ENSSAT LANNION
6 rue Kerampont
22300 LANNION
FRANCE

Rapport de Projet Technologique

Étude d'un modèle de classification d'images hyperspectrales entrainé sur des données non labélisées

Study of a hyperspectral image classification model trained on unlabeled data

Mattias Kockum
Informatique - ENSSAT Promotion 2023

Tuteur ENSSAT Benoît Vozel

6 Janvier 2024

1 Résumé

Résumé : Dans le but de mettre au point un modèle de classification d'images hyperspectrales pixel par pixel entraîné sur des données non labellisées, nous avons élaboré une méthode visant à démontrer la faisabilité d'un tel entraînement. Pour cela, nous avons pris un jeu de données labellisées dont nous avons retiré les labels, pour entraîner notre modèle constitant en un transformeur modifié pour la vision d'images et d'un modèle K-means. Nous avons ensuite comparé les prédictions de ce modèle aux labels originaux de la donnée. Les similarités que nous avons observées permettent d'être enthousiastes à l'idée de pouvoir classer des images hyperspectrales pixel par pixel sans labellisation préalable pour en tirer une représentation porteuse de sens.
Mots-clés : Classification, Apprentissage Automatique, Données Non Labellisées, Images Hyperspectrales, Transformeur.
Abstract : With the aim of developing a pixel-by-pixel hyperspectral image classification model trained on unlabeled data, we developed a method to demonstrate the feasibility of such training. To this end, we took a labeled dataset from which we removed the labels, to train our model consisting of a modified transformer for image processing and a K-means model. We then compared the predictions of this model with the original labels of the dataset. The similarities we observed allow us to be enthusiastic about the idea of being able to classify hyperspectral images pixel by pixel, without prior labeling, to derive a meaningful representation.
Keywords: Classification, Machine Learning, Unlabeled Data, Hyperspectral Images, Transformers.

2 Remerciements

Avant de continuer, je tiens à remercier M Vozel pour son encadrement tout au long de mon projet technologique. Je remercie aussi le corps enseignant de l'ENSSAT pour sa guidance et son aide.

Table des matières

Table des figures

3 Introduction

Les images hyperspectrales permettent, grâce à leurs données supplémentaires comparées aux images classiques, de distinguer deux objets différents qui ont la même couleur. Un exemple serait le gazon naturel et synthétique : dans le visible, les deux paraissent d'un même vert, mais leurs signatures spectrales étant différentes, il est possible de les distinguer ainsi. Il en va de même pour les feuilles de différentes essences d'arbres qui paraissent toutes aussi vertes à l'œil nu. Une image hyperspectrale se présente comme un bloc dont la largeur et la longueur correspondent à celles de l'image, et dont la profondeur correspond au nombre de longueurs d'ondes capturées. Réussir à classer des images hyperspectrales pixel par pixel (c'est à dire attribuer une classe à chaque pixel de l'image) est donc d'un intérêt majeur dans plusieurs domaines de reconnaissance du terrain vu du ciel, notamment dans le secteur de la surveillance et l'observation de l'écosystème.

L'état de l'art dans le domaine de la classification d'image hyperspectrale a longtemps été dominé par les réseaux de convolution, mais les modèles à base de transformeurs ont dernièrement vu grandir l'intérêt qui leur était porté [4]. Le principe d'un réseau de convolution est de faire passer des noyaux sur une image pour construire une représentation de celle-ci qui soit porteuse de sens. Cette approche est populaire car elle permet d'obtenir de bons résultats à un coup assez faible. Le modèle du transformeurs est quant à lui plus dur à entraîner, mais permet d'obtenir de meilleurs résultats dans les domaines nécessitant de capturer des liens de longue distance dans les données, sa popularité a donc explosé dans le domaine du traitement du langage naturel [5] et a ensuite gagné le domaine de l'analyse d'images [2, 1].

Cependant, les études au sujet de la classification d'images hyperspectrales partent toutes de jeux de données faiblement labellisés [4]. Or, l'ENSSAT travaille avec des images hyperspectrales prises du ciel. Ces images ne sont pas labellisées, mais doivent être étudiées. Nous avons donc cherché à savoir s'il était possible d'obtenir une représentation porteuse de sens à partir de données

non labellisées.

4 Méthode

4.1 Données

Les données à classifier sont des données hyperspectrales, ici notre but est de traiter les données de l'ENSSAT [fig 1], mais notre étude portera aussi sur les données Houston2018 [fig 2]. En effet, les données de l'ENSSAT sont totalement non labellisées, ce qui complique la tâche de savoir si notre modèle prédit des classifications intéressantes. En revanche, les données de Houston2018 sont labellisées, ce qui permet de comparer les résultats du modèle avec la réalité du terrain.

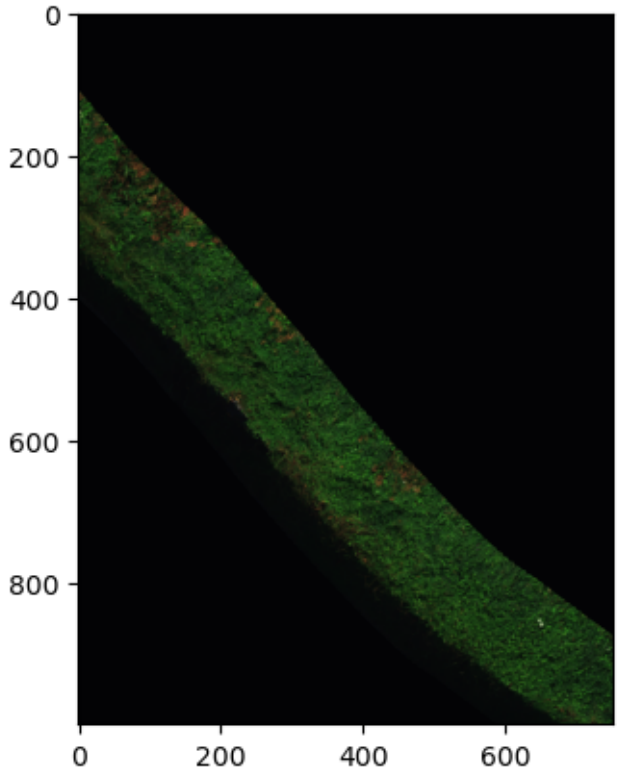


Figure 1: Représentation en couleurs visibles des données de l'ENSSAT.

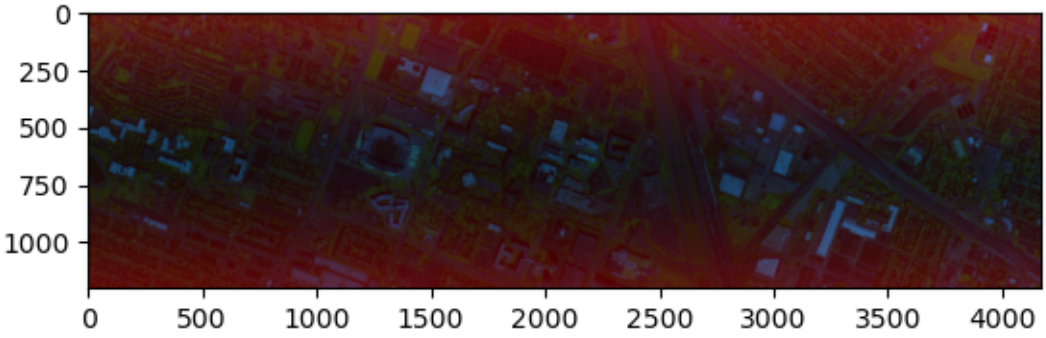


Figure 2: Représentation en couleurs visibles des données de Houston2018.

Cependant, ces données ne sont pas identiques, car les données de l'ENSSAT

comportent 224 bandes allant de 398 à 1004 nanomètres, soit légèrement au-dessus du violent et en dessous du rouge dans le spectre du visible (respectivement 380 et 750 nanomètres), là où les données de Houston2018 comprennent 48 bandes allant de 380 à 1050 nanomètres. Différence suplémmentaires, les données de Houston2018 remplissent un rectangle entier tandis que les données de l'ENSSAT correspondent à une langue de mesures, signifiant que de nombreux segments de cette donnée sont vides.

4.2 Outils

Pour classifier des images hyperspectrales non labellisées, nous avons opté pour la technologie des transformeurs en raison de ses récentes démonstrations de performance dans le domaine. De plus, les transformeurs permettent au modèle de créer des liens logique entre des données qui sont éloignées les unes des autres, ce qui est une propriété intéressante lorsque l'on veut analyser des images hyperspectrales. En effet, la valeur d'un pixel dans l'infrarouge et dans l'ultraviolet vont avoir un lien pour déduire l'objet représenté. Pour vérifier que notre méthode pouvait fonctionner sur des données non labellisées, nous avons décidé de créer une preuve de concept en l'appliquant à des données initialement labellisées dont nous avons retiré les labels pour finalement les comparer aux résultats. Pour cela, nous avons utilisé un NoteBook Python sur Amazon Web Service SageMaker car la plateforme permet d'utiliser les performances des ordinateurs du cloud pour nos expériences. Nous avons spécifiquement utilisé les bibliothèques PyTorch et SciKitLearn car ce sont les bibliothèques les plus modulaires et donc les plus utilisées en recherche. Aussi, utiliser PyTorch nous permettait de reprendre les modèles de transformeurs déjà implémentés dans PyTorch. Pour suivre les résultats des apprentissages nous avons utilisé Weights&Biases pour sa facilité d'utilisation.

4.3 Expérience

4.3.1 Principe de l'expérience

Le but de l'expérience est de savoir s'il est possible de classifier des images hyperspectrales non labellisées. Nous voulons dans un premier temps que le transformeur transforme une partie de l'image hyperspectrale (un pixel et ses alentours) en un point dans un espace vectoriel. Ce point doit être une représentation du pixel sélectionné de sorte que les points alentours soient des représentations de pixels similaires. On obtiendrait ainsi des nuages de points représentant chacun une classe d'objets à classifier (arbres, algues, route, *etc*...). Nous pouvons ensuite déduire la nature des objets représentés par les pixels en fonction du nuage de points dans lequel les projette le transformeur.

4.3.2 Déroulé de l'expérience

Pour mener à bien cette expérience, nous avons dû d'abord apprendre au modèle transformeur comment projeter deux points semblables au même endroit de l'espace vectoriel. Pour cela, nous avons émis l'hypothèse selon laquelle une image hyperspectrale et une image bruitée devaient avoir la même représentation dans l'espace vectoriel. En effet, un léger bruit ne doit pas changer la manière dont les pixels sont interprétés. Ensuite, pour obtenir une erreur mesurable de notre modèle et donc pouvoir le corriger et le faire apprendre, nous avons opté pour la technique de l'apprentissage contrastif. L'apprentissage contrastif est une méthode populaire quand il s'agit d'étudier des jeux de données non labellisées [3]. Cet apprentissage nécessite deux choses : une paire positive et une paire négative. Ici, la paire positive est le couple formé par deux images devant avoir la même représentation, et la paire négative, le couple formé par deux images devant avoir deux représentations différentes. L'apprentissage contrastif requiert qu'une image soit partagée entre les deux paires, elle s'appelle l'ancre. Nous avons donc formé nos paires ainsi : une image ancre, l'image ancre bruitée, et une autre image au hasard. L'erreur correspond alors à la distance entre les deux

images de la paire positive additionnée à la proximité des deux images de la paire négative. En diminuant l'erreur grâce à la méthode de la descente de gradient, les images similaires se rapprochent, et les images différentes s'éloignent. Une fois le transformeur entrainé, nous avons donc des nuages de points. Pour classifier ces nuages de points, nous avons utilisé la méthode des K-means avec la méthode du coude pour compter le nombre de nuages importants. Enfin, on applique successivement les deux modèles sur une fenêtre que l'on fait glisser sur l'image pour obtenir la représentation correspondante. La taille de fenêtre choisie est de 8x8 pixels.

4.4 Résultats

Dans un premier temps, l'application de la méthode exprimée précédemment sur les données de Houston2018 donne un résultat *a priori* satisfaisant. En effet, lorsque l'on compare ce que produit le modèle avec une image correspondante, il semble que le modèle reproduise les mêmes formes [fig 3, 4].

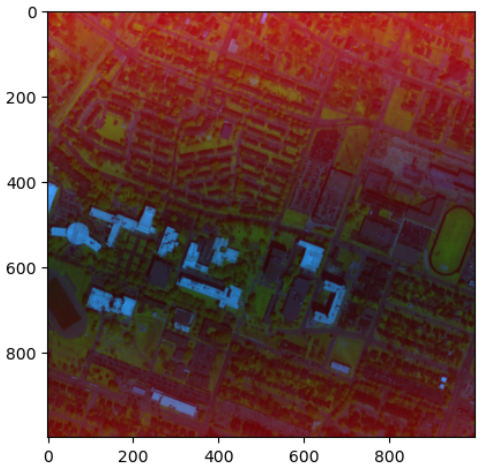


Figure 3: Image extraite de Houston2018 en couleurs réelles.

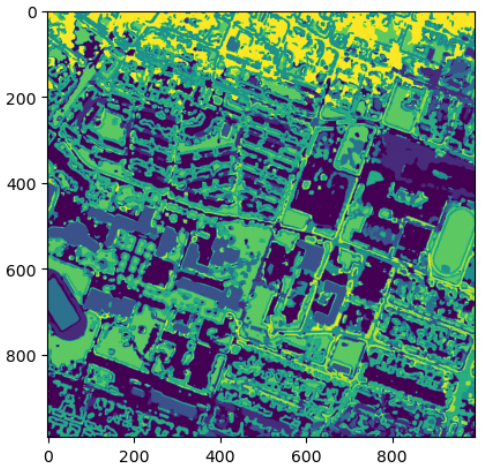


Figure 4: Production du modèle à partir d'une image extraite de Houston2018.

Cependant, pour ce qui est de la comparaison avec les labels de Houston2018,

notre méthode (méthode du coude [fig 5]) indique qu'il y a 9 nuages de points distincts, donc 9 classes, tandis que d'autres études utilisent une vingtaine de classes pour les mêmes données [4].

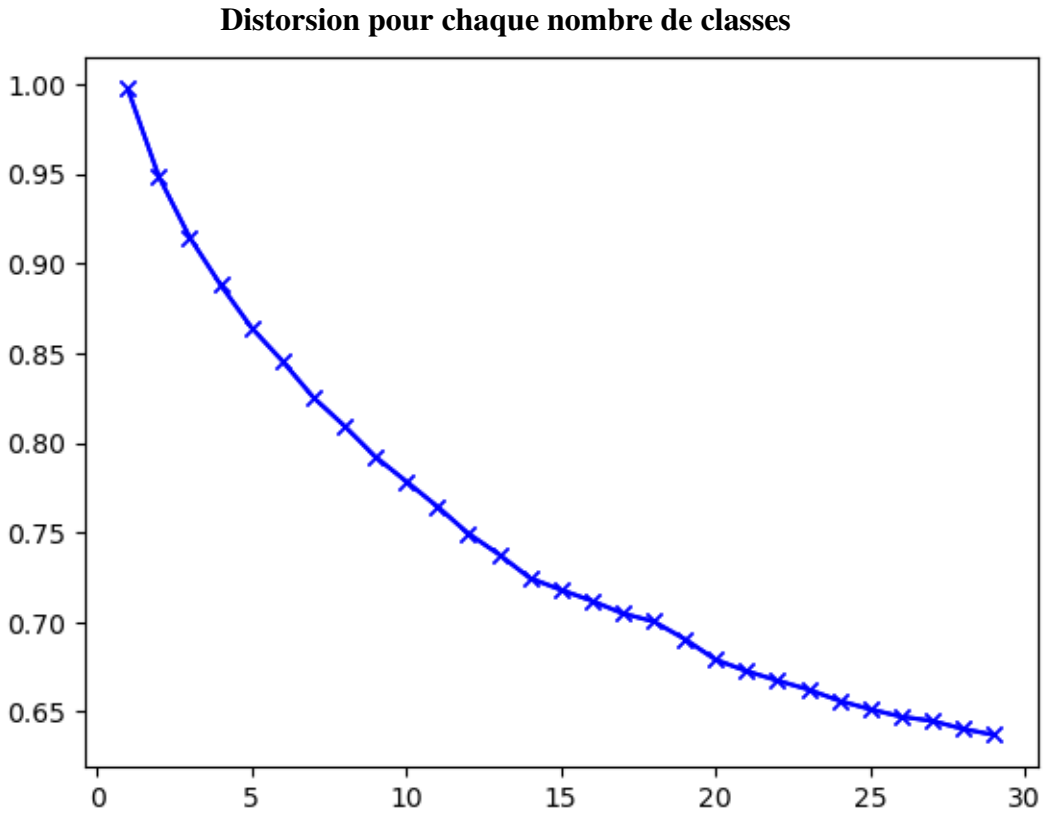


Figure 5: Résultats de la méthode du coude pour Houston2018.

Les résultats de l'expérience appliquée aux données de l'ENSSAT sont eux plus compliqués à interpréter étant donné le manque de points reconnaissables. Cependant, on peut observer des correspondances [fig 6, 7].

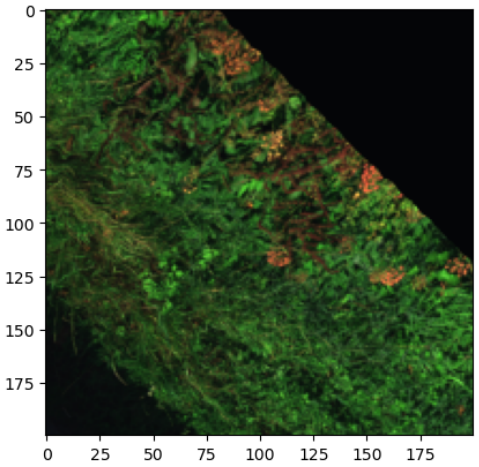


Figure 6: Image extraite des données de l'ENSSAT en couleurs réelles.

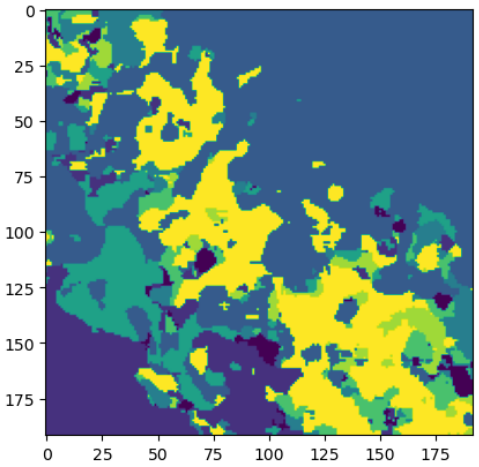


Figure 7: Production du modèle à partir d'une image extraite des données de l'ENSSAT.

L'entièreté de l'expérience sur les données de l'ENSSAT peut être trouvée en annexe A

5 Discussion

5.1 Découvertes

Premièrement, nous avons découvert qu'il était possible de classifier de manière consistante des images hyperspectrales sans s'entraîner sur un jeu de données labellisées. Cela s'observe surtout sur les données de Houston2018, de manière assez précise lorsque l'on regarde les stades à droite et à gauche de l'image. En effet, la fenêtre glissante de 8 par 8 pixels est trop petite pour les englober en entier, or on aperçoit que la classification est constante au niveau de l'herbe, de la terre battue autour, des sièges, *etc*. Si notre modèle n'avait pas réussi à apprendre correctement à reconnaître ces classes, alors on aurait observé plusieurs classes différentes être attribuées dans ces régions. Pour ce qui est des données de

l'ENSSAT en revanche, il est plus difficile de repérer ce genre de schémas, et il faudrait faire appel à un expert pour valider les résultats. Cependant, l'angle du coude permet d'être confiant sur le fait que le modèle ait réussi à trouver des classes assez distinctes car un coude plus franc indique une démarcation plus nette entre les groupes, donc moins "d'hésitation" de la part du modèle. Cela peut se voir lorsque l'on les nuages de points obtenus pour Houston2018 et pour l'ENSSAT [fig 8, 9] on remarque que les classes de l'ENSSAT sont plus éloignées les unes des autres tandis que les classes de Houston2018 sont plus proches entre elles.

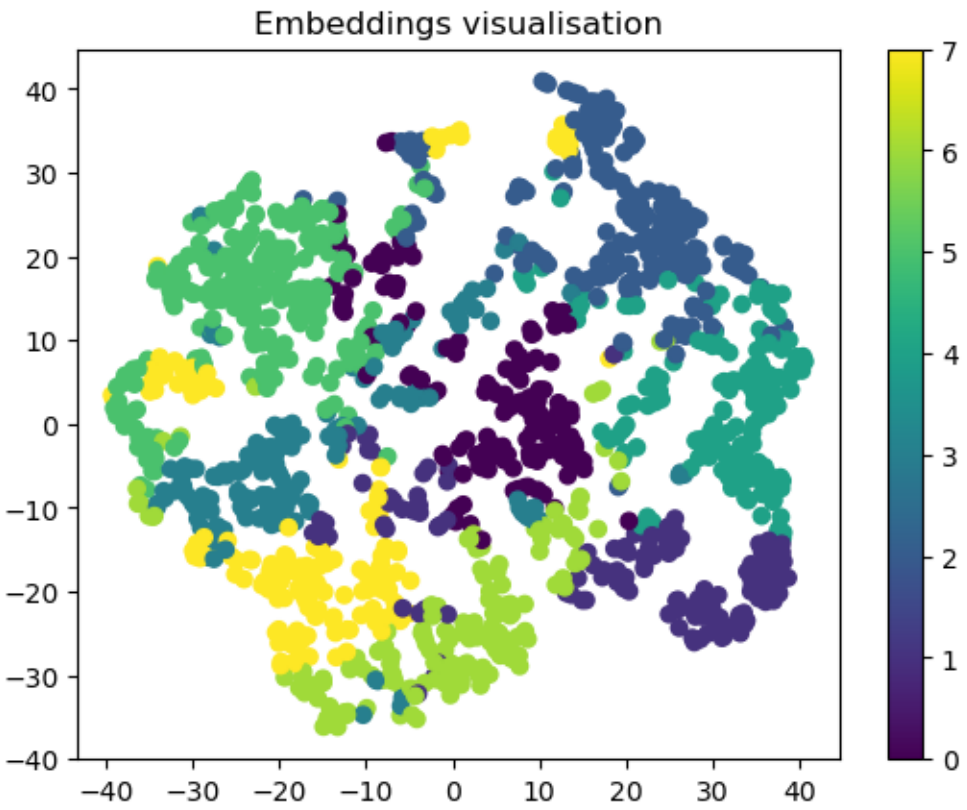


Figure 8: Projection du nuage de points pour les représentations des données de l'ENSSAT

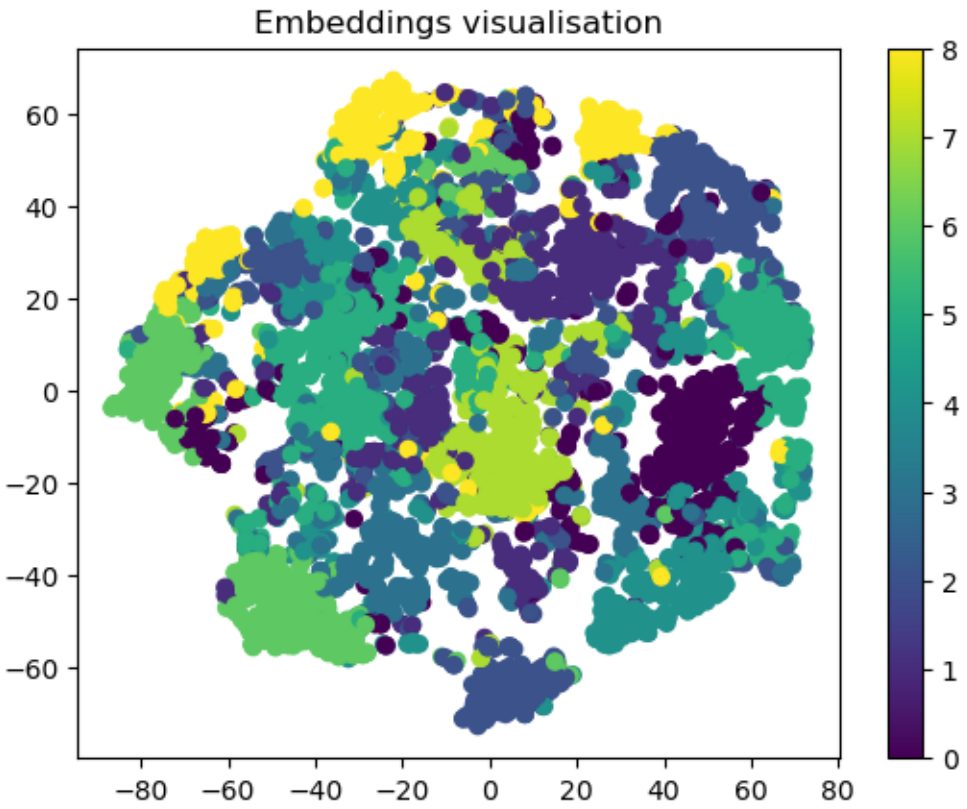


Figure 9: Projection du nuage de points pour les représentations des données de l'Houston2018

5.2 Limitations et Implications

Bien que nous ayons montré que notre modèle pouvait servir à classifier des images hyperspectrales de manières consistantes, la classification obtenue n'est pas facilement utilisable tel quel. En effet, on obtient simplement des groupes, mais on ne donne pas de nom à ces groupes. Cependant, un tel modèle pourrait être utile à un expert qui pourrait ensuite attribuer à chaque groupe un nom utile (exemple : classe 1 = route, classe 2 = lac ...), et trouver rapidement des groupes qu'il n'aurait pas vus seul.

5.3 Prochaines étapes

Pour de prochains travaux, nous visons à améliorer les paramètres du modèle pour obtenir des contrastes plus forts entre les classes. En effet, on peut observer sur les résultats de Houston2018 que les routes ne sont pas nettes au pixel près et qu'elles changent parfois de classe. Pour cela, nous allons appliquer des techniques de réglage fin des hyperparamètres du modèle. Le but étant d'approcher la précision des labellisations utilisées dans le domaine. [fig 10, 11]

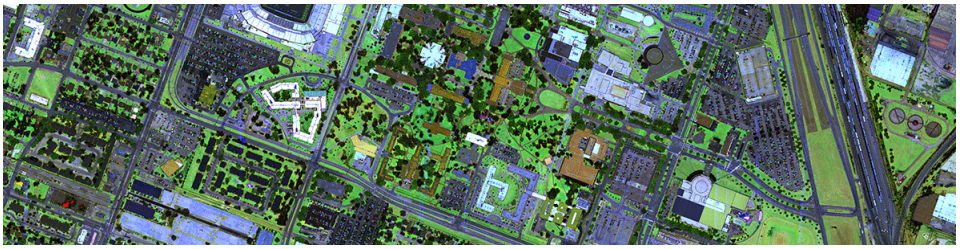


Figure 10: Projection en fausses couleurs des d'un partie des données de Houston2018.

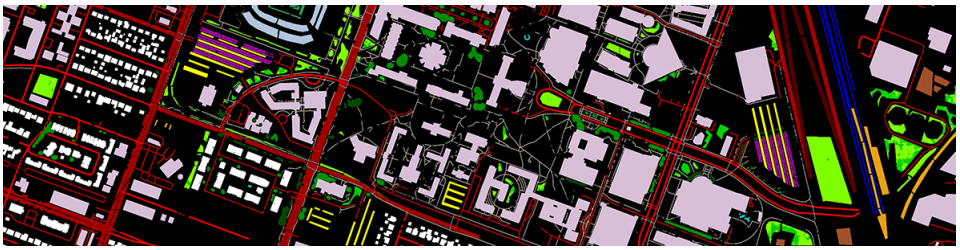


Figure 11: Labels d'une partie de Houston2018 définis par des experts.

6 Conclusion

En conclusion, nous avons observé un manque de connaissances dans le domaine de la classification d'images hyperspectrales à partir de données totalement non labellisées. Or ces connaissances seraient utiles pour les laboratoires et entreprises qui souhaitent tirer une représentation concise d'images hyperspectrales. Nous avons donc mis au point une expérience permettant de démontrer qu'il était possible de classifier des images hyperspectrales sans entraînement sur des données labellisées et d'obtenir une classification prometteuse. Cependant, il s'agit encore d'une preuve de concept et non d'un produit fini. Pour cela, il faut encore améliorer le modèle pour le rendre plus précis.

Bibliographie

[1] Yuanhao Cai, Jing Lin, Xiaowan Hu, Haoqian Wang, Xin Yuan, Yulun Zhang, Radu Timofte, and Luc Van Gool. Mask-guided spectral-wise transformer for efficient hyperspectral image reconstruction, 2022.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[3] Nikunj Saunshi, Jordan Ash, Surbhi Goel, Dipendra Misra, Cyril Zhang, Sanjeev Arora, Sham Kakade, and Akshay Krishnamurthy. Understanding contrastive learning requires incorporating inductive biases, 2022.

[4] Linus Scheibenreif, Michael Mommert, and Damian Borth. Masked vision transformers for hyperspectral image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2166–2176, June 2023.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

Glossaire

classe Groupe d'objets similaires. 7

hyperparamètres Paramètres non apprenables qui définissent un modèle : taille, vitesse d'apprentissage, *etc*. 13

objet Correspond à tout ce qui peut être sur une image : Végétation, eau, routes, *etc*. 6

pixel Enregistrement de l'intensité de l'objet pour chaque fréquence. 7

A Jupyter Notebook

[1]:
```
#!pip install torch torchvision transformers scikit-learn matplotlib einops␣
↪spectral wandb
```

[29]:
```
import os

import numpy as np

import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE

from scipy.spatial.distance import cdist

from tqdm import tqdm

import matplotlib.pyplot as plt

from einops import rearrange
from einops.layers.torch import Rearrange

import spectral.io.envi as envi

import wandb
```

[3]:
```
header_file = "ENSSAT_Dataset/header.hdr"
spectral_file = "ENSSAT_Dataset/data.pix"

data = envi.open(header_file, spectral_file)
```

[4]:
```
default_red, default_green, default_blue = data.metadata['default bands']
default_red = int(default_red)
default_green = int(default_green)
default_blue = int(default_blue)
default_red, default_green, default_blue
```

[4]:
```
(95, 58, 20)
```

[1]:
```
d = data[:, :, :]
mean_per_layer = np.mean(d, axis=(0, 1))
std_per_layer = np.std(d, axis=(0, 1))
```

```
d.shape
```

[1]: (100, 100, 224)

[6]:
```
def normalize_data(x):
    return (x - mean_per_layer[np.newaxis, np.newaxis, :]) / std_per_layer[np.
 ↪newaxis, np.newaxis, :]
```

[7]:
```
def update_type_float32(x):
    return x.type(torch.float32)
```

[8]:
```
transform = transforms.Compose([
    normalize_data,
    transforms.ToTensor(),
    update_type_float32,
])
```

[9]:
```
class SpectralDataset(Dataset):
    def __init__(self, data, pad_x, pad_y, transform):
        self.data = data
        self.pad_x = pad_x
        self.pad_y = pad_y
        self.transform = transform
        if data.shape[0] < pad_x or data.shape[1] < pad_y:
            raise ValueError(f"Padding should be lower than data shape {data.
 ↪shape=}")

    def __len__(self):
        return (self.data.shape[0] - self.pad_x) * (self.data.shape[1] - self.
 ↪pad_y)

    def __getitem__(self, index):
        x = index % (self.data.shape[0] - self.pad_x)
        y = index // (self.data.shape[0] - self.pad_x)
        img = self.data[x : x + self.pad_x, y : y + self.pad_y, :]
        transformed_img = self.transform(img).transpose(0, 2)
        return transformed_img
```

[10]:
```
def custom_collate(batch):
    # Filter out mostly black images based on a threshold
    threshold = 0.1
    filtered_batch = [sample for sample in batch if sample.mean() > threshold]

    samples = [sample for sample in filtered_batch]

    if not samples:
        return None
```

```
    stacked_samples = torch.stack(samples, dim=0)

    return stacked_samples
```

[11]:
```
pad_x, pad_y = 8, 8
dataset = SpectralDataset(data, 8, 8, transform=transform)
dataloader = DataLoader(dataset, batch_size=64, collate_fn=custom_collate,␣
↪shuffle=True)
```

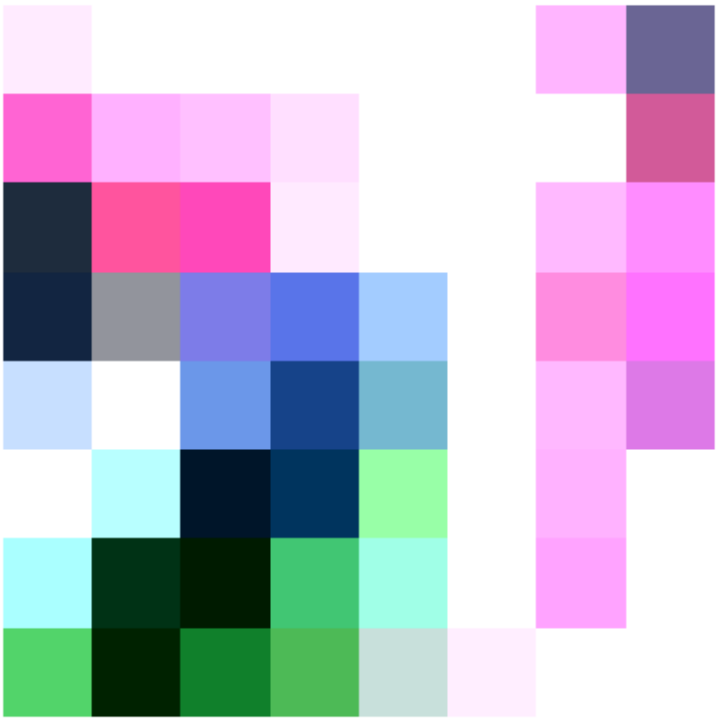
[12]:
```
for batch in dataloader:
    break
```

[13]:
```
def show_images(images, R=default_red, G=default_green, B=default_blue):
    if type(images) != list:
        images = [images]
    if len(list(images[0].shape)) == 4:
        images = [img for img in images[0]]
    fig, axes = plt.subplots(1, len(images))
    if len(images) == 1:
        axes = [axes]
    for i, img in enumerate(images):
        axes[i].imshow(img[:, :, [R, G, B]])
        axes[i].axis('off')
    plt.show()
```

[14]:
```
for idx, img in enumerate(batch):
    show_images(img)
    if idx > 3:
        break
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

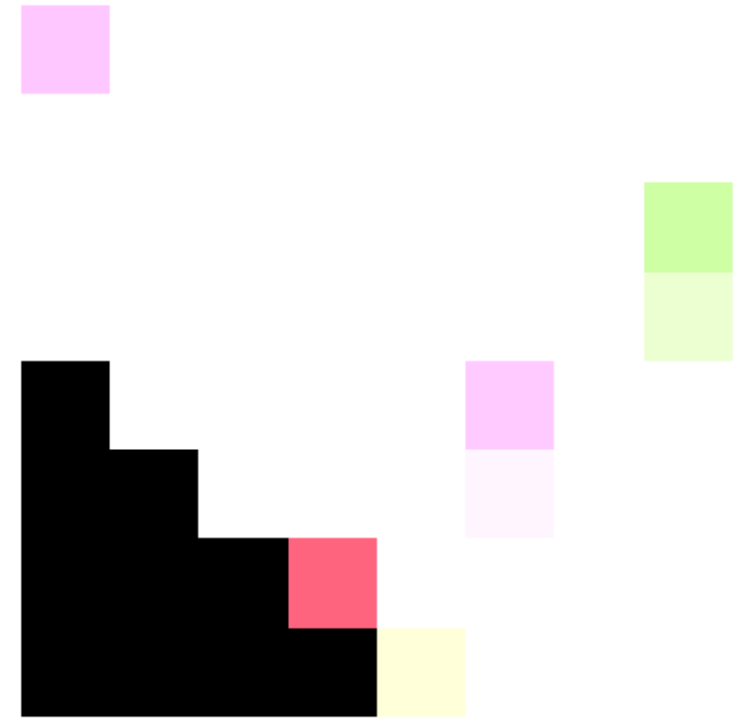
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

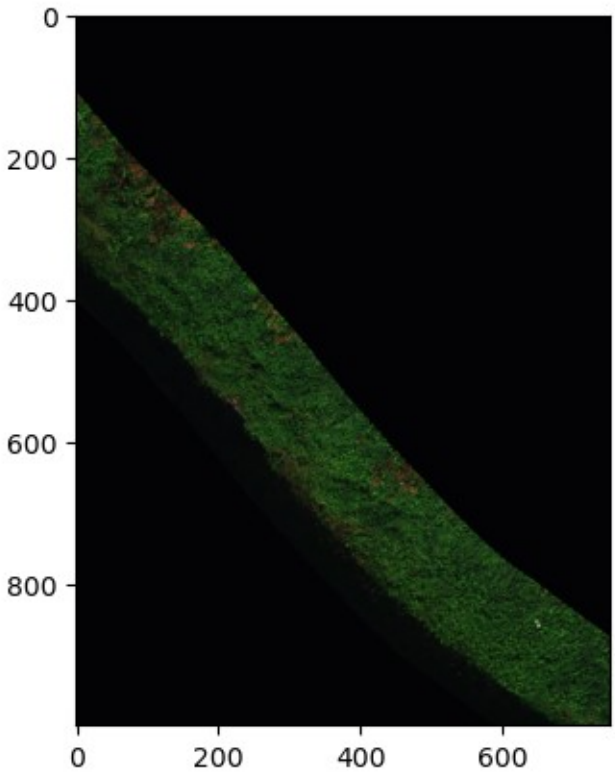
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



[15]:
```
def show_full_image(data):
    img = data[:, :, [default_red, default_green, default_blue]]
    img = img / img.max()
    fig, ax = plt.subplots(1, 1)
    ax.imshow(img)
    plt.show()
```

[16]:
```
show_full_image(data)
```



[17]:
```
def pair(t):
    return t if isinstance(t, tuple) else (t, t)

def posemb_sincos_2d(h, w, dim, temperature: int = 10000, dtype = torch.float32):
    y, x = torch.meshgrid(torch.arange(h), torch.arange(w), indexing="ij")
    assert (dim % 4) == 0, "feature dimension must be multiple of 4 for sincos␣
↪emb"
    omega = torch.arange(dim // 4) / (dim // 4 - 1)
    omega = 1.0 / (temperature ** omega)
```

```
    y = y.flatten()[:, None] * omega[None, :]
    x = x.flatten()[:, None] * omega[None, :]
    pe = torch.cat((x.sin(), x.cos(), y.sin(), y.cos()), dim=1)
    return pe.type(dtype)

# classes

class FeedForward(nn.Module):
    def __init__(self, dim, hidden_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.LayerNorm(dim),
            nn.Linear(dim, hidden_dim),
            nn.GELU(),
            nn.Linear(hidden_dim, dim),
        )
    def forward(self, x):
        return self.net(x)

class Attention(nn.Module):
    def __init__(self, dim, heads = 8, dim_head = 64):
        super().__init__()
        inner_dim = dim_head * heads
        self.heads = heads
        self.scale = dim_head ** -0.5
        self.norm = nn.LayerNorm(dim)

        self.attend = nn.Softmax(dim = -1)

        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias = False)
        self.to_out = nn.Linear(inner_dim, dim, bias = False)

    def forward(self, x):
        x = self.norm(x)

        qkv = self.to_qkv(x).chunk(3, dim = -1)
        q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', h = self.
 ↪heads), qkv)

        dots = torch.matmul(q, k.transpose(-1, -2)) * self.scale

        attn = self.attend(dots)

        out = torch.matmul(attn, v)
        out = rearrange(out, 'b h n d -> b n (h d)')
        return self.to_out(out)
```

```
class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, dim_head, mlp_dim):
        super().__init__()
        self.norm = nn.LayerNorm(dim)
        self.layers = nn.ModuleList([])
        for _ in range(depth):
            self.layers.append(nn.ModuleList([
                Attention(dim, heads = heads, dim_head = dim_head),
                FeedForward(dim, mlp_dim)
            ]))
    def forward(self, x):
        for attn, ff in self.layers:
            x = attn(x) + x
            x = ff(x) + x
        return self.norm(x)

class SimpleViT(nn.Module):
    def __init__(self, *, image_size, patch_size, num_classes, dim, depth,␣
 ↪heads, mlp_dim, channels = 3, dim_head = 64):
        super().__init__()
        image_height, image_width = pair(image_size)
        patch_height, patch_width = pair(patch_size)

        assert image_height % patch_height == 0 and image_width % patch_width ==␣
 ↪0, 'Image dimensions must be divisible by the patch size.'

        patch_dim = channels * patch_height * patch_width

        self.to_patch_embedding = nn.Sequential(
            Rearrange("b (h p1) (w p2) c -> b (h w) (p1 p2 c)", p1 =␣
 ↪patch_height, p2 = patch_width),
            nn.LayerNorm(patch_dim),
            nn.Linear(patch_dim, dim),
            nn.LayerNorm(dim),
        )

        self.pos_embedding = posemb_sincos_2d(
            h = image_height // patch_height,
            w = image_width // patch_width,
            dim = dim,
        )

        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim)

        self.pool = "mean"
        self.to_latent = nn.Identity()
```

```
            self.linear_head = nn.Linear(dim, num_classes)

        def forward(self, img):

            device = img.device

            x = self.to_patch_embedding(img)
            x += self.pos_embedding.to(device, dtype=x.dtype)

            x = self.transformer(x)
            x = x.mean(dim = 1)

            x = self.to_latent(x)
            return self.linear_head(x)
```

[18]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

[19]:
```
model = SimpleViT(
    image_size=pad_x,
    patch_size=1,
    num_classes=20,
    dim=20,
    depth=5,
    heads=3,
    mlp_dim=5,
    channels=int(data.metadata["bands"])
).to(device)
```

[20]:
```
model(batch).shape
```

[20]:
```
torch.Size([6, 20])
```

[21]:
```
# Contrastive loss
triplet_loss = nn.TripletMarginLoss(margin=1.0, p=2, eps=1e-7)

# Optimizer
optimizer = optim.Adam(model.parameters(), lr=3e-4)

# Training loop
num_epochs = 10
early_stopping_at = 0.003
```

[22]:
```
def apply_gaussian_noise(batch, mean=0, std=1):
    noise = torch.randn_like(batch) * std + mean
    noisy_batch = batch + noise
    return noisy_batch
```

[23]:
```
os.environ["WANDB_NOTEBOOK_NAME"] = "ENSSAT_DATA.ipynb"

wandb.init(
    # set the wandb project where this run will be logged
    project="ENSSAT_POC",

    # track hyperparameters and run metadata
    config={
    "learning_rate": 3e-4,
    "architecture": "ViT",
    "dataset": "ENSSAT",
    "epochs": 10,
    "early_stopping_at": early_stopping_at
    }
)
```

```
wandb: Currently logged in as: mattias-kockum
(enssat-projet-technologique). Use `wandb login --relogin` to
force relogin
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[23]:
```
<wandb.sdk.wandb_run.Run at 0x7f5f552f5900>
```

[24]:
```
for epoch in range(num_epochs):
    for batch_idx, batch in enumerate(dataloader):
        batch = batch.to(device)

        # Contrastive learning
        negative_pairs = batch.roll(shifts=-1, dims=0).clone()
        negative_pairs[-1] = batch[0].clone()
        positive_pairs = apply_gaussian_noise(batch, std=0.1)

        # Forward pass
        embeddings_anchor = F.normalize(model(batch), dim=-1)
        embeddings_positive = F.normalize(model(positive_pairs), dim=-1)
        embeddings_negative = F.normalize(model(negative_pairs), dim=-1)

        # Contrastive loss
        loss = triplet_loss(embeddings_anchor, embeddings_positive,␣
↪embeddings_negative)
```

```
        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch_idx % 100 == 0:
            wandb.log({"loss": loss.item(), "epoch": epoch})
            print(f"Epoch [{epoch + 1}/{num_epochs}], Batch [{batch_idx}/
↪{len(dataloader)}], Loss: {loss.item():.4f}")
            if loss.item() < early_stopping_at:
                break
    if loss.item() < early_stopping_at:
                break

wandb.finish()
```

```
Epoch [1/10], Batch [0/11501], Loss: 0.3814
Epoch [1/10], Batch [100/11501], Loss: 0.0419
Epoch [1/10], Batch [200/11501], Loss: 0.0128
Epoch [1/10], Batch [300/11501], Loss: 0.0376
Epoch [1/10], Batch [400/11501], Loss: 0.0486
Epoch [1/10], Batch [500/11501], Loss: 0.0627
Epoch [1/10], Batch [600/11501], Loss: 0.0243
Epoch [1/10], Batch [700/11501], Loss: 0.0168
Epoch [1/10], Batch [800/11501], Loss: 0.0000
```

```
VBox(children=(Label(value='0.001 MB of 0.017 MB uploaded\r'),␣
↪FloatProgress(value=0.05957139684654675, max=1....
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[25]:
```
torch.save(model.state_dict(), "ENSSAT_POC_Model")
```

[26]:
```
all_embeddings = []

model.eval()

with torch.no_grad():
    for batch_idx, batch in enumerate(dataloader):
        batch = batch.to(device)
        embedding = F.normalize(model(batch), dim=-1)
        all_embeddings.append(embedding)
        if batch_idx > 100:
            break
```

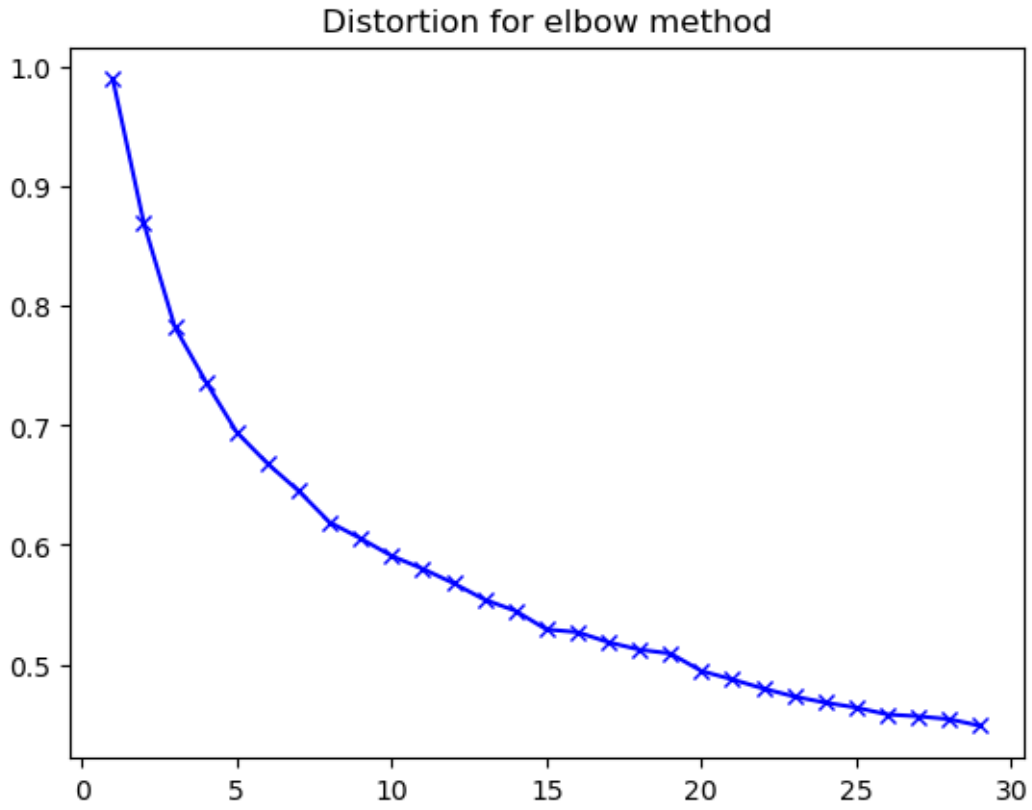
```
all_embeddings = torch.cat(all_embeddings, dim=0)
all_embeddings_np = all_embeddings.detach().numpy()
all_embeddings_flattened = all_embeddings_np.reshape(all_embeddings_np.shape[0],␣
↪-1)
```

[34]:
```
distortions = []

K = range(1, 30)

for k in K:
    kmeanModel = KMeans(n_clusters=k, n_init="auto", random_state=0)
    cluster_labels = kmeanModel.fit_predict(all_embeddings_flattened)
    distortions.append(
        sum(np.min(
            cdist(all_embeddings_flattened, kmeanModel.cluster_centers_,␣
↪'euclidean'),
            axis=1)
        ) / len(all_embeddings_flattened))
```

[57]:
```
plt.plot(K, distortions, "b-x")
plt.title('Distortion for elbow method')
plt.show()
```



[44]:
```
k = min([(i+1 , (i+1) ** 2 + (distortions[i] * len(distortions))**2) for i in␣
↪range(len(distortions))], key=lambda x: x[1])[0]
k
```

[44]: 8

[46]:
```
kmeanModel = KMeans(n_clusters=k, n_init="auto", random_state=0)
cluster_labels = kmeanModel.fit_predict(all_embeddings_flattened)
```

[47]:
```
X1 = 200
Y1 = 0
X2 = 400
Y2 = 200
dataset.data.shape
```

[47]: (1000, 750, 224)

[48]:
```
sub_dataset = SpectralDataset(data[X1:X2, Y1:Y2, :], pad_x, pad_y,␣
↪transform=transform)
```

[49]:
```
output_array = np.zeros(len(sub_dataset))
```

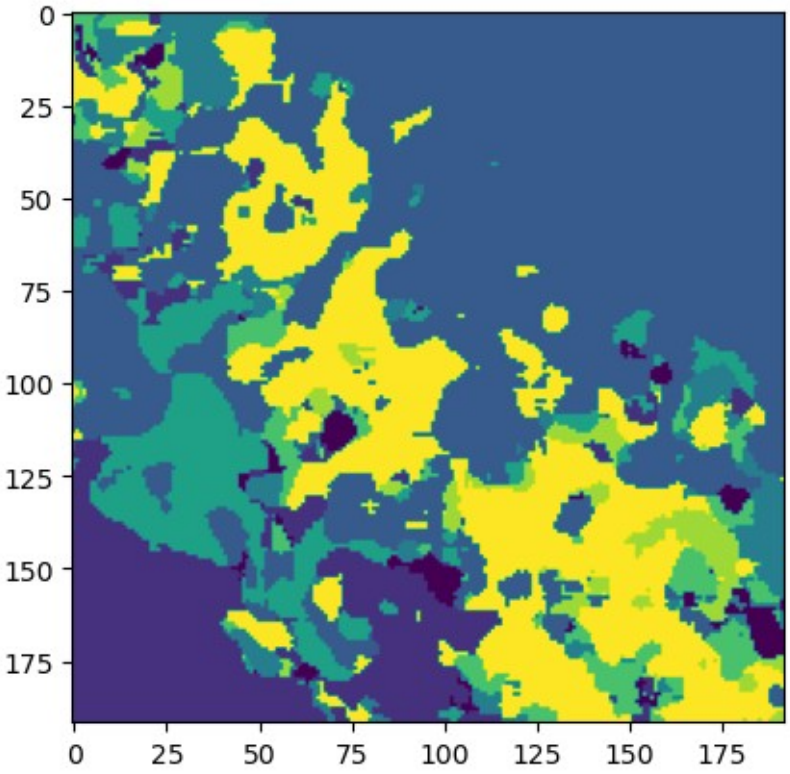
[51]:
```
with torch.no_grad():
    for i in tqdm(range(len(output_array))):
        embedding = model(sub_dataset[i].unsqueeze(0)).detach().numpy()
        pred = kmeanModel.predict(embedding)
        output_array[i] = pred[0]
```

100%|| 36864/36864 [03:47<00:00, 161.89it/s]

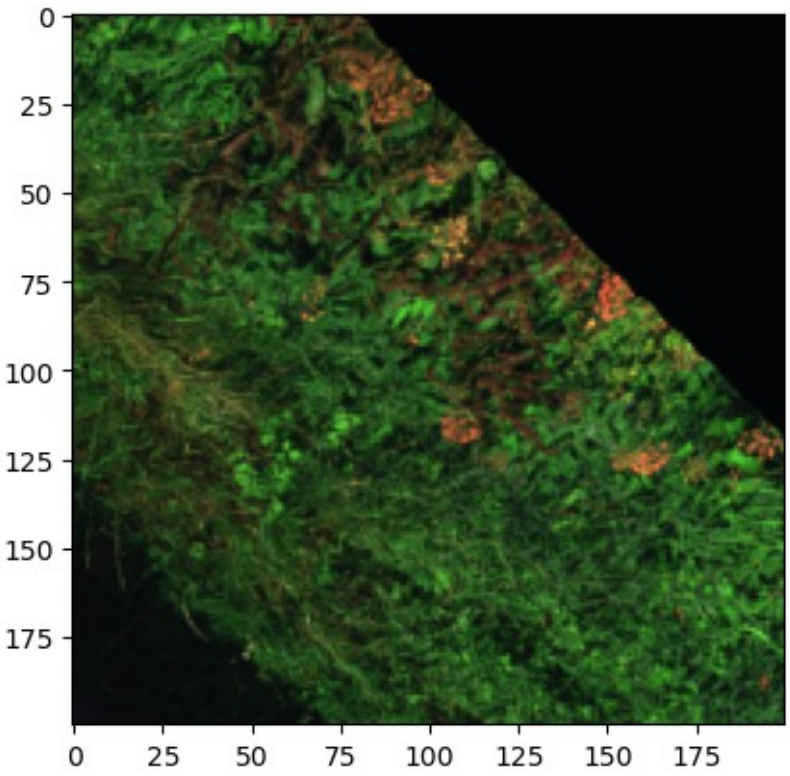
[52]:
```
out_img = output_array.reshape((Y2 - Y1 - pad_y, X2 - X1 - pad_x)).T
```

[53]:
```
plt.imshow(out_img)
```

[53]: <matplotlib.image.AxesImage at 0x7f5f3afe6680>

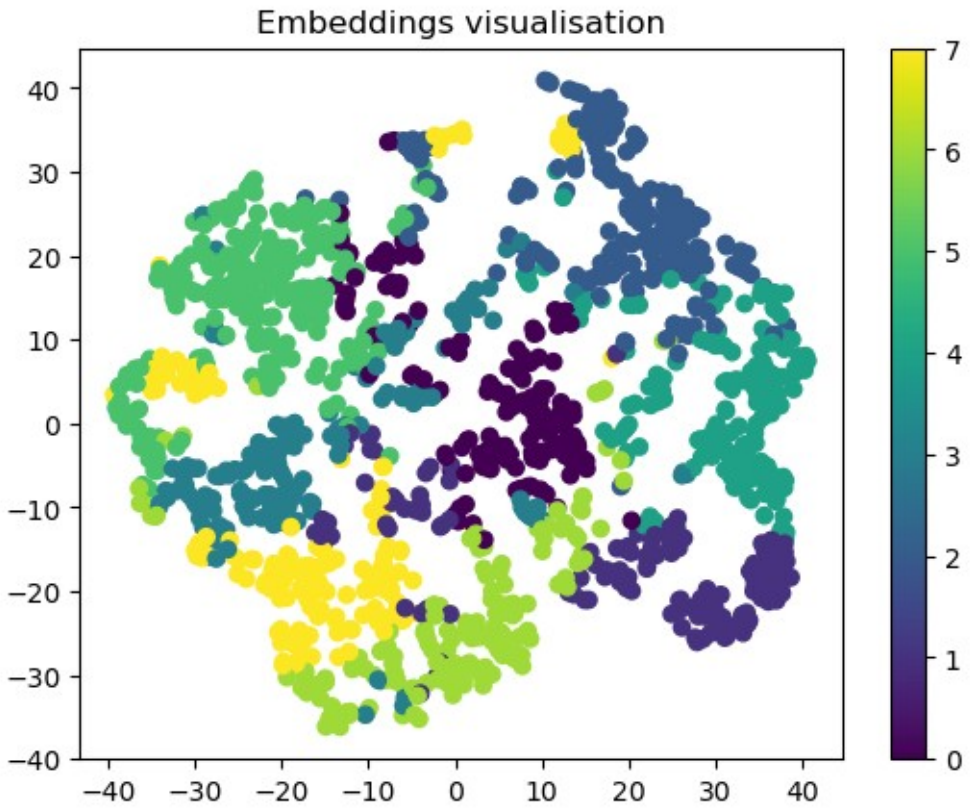


[54]:
```
show_full_image(data[X1:X2, Y1:Y2, :])
```



[55]:
```
tsne = TSNE(n_components=2)
embeddings_2d = tsne.fit_transform(all_embeddings_np)
```

[56]:
```
plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1], c=cluster_labels)
plt.title('Embeddings visualisation')
plt.colorbar()
plt.show()
```


Embeddings visualisation
40
30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40
7
6
5
4
3
2
1
0

[ ]: