import DANN
from adapt.instance_based import KMM

# Configuration de Matplotlib
rc('animation', html='jshtml')

```

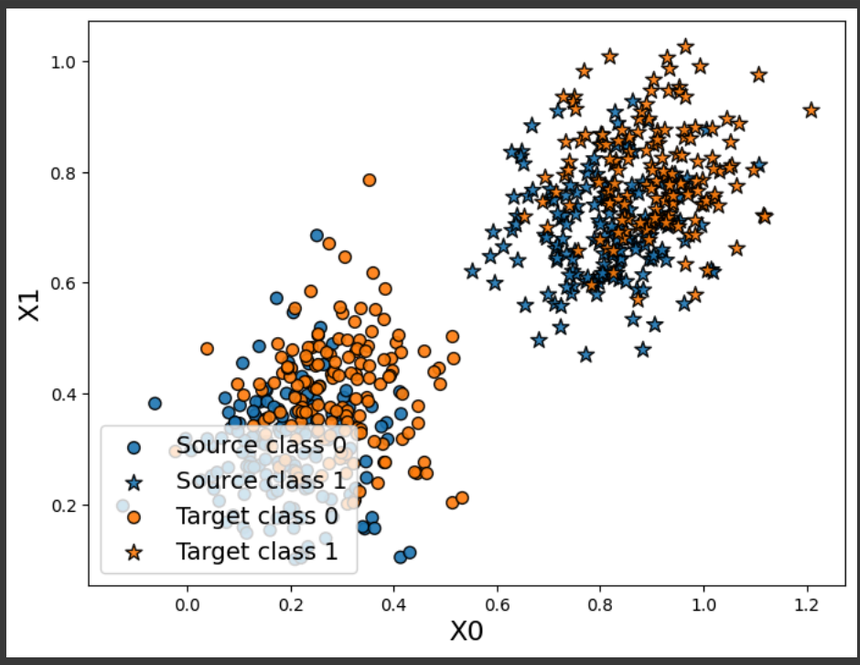
Configuration Expérimentale

```

# Génération des données source avec make_blobs
Xs, ys = make_blobs(n_samples=300, centers=[[0.2, 0.3], [0.8, 0.7]], cluster_std=0.1, random_state=42)

```

```
# Génération des données cible avec un décalage des centres pour simuler une adaptation
Xt, yt = make_blobs(n_samples=300, centers=[[0.3, 0.4], [0.9, 0.8]], cluster_std=0.1, random_state=42)

# Génération de la grille pour visualisation
x_grid, y_grid = np.meshgrid(np.linspace(-0.1, 1.1, 100),
                             np.linspace(-0.1, 1.1, 100))
X_grid = np.stack([x_grid.ravel(), y_grid.ravel()], -1)
```

```

def show(ax, yp_grid=None, yp_t=None, x_grid=x_grid, y_grid=y_grid, Xs=Xs, Xt=Xt,
         ys=ys, yt=yt, weights_src=None, disc_grid=None):
    cm = matplotlib.colors.ListedColormap(['w', 'r', 'w'])  # Colormap pour la grille
    if yp_grid is not None:
        ax.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=0.5)
        ax.plot([Xs[0, 0]], [Xs[0, 1]], c="red", label="Class separation")
    if disc_grid is not None:
        cm_disc = matplotlib.colors.ListedColormap([(1, 1, 1, 0), 'g', (1, 1, 1, 0)])
        ax.contourf(x_grid, y_grid, disc_grid, cmap=cm_disc, alpha=0.5)
        ax.plot([Xs[0, 0]], [Xs[0, 1]], c="green", label="Disc separation")
    if yp_t is not None:
        score = accuracy_score(yt, yp_t)
        score = " - Acc=%.2f" % score
    else:
        score = ""
    if weights_src is None:
        weights_src = 50 * np.ones_like(ys, dtype=float)

    # Points source
    ax.scatter(Xs[ys == 0, 0], Xs[ys == 0, 1], label="Source class 0", edgecolors='k',
               c="C0", s=weights_src[ys == 0], marker="o", alpha=0.9)
    ax.scatter(Xs[ys == 1, 0], Xs[ys == 1, 1], label="Source class 1", edgecolors='k',
               c="C0", s=2 * weights_src[ys == 1], marker="*", alpha=0.9)

    # Points cible
    ax.scatter(Xt[yt == 0, 0], Xt[yt == 0, 1], label="Target class 0" + score, edgecolors='k',
               c="C1", s=50, marker="o", alpha=0.9)
    ax.scatter(Xt[yt == 1, 0], Xt[yt == 1, 1], label="Target class 1", edgecolors='k',
               c="C1", s=100, marker="*", alpha=0.9)

    ax.legend(fontsize=14, loc="lower left")
    ax.set_xlabel("X0", fontsize=16)
    ax.set_ylabel("X1", fontsize=16)

```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
show(ax)
plt.show()

```




```

def get_model(input_shape=(2,)):
    model = Sequential()
    model.add(Dense(100, activation='elu',
                    input_shape=input_shape))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(1, activation="sigmoid"))
    model.compile(optimizer=Adam(0.01), loss='binary_crossentropy')
    return model

```

```

class SavePrediction(Callback):
    """
    Callbacks which stores predicted
    labels in history at each epoch.
    """
    def __init__(self, X_grid_=X_grid, Xt_=Xt):
        self.X_grid = X_grid_
        self.Xt = Xt_
```

```
        self.custom_history_grid_ = []
        self.custom_history_ = []
        super().__init__()

    def on_epoch_end(self, batch, logs={}):
        """Applied at the end of each epoch"""
        predictions = self.model.predict_on_batch(self.X_grid).reshape(100, 100)
        self.custom_history_grid_.append(predictions)
        predictions = self.model.predict_on_batch(self.Xt).ravel()
        self.custom_history_.append(predictions)
```

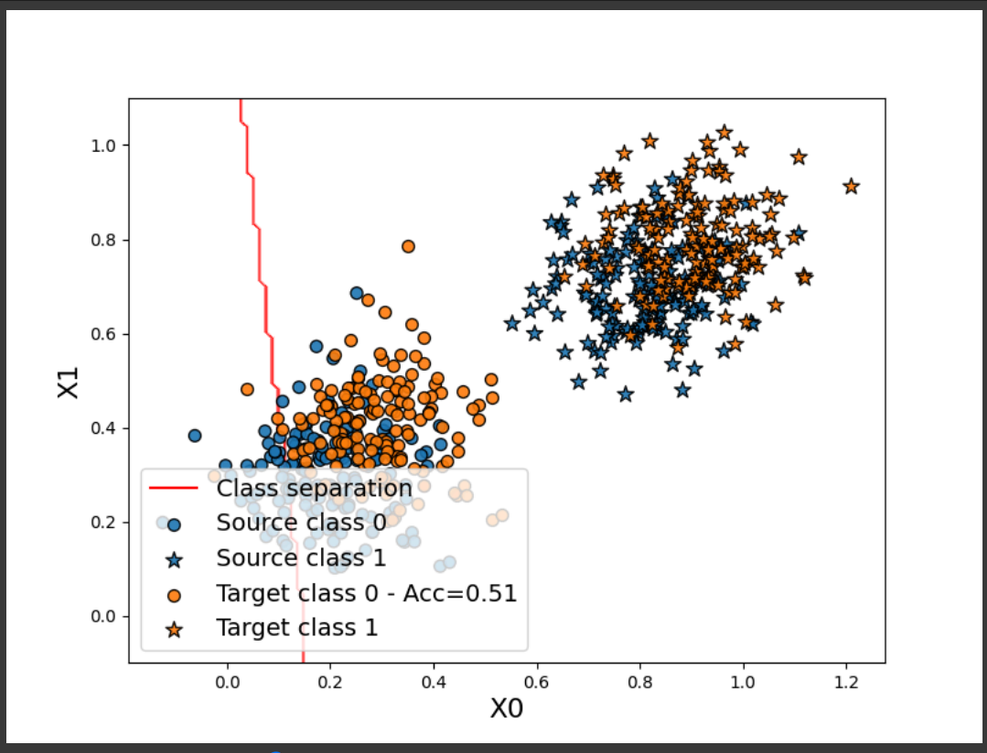
Méthode Source Only

```
np.random.seed(0)
tf.random.set_seed(0)

model = get_model()
save_preds = SavePrediction()
model.fit(Xs, ys, callbacks=[save_preds], epochs=100, batch_size=100, verbose=0);
```

```
def animate(i):
    ax.clear()
    yp_grid = (save_preds.custom_history_grid_[i]>0.5).astype(int)
    yp_t = save_preds.custom_history_[i]>0.5
    show(ax, yp_grid, yp_t)
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 6));
ani = animation.FuncAnimation(fig, animate, frames=100, interval=60, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```



Once Loop Reflect

DANN

```
def get_encoder(input_shape=(2,)):
    model = Sequential()
    model.add(Dense(100, activation='elu',
                    input_shape=input_shape))
    model.add(Dense(2, activation="sigmoid"))
    model.compile(optimizer=Adam(0.01), loss='mse')
    return model

def get_task(input_shape=(2,)):
    model = Sequential()
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1, activation="sigmoid"))
    model.compile(optimizer=Adam(0.01), loss='mse')
    return model

def get_discriminator(input_shape=(2,)):
    model = Sequential()
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1, activation="sigmoid"))
    model.compile(optimizer=Adam(0.01), loss='mse')
    return model
```

```

```

```
class SavePredictionDann(Callback):
    """
    Callbacks which stores predicted
    labels in history at each epoch.
    """
    def __init__(self, X_grid_=X_grid, Xt_=Xt, Xs_=Xs):
        self.X_grid = X_grid_
        self.Xt = Xt_
        self.Xs = Xs_
        self.custom_history_grid_ = []
        self.custom_history_ = []
        self.custom_history_enc_s = []
        self.custom_history_enc_t = []
        self.custom_history_enc_grid = []
        self.custom_history_disc = []
        super().__init__()

    def on_epoch_end(self, batch, logs={}):
        """Applied at the end of each epoch"""
        predictions = model.task_.predict_on_batch(
            model.encoder_.predict_on_batch(self.X_grid)).reshape(100, 100)
        self.custom_history_grid_.append(predictions)
        predictions = model.task_.predict_on_batch(
            model.encoder_.predict_on_batch(self.Xt)).ravel()
        self.custom_history_.append(predictions)
        predictions = model.encoder_.predict_on_batch(self.Xs)
        self.custom_history_enc_s.append(predictions)
        predictions = model.encoder_.predict_on_batch(self.Xt)
        self.custom_history_enc_t.append(predictions)
        predictions = model.encoder_.predict_on_batch(self.X_grid)
        self.custom_history_enc_grid.append(predictions)
        predictions = model.discriminator_.predict_on_batch(
            model.encoder_.predict_on_batch(self.X_grid)).reshape(100, 100)
        self.custom_history_disc.append(predictions)
```

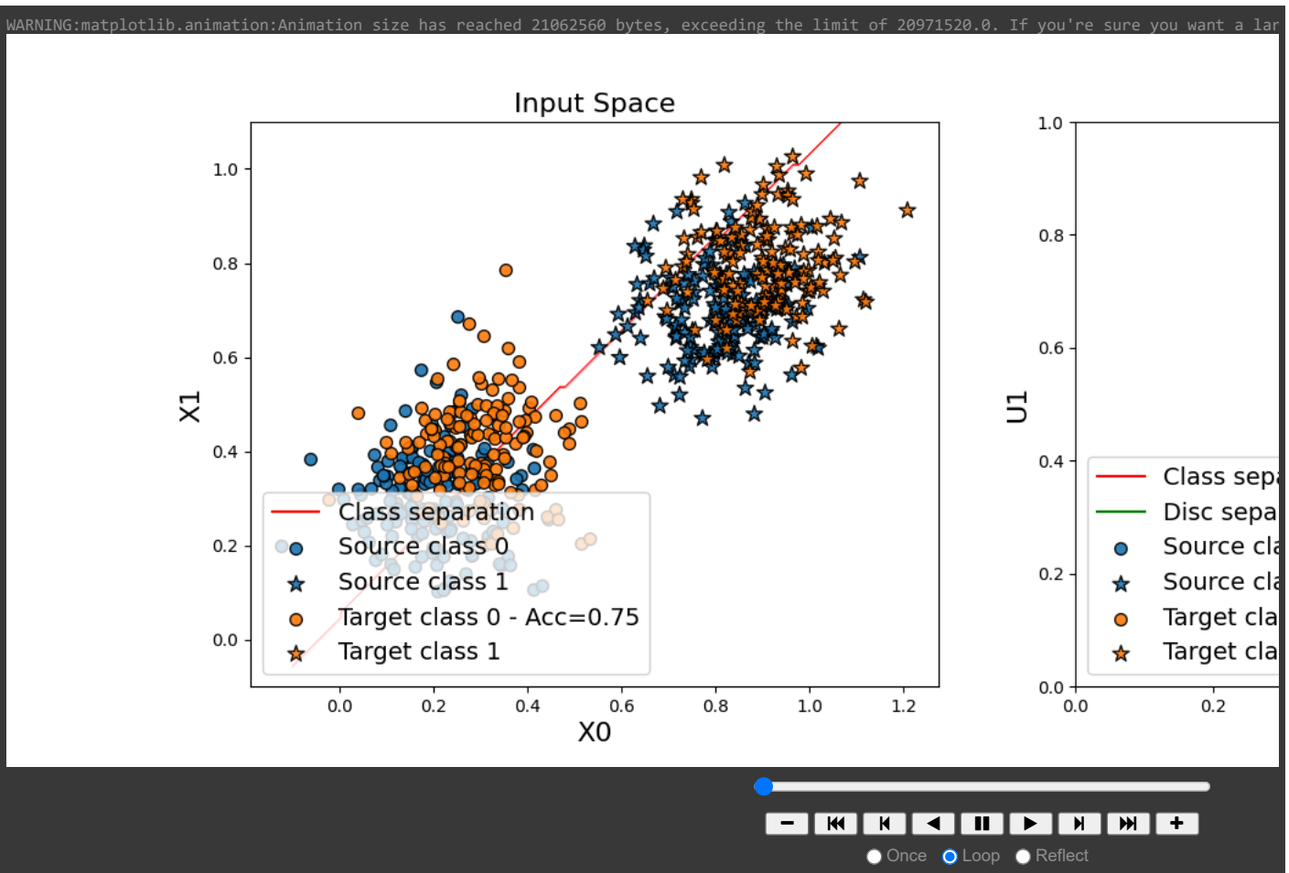
```
save_preds = SavePredictionDann()

model = DANN(
    get_encoder(),
    get_task(),
    get_discriminator(),
    lambda_=1.0,
    optimizer=Adam(0.001),
    random_state=0
)


model.fit(Xs, ys, Xt,
          callbacks=[save_preds],
          epochs=500, batch_size=100, verbose=0);
```

```
enc_s = np.concatenate(save_preds.custom_history_enc_s)
enc_t = np.concatenate(save_preds.custom_history_enc_t)
enc = np.concatenate((enc_s, enc_t))
x_min, y_min = enc.min(0)
x_max, y_max = enc.max(0)

x_min, y_min = (0., 0.)
x_max, y_max = (1., 1.)

def animate_dann(i):
    i *= 3
    yp_grid = (save_preds.custom_history_grid_[i]>0.5).astype(int)
    yp_t = save_preds.custom_history_[i]>0.5
    ax1.clear()
    ax2.clear()
    ax1.set_title("Input Space", fontsize=16)
    show(ax1, yp_grid, yp_t)
    ax2.set_title("Encoded Space", fontsize=16)
    Xs_enc = save_preds.custom_history_enc_s[i]
    Xt_enc = save_preds.custom_history_enc_t[i]
    X_grid_enc = save_preds.custom_history_enc_grid[i]
    x_grid_enc = X_grid_enc[:, 0].reshape(100,100)
    y_grid_enc = X_grid_enc[:, 1].reshape(100,100)
    disc_grid = (save_preds.custom_history_disc[i]>0.5).astype(int)
    show(ax2, yp_grid, yp_t,
         x_grid=x_grid_enc, y_grid=y_grid_enc,
         Xs=Xs_enc, Xt=Xt_enc, disc_grid=disc_grid)
    ax2.set_xlabel("U0", fontsize=16)
    ax2.set_ylabel("U1", fontsize=16)
    ax2.set_xlim(x_min, x_max)
    ax2.set_ylim(y_min, y_max)
```

```
fig, (ax1 , ax2) = plt.subplots(1, 2, figsize=(16, 6))
ani = animation.FuncAnimation(fig, animate_dann, frames=150, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```

```
WARNING:matplotlib.animation:Animation size has reached 21062560 bytes, exceeding the limit of 20971520.0. If you're sure you want a lar
```




```
ani.save('dann.gif', writer="imagemagick")
```

```
WARNING:matplotlib.animation:MovieWriter imagemagick unavailable; using Pillow instead.
```

KMM (Kernel Mean Matching)

Finally, we consider here the instance-based method KMM. This method consists in reweighting source instances in order to minimize the MMD distance between source and target domain. Then the algorithm trains a classifier using the reweighted source data.

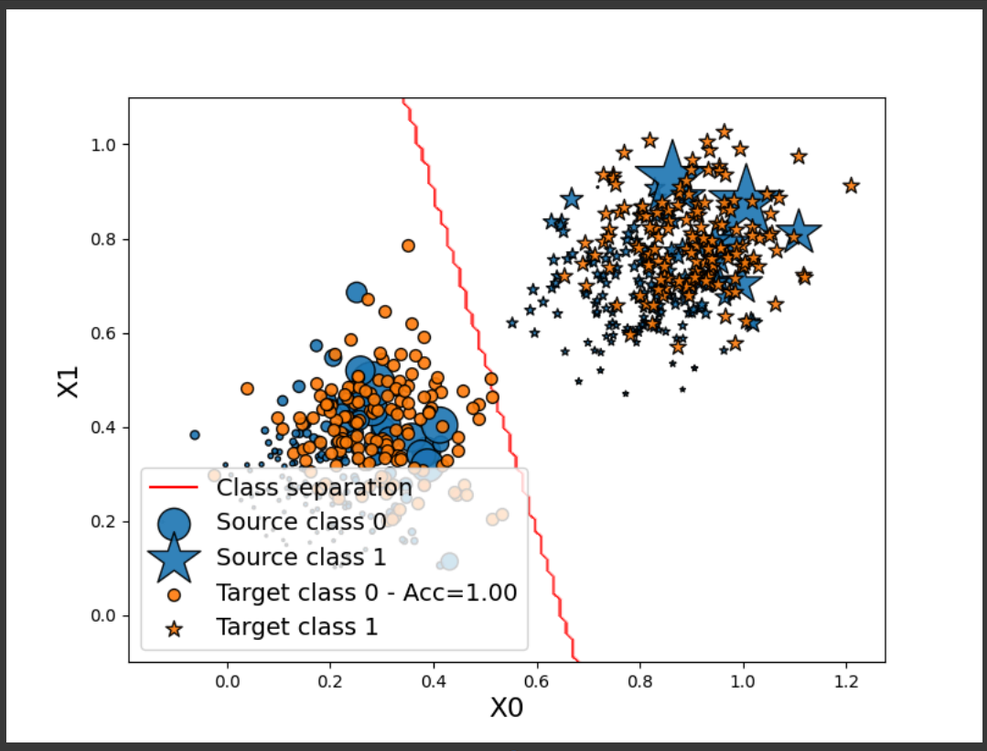
```


save_preds = SavePrediction()

model = KMM(
    get_model(),
    kernel="rbf",  # Utiliser 'rbf' pour le noyau gaussien
    gamma=1,       # Hyperparamètre du noyau
    random_state=0
)


model.fit(Xs, ys, Xt,
          callbacks=[save_preds],
          epochs=100, batch_size=100, verbose=0);
```

```
Fit weights...
      pcost       dcost       gap    pres   dres
 0:  8.1418e+03 -3.0829e+06  3e+07  1e-01  9e-16
 1: -3.7811e+03 -6.0684e+05  8e+05  8e-04  2e-13
 2: -3.8126e+03 -1.4133e+05  1e+05  2e-04  3e-14
 3: -2.6871e+04 -1.6685e+05  1e+05  1e-15  7e-15
 4: -3.4260e+04 -4.5510e+04  1e+04  2e-16  3e-16
 5: -3.4744e+04 -3.6355e+04  2e+03  2e-16  2e-16
 6: -3.4798e+04 -3.5323e+04  5e+02  2e-16  2e-16
 7: -3.4819e+04 -3.5037e+04  2e+02  2e-16  2e-16
 8: -3.4826e+04 -3.4882e+04  6e+01  2e-16  2e-16
 9: -3.4828e+04 -3.4868e+04  4e+01  2e-16  2e-16
10: -3.4829e+04 -3.4833e+04  4e+00  2e-16  2e-16
11: -3.4829e+04 -3.4830e+04  1e+00  2e-16  2e-16
12: -3.4829e+04 -3.4829e+04  2e-01  2e-16  2e-16
13: -3.4829e+04 -3.4829e+04  5e-02  2e-16  2e-16
14: -3.4829e+04 -3.4829e+04  1e-02  2e-16  2e-16
Optimal solution found.
Fit Estimator...
```

```
def animate_kmm(i):
    ax.clear()
    yp_grid = (save_preds.custom_history_grid_[i]>0.5).astype(int)
    yp_t = save_preds.custom_history_[i]>0.5
    weights_src = model.predict_weights().ravel() * 50
    show(ax, yp_grid, yp_t, weights_src=weights_src)
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ani = animation.FuncAnimation(fig, animate_kmm, frames=100, blit=False, repeat=True)
plt.close(fig)
```

```
1 ani
```



Adaptation de Domaine en Classification Binaire avec `make_moons`

Dans cette section, nous explorons l'application des méthodes d'adaptation de domaine (DA) fournies par le package **ADAPT** sur un problème de classification binaire en deux dimensions en utilisant les données générées par la fonction `make_moons` de `sklearn.datasets`. Ce scénario est particulièrement intéressant car les données en forme de deux lunes sont non linéaires et présentent un défi supplémentaire pour les modèles de classification classiques, surtout lorsqu'il s'agit d'adaptation de domaine.

Les méthodes abordées incluent :

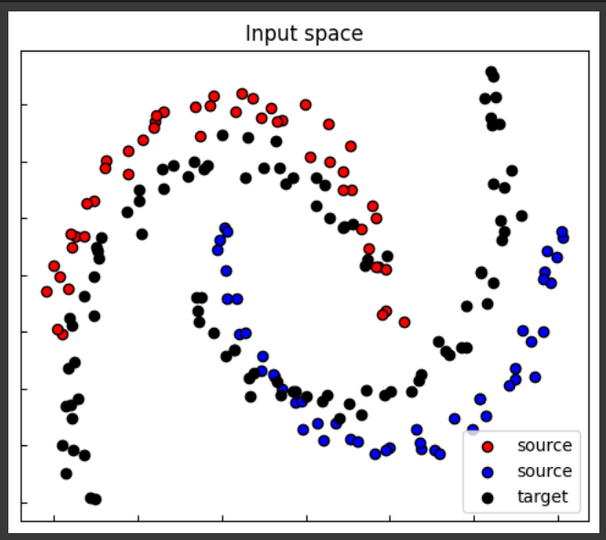
- **Source Only** : Entraînement sur les données source sans adaptation.
- **DANN (Domain-Adversarial Neural Network)** : Apprentissage de représentations invariantes au domaine via des techniques adversariales.
- **ADDA (Adversarial Discriminative Domain Adaptation)** : Une autre approche adversariale pour l'adaptation de domaine.
- **Deep CORAL** : Alignement des statistiques de second ordre entre les domaines source et cible.

```
1 # === Importation des bibliothèques ===
2
3 # Standard libraries
4 import os
5 import numpy as np
6 import pandas as pd
7
8 # Visualisation et animations
9 import matplotlib
10 import matplotlib.pyplot as plt
11 from matplotlib import rc
12
13 # Machine Learning
14 from sklearn.preprocessing import OneHotEncoder
15 from sklearn.decomposition import PCA
16 from sklearn.manifold import TSNE
17 from sklearn.metrics import accuracy_score
18 from sklearn.datasets import make_moons
19
20 # Deep Learning avec TensorFlow
21 import tensorflow as tf
22 from tensorflow.keras import Model, Sequential
23 from tensorflow.keras.layers import (
24     Dense, Input, Dropout, Conv2D, MaxPooling2D, Flatten, Reshape,
25     GaussianNoise, BatchNormalization
26 )
27 from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adagrad
28 from tensorflow.keras.optimizers import legacy as legacy_optimizers
29 from tensorflow.keras.constraints import MinMaxNorm
30 from tensorflow.keras.regularizers import l2
31 from tensorflow.keras.callbacks import Callback
32
33 # Adaptation de domaine avec ADAPT
34 import adapt
35 from adapt.feature_based import DANN, ADDA, DeepCORAL
36 from adapt.instance_based import KMM
37
38 # Configuration de Matplotlib
39 rc('animation', html='jshtml')
40
41
```

Configuration Expérimentale

```
def make_moons_da(n_samples=100, rotation=30, noise=0.05, random_state=0):
    Xs, ys = make_moons(n_samples=n_samples,
                        noise=noise,
                        random_state=random_state)
    Xs[:, 0] -= 0.5
    theta = np.radians(-rotation)
    cos_theta, sin_theta = np.cos(theta), np.sin(theta)
    rot_matrix = np.array(
        ((cos_theta, -sin_theta),
         (sin_theta, cos_theta))
    )
    Xt = Xs.dot(rot_matrix)
    yt = ys
    return Xs, ys, Xt, yt
```

```
Xs, ys, Xt, yt = make_moons_da()

x_min, y_min = np.min([Xs.min(0), Xt.min(0)], 0)
x_max, y_max = np.max([Xs.max(0), Xt.max(0)], 0)
x_grid, y_grid = np.meshgrid(np.linspace(x_min-0.1, x_max+0.1, 100),
                             np.linspace(y_min-0.1, y_max+0.1, 100))
X_grid = np.stack([x_grid.ravel(), y_grid.ravel()], -1)

fig, ax1 = plt.subplots(1, 1, figsize=(6, 5))
ax1.set_title("Input space")
ax1.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k', c="red")
ax1.scatter(Xs[ys==1, 0], Xs[ys==1, 1], label="source", edgecolors='k', c="blue")
ax1.scatter(Xt[:, 0], Xt[:, 1], label="target", edgecolors='k', c="black")
ax1.legend(loc="lower right")
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction ='in')
plt.show()
```



Définition du Modèle de Base

Nous définissons un réseau de neurones simple avec deux couches cachées pour apprendre la tâche de classification. De plus, nous créons un callback `SavePrediction` pour enregistrer les prédictions du modèle à chaque époque, ce qui nous permettra de visualiser l'évolution des performances au fil de l'entraînement.

```
def get_task():
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(10, activation="relu"))
    model.add(Dense(10, activation="relu"))
    model.add(Dense(1, activation="sigmoid"))
    return model
```

Méthode Source Only

```

# Instead of Adam(0.001, beta_1=0.5):
optimizer = optimizers.Adam(0.001, beta_1=0.5)

```

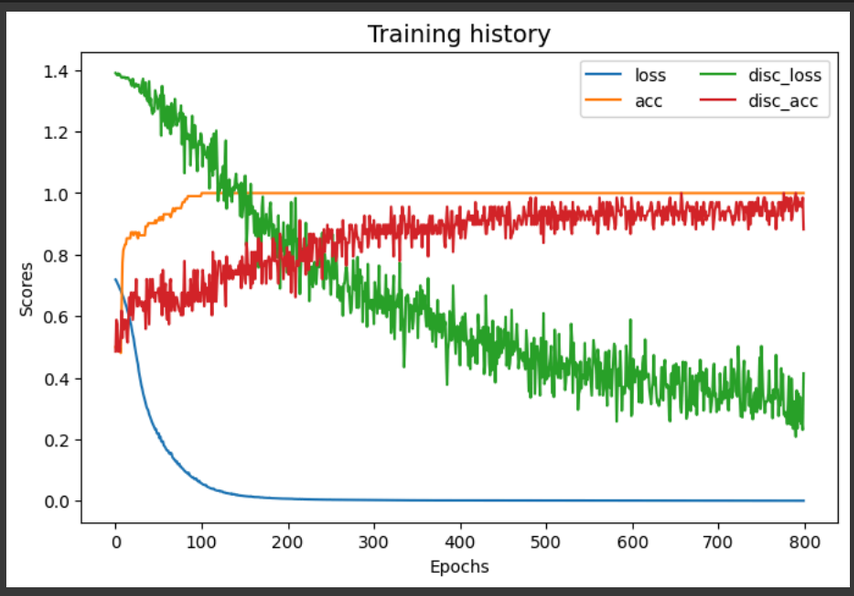
```
src_only = DANN(
    task=get_task(),
    loss="bce",
    optimizer=optimizer,
    copy=True,
    lambda_=0.,
    metrics=["acc"],
    gamma=10.,
    random_state=0
```

```
)

```

```
/usr/local/lib/python3.10/dist-packages/adapt/feature_based/_dann.py:114: UserWarning: the `gamma` argument has been removed from DANN.
  warnings.warn("the `gamma` argument has been removed from DANN."
```

```
# Build the model by calling it on a sample of the input data
model = get_task()
model.build((None, Xs.shape[1]))  # Ensure input shape is correct

```

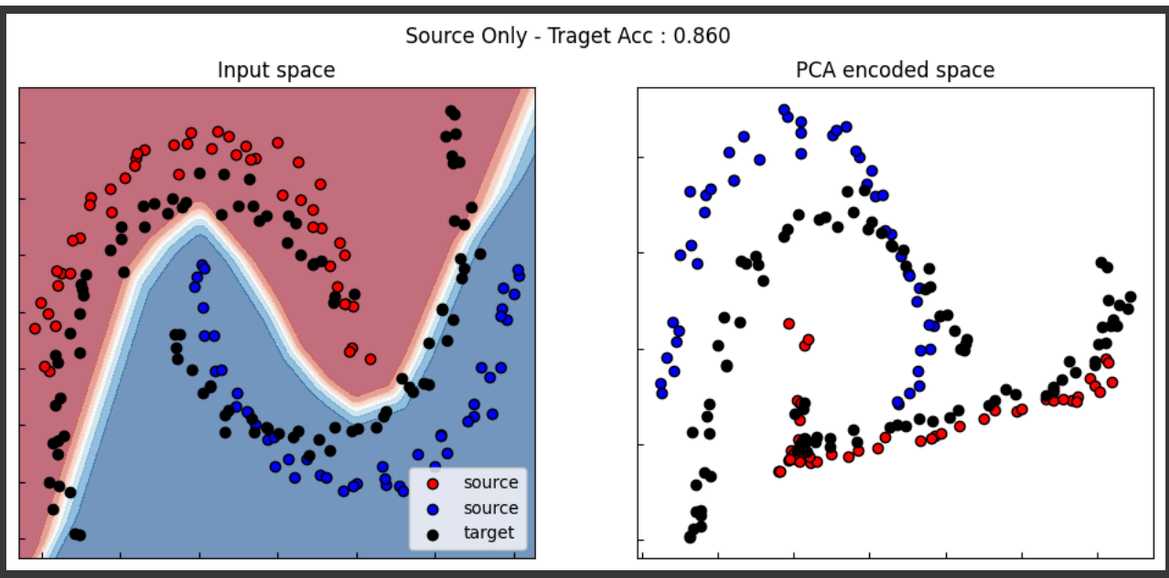
```

src_only.fit(Xs, ys, Xt, yt, epochs=800, batch_size=34, verbose=0);
pd.DataFrame(src_only.history_).plot(figsize=(8, 5))
plt.title("Training history", fontsize=14); plt.xlabel("Epochs"); plt.ylabel("Scores")
plt.legend(ncol=2)
plt.show()
```




```

yt_pred = src_only.predict(Xt)
acc = accuracy_score(yt, yt_pred>0.5)

yp_grid = src_only.predict(X_grid).reshape(100, 100)

# Obtenir les features encodées par l'encodeur du DANN
X_features_src = src_only.encoder_.predict(Xs)
X_features_tgt = src_only.encoder_.predict(Xt)


X_pca = np.concatenate((X_features_src, X_features_tgt))
X_pca = PCA(2).fit_transform(X_pca)


```

```
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
```

```
cm = plt.cm.RdBu
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.set_title("Input space")
ax1.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=0.6)
ax1.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k', c="red")
ax1.scatter(Xs[ys==1, 0], Xs[ys==1, 1], label="source", edgecolors='k', c="blue")
ax1.scatter(Xt[:, 0], Xt[:, 1], label="target", edgecolors='k', c="black")
ax1.legend()
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction ='in')

ax2.set_title("PCA encoded space")
ax2.scatter(X_pca[:len(Xs), 0][ys==0], X_pca[:len(Xs), 1][ys==0],
            label="source", edgecolors='k', c="red")
ax2.scatter(X_pca[:len(Xs), 0][ys==1], X_pca[:len(Xs), 1][ys==1],
            label="source", edgecolors='k', c="blue")
ax2.scatter(X_pca[len(Xs):, 0], X_pca[len(Xs):, 1],
            label="target", edgecolors='k', c="black")
ax2.set_yticklabels([])
ax2.set_xticklabels([])
ax2.tick_params(direction ='in')
fig.suptitle("Source Only - Traget Acc : %.3f"%acc)
plt.show()
```

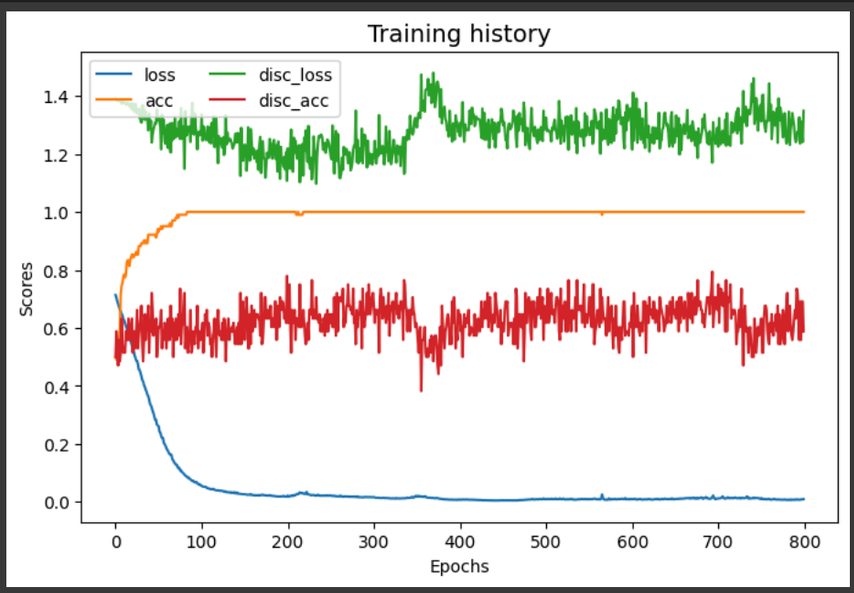


DANN

```
1
2 dann = DANN(
3     task=get_task(),
4     loss="bce",
5     optimizer=legacy_optimizers.Adam(0.001, beta_1=0.5),
6     copy=True,
7     lambda_=1.,
8     metrics=["acc"],
9     gamma=10.,
10    random_state=0
11 )
12
```

```
/usr/local/lib/python3.10/dist-packages/adapt/feature_based/_dann.py:114: UserWarning: the `gamma` argument has been removed from DANN.
  warnings.warn("the `gamma` argument has been removed from DANN."
```

```
1 dann.fit(Xs, ys, Xt, yt, epochs=800, batch_size=34, verbose=0);
2 pd.DataFrame(dann.history_).plot(figsize=(8, 5))
3 plt.title("Training history", fontsize=14); plt.xlabel("Epochs"); plt.ylabel("Scores")
4 plt.legend(ncol=2)
5 plt.show()
```

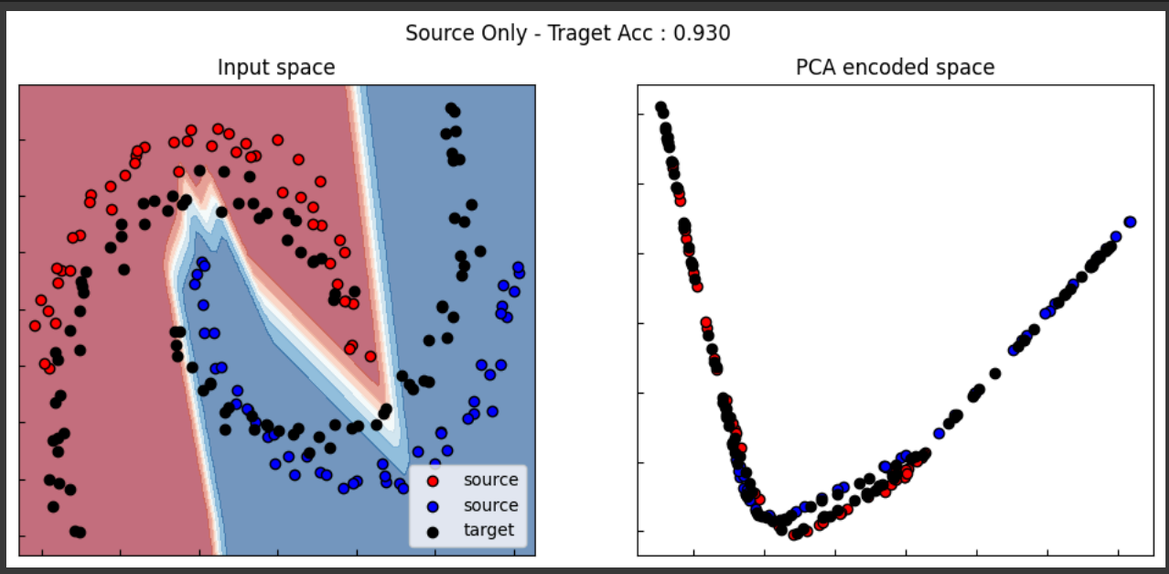



```
1 yt_pred = dann.predict(Xt)
2 acc = accuracy_score(yt, yt_pred>0.5)
3
4 yp_grid = dann.predict(X_grid).reshape(100, 100)
5
6 X_pca = np.concatenate((dann.encoder_.predict(Xs),
7                         dann.encoder_.predict(Xt)))
8 X_pca = PCA(2).fit_transform(X_pca)
```

```
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
```

```
1 cm = plt.cm.RdBu
2 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
3
4 ax1.set_title("Input space")
5 ax1.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=0.6)
6 ax1.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k', c="red")
7 ax1.scatter(Xs[ys==1, 0], Xs[ys==1, 1], label="source", edgecolors='k', c="blue")
8 ax1.scatter(Xt[:, 0], Xt[:, 1], label="target", edgecolors='k', c="black")
```

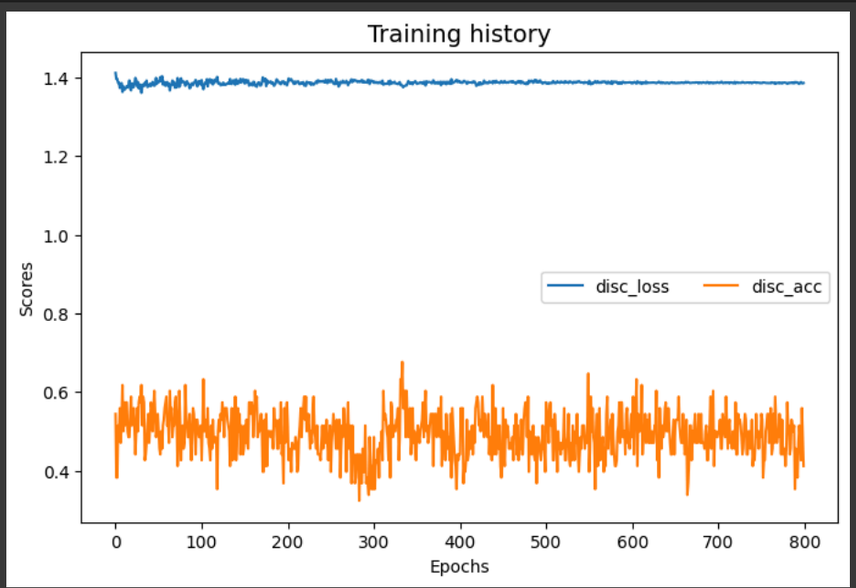
```
ax1.legend()
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction ='in')

ax2.set_title("PCA encoded space")
ax2.scatter(X_pca[:len(Xs), 0][ys==0], X_pca[:len(Xs), 1][ys==0],
            label="source", edgecolors='k', c="red")
ax2.scatter(X_pca[:len(Xs), 0][ys==1], X_pca[:len(Xs), 1][ys==1],
            label="source", edgecolors='k', c="blue")
ax2.scatter(X_pca[len(Xs):, 0], X_pca[len(Xs):, 1],
            label="target", edgecolors='k', c="black")
ax2.set_yticklabels([])
ax2.set_xticklabels([])
ax2.tick_params(direction ='in')
fig.suptitle("Source Only - Traget Acc : %.3f"%acc)
plt.show()
```



ADDA

```

adda = ADDA(task=get_task(),
            loss="bce", optimizer=legacy_optimizers.Adam(0.001, beta_1=0.5),
            copy=True, metrics=["acc"], random_state=0)
```

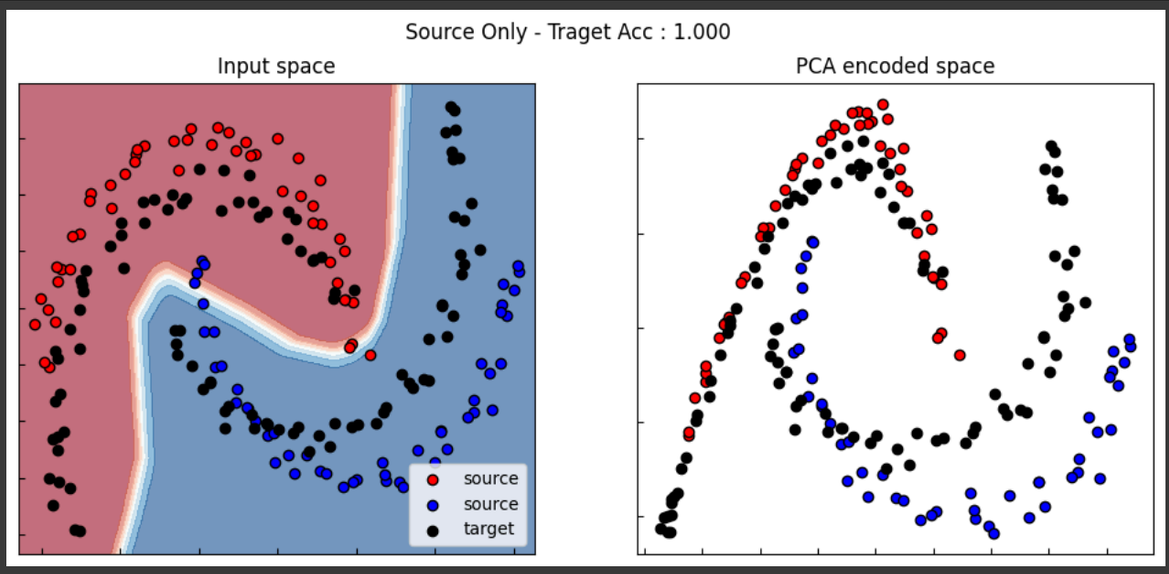
```
adda.fit(Xs, ys, Xt, yt, epochs=800, batch_size=34, verbose=0);
pd.DataFrame(adda.history_).plot(figsize=(8, 5))
plt.title("Training history", fontsize=14); plt.xlabel("Epochs"); plt.ylabel("Scores")
plt.legend(ncol=2)
plt.show()
```




```
yt_pred = adda.predict(Xt)
acc = accuracy_score(yt, yt_pred>0.5)

yp_grid = adda.predict(X_grid).reshape(100, 100)

X_pca = np.concatenate((adda.encoder_.predict(Xs),
                        adda.encoder_.predict(Xt)))
X_pca = PCA(2).fit_transform(X_pca)
```

```
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
```

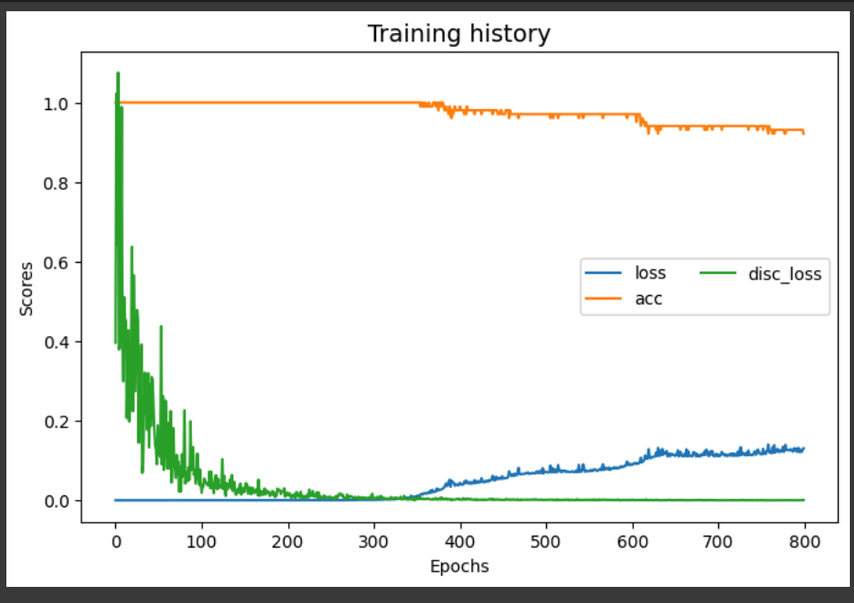
```
cm = plt.cm.RdBu
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.set_title("Input space")
ax1.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=0.6)
ax1.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k', c="red")
ax1.scatter(Xs[ys==1, 0], Xs[ys==1, 1], label="source", edgecolors='k', c="blue")
ax1.scatter(Xt[:, 0], Xt[:, 1], label="target", edgecolors='k', c="black")
ax1.legend()
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction ='in')

ax2.set_title("PCA encoded space")
ax2.scatter(X_pca[:len(Xs), 0][ys==0], X_pca[:len(Xs), 1][ys==0],
            label="source", edgecolors='k', c="red")
ax2.scatter(X_pca[:len(Xs), 0][ys==1], X_pca[:len(Xs), 1][ys==1],
            label="source", edgecolors='k', c="blue")
ax2.scatter(X_pca[len(Xs):, 0], X_pca[len(Xs):, 1],
            label="target", edgecolors='k', c="black")
ax2.set_yticklabels([])
ax2.set_xticklabels([])
ax2.tick_params(direction ='in')
fig.suptitle("Source Only - Traget Acc : %.3f"%acc)
plt.show()
```



DeepCORAL

```

dcoral = DeepCORAL(encoder=src_only.encoder_,
                   task=src_only.task_, lambda_=1000.,
            loss="bce", optimizer=legacy_optimizers.Adam(0.001, beta_1=0.5),
            copy=True, metrics=["acc"], random_state=0)
```

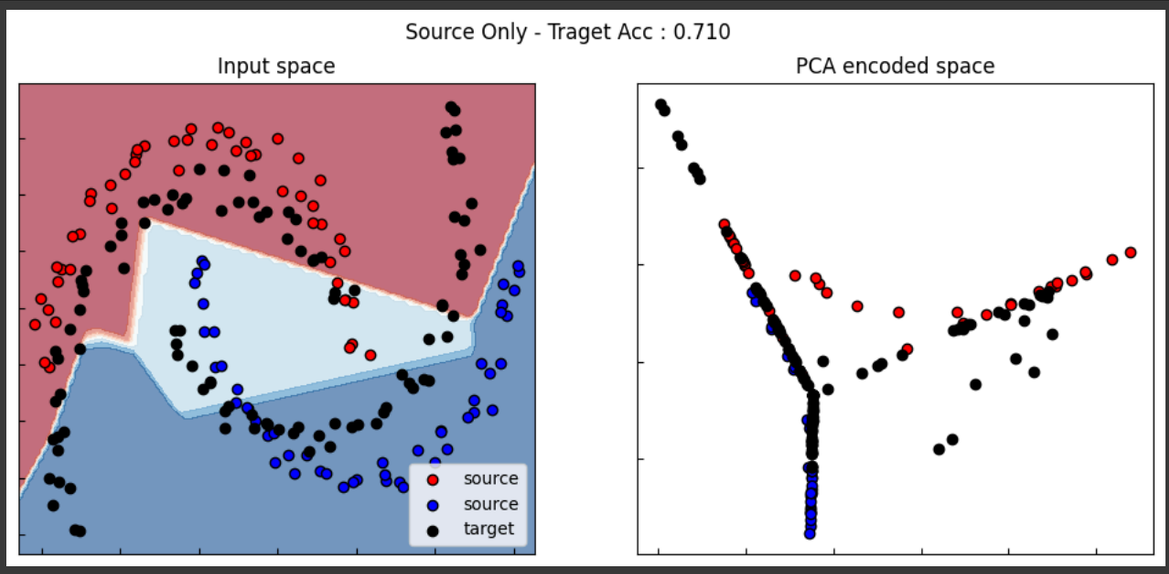
```
dcoral.fit(Xs, ys, Xt, yt, epochs=800, batch_size=34, verbose=0);
pd.DataFrame(dcoral.history_).plot(figsize=(8, 5))
plt.title("Training history", fontsize=14); plt.xlabel("Epochs"); plt.ylabel("Scores")
plt.legend(ncol=2)
plt.show()
```




```
yt_pred = dcoral.predict(Xt)
acc = accuracy_score(yt, yt_pred>0.5)

yp_grid = dcoral.predict(X_grid).reshape(100, 100)

```

```
X_pca = np.concatenate((dcoral.encoder_.predict(Xs),
                        dcoral.encoder_.predict(Xt)))
X_pca = PCA(2).fit_transform(X_pca)
```

```
cm = plt.cm.RdBu
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.set_title("Input space")
ax1.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=0.6)
ax1.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k', c="red")
ax1.scatter(Xs[ys==1, 0], Xs[ys==1, 1], label="source", edgecolors='k', c="blue")
ax1.scatter(Xt[:, 0], Xt[:, 1], label="target", edgecolors='k', c="black")
ax1.legend()
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction ='in')

ax2.set_title("PCA encoded space")
ax2.scatter(X_pca[:len(Xs), 0][ys==0], X_pca[:len(Xs), 1][ys==0],
            label="source", edgecolors='k', c="red")
ax2.scatter(X_pca[:len(Xs), 0][ys==1], X_pca[:len(Xs), 1][ys==1],
            label="source", edgecolors='k', c="blue")
ax2.scatter(X_pca[len(Xs):, 0], X_pca[len(Xs):, 1],
            label="target", edgecolors='k', c="black")
ax2.set_yticklabels([])
ax2.set_xticklabels([])
ax2.tick_params(direction ='in')
fig.suptitle("Source Only - Traget Acc : %.3f"%acc)
plt.show()
```




```
Commencez à coder ou à générer avec l'IA.
```

Adaptation de Domaine en Classification Binaire avec `make_circles`

Dans cette section, nous explorons l'application des méthodes d'adaptation de domaine (DA) fournies par le package **ADAPT** sur un problème de classification binaire en deux dimensions en utilisant les données générées par la fonction `make_circles` de `sklearn.datasets`. Ce scénario est particulièrement intéressant car les données en forme de cercles concentriques introduisent une séparation non linéaire, posant ainsi un défi supplémentaire pour les modèles de classification classiques, surtout dans le contexte de l'adaptation de domaine.

Les méthodes abordées incluent :

- **Source Only** : Entraînement sur les données source sans adaptation.
- **DANN (Domain-Adversarial Neural Network)** : Apprentissage de représentations invariantes au domaine via des techniques adversariales.
- **ADDA (Adversarial Discriminative Domain Adaptation)** : Une autre approche adversariale pour l'adaptation de domaine.
- **Deep CORAL** : Alignement des statistiques de second ordre entre les domaines source et cible. .

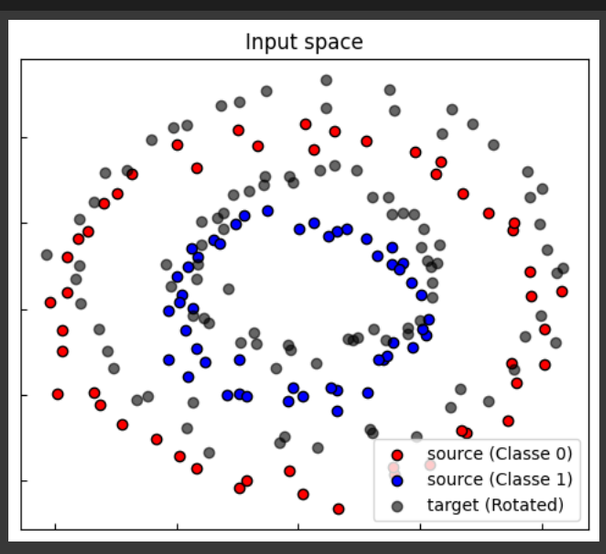
```
# === Importation des bibliothèques ===

# Standard libraries
import os
import numpy as np
import pandas as pd

# Visualisation et animations
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import rc

# Machine Learning
from sklearn.preprocessing import OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_circles

# Deep Learning avec TensorFlow
import tensorflow as tf
from tensorflow.keras import Model, Sequential
```

```
from tensorflow.keras.layers import (
    Dense, Input, Dropout, Conv2D, MaxPooling2D, Flatten, Reshape,
    GaussianNoise, BatchNormalization
)
from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adagrad
from tensorflow.keras.optimizers.legacy import Adam as LegacyAdam
from tensorflow.keras.constraints import MinMaxNorm
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import Callback

# Adaptation de domaine avec ADAPT
import adapt
from adapt.feature_based import DANN, ADDA, DeepCORAL
from adapt.instance_based import KMM

# Configuration de Matplotlib
rc('animation', html='jshtml')

```

Configuration Expérimentale

```
def make_circles_da(n_samples=100, rotation=30, noise=0.05, random_state=0):
    """
    Génère un problème d'adaptation de domaine avec make_circles.
    """
    # Générer les données sources
    Xs, ys = make_circles(n_samples=n_samples, noise=noise, random_state=random_state, factor=0.5)

    # Centrer les données source
    Xs[:, 0] -= 0.5

    # Appliquer la rotation aux données cibles
    theta = np.radians(rotation)
    cos_theta, sin_theta = np.cos(theta), np.sin(theta)
    rot_matrix = np.array(
        [[cos_theta, -sin_theta],
         [sin_theta, cos_theta]]
    )
    Xt = Xs.dot(rot_matrix)
    yt = ys

    return Xs, ys, Xt, yt
```

```


# Générer les données
Xs, ys, Xt, yt = make_circles_da()

# Définir les limites pour la visualisation
x_min, y_min = np.min([Xs.min(0), Xt.min(0)], 0)
x_max, y_max = np.max([Xs.max(0), Xt.max(0)], 0)
x_grid, y_grid = np.meshgrid(np.linspace(x_min-0.1, x_max+0.1, 100),
                             np.linspace(y_min-0.1, y_max+0.1, 100))
X_grid = np.stack([x_grid.ravel(), y_grid.ravel()], -1)

# Visualisation
fig, ax1 = plt.subplots(1, 1, figsize=(6, 5))
ax1.set_title("Input space")
ax1.scatter(Xs[ys == 0, 0], Xs[ys == 0, 1], label="source (Classe 0)", edgecolors='k', c="red")
ax1.scatter(Xs[ys == 1, 0], Xs[ys == 1, 1], label="source (Classe 1)", edgecolors='k', c="blue")
ax1.scatter(Xt[:, 0], Xt[:, 1], label="target (Rotated)", edgecolors='k', c="black", alpha=0.6)
ax1.legend(loc="lower right")
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction='in')
plt.show()
```



Définition du Modèle de Base

```
def get_task():
    model = Sequential()
```

```
    model.add(Flatten())
    model.add(Dense(10, activation="relu"))
    model.add(Dense(10, activation="relu"))
    model.add(Dense(1, activation="sigmoid"))
    return model
```

Méthode Source Only

Dans cette approche, nous entraînons le réseau de neurones uniquement sur les données source sans aucune adaptation de domaine. Cette méthode sert de référence pour évaluer l'impact des techniques d'adaptation de domaine. Comme attendu, le modèle peut mal classer les données cibles en raison de la différence entre les distributions source et cible.

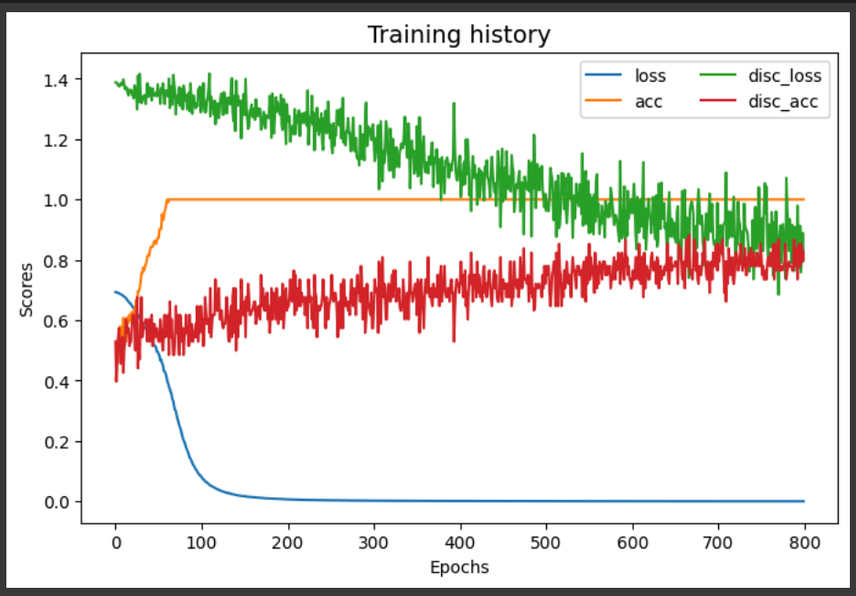
```

# Instead of Adam(0.001, beta_1=0.5):
optimizer = optimizers.Adam(0.001, beta_1=0.5)

```

```
src_only = DANN(
    task=get_task(),
    loss="bce",
    optimizer=optimizer,
    copy=True,
    lambda_=0.,
    metrics=["acc"],
    gamma=10.,
    random_state=0
)

```

```
/usr/local/lib/python3.10/dist-packages/adapt/feature_based/_dann.py:114: UserWarning: the `gamma` argument has been removed from DANN.
  warnings.warn("the `gamma` argument has been removed from DANN."
```

```
# Build the model by calling it on a sample of the input data
model = get_task()
model.build((None, Xs.shape[1]))  # Ensure input shape is correct

```

```

src_only.fit(Xs, ys, Xt, yt, epochs=800, batch_size=34, verbose=0);
pd.DataFrame(src_only.history_).plot(figsize=(8, 5))
plt.title("Training history", fontsize=14); plt.xlabel("Epochs"); plt.ylabel("Scores")
plt.legend(ncol=2)
plt.show()
```

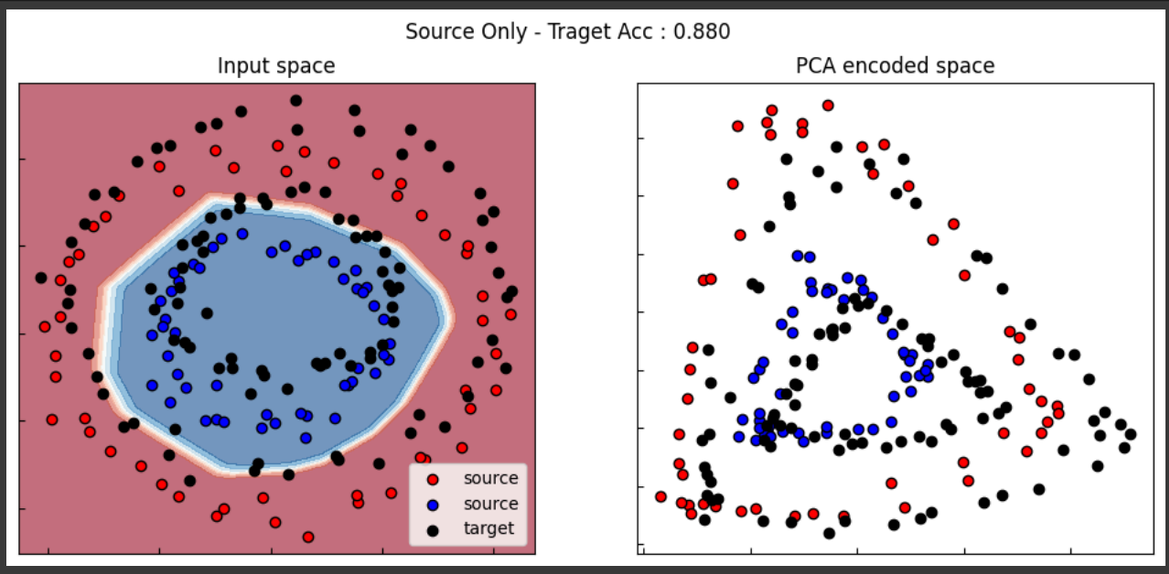



```

yt_pred = src_only.predict(Xt)
acc = accuracy_score(yt, yt_pred>0.5)

yp_grid = src_only.predict(X_grid).reshape(100, 100)

# Obtenir les features encodées par l'encodeur du DANN
X_features_src = src_only.encoder_.predict(Xs)
X_features_tgt = src_only.encoder_.predict(Xt)


X_pca = np.concatenate((X_features_src, X_features_tgt))
X_pca = PCA(2).fit_transform(X_pca)


```

```
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
```

```
cm = plt.cm.RdBu
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.set_title("Input space")
```

```
ax1.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=0.6)
ax1.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k', c="red")
ax1.scatter(Xs[ys==1, 0], Xs[ys==1, 1], label="source", edgecolors='k', c="blue")
ax1.scatter(Xt[:, 0], Xt[:, 1], label="target", edgecolors='k', c="black")
ax1.legend()
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction ='in')

ax2.set_title("PCA encoded space")
ax2.scatter(X_pca[:len(Xs), 0][ys==0], X_pca[:len(Xs), 1][ys==0],
            label="source", edgecolors='k', c="red")
ax2.scatter(X_pca[:len(Xs), 0][ys==1], X_pca[:len(Xs), 1][ys==1],
            label="source", edgecolors='k', c="blue")
ax2.scatter(X_pca[len(Xs):, 0], X_pca[len(Xs):, 1],
            label="target", edgecolors='k', c="black")
ax2.set_yticklabels([])
ax2.set_xticklabels([])
ax2.tick_params(direction ='in')
fig.suptitle("Source Only - Traget Acc : %.3f"%acc)
plt.show()
```

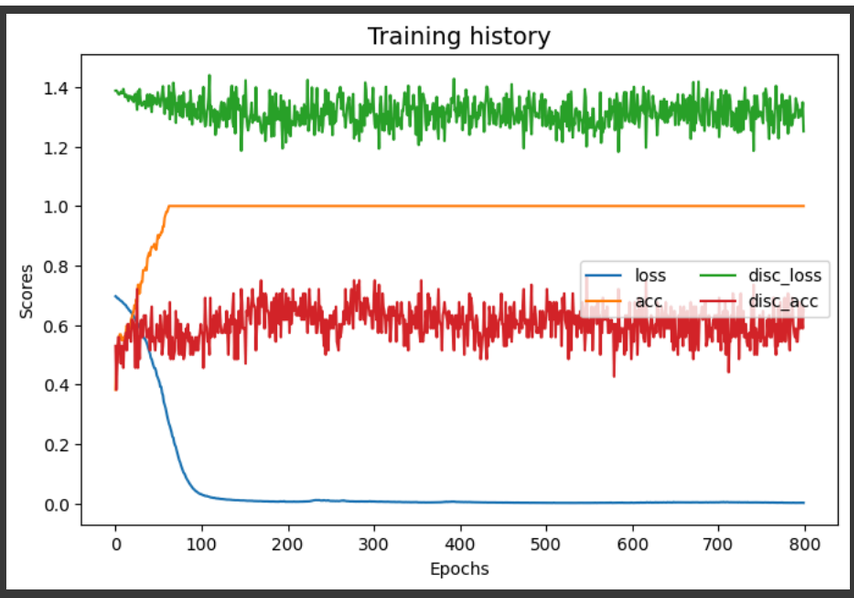


DANN

```

dann = DANN(
    task=get_task(),
    loss="bce",
    optimizer=Adam(0.001, beta_1=0.5),
    copy=True,
    lambda_=1.0,
    metrics=["acc"],
    gamma=10.0,
    random_state=0
)



```

```
/usr/local/lib/python3.10/dist-packages/adapt/feature_based/_dann.py:114: UserWarning: the `gamma` argument has been removed from DANN.
  warnings.warn("the `gamma` argument has been removed from DANN."
```

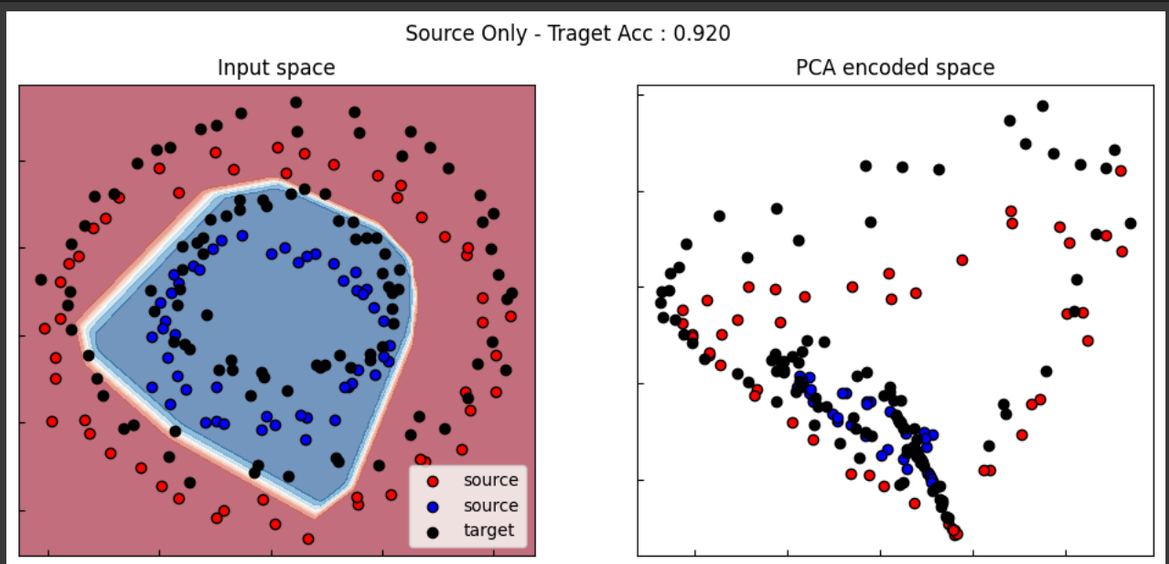
```
dann.fit(Xs, ys, Xt, yt, epochs=800, batch_size=34, verbose=0)
pd.DataFrame(dann.history_).plot(figsize=(8, 5))
plt.title("Training history", fontsize=14)
plt.xlabel("Epochs")
plt.ylabel("Scores")
plt.legend(ncol=2)
plt.show()
```




```
yt_pred = dann.predict(Xt)
acc = accuracy_score(yt, yt_pred>0.5)

yp_grid = dann.predict(X_grid).reshape(100, 100)

X_pca = np.concatenate((dann.encoder_.predict(Xs),
                        dann.encoder_.predict(Xt)))
X_pca = PCA(2).fit_transform(X_pca)
```

```
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
```

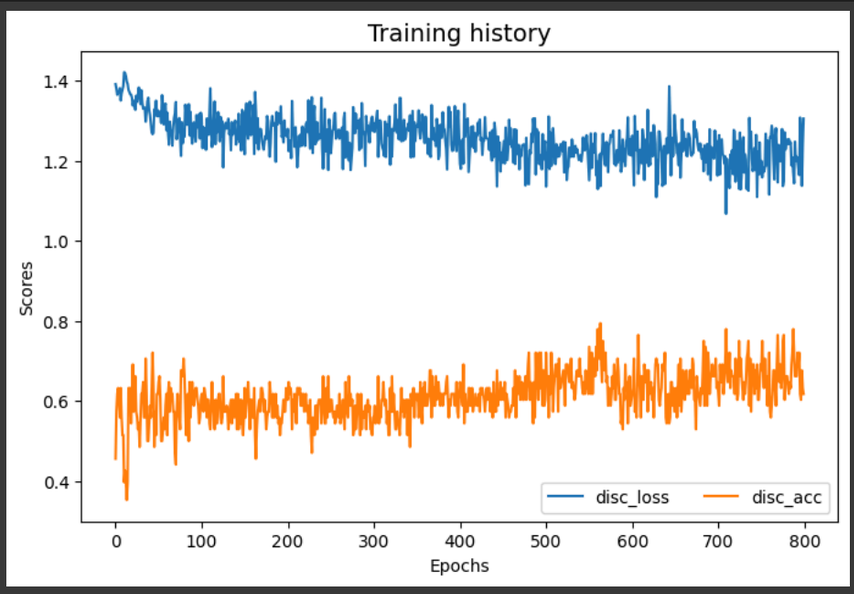
```
cm = plt.cm.RdBu
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.set_title("Input space")
ax1.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=0.6)
ax1.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k', c="red")
ax1.scatter(Xs[ys==1, 0], Xs[ys==1, 1], label="source", edgecolors='k', c="blue")
ax1.scatter(Xt[:, 0], Xt[:, 1], label="target", edgecolors='k', c="black")
ax1.legend()
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction ='in')

ax2.set_title("PCA encoded space")
ax2.scatter(X_pca[:len(Xs), 0][ys==0], X_pca[:len(Xs), 1][ys==0],
            label="source", edgecolors='k', c="red")
ax2.scatter(X_pca[:len(Xs), 0][ys==1], X_pca[:len(Xs), 1][ys==1],
            label="source", edgecolors='k', c="blue")
ax2.scatter(X_pca[len(Xs):, 0], X_pca[len(Xs):, 1],
            label="target", edgecolors='k', c="black")
ax2.set_yticklabels([])
ax2.set_xticklabels([])
ax2.tick_params(direction ='in')
fig.suptitle("Source Only - Traget Acc : %.3f"%acc)
plt.show()
```



ADDA

```
adda = ADDA(task=get_task(),
            loss="bce", optimizer=Adam(0.001, beta_1=0.5),
            copy=True, metrics=["acc"], random_state=0)
```

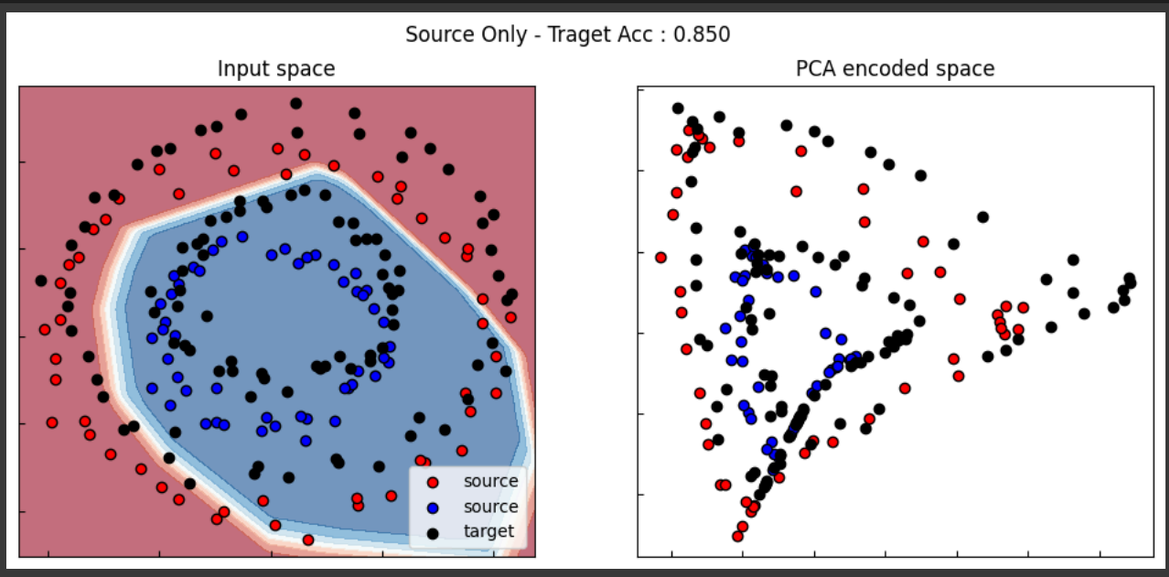
```
adda.fit(Xs, ys, Xt, yt, epochs=800, batch_size=34, verbose=0);
pd.DataFrame(adda.history_).plot(figsize=(8, 5))
plt.title("Training history", fontsize=14); plt.xlabel("Epochs"); plt.ylabel("Scores")
plt.legend(ncol=2)
plt.show()
```




```
yt_pred = adda.predict(Xt)
acc = accuracy_score(yt, yt_pred>0.5)

yp_grid = adda.predict(X_grid).reshape(100, 100)

X_pca = np.concatenate((adda.encoder_.predict(Xs),
                        adda.encoder_.predict(Xt)))
X_pca = PCA(2).fit_transform(X_pca)
```

```
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
```

```
cm = plt.cm.RdBu
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.set_title("Input space")
ax1.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=0.6)
ax1.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k', c="red")
ax1.scatter(Xs[ys==1, 0], Xs[ys==1, 1], label="source", edgecolors='k', c="blue")
ax1.scatter(Xt[:, 0], Xt[:, 1], label="target", edgecolors='k', c="black")
ax1.legend()
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction ='in')

ax2.set_title("PCA encoded space")
ax2.scatter(X_pca[:len(Xs), 0][ys==0], X_pca[:len(Xs), 1][ys==0],
            label="source", edgecolors='k', c="red")
ax2.scatter(X_pca[:len(Xs), 0][ys==1], X_pca[:len(Xs), 1][ys==1],
            label="source", edgecolors='k', c="blue")
ax2.scatter(X_pca[len(Xs):, 0], X_pca[len(Xs):, 1],
            label="target", edgecolors='k', c="black")
ax2.set_yticklabels([])
ax2.set_xticklabels([])
ax2.tick_params(direction ='in')
fig.suptitle("Source Only - Traget Acc : %.3f"%acc)
plt.show()
```



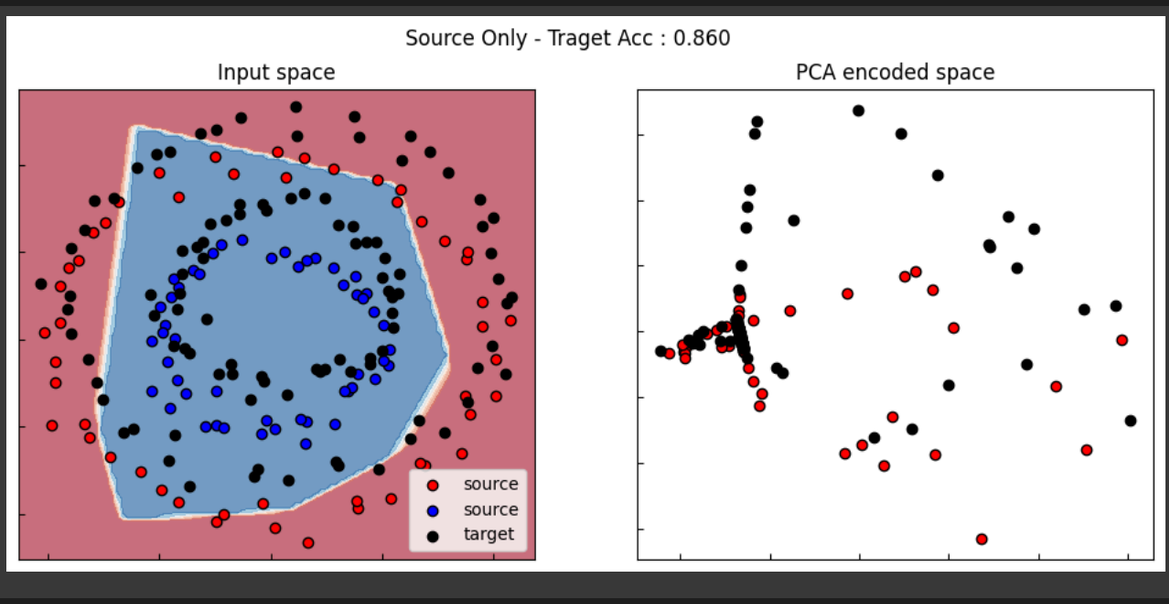
DeepCORAL

```
dcoral = DeepCORAL(encoder=src_only.encoder_,
                   task=src_only.task_, lambda_=1000.,
            loss="bce", optimizer=Adam(0.001, beta_1=0.5),
            copy=True, metrics=["acc"], random_state=0)
```

```
dcoral.fit(Xs, ys, Xt, yt, epochs=800, batch_size=34, verbose=0);
pd.DataFrame(dcoral.history_).plot(figsize=(8, 5))
plt.title("Training history", fontsize=14); plt.xlabel("Epochs"); plt.ylabel("Scores")
plt.legend(ncol=2)
plt.show()
```


```
yt_pred = dcoral.predict(Xt)
acc = accuracy_score(yt, yt_pred>0.5)

yp_grid = dcoral.predict(X_grid).reshape(100, 100)

X_pca = np.concatenate((dcoral.encoder_.predict(Xs),
                        dcoral.encoder_.predict(Xt)))
X_pca = PCA(2).fit_transform(X_pca)
```

```
cm = plt.cm.RdBu
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.set_title("Input space")
ax1.contourf(x_grid, y_grid, yp_grid, cmap=cm, alpha=0.6)
ax1.scatter(Xs[ys==0, 0], Xs[ys==0, 1], label="source", edgecolors='k', c="red")
ax1.scatter(Xs[ys==1, 0], Xs[ys==1, 1], label="source", edgecolors='k', c="blue")
ax1.scatter(Xt[:, 0], Xt[:, 1], label="target", edgecolors='k', c="black")
ax1.legend()
ax1.set_yticklabels([])
ax1.set_xticklabels([])
ax1.tick_params(direction ='in')

ax2.set_title("PCA encoded space")
ax2.scatter(X_pca[:len(Xs), 0][ys==0], X_pca[:len(Xs), 1][ys==0],
            label="source", edgecolors='k', c="red")
ax2.scatter(X_pca[:len(Xs), 0][ys==1], X_pca[:len(Xs), 1][ys==1],
            label="source", edgecolors='k', c="blue")
ax2.scatter(X_pca[len(Xs):, 0], X_pca[len(Xs):, 1],
            label="target", edgecolors='k', c="black")
ax2.set_yticklabels([])
ax2.set_xticklabels([])
ax2.tick_params(direction ='in')
fig.suptitle("Source Only - Traget Acc : %.3f"%acc)
plt.show()
```



```
Commencez à coder ou à générer avec l'IA.
```

Regression

Régression avec `make_regression_da`

Dans cette section, nous appliquons les méthodes d'adaptation de domaine (DA) du package **ADAPT** à un problème de régression unidimensionnelle. Nous utilisons des données synthétiques générées par la fonction `make_regression_da` de `adapt.utils` pour simuler un scénario d'adaptation de domaine où les distributions des données source et cible diffèrent.

Les méthodes abordées incluent :

- **Target Only** : Entraînement uniquement sur un petit ensemble de données cibles étiquetées.
- **Source Only** : Entraînement uniquement sur un grand ensemble de données source étiquetées.
- **All** : Entraînement sur la combinaison des données source et cible étiquetées.
- **RegularTransferNN** : Méthode basée sur les paramètres qui régularise les paramètres du modèle cible par rapport à un modèle pré-entraîné sur les données source.

```
# === Importation des bibliothèques ===

# Standard libraries
import numpy as np

# Visualisation et animations
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib import rc

# Configuration de Matplotlib
rc('animation', html='jshtml')

# Deep Learning avec TensorFlow
import tensorflow as tf
from tensorflow.keras import Sequential, Input, Model
from tensorflow.keras.layers import Dense, Reshape
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import Callback

# Adaptation de domaine avec ADAPT
from adapt.utils import make_regression_da
from adapt.parameter_based import RegularTransferNN

```

Configuration Expérimentale

Nous définissons un problème de régression synthétique unidimensionnelle en utilisant la fonction `make_regression_da` de `adapt.utils`. Cette fonction génère des données source et cible avec des distributions différentes pour simuler un scénario d'adaptation de domaine.

- **Données Source (`Xs`, `ys`)** : Un grand ensemble de données avec des étiquettes.
- **Données Cible (`Xt`, `yt`)** : Un petit ensemble de données avec des étiquettes, potentiellement différent en distribution par rapport aux données source.

Nous sélectionnons également un sous-ensemble de données cibles étiquetées pour les scénarios où seules quelques données cibles sont disponibles.

```


Xs, ys, Xt, yt = make_regression_da()
Xs = Xs.reshape(-1, 1).astype(np.float32)
Xt = Xt.reshape(-1, 1).astype(np.float32)
tgt_index_lab_ = np.random.choice(100,3)
Xt_lab = Xt[tgt_index_lab_]; yt_lab = yt[tgt_index_lab_]
```

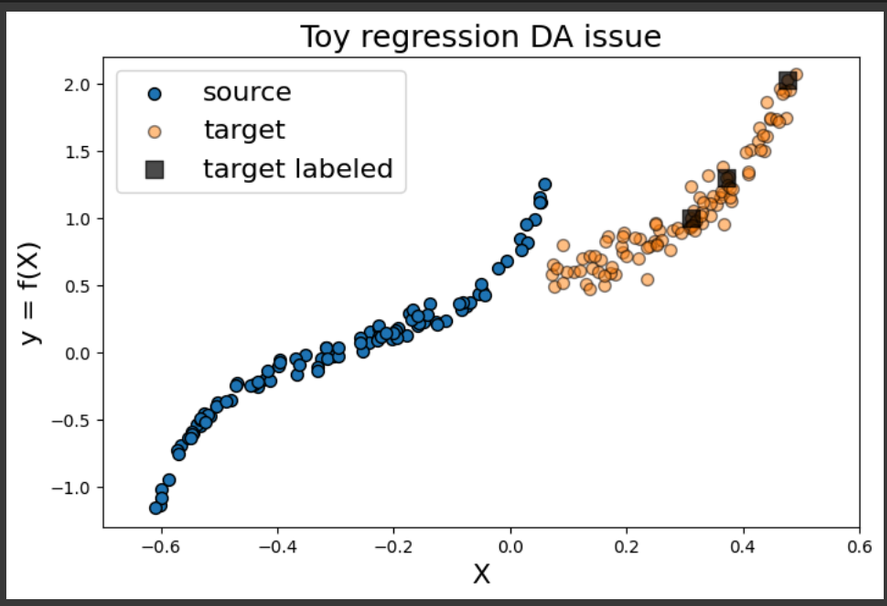
Fonction de Visualisation

La fonction `show` permet de visualiser les performances des algorithmes de régression sur le problème synthétique. Elle affiche les données source et cible, les prédictions du modèle, ainsi que les points cibles étiquetés. Cette visualisation facilite la compréhension de la manière dont le modèle s'adapte aux données cibles au fil des époques d'entraînement.

```

```

```
def show(ax, y_pred=None, X_src=Xs, weights_src=50, weights_tgt=100):
    ax.scatter(X_src, ys, s=weights_src, label="source", edgecolor="black")
    ax.scatter(Xt, yt, s=50, alpha=0.5, label="target", edgecolor="black")
    ax.scatter(Xt_lab, yt_lab, s=weights_tgt,
               c="black", marker="s", alpha=0.7, label="target labeled")
    if y_pred is not None:
        ax.plot(np.linspace(-0.7, 0.6, 100), y_pred, c="red", lw=3, label="predictions")
        index_ = np.abs(Xt - np.linspace(-0.7, 0.6, 100)).argmin(1)
        score = np.mean(np.abs(yt - y_pred[index_]))
        score = " -- Tgt MAE = %.2f"%score
    else:
        score = ""
    ax.set_xlim((-0.7,0.6))
    ax.set_ylim((-1.3, 2.2))
    ax.legend(fontsize=16)
    ax.set_xlabel("X", fontsize=16)
    ax.set_ylabel("y = f(X)", fontsize=16)
    ax.set_title("Toy regression DA issue"+score, fontsize=18)
    return ax
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
show(ax=ax)
plt.show()
```




```

def get_model():
    inputs = Input(shape=(1,))
    x = Dense(100, activation='elu')(inputs)
    x = Dense(100, activation='relu')(x)
    outputs = Dense(1)(x)
    model = Model(inputs, outputs)
    model.compile(optimizer=Adam(0.01), loss='mean_squared_error')
    return model
```

```

class SavePrediction(Callback):
    """
    Callbacks which stores predicted
    labels in history at each epoch.
    """
    def __init__(self):
        self.X = np.linspace(-0.7, 0.6, 100).reshape(-1, 1)
        self.custom_history_ = []
        super().__init__()

    def on_epoch_end(self, batch, logs={}):
        """Applied at the end of each epoch"""
        predictions = self.model.predict_on_batch(self.X).ravel()
        self.custom_history_.append(predictions)
```

Méthode Target Only

Dans cette approche, nous entraînons le réseau de neurones uniquement sur un petit ensemble de données cibles étiquetées. Comme prévu, cette méthode ne suffit pas à construire un modèle efficace sur l'ensemble du domaine cible en raison du nombre limité de données étiquetées.

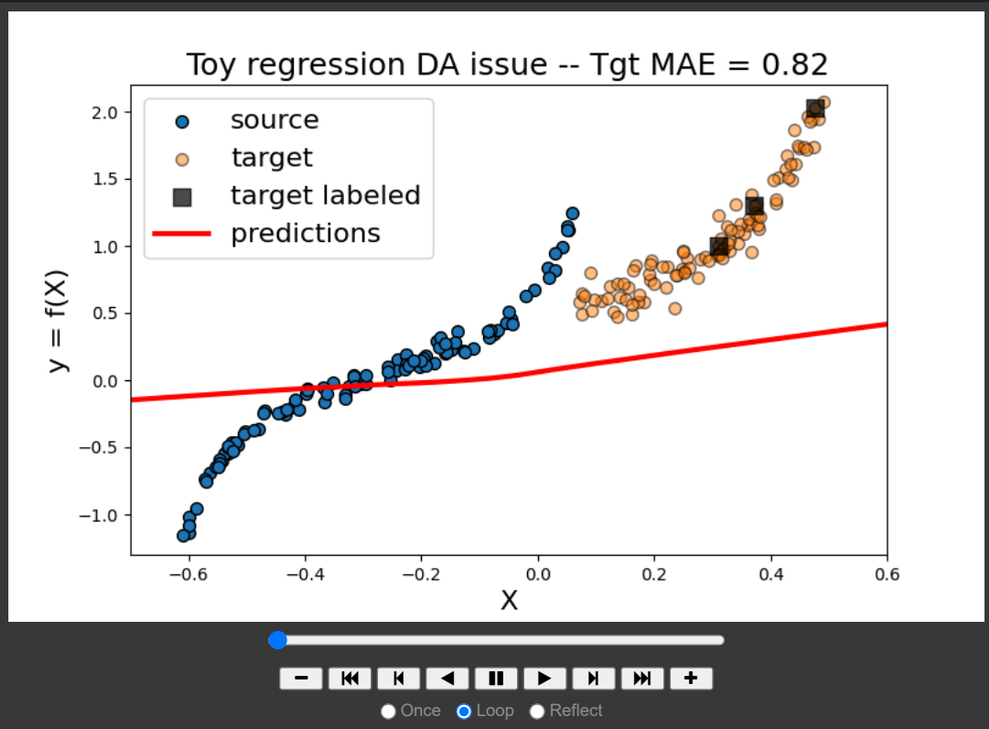
```
np.random.seed(0)
tf.random.set_seed(0)

model = get_model()
save_preds = SavePrediction()
model.fit(Xt_lab, yt_lab, callbacks=[save_preds], epochs=100, batch_size=64, verbose=0);
```

```
def animate(i, *fargs):
    ax.clear()
    y_pred = save_preds.custom_history_[i].ravel()
    if len(fargs)<1:
        show(ax, y_pred)
```

```
    else:
        show(ax, y_pred, **fargs[0])
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ani = animation.FuncAnimation(fig, animate, frames=100, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```



Once Loop Reflect

```
ani = animation.FuncAnimation(fig, animate, frames=100, repeat=False)
ani.save('tgtOnly.mp4', writer=writer)
```

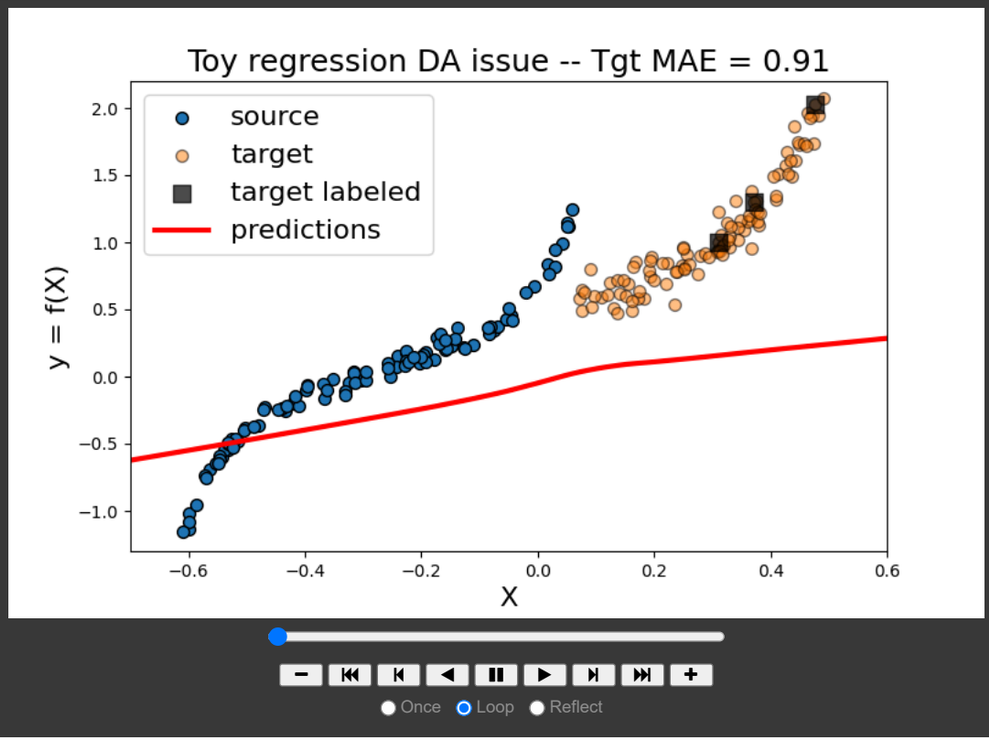
Méthode Source Only

Nous utilisons un grand ensemble de données source étiquetées pour entraîner le modèle de régression, sans aucune adaptation de domaine. Comme attendu, cette méthode échoue à fournir un modèle efficace sur le domaine cible en raison de la divergence entre les distributions des données source et cible.

```
np.random.seed(0)
tf.random.set_seed(0)

model = get_model()
save_preds = SavePrediction()
model.fit(Xs, ys, callbacks=[save_preds], epochs=100, batch_size=100, verbose=0);
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ani = animation.FuncAnimation(fig, animate, frames=100, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```



Once Loop Reflect

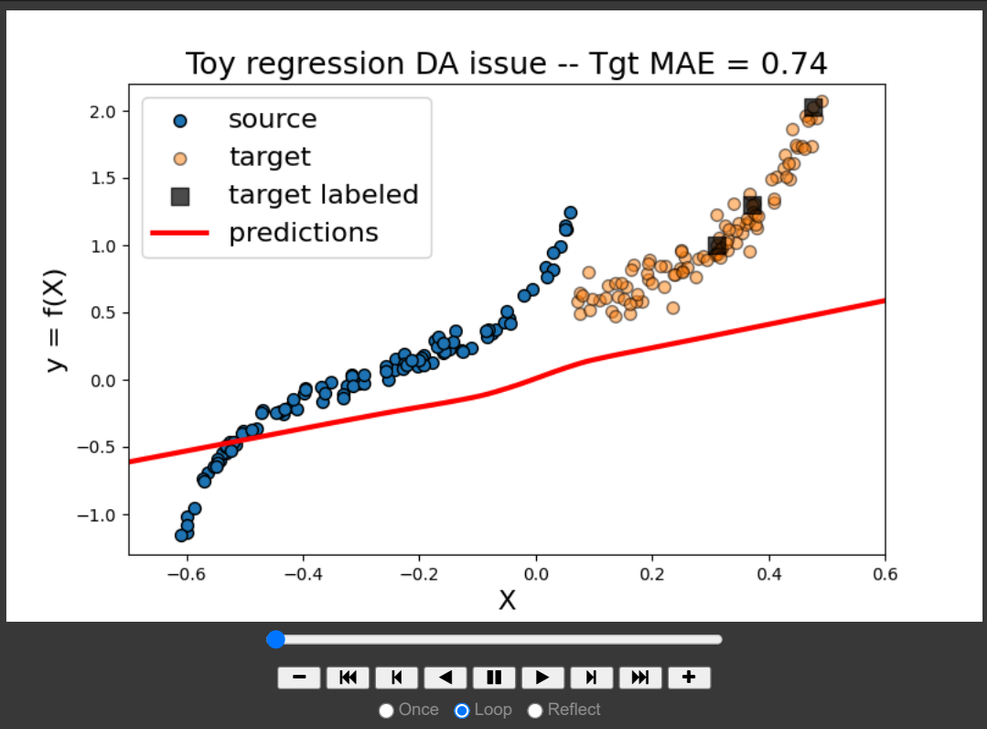
Méthode All

Dans cette approche, nous combinons les données source et les quelques données cibles étiquetées pour entraîner le modèle. Cependant, comme les données source dominent en nombre, le modèle est davantage influencé par les données source, ce qui empêche une adaptation suffisante au domaine cible.

```
np.random.seed(0)
tf.random.set_seed(0)

model = get_model()
save_preds = SavePrediction()
model.fit(np.concatenate((Xs, Xt_lab)),
          np.concatenate((ys, yt_lab)),
          callbacks=[save_preds],
          epochs=100, batch_size=110, verbose=0);
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ani = animation.FuncAnimation(fig, animate, frames=100, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```



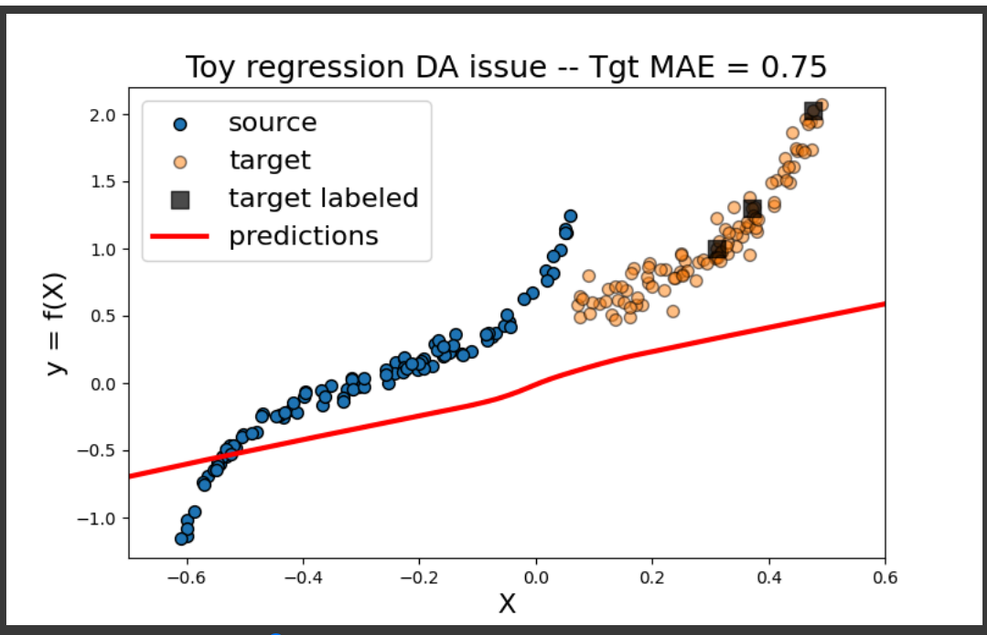
Méthode RegularTransferNN

Enfin, nous considérons la méthode basée sur les paramètres **RegularTransferNN**. Cette méthode entraîne le modèle cible en régularisant la distance euclidienne entre les paramètres du modèle cible et ceux d'un modèle pré-entraîné sur les données source. Cela permet au modèle cible de bénéficier des connaissances acquises sur les données source tout en s'adaptant aux spécificités du domaine cible.

```

np.random.seed(0)
tf.random.set_seed(0)

save_preds = SavePrediction()
model_0 = get_model()
model_0.fit(Xs.reshape(-1, 1), ys, callbacks=[save_preds], epochs=100, batch_size=110, verbose=0);
model = RegularTransferNN(model_0, lambdas=1.0, random_state=0)
model.fit(Xt_lab, yt_lab, callbacks=[save_preds], epochs=100, batch_size=110, verbose=0);
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ani = animation.FuncAnimation(fig, animate, frames=200, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```



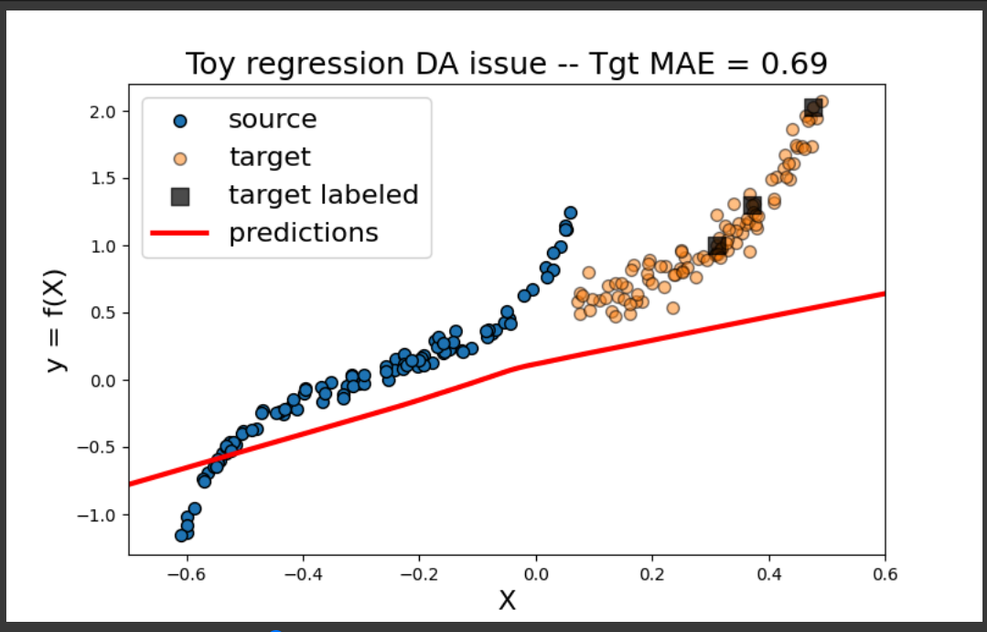
Once Loop Reflect

```
np.random.seed(0)
tf.random.set_seed(0)

model = get_model()
save_preds = SavePrediction()
model.fit(Xs, ys, callbacks=[save_preds], epochs=100, batch_size=100, verbose=0)

```

```
<keras.src.callbacks.History at 0x79d6eb5ed0f0>
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ani = animation.FuncAnimation(fig, animate, frames=100, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```



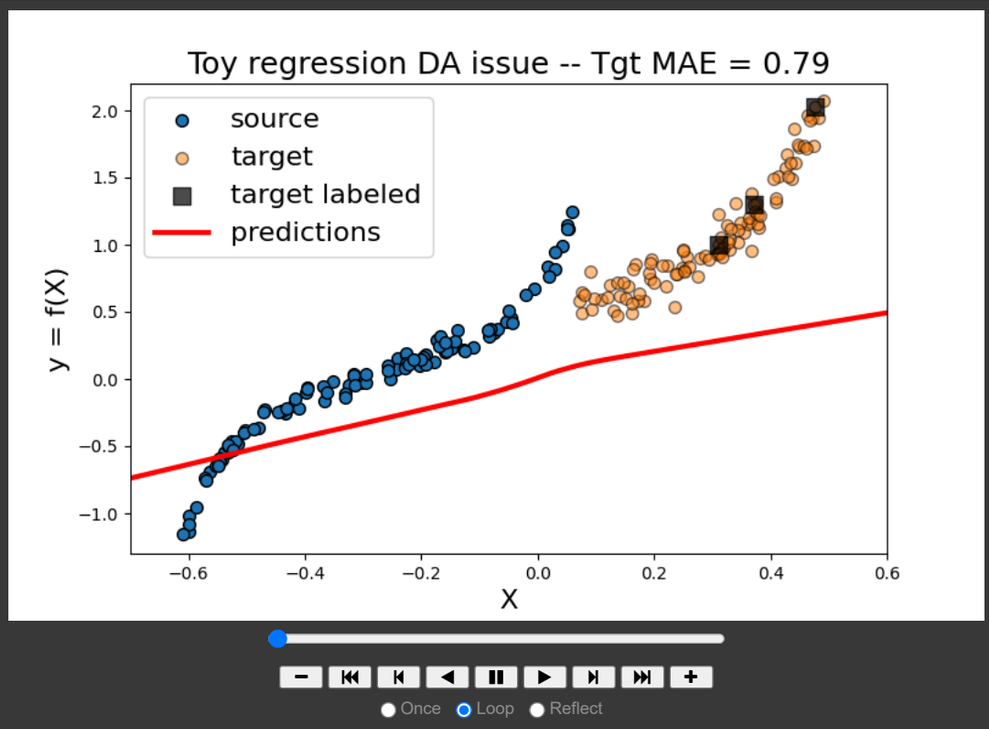
Once Loop Reflect

All

```
np.random.seed(0)
tf.random.set_seed(0)

model = get_model()
save_preds = SavePrediction()
model.fit(np.concatenate((Xs, Xt_lab)),
          np.concatenate((ys, yt_lab)),
          callbacks=[save_preds],
          epochs=100, batch_size=110, verbose=0);
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ani = animation.FuncAnimation(fig, animate, frames=100, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```



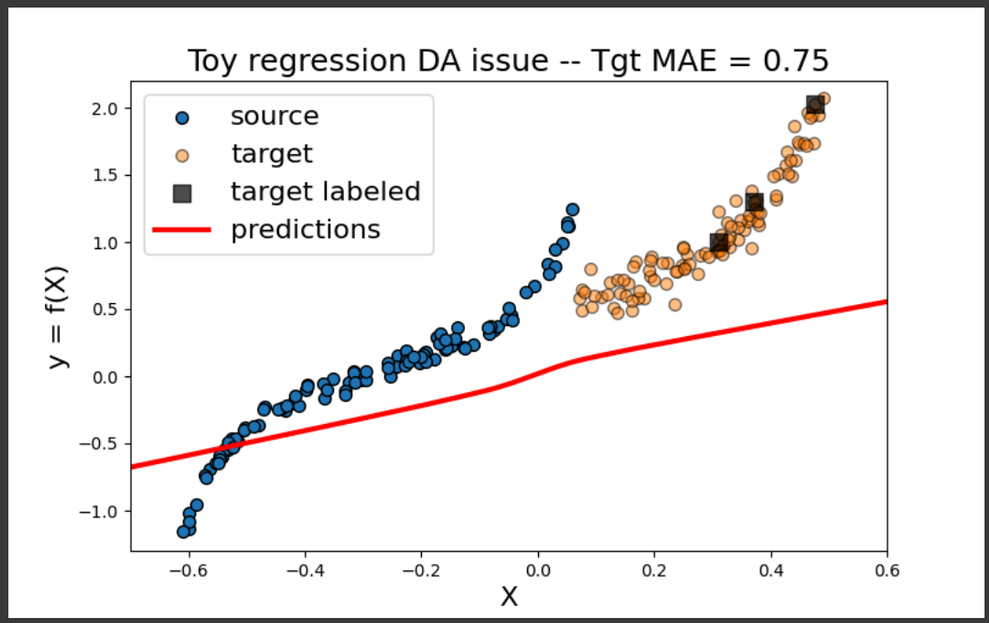
Once Loop Reflect

RegularTransferNN

```
np.random.seed(0)
tf.random.set_seed(0)

save_preds = SavePrediction()
model_0 = get_model()
model_0.fit(Xs.reshape(-1, 1), ys, callbacks=[save_preds], epochs=100, batch_size=110, verbose=0);
model = RegularTransferNN(model_0, lambdas=1.0, random_state=0)
model.fit(Xt_lab, yt_lab, callbacks=[save_preds], epochs=100, batch_size=110, verbose=0);
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ani = animation.FuncAnimation(fig, animate, frames=200, blit=False, repeat=True)
plt.close(fig)
```

```
ani
```



Once Loop Reflect

Domain Adaptation Real-World Examples

Sample_bias_examples

Ce notebook démontre comment gérer le biais d'échantillonnage en utilisant la toolbox `adapt`. Nous présentons deux exemples :

1. **Application des techniques de biais d'échantillonnage sur le jeu de données Diabetes.**
2. **Test sur un jeu de données différent (jeu de données d'images Office-31).**

Exemple 1 : Jeu de Données Diabetes avec la Toolbox `adapt`

Dans cet exemple, nous introduisons un biais d'échantillonnage dans le jeu de données Diabetes et appliquons des techniques d'adaptation de domaine en utilisant la toolbox `adapt` pour atténuer ce biais.

```
# === Importation des bibliothèques ===
# Standard libraries
import numpy as np
import pandas as pd

# Visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# Statistiques
from scipy.stats import gaussian_kde

# Scikit-learn
from sklearn.datasets import load_diabetes
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge

# Adaptation de domaine
from adapt.instance_based import KLIEP, KMM
import adapt.metrics
from adapt.metrics import make_uda_scorer, neg_j_score

# Reload nécessaire pour l'utilisation spécifique des métriques d'ADAPT
import importlib
importlib.reload(adapt.metrics)
```

```
# Charger le jeu de données Diabetes sous forme de DataFrame pandas
data = load_diabetes(as_frame=True)
print(data["DESCR"])

X = data["data"]
y = data['target']
```

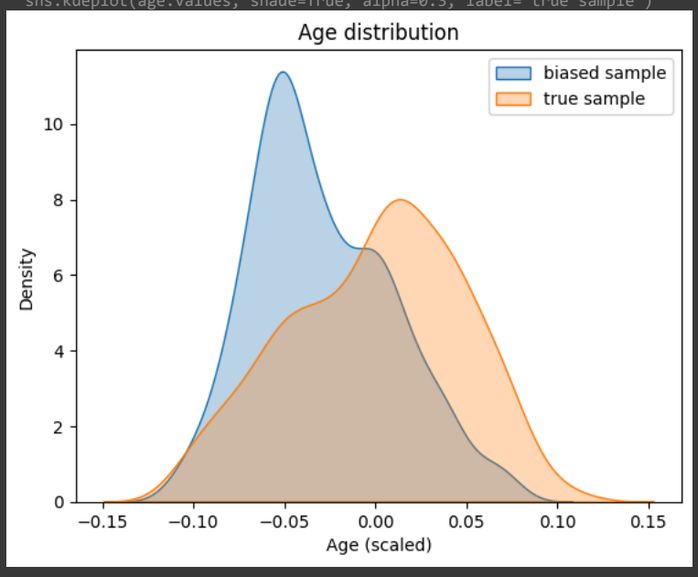
```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression one year after baseline

  :Attribute Information:
      - age     age in years
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, total serum cholesterol
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, total cholesterol / HDL
      - s5      ltg, possibly log of serum triglycerides level
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of `n_sample

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussic
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

Nous introduisons un biais basé sur la caractéristique `age` en modifiant la probabilité d'échantillonnage.

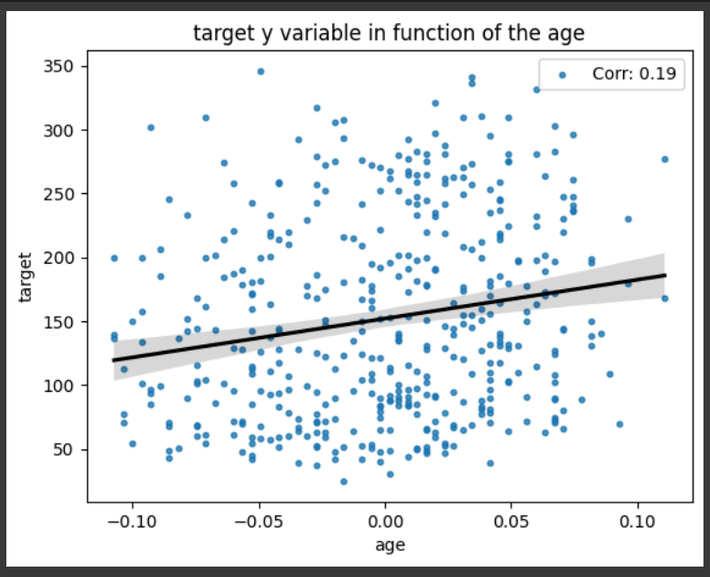
```
np.random.seed(123)
sample_bias = np.exp(-20*np.abs(X["age"]+0.06))
biased_index = np.random.choice(X.index, len(X), p=sample_bias/sample_bias.sum())
biased_X = X.loc[biased_index]
biased_y = y.loc[biased_index]

biased_age = biased_X["age"]
age = X["age"]
```

```
sns.kdeplot(biased_age.values, shade=True, alpha=0.3, label="biased sample")
sns.kdeplot(age.values, shade=True, alpha=0.3, label="true sample")
plt.title("Age distribution"); plt.xlabel("Age (scaled)");
plt.legend(); plt.show()
```

```
<ipython-input-63-d370cee291a0>:1: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(biased_age.values, shade=True, alpha=0.3, label="biased sample")
<ipython-input-63-d370cee291a0>:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(age.values, shade=True, alpha=0.3, label="true sample")
```

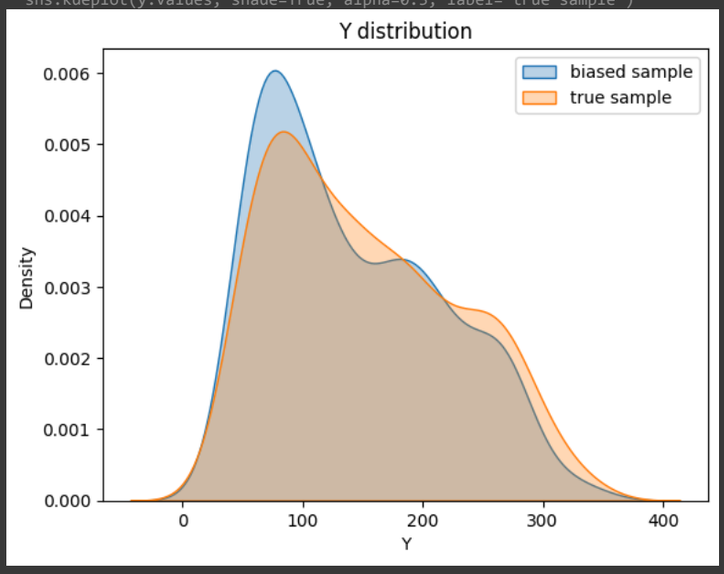



```
corr = np.corrcoef(X["age"], y)[0,1]
sns.regplot(x=X["age"], y=y, label="Corr: %.2f"%corr,
            scatter_kws={"s":10}, line_kws={"color":"k"})
plt.title("target y variable in function of the age");
plt.legend(); plt.show();
```




```
sns.kdeplot(biased_y.values, shade=True, alpha=0.3, label="biased sample")
sns.kdeplot(y.values, shade=True, alpha=0.3, label="true sample")
plt.title("Y distribution"); plt.xlabel("Y");
plt.legend(); plt.show()
```

```
<ipython-input-65-74b9b64d55fc>:1: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(biased_y.values, shade=True, alpha=0.3, label="biased sample")
<ipython-input-65-74b9b64d55fc>:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(y.values, shade=True, alpha=0.3, label="true sample")
```



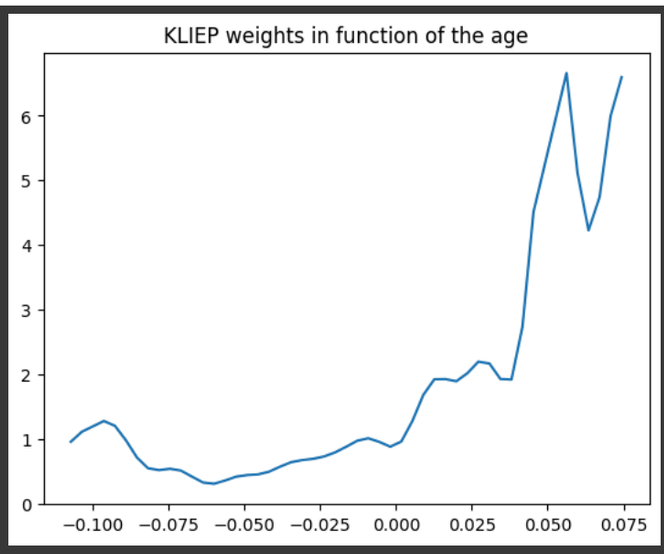

```
pd.concat((
pd.DataFrame(y).describe(percentiles=[0.5, 0.75, 0.9]),
pd.DataFrame(biased_y.values, columns=["biased target"]).describe(percentiles=[0.5, 0.75, 0.9])
), axis=1)
```

	target	biased target
count	442.000000	442.000000
mean	152.133484	142.389140
std	77.093005	74.338154
min	25.000000	25.000000
50%	140.500000	125.000000
75%	211.500000	200.000000
90%	265.000000	258.000000
max	346.000000	346.000000

```

kliep = KLIEP(kernel="rbf", gamma=[10**(i-4) for i in range(9)], random_state=0)
kliep_weights = kliep.fit_weights(biased_age.values.reshape(-1, 1), age.values.reshape(-1, 1))
```

```
Cross Validation process...
Parameters {'gamma': 0.0001} -- J-score = 0.000 (0.000)
Parameters {'gamma': 0.001} -- J-score = 0.000 (0.000)
Parameters {'gamma': 0.01} -- J-score = 0.000 (0.000)
Parameters {'gamma': 0.1} -- J-score = 0.001 (0.000)
Parameters {'gamma': 1} -- J-score = 0.006 (0.001)
Parameters {'gamma': 10} -- J-score = 0.058 (0.008)
Parameters {'gamma': 100} -- J-score = 0.265 (0.043)
Parameters {'gamma': 1000} -- J-score = 0.318 (0.064)
Parameters {'gamma': 10000} -- J-score = 0.444 (0.197)
```

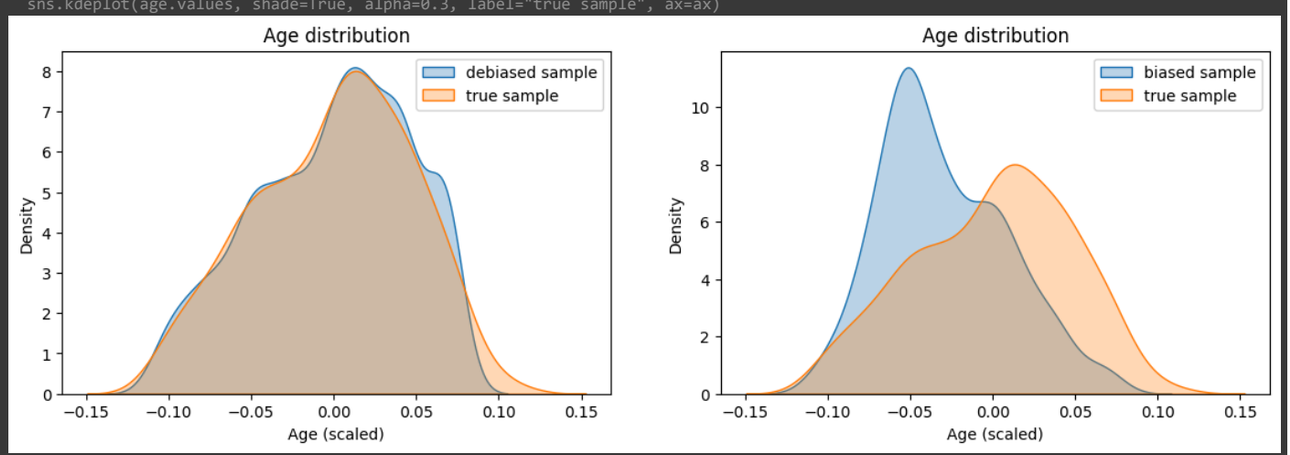
```
sns.lineplot(x=biased_age.values, y=kliep_weights)
plt.title("KLIEP weights in function of the age");
plt.show();
```




```
np.random.seed(123)
debiasing_index = np.random.choice(len(biased_age), 2000, p=kliep_weights/kliep_weights.sum())
debiased_age = biased_age.iloc[debiasing_index]
debiased_y = biased_y.iloc[debiasing_index]
```

```
fig, axes = plt.subplots(1, 2, figsize=(14, 4))
for ax, x, label in zip(axes, [debiased_age, biased_age], ["debiased sample", "biased sample"]):
    sns.kdeplot(x.values, shade=True, alpha=0.3, label=label, ax=ax)
    sns.kdeplot(age.values, shade=True, alpha=0.3, label="true sample", ax=ax)
    ax.set_title("Age distribution"); ax.set_xlabel("Age (scaled)"); ax.legend();
plt.show()
```

```
<ipython-input-70-92b41521e28a>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(x.values, shade=True, alpha=0.3, label=label, ax=ax)
<ipython-input-70-92b41521e28a>:4: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(age.values, shade=True, alpha=0.3, label="true sample", ax=ax)
<ipython-input-70-92b41521e28a>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(x.values, shade=True, alpha=0.3, label=label, ax=ax)
<ipython-input-70-92b41521e28a>:4: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(age.values, shade=True, alpha=0.3, label="true sample", ax=ax)
```

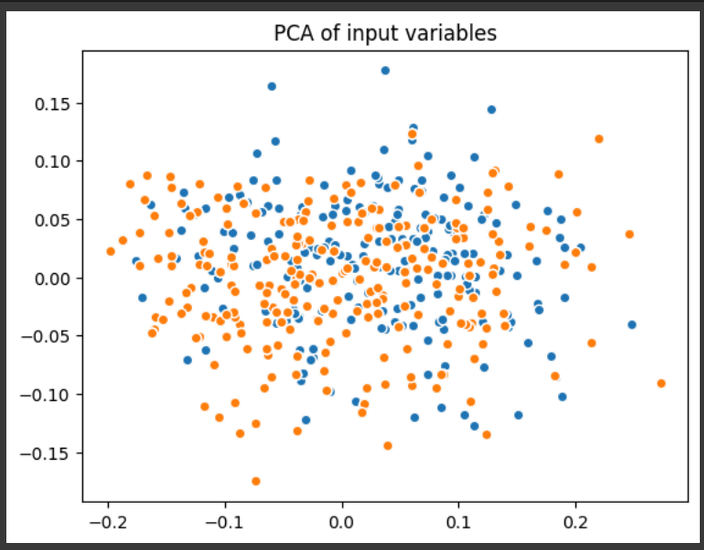



```
pd.concat((
pd.DataFrame(y.values, columns=["target"]).describe(percentiles=[0.5, 0.75, 0.9]),
pd.DataFrame(debiased_y.values, columns=["debiased target"]).describe(percentiles=[0.5, 0.75, 0.9]),
pd.DataFrame(biased_y.values, columns=["biased target"]).describe(percentiles=[0.5, 0.75, 0.9])
), axis=1)
```

	target	debiased target	biased target
count	442.000000	2000.000000	442.000000
mean	152.133484	149.863500	142.389140
std	77.093005	77.334075	74.338154
min	25.000000	25.000000	25.000000
50%	140.500000	134.500000	125.000000
75%	211.500000	219.000000	200.000000
90%	265.000000	265.000000	258.000000
max	346.000000	346.000000	346.000000

```
X = data["data"]
y = data['target']

np.random.seed(0)
sample_bias = np.ones(len(X))
sample_bias[X.sex < 0] *= 5.
X = X.drop(["sex"], axis=1)
biased_index = np.random.choice(X.index, len(X), p=sample_bias/sample_bias.sum())
biased_X = X.loc[biased_index]
biased_y = y.loc[biased_index]
```

```

X_pca = PCA(2).fit_transform(np.concatenate((X, biased_X)))
plt.scatter(X_pca[:len(X), 0], X_pca[:len(X), 1], edgecolor="w", label="Unbiased dataset")
plt.scatter(X_pca[len(X):, 0], X_pca[len(X):, 1], edgecolor="w", label="Biased dataset")
plt.title("PCA of input variables"); plt.show();
```




```
kmm = KMM(
    estimator=Ridge(0.1),
    Xt=X,
    kernel="rbf",  # Gaussian kernel
    gamma=0.,
    verbose=0,
    random_state=0
)

# Create a score function from the neg_j_score metric and biased_X, X
score = make_uda_scorer(neg_j_score, biased_X, X)

# Launch the gridsearch on four gamma parameters [0.1, 1., 10., 100., 1000.]
gs = GridSearchCV(kmm, {"gamma": [0.1, 1., 10., 100., 1000.]},
                  scoring=score,
                  return_train_score=True, cv=5, verbose=0)
gs.fit(biased_X, biased_y);

# Print results (the neg_j_score is given in "train_score")
keys = ["params", 'mean_train_score', 'std_train_score']
results =  [v for k, v in gs.cv_results_.items() if k in keys]
best = results[1].argmin()
print("Best Params %s -- Score %.3f (%.3f)"%
      (str(results[0][best]), results[1][best], results[2][best]))
print("-"*50)
for p, mu, std in zip(*results):
    print("Params %s -- Score %.3f (%.3f)"%(str(p), mu*1000., std*1000.))
```

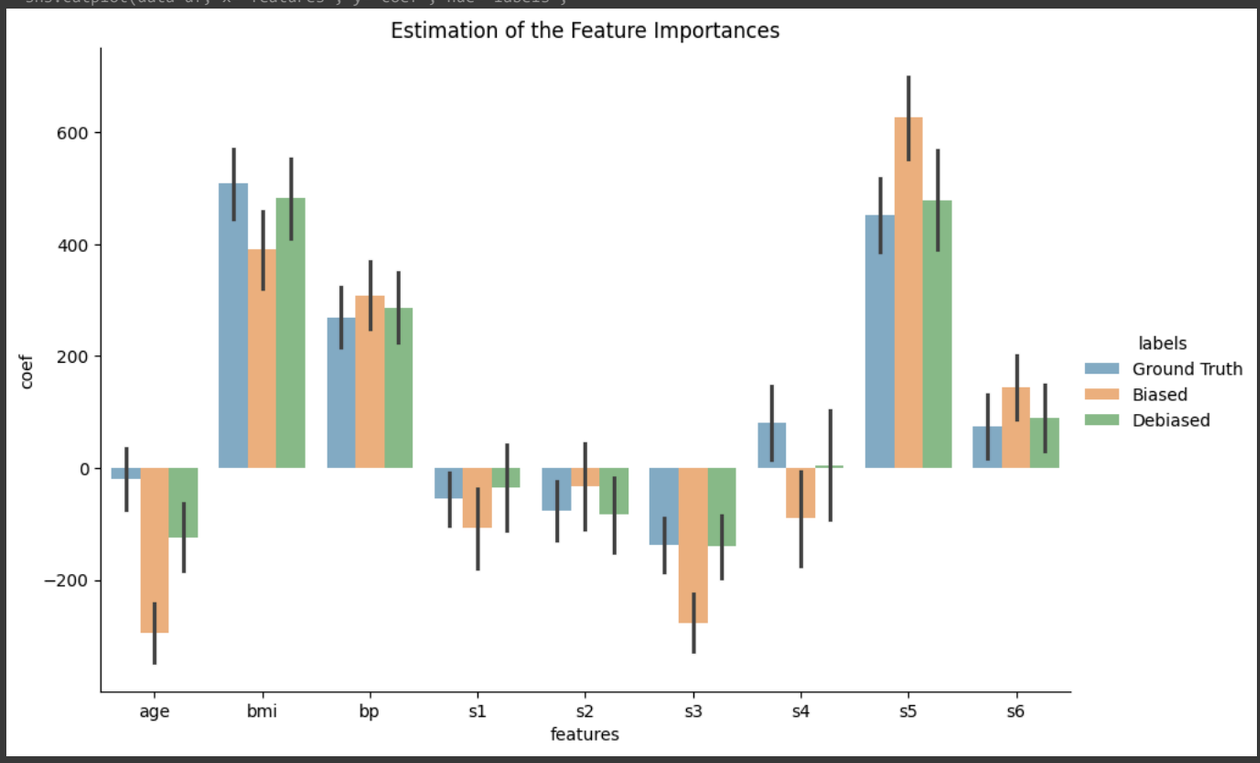
Afficher la sortie masquée

```
kmm = KMM(
    estimator=Ridge(),
    Xt=X.values,
    kernel="rbf",
    gamma=100.,
    verbose=0,
    random_state=0
)
weights = kmm.fit_weights(biased_X, X);
```

```
true_coefs = []
biased_coefs = []
debiased_coefs = []

for _ in range(100):
    bs = np.random.choice(X.index, len(X))
    biased_bs = np.random.choice(biased_X.index, len(X))
    debiased_bs = np.random.choice(biased_X.index, len(biased_X), p=weights/weights.sum())
    true_coefs.append(Ridge(0.1).fit(X.loc[bs], y.loc[bs]).coef_)
    biased_coefs.append(Ridge(0.1).fit(biased_X.loc[biased_bs], biased_y.loc[biased_bs]).coef_)
    debiased_coefs.append(Ridge(0.1).fit(biased_X.loc[debiased_bs], biased_y.loc[debiased_bs]).coef_)
```

```
arr = np.concatenate((true_coefs, biased_coefs, debiased_coefs)).ravel()
x = list(X.columns)*(len(true_coefs)+len(biased_coefs)+len(debiased_coefs))
labels = (["Ground Truth"]*len(true_coefs)*9 + ["Biased"]*len(biased_coefs)*9 +
          ["Debiased"]*len(debiased_coefs)*9)
df = pd.DataFrame([arr, x, labels], index=["coef", "features", "labels"]).transpose()
sns.catplot(data=df, x="features", y="coef", hue="labels",
            kind="bar", ci="sd", alpha=.6, height=6, aspect=1.5)
plt.title("Estimation of the Feature Importances"); plt.show();
```

```
<ipython-input-78-efb64bc383a6>:6: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  sns.catplot(data=df, x="features", y="coef", hue="labels",
```




```
Commencez à coder ou à générer avec l'IA.
```

Exemple 2 : Biais d'Échantillonnage sur un Jeu de Données Différent (Office-31)

Dans cet exemple, nous appliquons des techniques de biais d'échantillonnage au jeu de données d'images Office-31. Le jeu de données Office-31 comprend des images provenant de trois domaines : Amazon, DSLR et Webcam.

```
# === Importation des bibliothèques ===

# Suppression des avertissements
import warnings
warnings.filterwarnings("ignore")

# Bibliothèques standards
import os
import numpy as np
import pandas as pd

# Visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# Traitement d'images
from PIL import Image
from torchvision import transforms

# Machine Learning
from sklearn.linear_model import Ridge
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.preprocessing import LabelEncoder

# Adaptation de domaine
from adapt.instance_based import KLIEP, KMM

```

```
# Gestion des datasets
from torch.utils.data import Dataset, DataLoader

```

```
import tarfile

tar_file_path = "/content/domain_adaptation_images.tar.gz"

with tarfile.open(tar_file_path, "r:gz") as tar_ref:
    tar_ref.extractall("office31")

```

```
data_dir = "office31"  # Dossier principal contenant amazon, dslr, webcam
domains = ["amazon", "dslr", "webcam"]

image_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

class Office31Dataset(Dataset):
    def __init__(self, root_dir, domain, transform=None):
        self.root_dir = os.path.join(root_dir, domain, "images")
        self.transform = transform
        self.images = []
        self.labels = []

        for label in os.listdir(self.root_dir):
            label_path = os.path.join(self.root_dir, label)
            if os.path.isdir(label_path):
                for img_file in os.listdir(label_path):
                    img_path = os.path.join(label_path, img_file)
                    self.images.append(img_path)
                    self.labels.append(label)

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_path = self.images[idx]
        label = self.labels[idx]

        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)

        return image, label

datasets = {}
for domain in domains:
    datasets[domain] = Office31Dataset(data_dir, domain, transform=image_transforms)

dataloaders = {domain: DataLoader(datasets[domain], batch_size=32, shuffle=True) for domain in domains}

# Afficher un batch pour validation
for images, labels in dataloaders['amazon']:
    print(f"Batch d'images: {images.shape}")
    print(f"Batch de labels: {labels[:5]}")
    break
```

```
Batch d'images: torch.Size([32, 3, 224, 224])
Batch de labels: ('paper_notebook', 'bike', 'mobile_phone', 'trash_can', 'headphones')
```

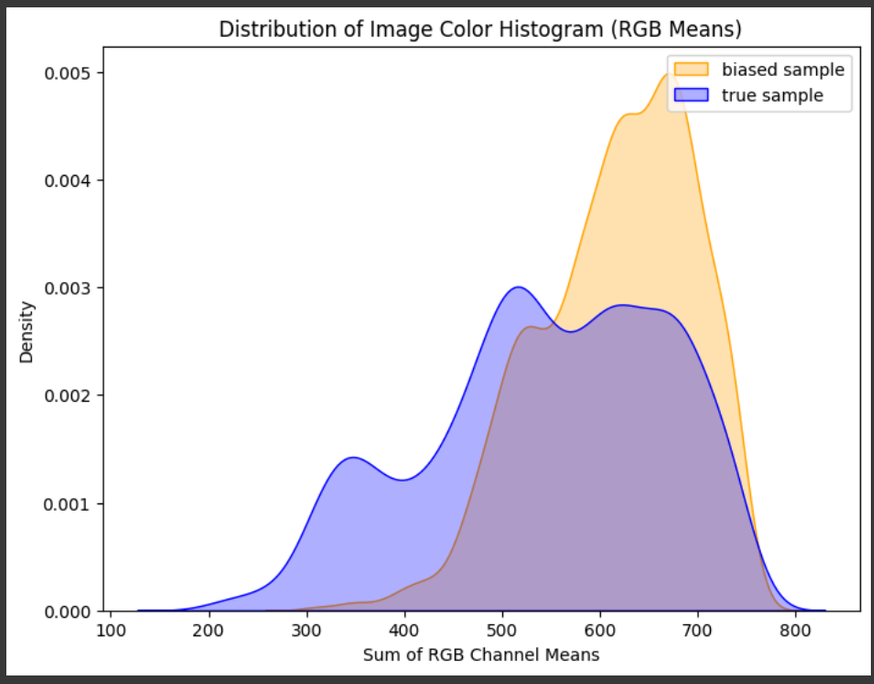
```
def compute_color_histogram(image_path):
    image = Image.open(image_path).convert("RGB")
    r, g, b = image.split()
    r_hist = np.array(r).mean()
    g_hist = np.array(g).mean()
    b_hist = np.array(b).mean()
    return r_hist, g_hist, b_hist

image_paths = []
image_labels = []
r_means, g_means, b_means = [], [], []

for domain in domains:  # amazon, dslr, webcam
    domain_dir = os.path.join(data_dir, domain, "images")
    for label in os.listdir(domain_dir):
        label_path = os.path.join(domain_dir, label)
        if os.path.isdir(label_path):
            for img_file in os.listdir(label_path):
                img_path = os.path.join(label_path, img_file)
                r_mean, g_mean, b_mean = compute_color_histogram(img_path)

                image_paths.append(img_path)
                image_labels.append(label)
                r_means.append(r_mean)
                g_means.append(g_mean)
                b_means.append(b_mean)

# Créer un DataFrame
df_images = pd.DataFrame({
    "path": image_paths,
    "r_mean": r_means,
    "g_mean": g_means,
    "b_mean": b_means,
    "label": image_labels
```

```
})

feature = df_images["r_mean"] + df_images["g_mean"] + df_images["b_mean"]
feature_mean = feature.mean()

# Distribution biaisée
sample_bias = np.exp(-(feature - (feature_mean + feature.std()))**2 / (2 * feature.std()**2))

# Échantillon biaisé
np.random.seed(123)
biased_index = np.random.choice(df_images.index, len(df_images), p=sample_bias / sample_bias.sum())
biased_df = df_images.loc[biased_index]
```

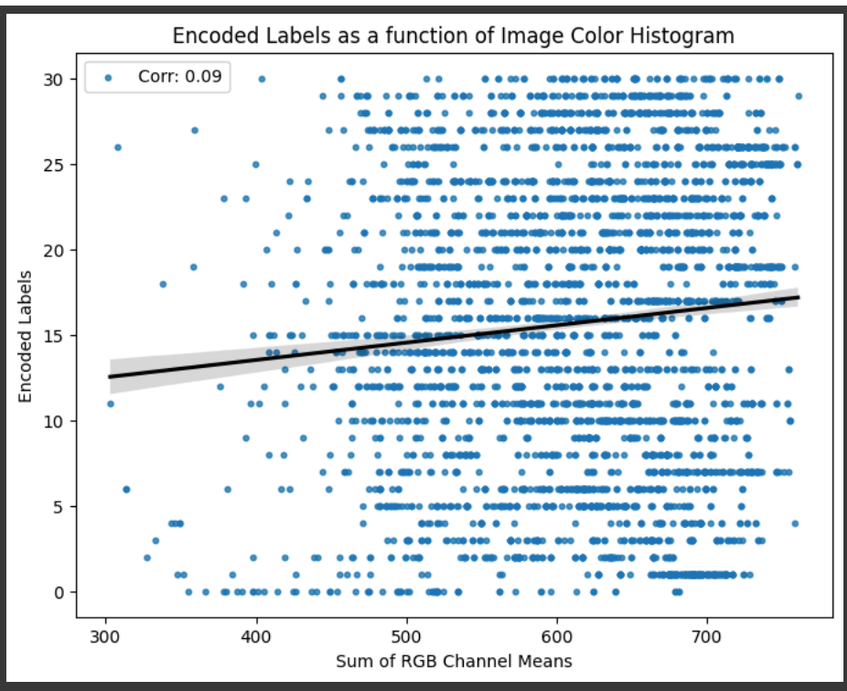
```
print("Distribution originale :")
print((df_images["r_mean"] + df_images["g_mean"] + df_images["b_mean"]).describe())

print("\nDistribution biaisée :")
print((biased_df["r_mean"] + biased_df["g_mean"] + biased_df["b_mean"]).describe())
```

```
Distribution originale :
count    4110.000000
mean      545.078949
std       122.879820
min       196.641609
25%       465.665392
50%       553.229878
75%       645.127153
max       760.424356
dtype: float64

Distribution biaisée :
count    4110.000000
mean      616.526152
std        79.563487
min       302.919619
25%       563.095678
50%       625.819433
75%       678.714144
max       760.424356
dtype: float64
```

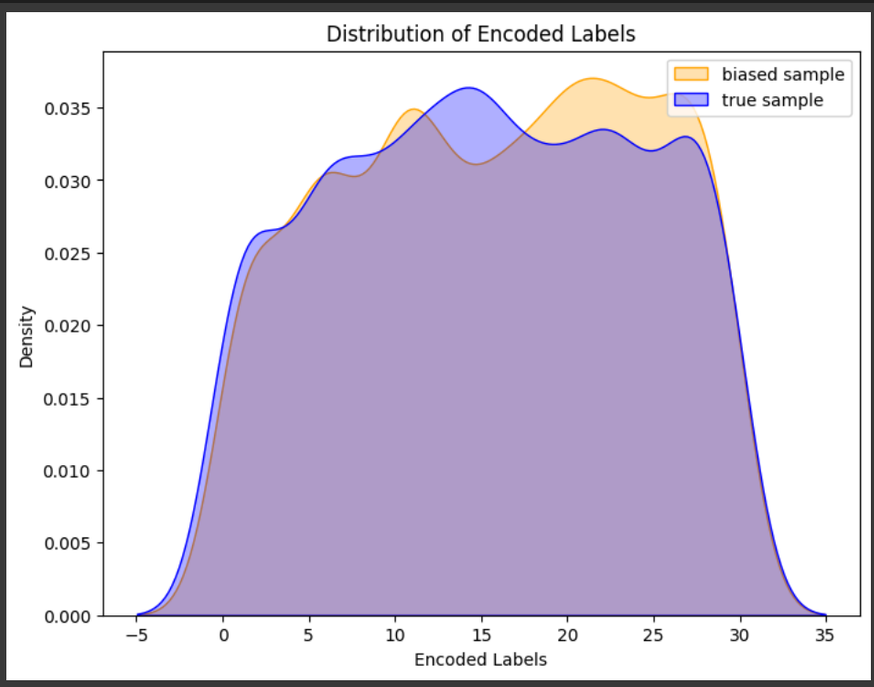
```
plt.figure(figsize=(8, 6))
sns.kdeplot(biased_df["r_mean"] + biased_df["g_mean"] + biased_df["b_mean"], shade=True, alpha=0.3, label="biased sample", color="orange")
sns.kdeplot(df_images["r_mean"] + df_images["g_mean"] + df_images["b_mean"], shade=True, alpha=0.3, label="true sample", color="blue")

plt.title("Distribution of Image Color Histogram (RGB Means)")
plt.xlabel("Sum of RGB Channel Means")
plt.ylabel("Density")
plt.legend()
plt.show()
```




```
X_var = biased_df["r_mean"] + biased_df["g_mean"] + biased_df["b_mean"]
y = LabelEncoder().fit_transform(biased_df["label"])
corr = np.corrcoef(X_var, y)[0, 1]

plt.figure(figsize=(8, 6))
sns.regplot(x=X_var, y=y, label="Corr: %.2f" % corr, scatter_kws={"s": 10}, line_kws={"color": "k"})
plt.title("Encoded Labels as a function of Image Color Histogram")
plt.xlabel("Sum of RGB Channel Means")
plt.ylabel("Encoded Labels")
plt.legend()
plt.show()
```




```
label_encoder = LabelEncoder()
y_true = label_encoder.fit_transform(df_images["label"])
y_biased = label_encoder.transform(biased_df["label"])

plt.figure(figsize=(8, 6))
sns.kdeplot(y_biased, shade=True, alpha=0.3, label="biased sample", color="orange")
sns.kdeplot(y_true, shade=True, alpha=0.3, label="true sample", color="blue")

plt.title("Distribution of Encoded Labels")
plt.xlabel("Encoded Labels")
plt.ylabel("Density")
plt.legend()
plt.show()

```




```
y_true = label_encoder.fit_transform(df_images["label"])  # Labels encodés pour les données originales
y_biased = label_encoder.transform(biased_df["label"])    # Labels encodés pour les données biaisées

print("Statistiques descriptives cibles")
display(pd.concat((
    pd.DataFrame(y_true, columns=["target"]).describe(percentiles=[0.5, 0.75, 0.9]),
    pd.DataFrame(y_biased, columns=["biased target"]).describe(percentiles=[0.5, 0.75, 0.9])
), axis=1))

```

Statistiques descriptives cibles

	target	biased target
count	4110.000000	4110.000000
mean	15.273966	15.717032
std	8.736754	8.646289
min	0.000000	0.000000
50%	15.000000	16.000000
75%	23.000000	23.000000
90%	27.000000	27.000000
max	30.000000	30.000000

```
df_y_true = pd.DataFrame(y_true, columns=["target"])
df_y_biased = pd.DataFrame(y_biased, columns=["biased target"])

result = pd.concat((
    df_y_true.describe(percentiles=[0.5, 0.75, 0.9]),
    df_y_biased.describe(percentiles=[0.5, 0.75, 0.9])
), axis=1)

print("Statistiques descriptives des labels encodés :")
display(result)

```

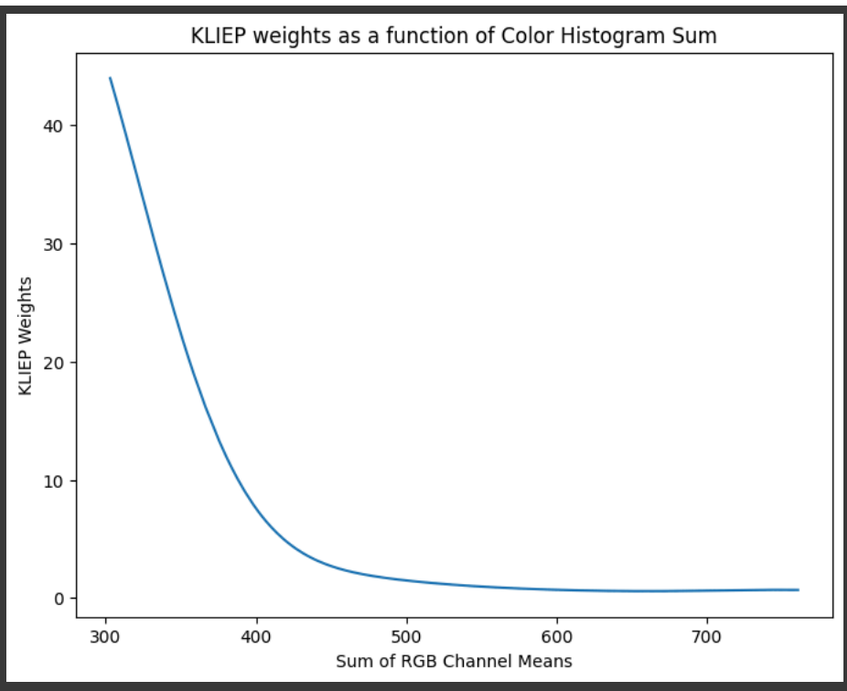
Statistiques descriptives des labels encodés :

	target	biased target
count	4110.000000	4110.000000
mean	15.273966	15.717032
std	8.736754	8.646289
min	0.000000	0.000000
50%	15.000000	16.000000
75%	23.000000	23.000000
90%	27.000000	27.000000
max	30.000000	30.000000

```
var_biais = "color_hist_sum"
biased_X_var = biased_df["r_mean"] + biased_df["g_mean"] + biased_df["b_mean"]
true_X_var = df_images["r_mean"] + df_images["g_mean"] + df_images["b_mean"]

# Ajustement des poids via KLIEP
kliep = KLIEP(sigmas=[10**(i-4) for i in range(9)], random_state=0)
kliep_weights = kliep.fit_weights(biased_X_var.values.reshape(-1, 1),
                                  true_X_var.values.reshape(-1, 1))
```

```
Cross Validation process...
Parameters {'gamma': 0.0001} -- J-score = 0.459 (0.045)
Parameters {'gamma': 0.001} -- J-score = 0.459 (0.057)
Parameters {'gamma': 0.01} -- J-score = 0.429 (0.185)
Parameters {'gamma': 0.1} -- J-score = -0.392 (0.127)
Parameters {'gamma': 1} -- J-score = -5.461 (0.952)
Parameters {'gamma': 10} -- J-score = -19.269 (0.480)
Parameters {'gamma': 100} -- J-score = -28.828 (0.200)
Parameters {'gamma': 1000} -- J-score = -33.366 (0.108)
Parameters {'gamma': 10000} -- J-score = -35.116 (0.167)
```

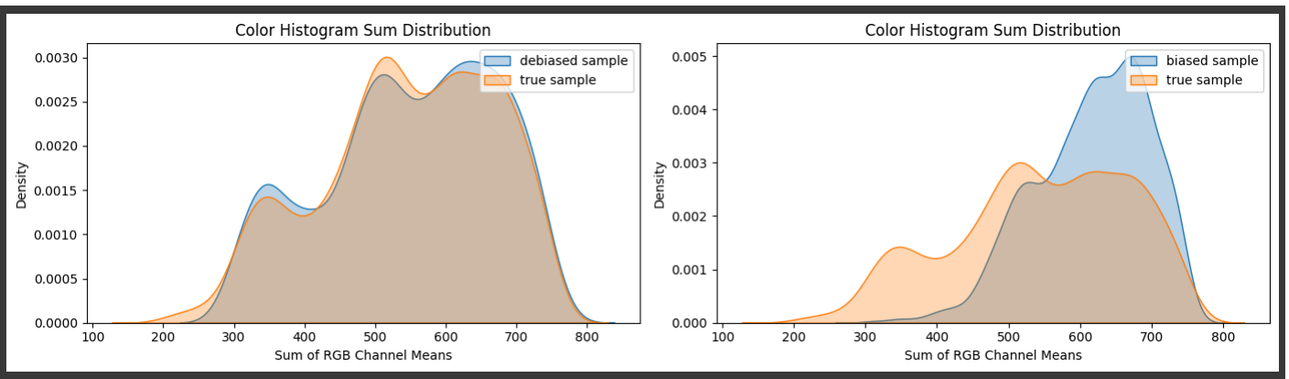
```
#  Visualisation des poids KLIEP
plt.figure(figsize=(8, 6))
sns.lineplot(x=biased_X_var.values, y=kliep_weights)
plt.title("KLIEP weights as a function of Color Histogram Sum")
plt.xlabel("Sum of RGB Channel Means")
plt.ylabel("KLIEP Weights")
plt.show()
```




```
# Rééchantillonnage avec les poids KLIEP
np.random.seed(123)
debiasing_index = np.random.choice(len(biased_X_var), 2000, p=kliep_weights / kliep_weights.sum())

debiased_X = biased_df.iloc[debiasing_index][["r_mean", "g_mean", "b_mean"]]
debiased_y = biased_df.iloc[debiasing_index]["label"]

print("Taille des données rééchantillonnées :", debiased_X.shape)
print("Exemples de labels après rééchantillonnage :")
print(debiased_y.value_counts())

```

```
Taille des données rééchantillonnées : (2000, 3)
Exemples de labels après rééchantillonnage :
label
desk_chair          104
bike                 94
scissors             88
paper_notebook       87
laptop_computer      86
punchers             82
back_pack            78
bottle               78
headphones           76
keyboard             75
projector            71
bike_helmet          70
speaker              70
mobile_phone         68
printer              66
ring_binder          64
monitor              63
mug                  62
bookcase             60
phone                59
desk_lamp            56
calculator           55
pen                  51
mouse                50
letter_tray          49
stapler              48
ruler                46
tape_dispenser       38
file_cabinet         36
trash_can            36
desktop_computer     34
Name: count, dtype: int64
```

```
#  Comparaison des distributions avant et après débiaisage
fig, axes = plt.subplots(1, 2, figsize=(14, 4))

for ax, x_data, label in zip(
    axes,
    [debiased_X["r_mean"] + debiased_X["g_mean"] + debiased_X["b_mean"],
     biased_df["r_mean"] + biased_df["g_mean"] + biased_df["b_mean"]],
    ["debiased sample", "biased sample"]
):
    sns.kdeplot(x_data.values, shade=True, alpha=0.3, label=label, ax=ax)
    sns.kdeplot((df_images["r_mean"] + df_images["g_mean"] + df_images["b_mean"]).values,
                shade=True, alpha=0.3, label="true sample", ax=ax)
    ax.set_title("Color Histogram Sum Distribution")
    ax.set_xlabel("Sum of RGB Channel Means")
    ax.legend()

plt.tight_layout()
plt.show()
```




```
# Calcul des statistiques descriptives en utilisant y_biased
print("Statistiques descriptives des cibles après débiaisage")
display(pd.concat((
    pd.DataFrame(y_true, columns=["target"]).describe(percentiles=[0.5, 0.75, 0.9]),
    pd.DataFrame(debiased_y.values, columns=["debiased target"]).describe(percentiles=[0.5, 0.75, 0.9]),
    pd.DataFrame(y_biased, columns=["biased target"]).describe(percentiles=[0.5, 0.75, 0.9])
), axis=1))

```

```
Statistiques descriptives des cibles après débiaisage
```

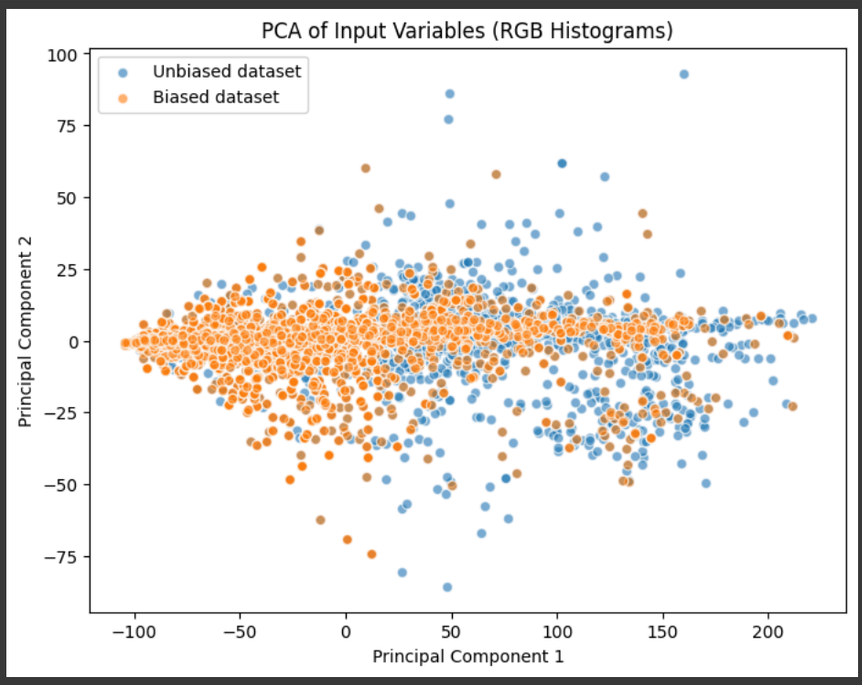
	target	debiased target	biased target
count	4110.000000	2000	4110.000000
mean	15.273966	NaN	15.717032
std	8.736754	NaN	8.646289
min	0.000000	NaN	0.000000
50%	15.000000	NaN	16.000000
75%	23.000000	NaN	23.000000
90%	27.000000	NaN	27.000000
max	30.000000	NaN	30.000000
unique	NaN	31	NaN
top	NaN	desk_chair	NaN
freq	NaN	104	NaN

```

np.random.seed(0)

#  Définir le DataFrame X avec les valeurs RGB
X = df_images[["r_mean", "g_mean", "b_mean"]]
y = label_encoder.fit_transform(df_images["label"])  # Encodage des labels

sample_bias = np.ones(len(X))

rgb_sum = X["r_mean"] + X["g_mean"] + X["b_mean"]
median_rgb_sum = rgb_sum.median()

sample_bias[rgb_sum > median_rgb_sum] *= 5.

biased_index = np.random.choice(X.index, len(X), p=sample_bias / sample_bias.sum())
biased_X = X.loc[biased_index]
biased_y = y[biased_index]
print("Taille des données biaisées :", biased_X.shape)
print("Distribution des labels après biaisage :")
print(pd.Series(biased_y).value_counts())

```

```
Taille des données biaisées : (4110, 3)
Distribution des labels après biaisage :
23    177
26    173
10    173
24    171
11    166
28    165
19    164
6     161
16    156
7     154
22    151
21    146
29    143
1     139
20    138
8     137
27    137
5     129
12    128
17    124
18    123
30    123
13    121
25    116
```

```
3     107
15     98
2      93
14     89
9      82
0      66
4      60
Name: count, dtype: int64
```

```
X_concat = np.concatenate((X, biased_X), axis=0)

X_pca = PCA(2).fit_transform(X_concat)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:len(X), 0], X_pca[:len(X), 1], edgecolor="w", label="Unbiased dataset", alpha=0.6)
plt.scatter(X_pca[len(X):, 0], X_pca[len(X):, 1], edgecolor="w", label="Biased dataset", alpha=0.6)

plt.title("PCA of Input Variables (RGB Histograms)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.show()

```




```
def neg_j_scorer(estimator, X_, y_, Xt_):
    y_pred = estimator.predict(X_)
    return -np.mean((y_ - y_pred)**2)

kmm = KMM(
    estimator=Ridge(alpha=0.1),
    Xt=X,  # Données originales
    gamma=0.0,
    verbose=0,
    random_state=0
)

grid_params = {"gamma": [0.1, 1.0, 10.0, 100.0, 1000.0]}
gs = GridSearchCV(
    kmm,
    param_grid=grid_params,
    scoring=make_scorer(neg_j_scorer, Xt=X),  # Inclure les données cibles originales
    return_train_score=True,
    cv=5,
    verbose=0
)

gs.fit(biased_X, biased_y)

keys = ["params", "mean_train_score", "std_train_score"]
results = [v for k, v in gs.cv_results_.items() if k in keys]
best = np.argmin(results[1])

print("Best Params %s -- Score %.3f (%.3f)" %
      (str(results[0][best]), results[1][best], results[2][best]))
print("-" * 50)

for p, mu, std in zip(*results):
    print("Params %s -- Score %.3f (%.3f)" % (str(p), mu, std))

```

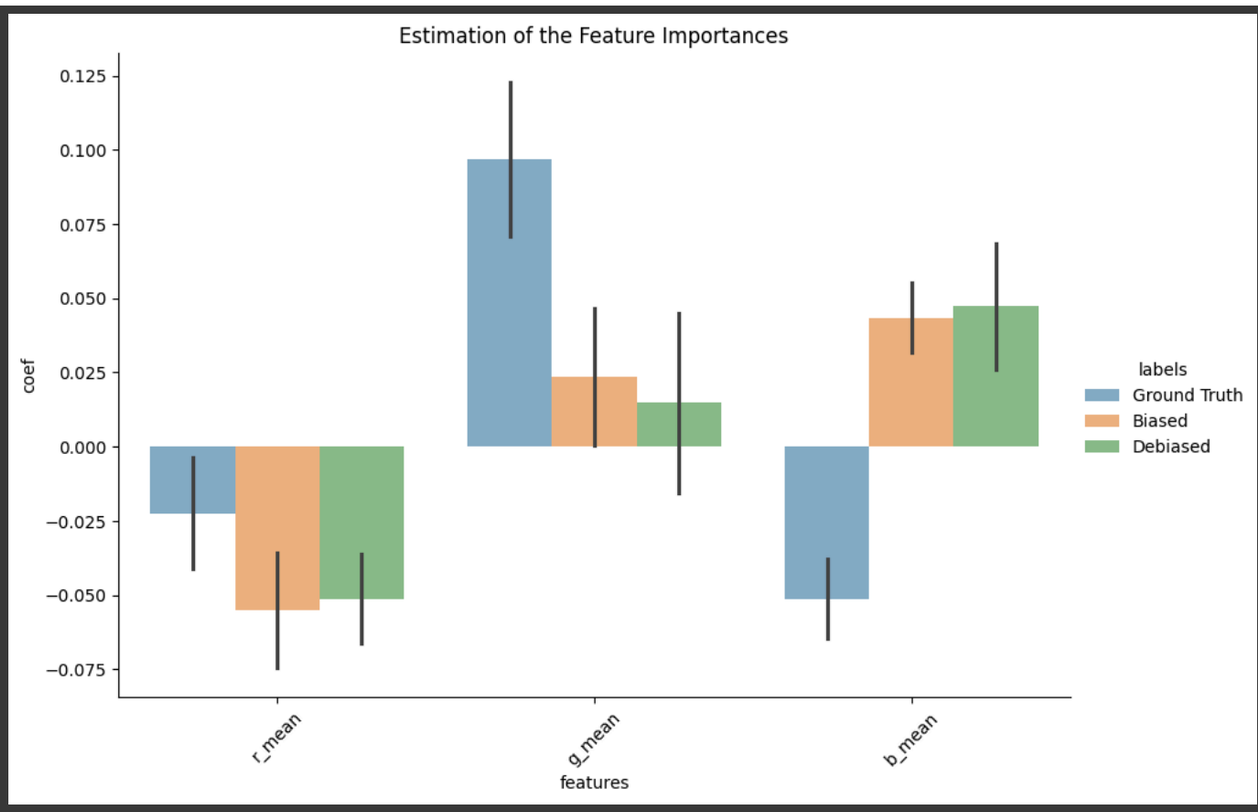
```
Best Params {'gamma': 0.1} -- Score nan (nan)
--------------------------------------------------
Params {'gamma': 0.1} -- Score nan (nan)
Params {'gamma': 1.0} -- Score nan (nan)
Params {'gamma': 10.0} -- Score nan (nan)
Params {'gamma': 100.0} -- Score nan (nan)
Params {'gamma': 1000.0} -- Score nan (nan)
```

```

#  Initialiser KMM avec un noyau RBF
kmm = KMM(
    estimator=Ridge(),
    Xt=X.values,  # Données originales (non biaisées)
    kernel="rbf",  # Noyau gaussien
    gamma=100.0,   # Paramètre de bande passante
    verbose=0,
    random_state=0
)

#  Calculer les poids avec KMM
weights = kmm.fit_weights(biased_X.values, X.values)
```

```
if not isinstance(biased_X, pd.DataFrame):
    biased_X = pd.DataFrame(biased_X, columns=["r_mean", "g_mean", "b_mean"])
if not isinstance(biased_y, pd.Series):
    biased_y = pd.Series(biased_y, name="label")

biased_y = biased_y.loc[biased_X.index]

assert (biased_X.index == biased_y.index).all(), "Indices de biased_X et biased_y ne sont pas alignés après réalignement."

# Comparaison des coefficients estimés
true_coefs = []
biased_coefs = []
debiased_coefs = []

for _ in range(20):
    bs = np.random.choice(X.index, len(X), replace=True)
    biased_bs = np.random.choice(biased_X.index, len(X), replace=True)
    debiased_bs = np.random.choice(
        biased_X.index, len(biased_X), replace=True, p=weights / weights.sum()
    )

    true_coefs.append(Ridge(alpha=0.1).fit(X.loc[bs], y[bs]).coef_)
    biased_coefs.append(Ridge(alpha=0.1).fit(biased_X.loc[biased_bs], biased_y.loc[biased_bs]).coef_)
    debiased_coefs.append(Ridge(alpha=0.1).fit(biased_X.loc[debiased_bs], biased_y.loc[debiased_bs]).coef_)

print("Nombre de coefficients estimés (non biaisés, biaisés, débiaisés) :",
      len(true_coefs), len(biased_coefs), len(debiased_coefs))

```

```
Nombre de coefficients estimés (non biaisés, biaisés, débiaisés) : 20 20 20
```

```
features_subset = biased_X.columns

def filter_coefs(coefs_list, X, features_subset):
    coefs_array = np.array(coefs_list)
    indices = [X.columns.get_loc(c) for c in features_subset]
    return coefs_array[:, indices]

true_arr = filter_coefs(true_coefs, biased_X, features_subset).ravel()
biased_arr = filter_coefs(biased_coefs, biased_X, features_subset).ravel()
debiased_arr = filter_coefs(debiased_coefs, biased_X, features_subset).ravel()

x = list(features_subset) * (len(true_coefs) + len(biased_coefs) + len(debiased_coefs))
labels = (["Ground Truth"] * len(true_coefs) * len(features_subset) +
          ["Biased"] * len(biased_coefs) * len(features_subset) +
          ["Debiased"] * len(debiased_coefs) * len(features_subset))
arr = np.concatenate((true_arr, biased_arr, debiased_arr))
df_plot = pd.DataFrame({"coef": arr, "features": x, "labels": labels})

sns.catplot(data=df_plot, x="features", y="coef", hue="labels",
            kind="bar", ci="sd", alpha=.6, height=6, aspect=1.5)
plt.title("Estimation of the Feature Importances")
plt.xticks(rotation=45)
plt.show()

```



Conclusion

Dans ce notebook, nous avons démontré comment introduire et atténuer le biais d'échantillonnage en utilisant la toolbox adapt sur deux jeux de données différents :

Jeu de Données Diabetes : Introduction du biais basé sur la caractéristique age et utilisation de KLIEP et KMM pour l'ajustement des poids, suivie de la visualisation de l'impact sur les importances des caractéristiques.

Jeu de Données Office-31 : Application de techniques similaires d'introduction de biais basées sur les histogrammes de couleurs des images et utilisation des méthodes d'adaptation de domaine pour corriger le biais, culminant par la comparaison des importances des caractéristiques entre les échantillons non biaisés, biaisés et débiaisés.

Ces exemples illustrent l'efficacité des techniques d'adaptation de domaine pour traiter le biais d'échantillonnage, assurant ainsi des modèles plus robustes et généralisables.

Exemple de Fine-Tuning

Ce notebook démontre comment effectuer du fine-tuning en utilisant la toolbox `adapt`. Nous présentons deux exemples :

1. **Exemple original avec le jeu de données Flowers Recognition.**
2. **Test sur un jeu de données différent (Intel Image Classification).**

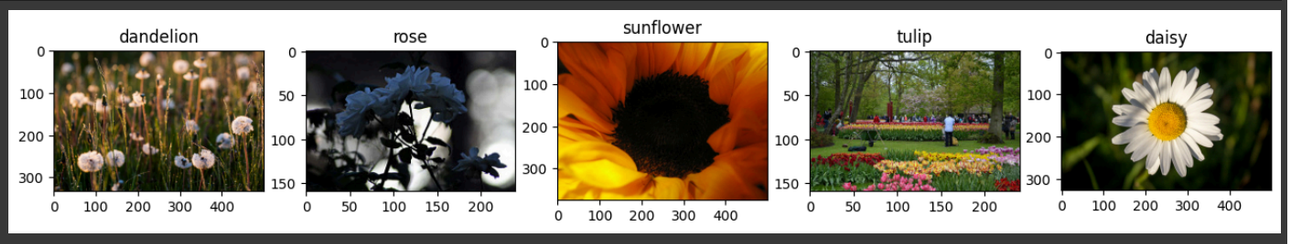
Exemple 1 : Fine-Tuning avec le Jeu de Données Flowers et la Toolbox `adapt`

Dans cet exemple, nous réalisons un fine-tuning sur le jeu de données Flowers en utilisant un modèle de réseau de neurones convolutifs (CNN) personnalisé, puis nous appliquons la toolbox `adapt` pour affiner davantage le modèle.

```
# === Importation des bibliothèques ===

# Standard libraries
import os
import numpy as np

# Deep Learning avec TensorFlow
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.models import load_model
from tensorflow.keras import Model, Sequential
from tensorflow.keras.layers import (
    Dense, Input, Dropout, Conv2D, MaxPooling2D, Flatten, Reshape, BatchNormalization, Layer
)
from tensorflow.keras.optimizers import Adam

# Visualisation
import matplotlib.pyplot as plt

# Traitement d'images
from PIL import Image

# Machine Learning
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split


```

```
import zipfile

# Décompresser le fichier ZIP
zip_file_path = "/content/archive (1).zip"
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall("flowers_extracted")
```

```
path = "/content/flowers_extracted/flowers"

X_path = []
y = []

fig, axes = plt.subplots(1, 5, figsize=(16, 5))
i = 0
for r, d, f in os.walk(path):
    for direct in d:
        if not ".ipynb_checkpoints" in direct:
            for r, d, f in os.walk(os.path.join(path , direct)):
                for file in f:
                    path_to_image = os.path.join(r, file)
                    if not ".ipynb_checkpoints" in path_to_image:
                        X_path.append(path_to_image)
                        y.append(direct)
            axes[i].imshow(plt.imread(X_path[-1]))
            axes[i].set_title(y[-1])
            i+=1
plt.show()
```




```
one = OneHotEncoder(sparse_output=False)

y_lab = one.fit_transform(np.array(y).reshape(-1, 1))

np.random.seed(0)
train_indexes, test_indexes = train_test_split(np.arange(len(X_path)), train_size=0.2, shuffle=True)
print("Train size: %i, Test size: %i"%(len(train_indexes), len(test_indexes)))
```

```
Train size: 863, Test size: 3454
```

```


def generator_train(only_X=False):
    for i in train_indexes:
        image = Image.open(X_path[i])
        image = image.resize((224, 224), Image.Resampling.LANCZOS)
        X = np.array(image, dtype=int)
        if only_X:
            yield preprocess_input(X)
        else:
            yield (preprocess_input(X), y_lab[i])

def generator_test(only_X=False):
    for i in test_indexes:
        image = Image.open(X_path[i])
        image = image.resize((224, 224), Image.Resampling.LANCZOS)
        X = np.array(image, dtype=int)
        if only_X:
            yield preprocess_input(X)
        else:
            yield (preprocess_input(X), y_lab[i])

data_train = tf.data.Dataset.from_generator(generator_train,
                                        output_types=(tf.float32, tf.float32),
                                        output_shapes=([224,224,3], [5]))

data_test = tf.data.Dataset.from_generator(generator_test,
                                        output_types=(tf.float32, tf.float32),
                                        output_shapes=([224,224,3], [5]))

X_train = tf.data.Dataset.from_generator(generator_train, args=(True,),
                                        output_types=tf.float32,
                                        output_shapes=[224,224,3])

X_test = tf.data.Dataset.from_generator(generator_test, args=(True,),
                                        output_types=tf.float32,
                                        output_shapes=[224,224,3])

def load_resnet50(path="resnet50.hdf5"):
    model = ResNet50(include_top=False, input_shape=(224, 224, 3), pooling="avg")
    for i in range(len(model.layers)):
        if model.layers[i].__class__.__name__ == "BatchNormalization":
            model.layers[i].trainable = False
    return model
```

```
WARNING:tensorflow:From <ipython-input-6-b716bcc8b8f0>:24: calling DatasetV2.from_generator (from tensorflow.python.data.ops.dataset_ops
Instructions for updating:
Use output_signature instead
```

```
WARNING:tensorflow:From <ipython-input-6-b716bcc8b8f0>:24: calling DatasetV2.from_generator (from tensorflow.python.data.ops.dataset_ops
Instructions for updating:
Use output_signature instead
```

```
class Rescaling(Layer):

    def __init__(self, scale=1., offset=0.):
        super().__init__()
        self.scale = scale
        self.offset = offset

    def call(self, inputs):
        return inputs*self.scale + self.offset


def get_model(input_shape=(224, 224, 3)):
    inputs = Input(input_shape)
    modeled = Rescaling(1./127.5, offset=-1.0)(inputs)
    modeled = Conv2D(32, 5, activation='relu')(modeled)
    modeled = MaxPooling2D(2, 2)(modeled)
    modeled = BatchNormalization()(modeled)
    modeled = Conv2D(48, 5, activation='relu')(modeled)
    modeled = BatchNormalization()(modeled)
    modeled = MaxPooling2D(2, 2)(modeled)
    modeled = Conv2D(64, 5, activation='relu')(modeled)
    modeled = BatchNormalization()(modeled)
    modeled = MaxPooling2D(2, 2)(modeled)
    modeled = Conv2D(128, 5, activation='relu')(modeled)
    modeled = BatchNormalization()(modeled)
    modeled = MaxPooling2D(2, 2)(modeled)
    modeled = Flatten()(modeled)
    modeled = Dropout(0.5)(modeled)
    modeled = Dense(5, activation="softmax")(modeled)
    model = Model(inputs, modeled)
    model.compile(optimizer=Adam(0.001), loss='categorical_crossentropy', metrics=["acc"])
    return model

model = get_model()
model.summary()
```

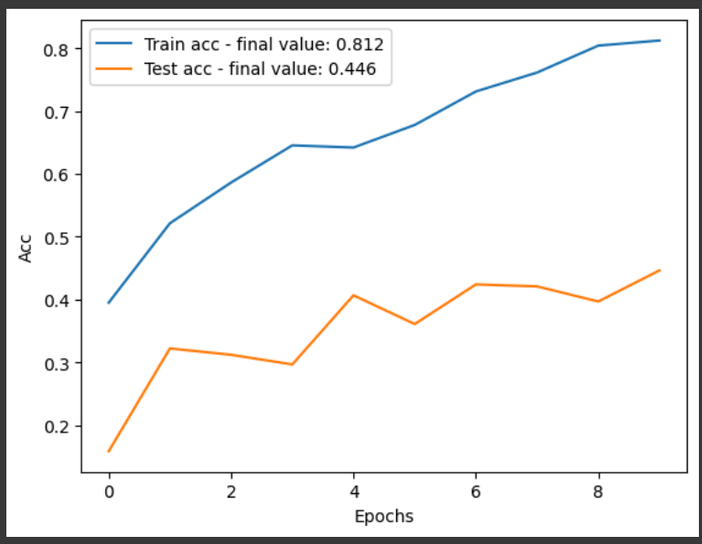
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 rescaling (Rescaling)       (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 220, 220, 32)      2432

 max_pooling2d (MaxPooling2  (None, 110, 110, 32)      0
 D)

 batch_normalization (Batch  (None, 110, 110, 32)      128
 Normalization)

 conv2d_1 (Conv2D)           (None, 106, 106, 48)      38448

 batch_normalization_1 (Bat  (None, 106, 106, 48)      192
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 53, 53, 48)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 49, 49, 64)        76864

 batch_normalization_2 (Bat  (None, 49, 49, 64)        256
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 24, 24, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 20, 20, 128)       204928

 batch_normalization_3 (Bat  (None, 20, 20, 128)       512
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 10, 10, 128)       0
 g2D)

 flatten (Flatten)           (None, 12800)             0

 dropout (Dropout)           (None, 12800)             0

 dense (Dense)               (None, 5)                 64005

=================================================================
Total params: 387765 (1.48 MB)
Trainable params: 387221 (1.48 MB)
Non-trainable params: 544 (2.12 KB)
_________________________________________________________________
```

```
model.fit(data_train.batch(32), epochs=10, validation_data=data_test.batch(32))

```

```
Epoch 1/10
27/27 [==============================] - 44s 1s/step - loss: 2.7279 - acc: 0.3951 - val_loss: 2.9393 - val_acc: 0.1587
Epoch 2/10
27/27 [==============================] - 25s 958ms/step - loss: 1.8935 - acc: 0.5214 - val_loss: 2.0136 - val_acc: 0.3222
Epoch 3/10
27/27 [==============================] - 25s 945ms/step - loss: 1.6160 - acc: 0.5863 - val_loss: 1.9107 - val_acc: 0.3121
Epoch 4/10
```

```
27/27 [==============================] - 24s 910ms/step - loss: 1.4520 - acc: 0.6454 - val_loss: 2.5594 - val_acc: 0.2968
Epoch 5/10
27/27 [==============================] - 26s 978ms/step - loss: 1.3379 - acc: 0.6419 - val_loss: 2.1632 - val_acc: 0.4068
Epoch 6/10
27/27 [==============================] - 25s 945ms/step - loss: 1.1333 - acc: 0.6779 - val_loss: 2.1286 - val_acc: 0.3610
Epoch 7/10
27/27 [==============================] - 25s 944ms/step - loss: 1.1102 - acc: 0.7312 - val_loss: 2.3541 - val_acc: 0.4241
Epoch 8/10
27/27 [==============================] - 24s 890ms/step - loss: 0.8821 - acc: 0.7613 - val_loss: 2.5457 - val_acc: 0.4210
Epoch 9/10
27/27 [==============================] - 25s 939ms/step - loss: 0.8478 - acc: 0.8042 - val_loss: 2.5023 - val_acc: 0.3969
Epoch 10/10
27/27 [==============================] - 24s 927ms/step - loss: 0.6376 - acc: 0.8123 - val_loss: 2.8045 - val_acc: 0.4464
<keras.src.callbacks.History at 0x7bec983450c0>
```

```
acc = model.history.history["acc"]; val_acc = model.history.history["val_acc"]
plt.plot(acc, label="Train acc - final value: %.3f"%acc[-1])
plt.plot(val_acc, label="Test acc - final value: %.3f"%val_acc[-1])
plt.legend(); plt.xlabel("Epochs"); plt.ylabel("Acc"); plt.show()
```




```
def get_task():
    model = Sequential()
    model.add(Dense(1024, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(1024, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(5, activation="softmax"))
    return model
```

```
resnet50 = load_resnet50()
X_train_enc = resnet50.predict(X_train.batch(32))
X_test_enc = resnet50.predict(X_test.batch(32))

print("X train shape: %s"%str(X_train_enc.shape))
```

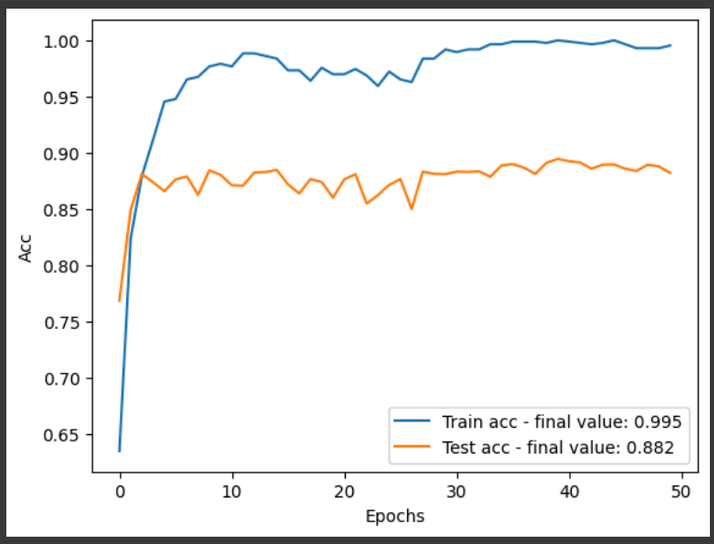
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_nc
94765736/94765736 [==============================] - 3s 0us/step
27/27 [==============================] - 9s 229ms/step
108/108 [==============================] - 25s 227ms/step
X train shape: (863, 2048)
```

```
task = get_task()
task.compile(loss="categorical_crossentropy", optimizer=Adam(0.001), metrics=["acc"])
task.fit(X_train_enc, y_lab[train_indexes], epochs=50, batch_size=32,
         validation_data=(X_test_enc, y_lab[test_indexes]))
```

```
Epoch 1/50
27/27 [==============================] - 3s 29ms/step - loss: 1.1674 - acc: 0.6350 - val_loss: 0.7535 - val_acc: 0.7687
Epoch 2/50
27/27 [==============================] - 0s 14ms/step - loss: 0.5415 - acc: 0.8239 - val_loss: 0.4403 - val_acc: 0.8494
Epoch 3/50
27/27 [==============================] - 0s 14ms/step - loss: 0.3541 - acc: 0.8795 - val_loss: 0.3600 - val_acc: 0.8813
Epoch 4/50
27/27 [==============================] - 0s 17ms/step - loss: 0.2457 - acc: 0.9119 - val_loss: 0.3860 - val_acc: 0.8738
Epoch 5/50
27/27 [==============================] - 0s 13ms/step - loss: 0.1663 - acc: 0.9455 - val_loss: 0.4358 - val_acc: 0.8660
Epoch 6/50
27/27 [==============================] - 0s 14ms/step - loss: 0.1342 - acc: 0.9479 - val_loss: 0.3892 - val_acc: 0.8764
Epoch 7/50
27/27 [==============================] - 0s 13ms/step - loss: 0.0970 - acc: 0.9652 - val_loss: 0.4423 - val_acc: 0.8790
Epoch 8/50
27/27 [==============================] - 0s 13ms/step - loss: 0.0876 - acc: 0.9676 - val_loss: 0.5473 - val_acc: 0.8628
Epoch 9/50
27/27 [==============================] - 0s 16ms/step - loss: 0.0787 - acc: 0.9768 - val_loss: 0.4280 - val_acc: 0.8845
Epoch 10/50
27/27 [==============================] - 0s 18ms/step - loss: 0.0573 - acc: 0.9791 - val_loss: 0.4892 - val_acc: 0.8804
Epoch 11/50
27/27 [==============================] - 1s 20ms/step - loss: 0.0857 - acc: 0.9768 - val_loss: 0.5640 - val_acc: 0.8712
Epoch 12/50
27/27 [==============================] - 0s 18ms/step - loss: 0.0450 - acc: 0.9884 - val_loss: 0.5955 - val_acc: 0.8709
Epoch 13/50
27/27 [==============================] - 1s 31ms/step - loss: 0.0293 - acc: 0.9884 - val_loss: 0.5655 - val_acc: 0.8825
Epoch 14/50
27/27 [==============================] - 0s 18ms/step - loss: 0.0458 - acc: 0.9861 - val_loss: 0.6113 - val_acc: 0.8830
Epoch 15/50
27/27 [==============================] - 0s 14ms/step - loss: 0.0403 - acc: 0.9838 - val_loss: 0.5627 - val_acc: 0.8848
Epoch 16/50
```

```
27/27 [==============================] - 0s 13ms/step - loss: 0.0869 - acc: 0.9733 - val_loss: 0.6241 - val_acc: 0.8720
Epoch 17/50
27/27 [==============================] - 0s 14ms/step - loss: 0.0835 - acc: 0.9733 - val_loss: 0.7546 - val_acc: 0.8639
Epoch 18/50
27/27 [==============================] - 0s 14ms/step - loss: 0.1009 - acc: 0.9641 - val_loss: 0.5853 - val_acc: 0.8767
Epoch 19/50
27/27 [==============================] - 0s 13ms/step - loss: 0.0726 - acc: 0.9757 - val_loss: 0.6483 - val_acc: 0.8741
Epoch 20/50
27/27 [==============================] - 0s 17ms/step - loss: 0.1279 - acc: 0.9699 - val_loss: 0.8089 - val_acc: 0.8602
Epoch 21/50
27/27 [==============================] - 0s 13ms/step - loss: 0.0818 - acc: 0.9699 - val_loss: 0.7071 - val_acc: 0.8764
Epoch 22/50
27/27 [==============================] - 0s 14ms/step - loss: 0.0711 - acc: 0.9745 - val_loss: 0.7624 - val_acc: 0.8810
Epoch 23/50
27/27 [==============================] - 0s 17ms/step - loss: 0.1444 - acc: 0.9687 - val_loss: 0.9615 - val_acc: 0.8550
Epoch 24/50
27/27 [==============================] - 0s 17ms/step - loss: 0.1469 - acc: 0.9594 - val_loss: 0.8176 - val_acc: 0.8625
Epoch 25/50
27/27 [==============================] - 0s 17ms/step - loss: 0.1240 - acc: 0.9722 - val_loss: 0.8170 - val_acc: 0.8715
Epoch 26/50
27/27 [==============================] - 0s 13ms/step - loss: 0.1473 - acc: 0.9652 - val_loss: 0.7300 - val_acc: 0.8767
Epoch 27/50
27/27 [==============================] - 0s 17ms/step - loss: 0.1848 - acc: 0.9629 - val_loss: 0.9514 - val_acc: 0.8500
Epoch 28/50
27/27 [==============================] - 0s 17ms/step - loss: 0.1024 - acc: 0.9838 - val_loss: 0.6651 - val_acc: 0.8833
Epoch 29/50
27/27 [==============================] - 0s 14ms/step - loss: 0.0457 - acc: 0.9838 - val_loss: 0.6464 - val_acc: 0.8813
```

```
acc = task.history.history["acc"]; val_acc = task.history.history["val_acc"]
plt.plot(acc, label="Train acc - final value: %.3f"%acc[-1])
plt.plot(val_acc, label="Test acc - final value: %.3f"%val_acc[-1])
plt.legend(); plt.xlabel("Epochs"); plt.ylabel("Acc"); plt.show()
```

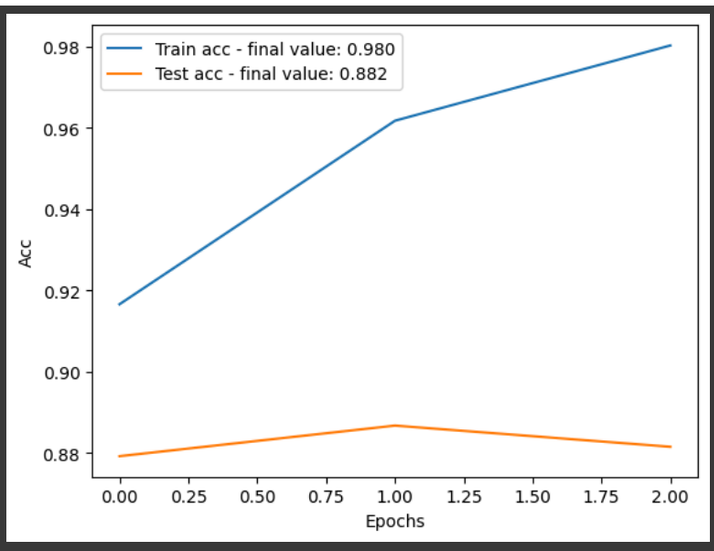



```
from adapt.parameter_based import FineTuning

encoder = load_resnet50()
task = get_task()

optimizer = Adam(0.001)
optimizer_enc = Adam(0.00001)

finetunig = FineTuning(encoder=encoder,
                       task=task,
                       optimizer=optimizer,
                       optimizer_enc=optimizer_enc,
                       loss="categorical_crossentropy",
                       metrics=["acc"],
                       copy=False,
                       pretrain=True,
                       pretrain__epochs=3)
```

```
finetunig.fit(data_train, epochs=3, batch_size=32, validation_data=data_test.batch(32))

```

```
Computing src dataset size...
Done!
Computing tgt dataset size...
Done!
Epoch 1/3
27/27 [==============================] - 49s 1s/step - loss: 1.1342 - acc: 0.6470 - val_loss: 0.5381 - val_acc: 0.8118
Epoch 2/3
27/27 [==============================] - 42s 1s/step - loss: 0.5111 - acc: 0.8264 - val_loss: 0.5462 - val_acc: 0.8179
Epoch 3/3
27/27 [==============================] - 62s 2s/step - loss: 0.3397 - acc: 0.8889 - val_loss: 0.3843 - val_acc: 0.8709
Epoch 1/3
27/27 [==============================] - 83s 1s/step - loss: 0.2478 - acc: 0.9167 - val_loss: 0.3838 - val_acc: 0.8793
Epoch 2/3
27/27 [==============================] - 45s 1s/step - loss: 0.1180 - acc: 0.9618 - val_loss: 0.3344 - val_acc: 0.8868
Epoch 3/3
27/27 [==============================] - 44s 1s/step - loss: 0.0526 - acc: 0.9803 - val_loss: 0.4004 - val_acc: 0.8816
<adapt.parameter_based._finetuning.FineTuning at 0x7bebb8491750>
```

```
acc = finetunig.history.history["acc"]; val_acc = finetunig.history.history["val_acc"]
plt.plot(acc, label="Train acc - final value: %.3f"%acc[-1])
plt.plot(val_acc, label="Test acc - final value: %.3f"%val_acc[-1])
plt.legend(); plt.xlabel("Epochs"); plt.ylabel("Acc"); plt.show()
```




```
Commencez à coder ou à générer avec l'IA.
```

Exemple 2 : Fine-Tuning avec un Jeu de Données Différent (Intel Classification)

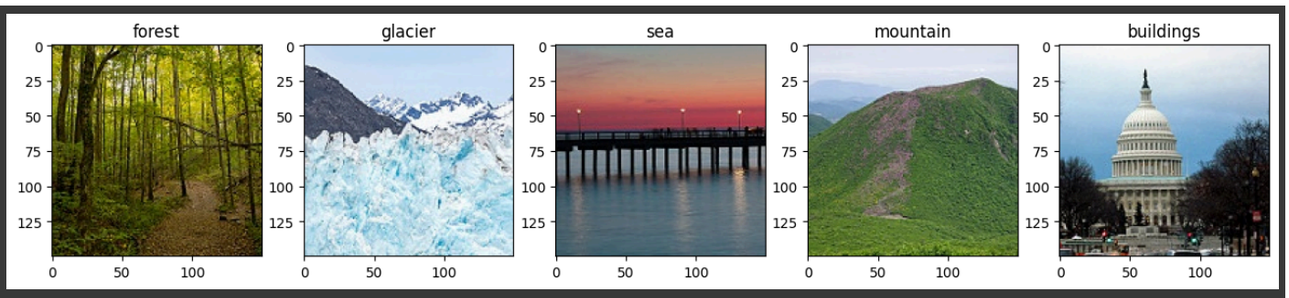
Dans cet exemple, nous réalisons un fine-tuning sur un jeu de données différent, le jeu de données Intel pour la classification des images, afin de tester la généralisation des techniques apprises.

```
# === Importation des bibliothèques ===

# Standard libraries
import os
import zipfile
import numpy as np

# Deep Learning avec TensorFlow
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import (
    Dense, Input, Dropout, Conv2D, MaxPooling2D, Flatten, Reshape, BatchNormalization, Layer
)
from tensorflow.keras.optimizers import Adam

# Visualisation
import matplotlib.pyplot as plt

# Traitement d'images
from PIL import Image

# Machine Learning
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

```

```
zip_file_path = "/content/archive (2).zip"
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall("intel")


```

```
train_path = "/content/intel/seg_train/seg_train"
test_path = "/content/intel/seg_test/seg_test"

```

```
X_path = []
y = []

fig, axes = plt.subplots(1, 5, figsize=(16, 5))
i = 0
for class_dir in os.listdir(train_path):
    if not class_dir.startswith('.'):
        class_folder = os.path.join(train_path, class_dir)
        files = os.listdir(class_folder)
        for file in files:
            if not file.startswith('.'):
                path_to_image = os.path.join(class_folder, file)
                X_path.append(path_to_image)
                y.append(class_dir)
        if i < 5:
            axes[i].imshow(plt.imread(path_to_image))
            axes[i].set_title(class_dir)
            i+=1
plt.show()
```




```
1
2 one = OneHotEncoder(sparse_output=False)
3 y_lab = one.fit_transform(np.array(y).reshape(-1, 1))
4
5 X_path_test = []
6 y_test = []
7 for class_dir in os.listdir(test_path):
8     if not class_dir.startswith('.'):
9         class_folder = os.path.join(test_path, class_dir)
10        files = os.listdir(class_folder)
11        for file in files:
12            if not file.startswith('.'):
13                path_to_image = os.path.join(class_folder, file)
14                X_path_test.append(path_to_image)
15                y_test.append(class_dir)
16
17 y_lab_test = one.transform(np.array(y_test).reshape(-1, 1))
18
19 print("Train size: %i, Test size: %i"%(len(X_path), len(X_path_test)))
```

```
Train size: 14034, Test size: 3000
```

```
1 subset_size = 1000
2 X_path = X_path[:subset_size]
3 y_lab = y_lab[:subset_size]
4
5 class Rescaling(Layer):
6     def __init__(self, scale=1., offset=0.):
7         super().__init__()
8         self.scale = scale
9         self.offset = offset
10
11    def call(self, inputs):
12        return inputs*self.scale + self.offset
13
14 # Générateurs
15 def generator_train(only_X=False):
16     for i in range(len(X_path)):
17         image = Image.open(X_path[i]).convert('RGB')
18         image = image.resize((224, 224), Image.Resampling.LANCZOS)
19         X_ = np.array(image, dtype=int)
20         if only_X:
21             yield preprocess_input(X_)
22         else:
23             yield (preprocess_input(X_), y_lab[i])
24
25 def generator_test(only_X=False):
26     for i in range(len(X_path_test)):
27         image = Image.open(X_path_test[i]).convert('RGB')
28         image = image.resize((224, 224), Image.Resampling.LANCZOS)
29         X_ = np.array(image, dtype=int)
30         if only_X:
31             yield preprocess_input(X_)
32         else:
33             yield (preprocess_input(X_), y_lab_test[i])
34
35 data_train = tf.data.Dataset.from_generator(generator_train,
36                                             output_types=(tf.float32, tf.float32),
37                                             output_shapes=([224,224,3], [len(one.categories_[0])]))
38
39 data_test = tf.data.Dataset.from_generator(generator_test,
40                                            output_types=(tf.float32, tf.float32),
41                                            output_shapes=([224,224,3], [len(one.categories_[0])]))
42
43 X_train = tf.data.Dataset.from_generator(generator_train, args=(True,),
44                                          output_types=tf.float32,
45                                          output_shapes=[224,224,3])
46
47 X_test = tf.data.Dataset.from_generator(generator_test, args=(True,),
48                                         output_types=tf.float32,
49                                         output_shapes=[224,224,3])
50
51 def get_model(input_shape=(224, 224, 3)):
52     inputs = Input(input_shape)
53     modeled = Rescaling(1./127.5, offset=-1.0)(inputs)
54     modeled = Conv2D(32, 5, activation='relu')(modeled)
55     modeled = MaxPooling2D(2, 2)(modeled)
56     modeled = BatchNormalization()(modeled)
57     modeled = Conv2D(48, 5, activation='relu')(modeled)
58     modeled = BatchNormalization()(modeled)
59     modeled = MaxPooling2D(2, 2)(modeled)
60     modeled = Conv2D(64, 5, activation='relu')(modeled)
61     modeled = BatchNormalization()(modeled)
```

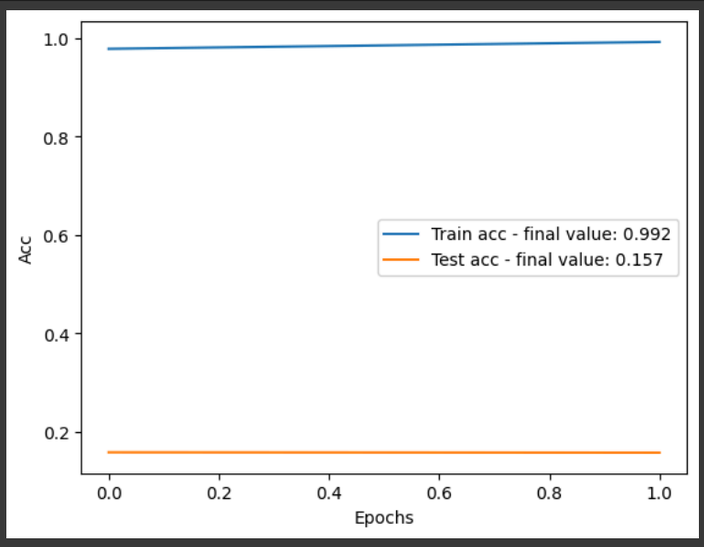
```
    modeled = MaxPooling2D(2, 2)(modeled)
    modeled = Conv2D(128, 5, activation='relu')(modeled)
    modeled = BatchNormalization()(modeled)
    modeled = MaxPooling2D(2, 2)(modeled)
    modeled = Flatten()(modeled)
    modeled = Dropout(0.5)(modeled)
    modeled = Dense(len(one.categories_[0]), activation="softmax")(modeled)
    model = Model(inputs, modeled)
    model.compile(optimizer=Adam(0.001), loss='categorical_crossentropy', metrics=["acc"])
    return model

model = get_model()
model.summary()
```

Afficher la sortie masquée

```
model.fit(data_train.batch(8), epochs=2, validation_data=data_test.batch(8))

```

```
Epoch 1/2
125/125 [==============================] - 32s 144ms/step - loss: 0.0922 - acc: 0.9780 - val_loss: 30.0200 - val_acc: 0.1580
Epoch 2/2
125/125 [==============================] - 26s 209ms/step - loss: 0.0286 - acc: 0.9920 - val_loss: 59.9577 - val_acc: 0.1573
<keras.src.callbacks.History at 0x7d271f261480>
```

```
acc = model.history.history["acc"]; val_acc = model.history.history["val_acc"]
plt.plot(acc, label="Train acc - final value: %.3f"%acc[-1])
plt.plot(val_acc, label="Test acc - final value: %.3f"%val_acc[-1])
plt.legend(); plt.xlabel("Epochs"); plt.ylabel("Acc"); plt.show()
```



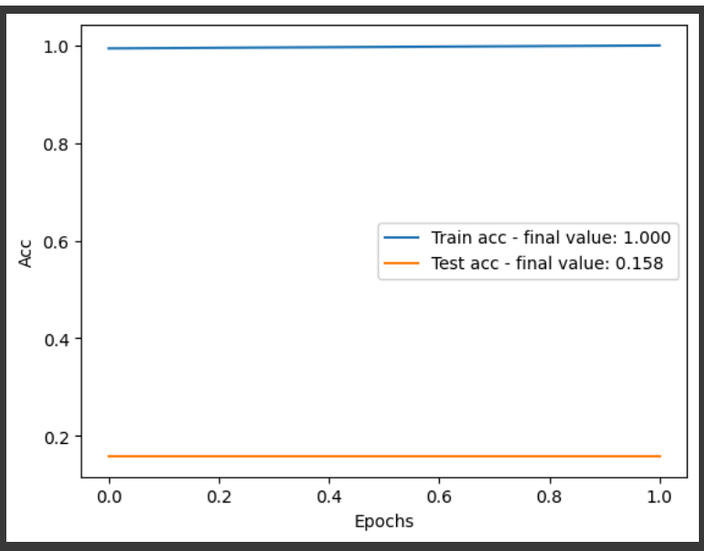

```
# Maintenant, on passe au fine-tuning avec ResNet50
def load_resnet50(path=None):
    model = ResNet50(include_top=False, input_shape=(224, 224, 3), pooling="avg")
    for i in range(len(model.layers)):
        if model.layers[i].__class__.__name__ == "BatchNormalization":
            model.layers[i].trainable = False
    return model

resnet50 = load_resnet50()
X_train_enc = resnet50.predict(X_train.batch(8))
X_test_enc = resnet50.predict(X_test.batch(8))

print("X train shape: %s"%str(X_train_enc.shape))

def get_task():
    model = Sequential()
    model.add(Dense(1024, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(1024, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(len(one.categories_[0]), activation="softmax"))
    return model

```

```
125/125 [==============================] - 6s 35ms/step
375/375 [==============================] - 13s 35ms/step
X train shape: (1000, 2048)
```

```
task = get_task()
task.compile(loss="categorical_crossentropy", optimizer=Adam(0.001), metrics=["acc"])
task.fit(X_train_enc, y_lab, epochs=2, batch_size=8,
         validation_data=(X_test_enc, y_lab_test))

```

```
Epoch 1/2
125/125 [==============================] - 3s 11ms/step - loss: 0.0193 - acc: 0.9940 - val_loss: 130.8804 - val_acc: 0.1580
Epoch 2/2
125/125 [==============================] - 1s 9ms/step - loss: 0.0000e+00 - acc: 1.0000 - val_loss: 130.8811 - val_acc: 0.1580
<keras.src.callbacks.History at 0x7d26a0231b40>
```

```
acc = task.history.history["acc"]; val_acc = task.history.history["val_acc"]
plt.plot(acc, label="Train acc - final value: %.3f"%acc[-1])
plt.plot(val_acc, label="Test acc - final value: %.3f"%val_acc[-1])
plt.legend(); plt.xlabel("Epochs"); plt.ylabel("Acc"); plt.show()
```

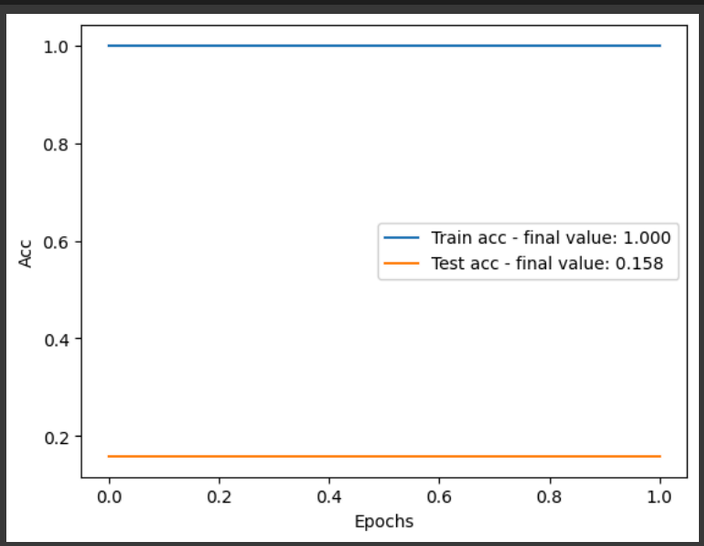



```
from adapt.parameter_based import FineTuning

encoder = load_resnet50()
task = get_task()

optimizer = Adam(0.001)
optimizer_enc = Adam(0.00001)

finetunig = FineTuning(encoder=encoder,
                       task=task,
                       optimizer=optimizer,
                       optimizer_enc=optimizer_enc,
                       loss="categorical_crossentropy",
                       metrics=["acc"],
                       copy=False,
                       pretrain=True,
                       pretrain__epochs=2)

```

```
finetunig.fit(data_train.batch(8), epochs=2, validation_data=data_test.batch(8))

```

```
Computing src dataset size...
Done!
Computing tgt dataset size...
Done!
Epoch 1/2
128/128 [==============================] - 37s 193ms/step - loss: 0.0178 - acc: 0.9922 - val_loss: 115.0975 - val_acc: 0.1580
Epoch 2/2
128/128 [==============================] - 30s 187ms/step - loss: 1.1921e-07 - acc: 1.0000 - val_loss: 115.0979 - val_acc: 0.1580
Epoch 1/2
128/128 [==============================] - 73s 243ms/step - loss: 1.1921e-07 - acc: 1.0000 - val_loss: 115.0979 - val_acc: 0.1580
Epoch 2/2
128/128 [==============================] - 35s 225ms/step - loss: 1.1921e-07 - acc: 1.0000 - val_loss: 115.0979 - val_acc: 0.1580
<adapt.parameter_based._finetuning.FineTuning at 0x7d268deba470>
```

```
acc = finetunig.history.history["acc"]; val_acc = finetunig.history.history["val_acc"]
plt.plot(acc, label="Train acc - final value: %.3f"%acc[-1])
plt.plot(val_acc, label="Test acc - final value: %.3f"%val_acc[-1])
plt.legend(); plt.xlabel("Epochs"); plt.ylabel("Acc"); plt.show()
```



Note importante : En raison de limitations matérielles (problèmes de GPU et de RAM insuffisants), il n'a pas été possible d'effectuer des entraînements avec un nombre élevé d'epochs. Par conséquent, les résultats présentés ici sont basés sur un entraînement avec un nombre réduit d'epochs(2) , ce qui peut affecter la qualité des performances finales. Pour obtenir des résultats plus robustes, il serait idéal de réexécuter ces expériences sur une machine équipée d'un GPU performant et de plus de mémoire RAM.

```
Commencez à coder ou à générer avec l'IA.
```

Conclusion

Dans ce notebook, nous avons démontré comment effectuer du fine-tuning en utilisant la toolbox adapt sur deux jeux de données différents :

Jeu de Données Flowers : Entraînement d'un modèle CNN personnalisé, extraction des features avec ResNet50, et application du fine-tuning avec adapt pour améliorer les performances de reconnaissance des fleurs. Jeu de Données Intel pour la Classification : Entraînement similaire sur un autre jeu de données de classification d'images, démontrant la flexibilité et l'efficacité des techniques de fine-tuning pour différents types de données. Ces exemples illustrent l'importance du fine-tuning et de l'adaptation de domaine pour améliorer la généralisation et les performances des modèles de deep learning sur divers jeux de données.

Exemple de Fine-Tuning Avancé avec la Toolbox `adapt` (deep domain adaptation)

Ce notebook présente deux exemples avancés de fine-tuning en utilisant la toolbox `adapt` :

1. **Utilisation du jeu de données Office31** (comme dans l'exemple original de la toolbox).
2. **Test sur un jeu de données différent, Mini-VisDA2017**, en se concentrant uniquement sur les classes `plant`, `truck` et `bicycle`.

```
## Remarque Importante
'''
Pour éviter les problèmes de compatibilité de versions, il est recommandé d'exécuter les
commandes suivantes:( c'est que pour le deep domain adaptation!!)
'''

!pip uninstall -y numpy
!pip install numpy==1.24.0

```

```
Found existing installation: numpy 1.26.2
Uninstalling numpy-1.26.2:
  Successfully uninstalled numpy-1.26.2
Collecting numpy==1.24.0
  Downloading numpy-1.24.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.6 kB)
Downloading numpy-1.24.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 64.8 MB/s eta 0:00:00
Installing collected packages: numpy
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source
albucore 0.0.19 requires numpy>=1.24.4, but you have numpy 1.24.0 which is incompatible.
albumentations 1.4.20 requires numpy>=1.24.4, but you have numpy 1.24.0 which is incompatible.
chex 0.1.88 requires numpy>=1.24.1, but you have numpy 1.24.0 which is incompatible.
pymc 5.19.1 requires numpy>=1.25.0, but you have numpy 1.24.0 which is incompatible.
seaborn 0.13.2 requires numpy!=1.24.0,>=1.20, but you have numpy 1.24.0 which is incompatible.
tensorstore 0.1.71 requires ml_dtypes>=0.3.1, but you have ml-dtypes 0.2.0 which is incompatible.
tf-keras 2.17.0 requires tensorflow<2.18,>=2.17, but you have tensorflow 2.15.0 which is incompatible.
Successfully installed numpy-1.24.0
WARNING: The following packages were previously imported in this runtime:
  [numpy]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

Exemple 1 : testerle code original avec le Jeu de Données Office31 et la Toolbox `adapt`

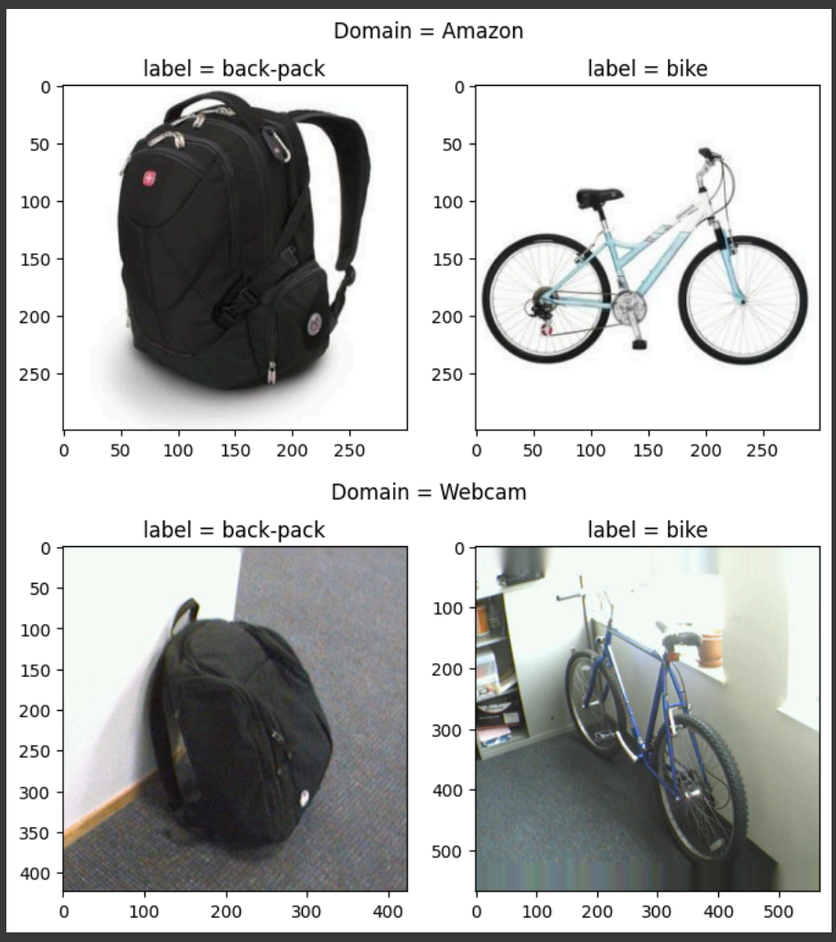
Dans cet exemple, nous effectuons du fine-tuning sur le jeu de données Office31 en utilisant un modèle ResNet50 pré-entraîné, puis nous appliquons des techniques d'adaptation de domaine pour améliorer les performances de classification entre les domaines `Amazon` et `Webcam`.

```
# === Importation des bibliothèques ===

# Standard libraries
import os
import numpy as np

# Deep Learning avec TensorFlow
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import preprocess_input, ResNet50
from tensorflow.keras.models import Model, load_model
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.constraints import MaxNorm
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers.schedules import LearningRateSchedule

# Adaptation de domaine avec ADAPT
from adapt.parameter_based import FineTuning
from adapt.feature_based import MDD
from adapt.utils import UpdateLambda

# Machine Learning
from sklearn.preprocessing import OneHotEncoder
from sklearn.manifold import TSNE

# Visualisation
import matplotlib.pyplot as plt
from PIL import Image

```

```
import tarfile

tar_file_path = "/content/domain_adaptation_images.tar.gz"

# Extraire le fichier tar.gz
with tarfile.open(tar_file_path, "r:gz") as tar_ref:
```

```
    tar_ref.extractall("office31")

```

```
path = "office31"
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))
ax1.imshow(plt.imread(path+"/amazon/images/back_pack/frame_0001.jpg"))
ax2.imshow(plt.imread(path+"/amazon/images/bike/frame_0001.jpg"))
ax1.set_title("label = back-pack"); ax2.set_title("label = bike");
plt.suptitle("Domain = Amazon"); plt.show()
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))
ax1.imshow(plt.imread(path+"/webcam/images/back_pack/frame_0001.jpg"))
ax2.imshow(plt.imread(path+"/webcam/images/bike/frame_0001.jpg"))
ax1.set_title("label = back-pack"); ax2.set_title("label = bike");
plt.suptitle("Domain = Webcam"); plt.show()
```




```
def get_Xy(domain, path_to_folder="office31/", max_images_per_class=100):

    path = os.path.join(path_to_folder, domain, "images")
    X = []
    y = []
    # Dictionnaire pour compter le nombre d'images par classe
    class_count = {}

    for root, dirs, files in os.walk(path):
        for file in files:
            if file.lower().endswith(('.jpg', '.jpeg', '.png')):
                class_name = os.path.basename(root)
                if ".ipynb_checkpoints" in class_name:
                    continue
                if class_name not in class_count:
                    class_count[class_name] = 0
                if class_count[class_name] < max_images_per_class:
                    path_to_image = os.path.join(root, file)
                    if ".ipynb_checkpoints" not in path_to_image:
                        image = Image.open(path_to_image)
                        image = image.resize((224, 224), Image.Resampling.LANCZOS)
                        image = np.array(image, dtype=int)
                        X.append(image)
                        y.append(class_name)
                        class_count[class_name] += 1

    return np.array(X), np.array(y)

Xs, ys = get_Xy("amazon", max_images_per_class=20)
Xt, yt = get_Xy("webcam", max_images_per_class=20)

```

```

resnet50 = ResNet50(include_top=False, input_shape=(224, 224, 3), pooling="avg")

first_layer = resnet50.get_layer('conv5_block2_out')
inputs = Input(first_layer.output_shape[1:])

for layer in resnet50.layers[resnet50.layers.index(first_layer)+1:]:
    if layer.name == "conv5_block3_1_conv":
        x = layer(inputs)
    elif layer.name == "conv5_block3_add":
```

```
        x = layer([inputs, x])
    else:
        x = layer(x)

first_blocks = Model(resnet50.input, first_layer.output)
last_block = Model(inputs, x)
```

```
def load_resnet50(path="resnet50_last_block.hdf5"):
    model = load_model(path)
    for i in range(len(model.layers)):
        if model.layers[i].__class__.__name__ == "BatchNormalization":
            model.layers[i].trainable = False
    return model

last_block.summary()
last_block.save("resnet50_last_block.hdf5")
```

```
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_6 (InputLayer)        [(None, 7, 7, 2048)]         0         []

 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)            1049088   ['input_6[0][0]']
 D)

 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block3_1_conv[1][0]']
 rmalization)

 conv5_block3_1_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block3_1_bn[1][0]']
 ation)

 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)            2359808   ['conv5_block3_1_relu[1][0]']
 D)

 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block3_2_conv[1][0]']
 rmalization)

 conv5_block3_2_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block3_2_bn[1][0]']
 ation)

 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)           1050624   ['conv5_block3_2_relu[1][0]']
 D)

 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)           8192      ['conv5_block3_3_conv[1][0]']
 rmalization)

 conv5_block3_add (Add)      (None, 7, 7, 2048)           0         ['input_6[0][0]',
                                                                     'conv5_block3_3_bn[1][0]']

 conv5_block3_out (Activati  (None, 7, 7, 2048)           0         ['conv5_block3_add[1][0]']
 on)

 avg_pool (GlobalAveragePoo  (None, 2048)                 0         ['conv5_block3_out[1][0]']
 ling2D)

==================================================================================================
Total params: 4471808 (17.06 MB)
Trainable params: 4465664 (17.04 MB)
Non-trainable params: 6144 (24.00 KB)
__________________________________________________________________________________________________
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until
```

```

def batch_predict(images, batch_size=100):
    features = []
    for i in range(0, len(images), batch_size):
        batch = images[i:i+batch_size]
        batch = preprocess_input(np.stack(batch))
        batch_features = first_blocks.predict(batch)
        features.append(batch_features)
    return np.concatenate(features, axis=0)

Xs_features = batch_predict(Xs)
Xt_features = batch_predict(Xt)


```

```
4/4 [==============================] - 5s 323ms/step
4/4 [==============================] - 0s 74ms/step
4/4 [==============================] - 0s 72ms/step
4/4 [==============================] - 0s 72ms/step
4/4 [==============================] - 0s 73ms/step
4/4 [==============================] - 0s 74ms/step
1/1 [==============================] - 1s 1s/step
4/4 [==============================] - 0s 75ms/step
4/4 [==============================] - 0s 74ms/step
4/4 [==============================] - 0s 74ms/step
4/4 [==============================] - 0s 77ms/step
4/4 [==============================] - 0s 76ms/step
3/3 [==============================] - 2s 976ms/step
```

```
one = OneHotEncoder(sparse_output=False)
one.fit(np.array(ys).reshape(-1, 1))

ys_lab = one.transform(np.array(ys).reshape(-1, 1))
yt_lab = one.transform(np.array(yt).reshape(-1, 1))

print("X source shape: %s" % str(Xs_features.shape))
print("X target shape: %s" % str(Xt_features.shape))

```

```
X source shape: (620, 7, 7, 2048)
X target shape: (591, 7, 7, 2048)
```

```

def get_task(dropout=0.5, max_norm=0.5):
    model = Sequential()
    model.add(Dense(1024, activation="relu",
                    kernel_constraint=MaxNorm(max_norm),
                    bias_constraint=MaxNorm(max_norm)))
    model.add(Dropout(dropout))
    model.add(Dense(1024, activation="relu",
                    kernel_constraint=MaxNorm(max_norm),
                    bias_constraint=MaxNorm(max_norm)))
    model.add(Dropout(dropout))
    model.add(Dense(31, activation="softmax",
                    kernel_constraint=MaxNorm(max_norm),
                    bias_constraint=MaxNorm(max_norm)))
    return model
```

```

class MyDecay(LearningRateSchedule):

    def __init__(self, max_steps=1000, mu_0=0.01, alpha=10, beta=0.75):
        self.mu_0 = mu_0
        self.alpha = alpha
        self.beta = beta
        self.max_steps = float(max_steps)

    def __call__(self, step):
        p = tf.cast(step, tf.float32) / self.max_steps
        return self.mu_0 / (1 + self.alpha * p)**self.beta
```

```
lr = 0.04
momentum = 0.9
alpha = 0.0002

optimizer_task = SGD(learning_rate=MyDecay(mu_0=float(lr), alpha=float(alpha)),
                     momentum=momentum, nesterov=True)
optimizer_enc = SGD(learning_rate=MyDecay(mu_0=float(lr)/10., alpha=float(alpha)),
                    momentum=momentum, nesterov=True)
```

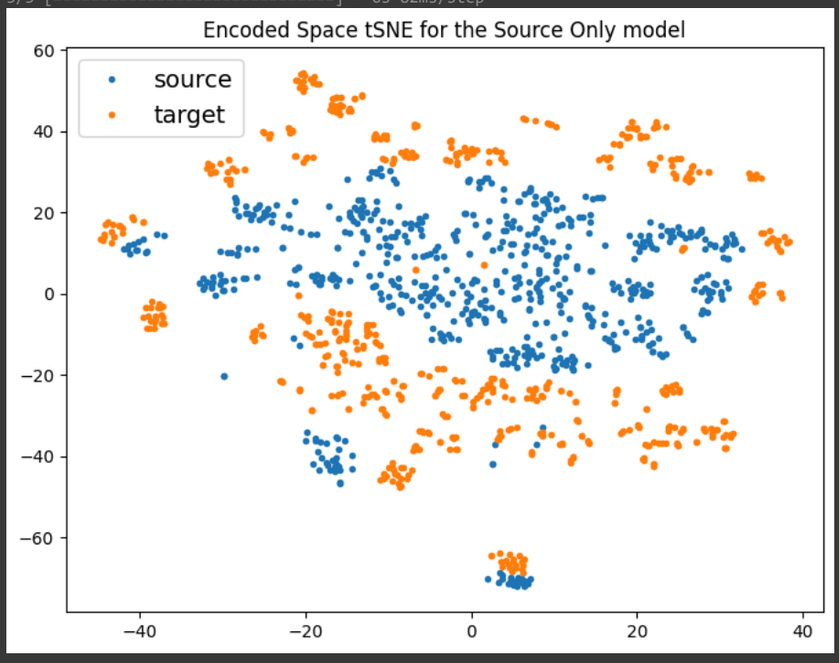
```


finetunig = FineTuning(encoder=load_resnet50(),
                        task=get_task(),
                        optimizer=optimizer_task,
                        optimizer_enc=optimizer_enc,
                        loss="categorical_crossentropy",
                        metrics=["acc"],
                        copy=False,
                        pretrain=True,
                        pretrain__epochs=5)

finetunig.fit(Xs_features[:10], ys_lab[:10], epochs=1, batch_size=8, validation_data=(Xt_features[:10], yt_lab[:10]))
finetunig.fit(Xs_features, ys_lab, epochs=100, batch_size=8, validation_data=(Xt_features, yt_lab))
```

```
78/78 [==============================] - 1s 15ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6814 - val_acc: 0.0338
Epoch 82/100
78/78 [==============================] - 2s 20ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6775 - val_acc: 0.0338
Epoch 83/100
78/78 [==============================] - 1s 15ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6736 - val_acc: 0.0338
Epoch 84/100
78/78 [==============================] - 1s 15ms/step - loss: 15.5757 - acc: 0.0337 - val_loss: 88.6697 - val_acc: 0.0338
Epoch 85/100
78/78 [==============================] - 1s 14ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6658 - val_acc: 0.0338
Epoch 86/100
78/78 [==============================] - 1s 17ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6619 - val_acc: 0.0338
Epoch 87/100
78/78 [==============================] - 1s 19ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6581 - val_acc: 0.0338
Epoch 88/100
78/78 [==============================] - 1s 16ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6542 - val_acc: 0.0338
Epoch 89/100
78/78 [==============================] - 1s 15ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6503 - val_acc: 0.0338
Epoch 90/100
78/78 [==============================] - 1s 16ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6464 - val_acc: 0.0338
Epoch 91/100
78/78 [==============================] - 1s 15ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6425 - val_acc: 0.0338
Epoch 92/100
78/78 [==============================] - 1s 15ms/step - loss: 15.6015 - acc: 0.0321 - val_loss: 88.6386 - val_acc: 0.0338
Epoch 93/100
```

```
acc = finetunig.history.history["acc"]; val_acc = finetunig.history.history["val_acc"]
plt.plot(acc, label="Train acc - final value: %.3f"%acc[-1])
plt.plot(val_acc, label="Test acc - final value: %.3f"%val_acc[-1])
plt.legend(); plt.xlabel("Epochs"); plt.ylabel("Acc"); plt.show()
```

```

Xs_enc = finetunig.transform(batch_predict(Xs))
Xt_enc = finetunig.transform(batch_predict(Xt))

np.random.seed(0)
X_ = np.concatenate((Xs_enc, Xt_enc))
X_tsne = TSNE(2).fit_transform(X_)
plt.figure(figsize=(8, 6))
plt.plot(X_tsne[:len(Xs), 0], X_tsne[:len(Xs), 1], '.', label="source")
plt.plot(X_tsne[len(Xs):, 0], X_tsne[len(Xs):, 1], '.', label="target")
plt.legend(fontsize=14)
plt.title("Encoded Space tSNE for the Source Only model")
plt.show()
```

```
4/4 [==============================] - 0s 101ms/step
4/4 [==============================] - 0s 78ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 27ms/step
4/4 [==============================] - 0s 79ms/step
4/4 [==============================] - 0s 79ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 81ms/step
3/3 [==============================] - 0s 82ms/step
```




```
np.random.seed(0)
shuffle_src = np.random.choice(len(Xs), len(Xs), replace=False)
shuffle_tgt = np.random.choice(len(Xt), len(Xt), replace=False)

Xs = Xs[shuffle_src]
ys_lab = ys_lab[shuffle_src]
Xt = Xt[shuffle_tgt]
yt_lab = yt_lab[shuffle_tgt]
```

```
np.random.seed(123)
tf.random.set_seed(123)

lr = 0.04
momentum = 0.9
```

```
alpha = 0.0002

encoder = load_resnet50()
task = get_task()

optimizer_task = SGD(learning_rate=MyDecay(mu_0=lr, alpha=alpha),
                     momentum=momentum, nesterov=True)
optimizer_enc = SGD(learning_rate=MyDecay(mu_0=lr/10., alpha=alpha),
                    momentum=momentum, nesterov=True)
optimizer_disc = SGD(learning_rate=MyDecay(mu_0=lr/10., alpha=alpha))


mdd = MDD(encoder, task,
            loss="categorical_crossentropy",
            metrics=["acc"],
            copy=False,
            lambda_=tf.Variable(0.),
            gamma=2.,
            optimizer=optimizer_task,
            optimizer_enc=optimizer_enc,
            optimizer_disc=optimizer_disc,
            callbacks=[UpdateLambda(lambda_max=0.1)])
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
```

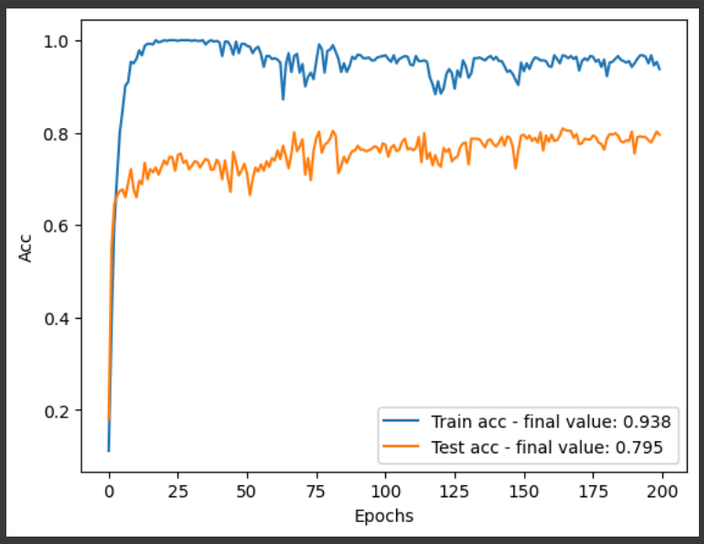
```
Xs_features = batch_predict(Xs[:-1])
Xt_features = batch_predict(Xt)
```

```
4/4 [==============================] - 1s 105ms/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 80ms/step
1/1 [==============================] - 2s 2s/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 81ms/step
3/3 [==============================] - 0s 81ms/step
```

```
mdd.fit(X=Xs_features, y=ys_lab[:-1], Xt=Xt_features, epochs=200, batch_size=32, validation_data=(Xt_features, yt_lab))
```

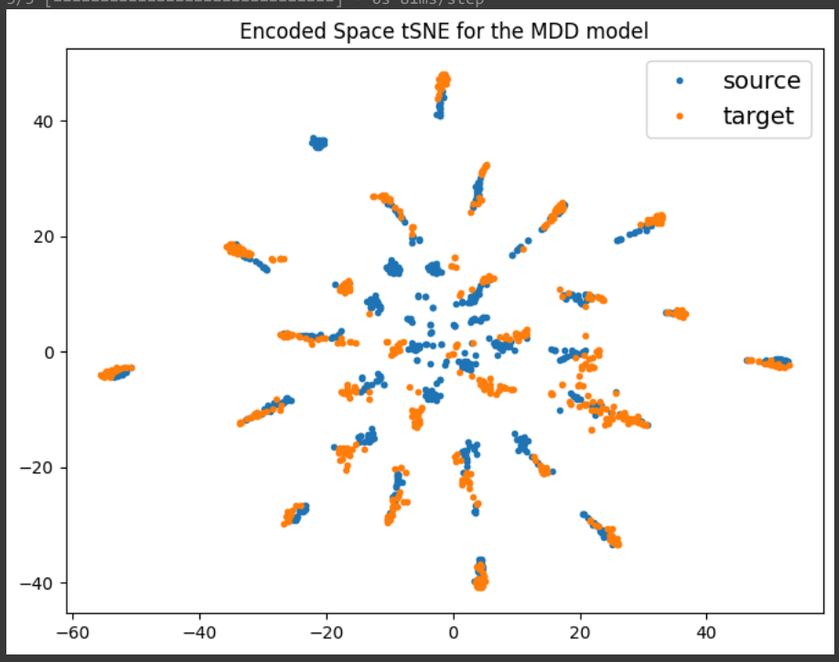
Afficher la sortie masquée

```
acc = mdd.history.history["acc"]; val_acc = mdd.history.history["val_acc"]
plt.plot(acc, label="Train acc - final value: %.3f"%acc[-1])
plt.plot(val_acc, label="Test acc - final value: %.3f"%val_acc[-1])
plt.legend(); plt.xlabel("Epochs"); plt.ylabel("Acc"); plt.show()

```




```

Xs_features = batch_predict(Xs)
Xt_features = batch_predict(Xt)
Xs_enc = mdd.transform(Xs_features)
Xt_enc = mdd.transform(Xt_features)

np.random.seed(0)
X_ = np.concatenate((Xs_enc, Xt_enc))
X_tsne = TSNE(2).fit_transform(X_)
plt.figure(figsize=(8, 6))
plt.plot(X_tsne[:len(Xs), 0], X_tsne[:len(Xs), 1], '.', label="source")
plt.plot(X_tsne[len(Xs):, 0], X_tsne[len(Xs):, 1], '.', label="target")
plt.legend(fontsize=14)
plt.title("Encoded Space tSNE for the MDD model")
plt.show()
```

```
4/4 [==============================] - 0s 97ms/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 77ms/step
1/1 [==============================] - 0s 34ms/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 80ms/step
4/4 [==============================] - 0s 81ms/step
4/4 [==============================] - 0s 79ms/step
4/4 [==============================] - 0s 80ms/step
3/3 [==============================] - 0s 81ms/step
```




```
Commencez à coder ou à générer avec l'IA.
```

```
Commencez à coder ou à générer avec l'IA.
```

Exemple 2 : Fine-Tuning avec le Jeu de Données Mini-VisDA2017

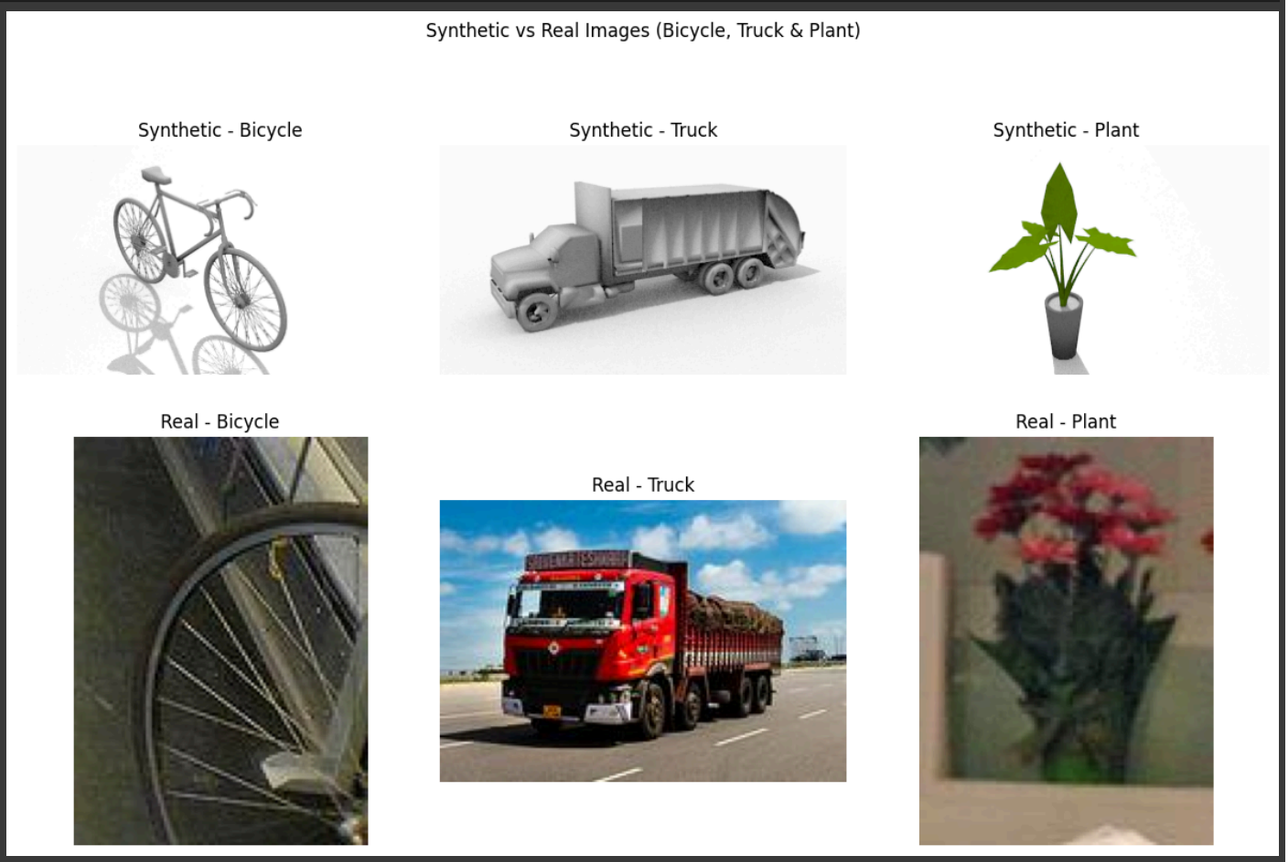
Dans cet exemple, nous testons les techniques d'adaptation de domaine sur un jeu de données différent, Mini-VisDA2017, en se concentrant uniquement sur les classes `plant`, `truck` et `bicycle`.

```python
# === Importation des bibliothèques ===

# Standard libraries
import os
import numpy as np

# Deep Learning avec TensorFlow
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import preprocess_input, ResNet50
from tensorflow.keras.models import Model, load_model
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.constraints import MaxNorm
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers.schedules import LearningRateSchedule

# Adaptation de domaine avec ADAPT
from adapt.feature_based import MDD
from adapt.utils import UpdateLambda

# Machine Learning
from sklearn.preprocessing import OneHotEncoder
from sklearn.manifold import TSNE

# Visualisation
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image

```

```python
import zipfile

zip_file_path = "/content/mini_visda2017.zip"

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall("mini_visda")

```

```python
path = "mini_visda"

def get_first_image(folder):
    for file in os.listdir(folder):
        if file.lower().endswith(('.jpg', '.jpeg', '.png')):
            return os.path.join(folder, file)
```

```
    return None

fig, axes = plt.subplots(2, 3, figsize=(12, 8))

classes = ["bicycle", "truck", "plant"]

for i, cls in enumerate(classes):
    # Images synthetic (train)
    synthetic_img = get_first_image(os.path.join(path, "train", cls))
    if synthetic_img:
        axes[0, i].imshow(mpimg.imread(synthetic_img))
        axes[0, i].set_title(f"Synthetic - {cls.capitalize()}")
    else:
        axes[0, i].set_title(f"No image found for {cls} (train)")
    axes[0, i].axis('off')

    # Images real (validation)
    real_img = get_first_image(os.path.join(path, "validation", cls))
    if real_img:
        axes[1, i].imshow(mpimg.imread(real_img))
        axes[1, i].set_title(f"Real - {cls.capitalize()}")
    else:
        axes[1, i].set_title(f"No image found for {cls} (validation)")
    axes[1, i].axis('off')

plt.suptitle("Synthetic vs Real Images (Bicycle, Truck & Plant)")
plt.tight_layout()
plt.show()

```




```


def get_Xy(domain, path_to_folder="mini_visda", max_images_per_class=100):

    path = os.path.join(path_to_folder, domain)
    X, y = [], []
    class_count = {}

    for class_name in ["bicycle", "plant", "truck"]:
        class_path = os.path.join(path, class_name)
        class_count[class_name] = 0

        for file in os.listdir(class_path):
            if file.lower().endswith(('.jpg', '.jpeg', '.png')):
                if class_count[class_name] < max_images_per_class:
                    path_to_image = os.path.join(class_path, file)
                    try:
                        image = Image.open(path_to_image).resize((224, 224), Image.Resampling.LANCZOS)
                        image = np.array(image, dtype=np.float32)
                        X.append(image)
                        y.append(class_name)
                        class_count[class_name] += 1
                    except Exception as e:
                        print(f"Erreur de lecture d'image {path_to_image}: {e}")

    return np.array(X), np.array(y)

```

```
Xs, ys = get_Xy("train", max_images_per_class=50)       # Train (synthetic)
Xt, yt = get_Xy("validation", max_images_per_class=50)   # Domaine real (validation)

print("Shape of Xs (train):", Xs.shape)
print("Shape of ys (train):", ys.shape)
print("Shape of Xt (validation):", Xt.shape)
print("Shape of yt (validation):", yt.shape)

```

```
Shape of Xs (train): (150, 224, 224, 3)
Shape of ys (train): (150,)
Shape of Xt (validation): (150, 224, 224, 3)
Shape of yt (validation): (150,)
```

```

resnet50 = ResNet50(include_top=False, input_shape=(224, 224, 3), pooling="avg")

first_layer = resnet50.get_layer('conv5_block2_out')
inputs = Input(first_layer.output_shape[1:])

for layer in resnet50.layers[resnet50.layers.index(first_layer)+1:]:
    if layer.name == "conv5_block3_1_conv":
        x = layer(inputs)
    elif layer.name == "conv5_block3_add":
        x = layer([inputs, x])
    else:
        x = layer(x)

first_blocks = Model(resnet50.input, first_layer.output)
last_block = Model(inputs, x)
```

```
def load_resnet50(path="resnet50_last_block.hdf5"):
    model = load_model(path)
    for i in range(len(model.layers)):
        if model.layers[i].__class__.__name__ == "BatchNormalization":
            model.layers[i].trainable = False
    return model

last_block.summary()
last_block.save("resnet50_last_block.hdf5")
```

```
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_6 (InputLayer)        [(None, 7, 7, 2048)]      0         []

 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)         1049088   ['input_6[0][0]']
 D)

 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)         2048      ['conv5_block3_1_conv[1][0]']
 rmalization)

 conv5_block3_1_relu (Activ  (None, 7, 7, 512)         0         ['conv5_block3_1_bn[1][0]']
 ation)

 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)         2359808   ['conv5_block3_1_relu[1][0]']
 D)

 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)         2048      ['conv5_block3_2_conv[1][0]']
 rmalization)

 conv5_block3_2_relu (Activ  (None, 7, 7, 512)         0         ['conv5_block3_2_bn[1][0]']
 ation)

 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)        1050624   ['conv5_block3_2_relu[1][0]']
 D)

 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)        8192      ['conv5_block3_3_conv[1][0]']
 rmalization)

 conv5_block3_add (Add)      (None, 7, 7, 2048)        0         ['input_6[0][0]',
                                                                  'conv5_block3_3_bn[1][0]']

 conv5_block3_out (Activati  (None, 7, 7, 2048)        0         ['conv5_block3_add[1][0]']
 on)

 avg_pool (GlobalAveragePoo  (None, 2048)              0         ['conv5_block3_out[1][0]']
 ling2D)

==================================================================================================
Total params: 4471808 (17.06 MB)
Trainable params: 4465664 (17.04 MB)
Non-trainable params: 6144 (24.00 KB)
__________________________________________________________________________________________________
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until
```

```

def batch_predict(images, batch_size=100):
    features = []
    for i in range(0, len(images), batch_size):
        batch = images[i:i+batch_size]
        batch = preprocess_input(np.stack(batch))
        batch_features = first_blocks.predict(batch)
        features.append(batch_features)
    return np.concatenate(features, axis=0)

one = OneHotEncoder(sparse_output=False)
ys_lab = one.fit_transform(np.array(ys).reshape(-1, 1))
```

```
yt_lab = one.transform(np.array(yt).reshape(-1, 1))

print("Classes encodées:", one.categories_)

# Extraction des features avec ResNet50
Xs_features = batch_predict(Xs)
Xt_features = batch_predict(Xt)

print("Shape of Xs_features:", Xs_features.shape)
print("Shape of Xt_features:", Xt_features.shape)

```

```
Classes encodées: [array(['bicycle', 'plant', 'truck'], dtype='<U7')]
4/4 [==============================] - 2s 414ms/step
2/2 [==============================] - 2s 2s/step
4/4 [==============================] - 0s 79ms/step
2/2 [==============================] - 0s 77ms/step
Shape of Xs_features: (150, 7, 7, 2048)
Shape of Xt_features: (150, 7, 7, 2048)
```

```

def get_task(dropout=0.5, max_norm=0.5):
    model = Sequential()
    model.add(Dense(1024, activation="relu",
                    kernel_constraint=MaxNorm(max_norm),
                    bias_constraint=MaxNorm(max_norm)))
    model.add(Dropout(dropout))
    model.add(Dense(1024, activation="relu",
                    kernel_constraint=MaxNorm(max_norm),
                    bias_constraint=MaxNorm(max_norm)))
    model.add(Dropout(dropout))
    # Ajustement à 3 classes
    model.add(Dense(3, activation="softmax",
                    kernel_constraint=MaxNorm(max_norm),
                    bias_constraint=MaxNorm(max_norm)))
    return model
```

```
class MyDecay(LearningRateSchedule):

    def __init__(self, max_steps=1000, mu_0=0.01, alpha=10, beta=0.75):
        self.mu_0 = mu_0
        self.alpha = alpha
        self.beta = beta
        self.max_steps = float(max_steps)

    def __call__(self, step):
        p = tf.cast(step, tf.float32) / self.max_steps
        return self.mu_0 / (1 + self.alpha * p)**self.beta
```

```
lr = 0.04
momentum = 0.9
alpha = 0.0002


optimizer_task = SGD(learning_rate=MyDecay(mu_0=float(lr), alpha=float(alpha)),
                     momentum=momentum, nesterov=True)
optimizer_enc = SGD(learning_rate=MyDecay(mu_0=float(lr)/10., alpha=float(alpha)),
                    momentum=momentum, nesterov=True)
```

```
# Fine-tuning
finetuning = FineTuning(
    encoder=load_resnet50(),
    task=get_task(),
    optimizer=optimizer_task,
    optimizer_enc=optimizer_enc,
    loss="categorical_crossentropy",
    metrics=["accuracy"],
    copy=False,
    pretrain=True,
    pretrain__epochs=5
)

```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
```

```
# Pré-entraînement rapide sur un sous-ensemble
finetuning.fit(
    Xs_features[:10], ys_lab[:10],
    epochs=1, batch_size=8,
    validation_data=(Xt_features[:10], yt_lab[:10])
)

```

```
Epoch 1/5
2/2 [==============================] - 4s 896ms/step - loss: 1.2184 - accuracy: 0.5000 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 2/5
2/2 [==============================] - 0s 38ms/step - loss: 1.2923e-04 - accuracy: 1.0000 - val_loss: 9.7752e-07 - val_accuracy: 1.0000
Epoch 3/5
2/2 [==============================] - 0s 35ms/step - loss: 1.1921e-07 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/5
2/2 [==============================] - 0s 42ms/step - loss: 1.1921e-07 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/5
2/2 [==============================] - 0s 49ms/step - loss: 1.1921e-07 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
2/2 [==============================] - 3s 243ms/step - loss: 1.1921e-07 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
<adapt.parameter_based._finetuning.FineTuning at 0x7af0acfaa050>
```

```
# Entraînement complet
finetuning.fit(
    Xs_features, ys_lab,
    epochs=100, batch_size=8,
    validation_data=(Xt_features, yt_lab)
)
```

```
Epoch 1/100
19/19 [==============================] - 1s 49ms/step - loss: 10.7100 - accuracy: 0.3355 - val_loss: 26.0824 - val_accuracy: 0.3333
Epoch 2/100
19/19 [==============================] - 0s 15ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.1905 - val_accuracy: 0.3333
Epoch 3/100
19/19 [==============================] - 0s 16ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3401 - val_accuracy: 0.3333
Epoch 4/100
19/19 [==============================] - 0s 16ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3603 - val_accuracy: 0.3333
Epoch 5/100
19/19 [==============================] - 0s 15ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3630 - val_accuracy: 0.3333
Epoch 6/100
19/19 [==============================] - 0s 16ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3633 - val_accuracy: 0.3333
Epoch 7/100
19/19 [==============================] - 0s 16ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3632 - val_accuracy: 0.3333
Epoch 8/100
19/19 [==============================] - 0s 16ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3629 - val_accuracy: 0.3333
Epoch 9/100
19/19 [==============================] - 0s 16ms/step - loss: 10.7100 - accuracy: 0.3355 - val_loss: 27.3626 - val_accuracy: 0.3333
Epoch 10/100
19/19 [==============================] - 0s 16ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3623 - val_accuracy: 0.3333
Epoch 11/100
19/19 [==============================] - 0s 16ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3621 - val_accuracy: 0.3333
Epoch 12/100
19/19 [==============================] - 0s 16ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3618 - val_accuracy: 0.3333
Epoch 13/100
19/19 [==============================] - 0s 16ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3615 - val_accuracy: 0.3333
Epoch 14/100
19/19 [==============================] - 0s 23ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3612 - val_accuracy: 0.3333
Epoch 15/100
19/19 [==============================] - 0s 19ms/step - loss: 10.7101 - accuracy: 0.3355 - val_loss: 27.3609 - val_accuracy: 0.3333
Epoch 16/100
19/19 [==============================] - 0s 19ms/step - loss: 10.7101 - accuracy: 0.3355 - val_loss: 27.3606 - val_accuracy: 0.3333
Epoch 17/100
19/19 [==============================] - 0s 21ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3603 - val_accuracy: 0.3333
Epoch 18/100
19/19 [==============================] - 0s 19ms/step - loss: 10.7101 - accuracy: 0.3355 - val_loss: 27.3600 - val_accuracy: 0.3333
Epoch 19/100
19/19 [==============================] - 0s 19ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3597 - val_accuracy: 0.3333
Epoch 20/100
19/19 [==============================] - 0s 16ms/step - loss: 10.7100 - accuracy: 0.3355 - val_loss: 27.3594 - val_accuracy: 0.3333
Epoch 21/100
19/19 [==============================] - 0s 17ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3591 - val_accuracy: 0.3333
Epoch 22/100
19/19 [==============================] - 0s 16ms/step - loss: 10.7100 - accuracy: 0.3355 - val_loss: 27.3588 - val_accuracy: 0.3333
Epoch 23/100
19/19 [==============================] - 0s 16ms/step - loss: 10.7100 - accuracy: 0.3355 - val_loss: 27.3585 - val_accuracy: 0.3333
Epoch 24/100
19/19 [==============================] - 0s 15ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3583 - val_accuracy: 0.3333
Epoch 25/100
19/19 [==============================] - 0s 16ms/step - loss: 10.7101 - accuracy: 0.3355 - val_loss: 27.3580 - val_accuracy: 0.3333
Epoch 26/100
19/19 [==============================] - 0s 15ms/step - loss: 10.7100 - accuracy: 0.3355 - val_loss: 27.3577 - val_accuracy: 0.3333
Epoch 27/100
19/19 [==============================] - 0s 15ms/step - loss: 10.8161 - accuracy: 0.3289 - val_loss: 27.3574 - val_accuracy: 0.3333
Epoch 28/100
19/19 [==============================] - 0s 16ms/step - loss: 10.7100 - accuracy: 0.3355 - val_loss: 27.3571 - val_accuracy: 0.3333
Epoch 29/100
19/19 [==============================] - 0s 15ms/step - loss: 10.7100 - accuracy: 0.3355 - val_loss: 27.3568 - val_accuracy: 0.3333
```

```
# Récupération des métriques d'entraînement et validation
acc = finetuning.history.history.get("accuracy", [])  # Train accuracy
val_acc = finetuning.history.history.get("val_accuracy", [])  # Validation accuracy

if acc and val_acc:
    plt.plot(acc, label="Train accuracy - final value: %.3f" % acc[-1])
    plt.plot(val_acc, label="Validation accuracy - final value: %.3f" % val_acc[-1])
    plt.legend()
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.title("FineTuning - Train vs Validation Accuracy")
    plt.show()
else:
```