RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

METEO FRANCE
À VOS CÔTÉS, DANS UN CLIMAT QUI CHANGE

ENM
École Nationale de la Météorologie

23/01/2026

Détection de sargasses par IA au large des Antilles

Détection automatisée des bancs de sargasses par intelligence artificielle pour la prévision du risque d'échouages en Antilles-Guyane.

Florent PUY & Joris SAINT-GENÈS, IENM3

Plan

1. Introduction
2. Méthodologie
    a. Création de la base de données
    b. Réseau de neurones
3. Résultats
4. Conclusion
5. Bibliographie

1. Introduction

Contexte :

Météo France doit anticiper les échouages de sargasses (détection → modélisation → bulletin).

Échouages ⇒ émissions de H2S ⇒ Risques
Écologique, économique, sanitaire

Prévision du danger essentielle, ainsi que couplage avec études sur la qualité de l'air

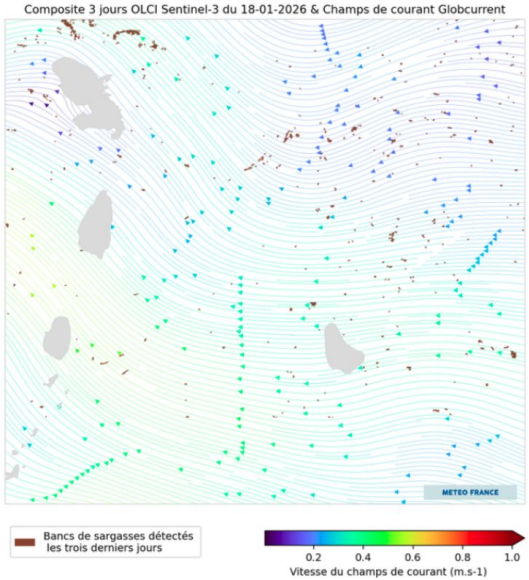
Septembre 2024 - Le François, Martinique

1. Introduction

Travaux opérationnels actuels :

- détection par indices spectraux (proche infrarouge) avec seuils empiriques et post-traitement manuel

- limites : long, pas optimisé, dépendance aux pré-traitements, confusion avec autres fréquente



Bulletin à partir d'une image composite tous les 3-4 jours
Source : meteofrance.mq

Plan

1. Introduction
2. Méthodologie
    a. Création de la base de données
    b. Réseau de neurones
3. Résultats
4. Conclusion
5. Bibliographie

2.a. Création de la base de données

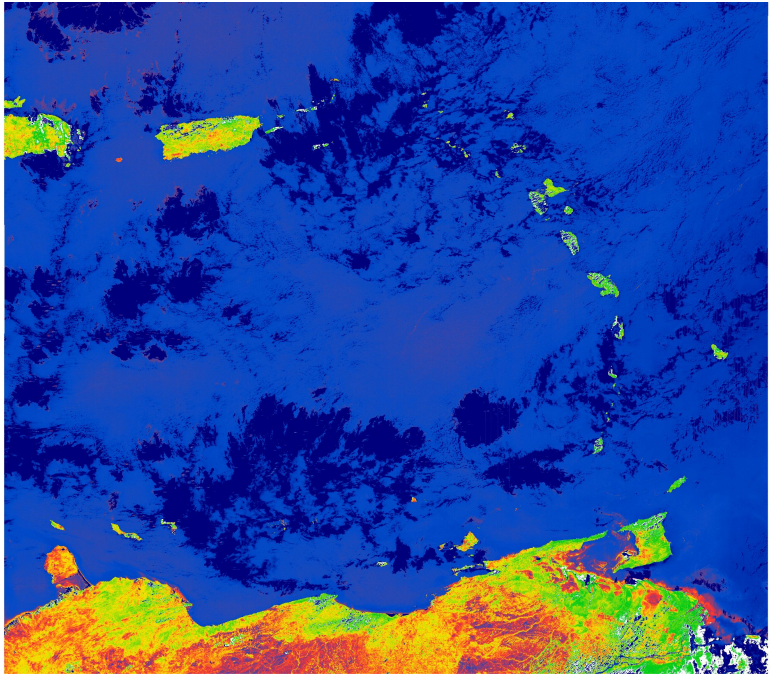
Données disponibles utilisables :

Images satellites des capteurs :

- **MSI (Sentinel-2)** : 10–20 m (côtier, suivi fin).
- **OLCI (Sentinel-3)** : ≈ 300 m (régional, couverture large).
- **VIIRS (NOAA-20 / Suomi-NPP)** : ≈ 750 m (large échelle, revisite fréquente).

Bandes utilisées :

- **RED** ≈ 665 nm
- **NIR** ≈ 859 nm
- **SWIR** ≈ 1240 nm



Images au format OTCI de la région Caraïbe
Source : Météo France

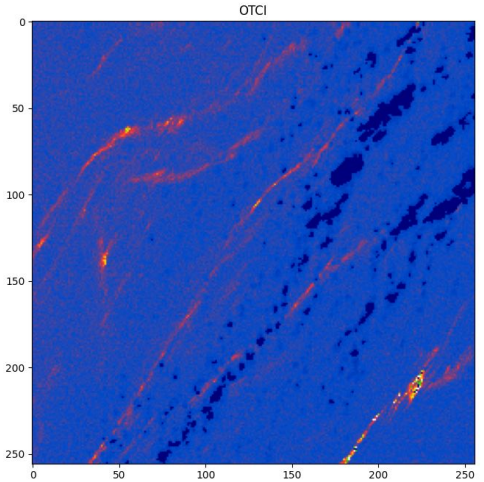
2.a. Création de la base de données

Création de la base de donnée :

50 crops aléatoires de 256*256 sur les 203 images OCTI ⇒ Total de 10150 éléments

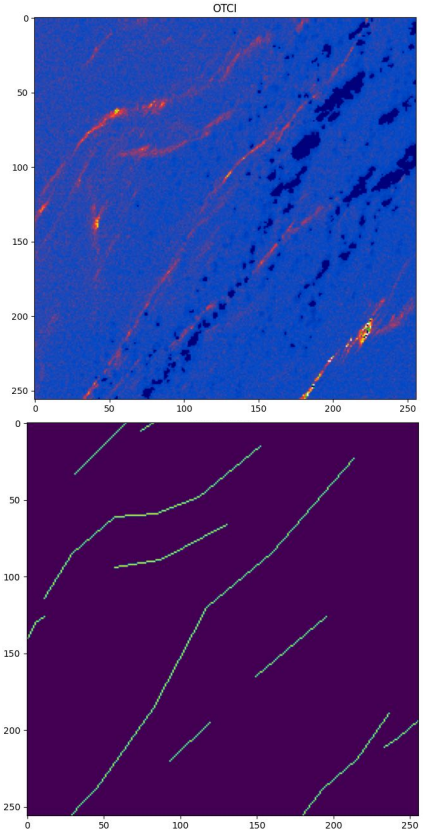
Filtrage par seuil de présence de sargasse sur chaque crop
(pour pas entraîner sur du vide)
~2500 images en entrée de modèle

Si crop vide (sans sargasse), crop éliminé



Exemple d'une image 256x256 cropée à partir d'une image OCTI

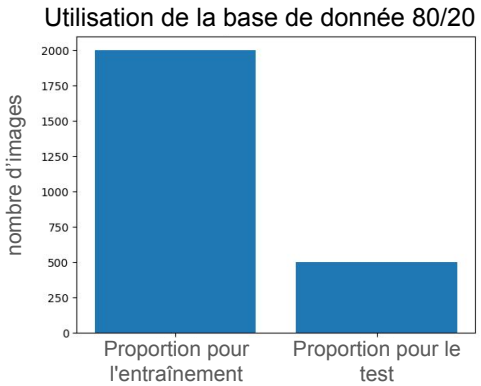
2.a. Création de la base de données



images satellites composites OCTI recadrées

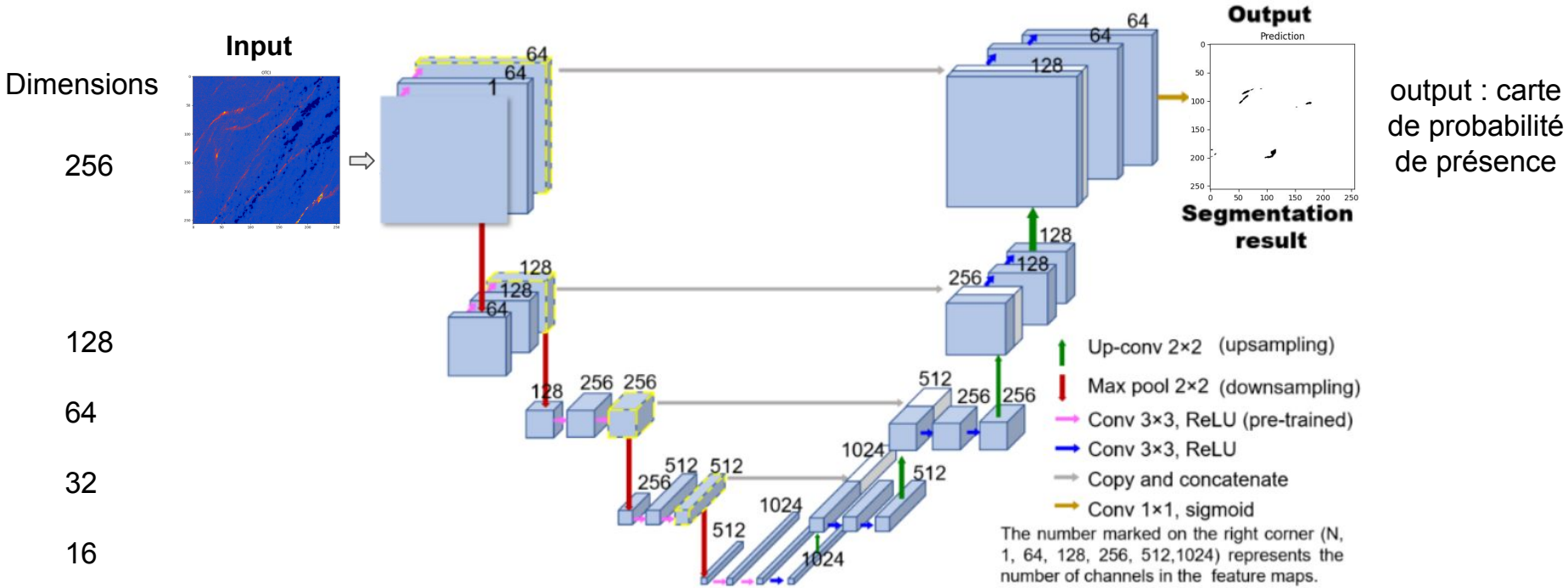
annotations correspondantes aux sargasses présentes
(recouvrement moyen de 0.27%)

= donnée d'entraînement



2.b. Réseau de neurones

Réseau VGGU-Net : réseaux de neurones convolutionnels en segmentation binaire



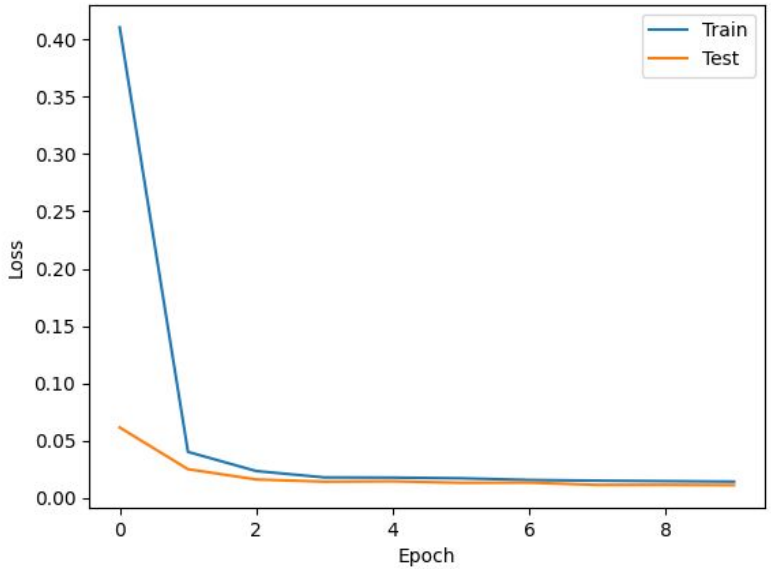
output : carte de probabilité de présence

Plan

1. Introduction
2. Méthodologie
    a. Création de la base de données
    b. Réseau de neurones
3. Résultats
4. Conclusion
5. Bibliographie

4. Premiers résultats

Fonctions de coût (BCELoss)
entrainement et test sur 9 époque



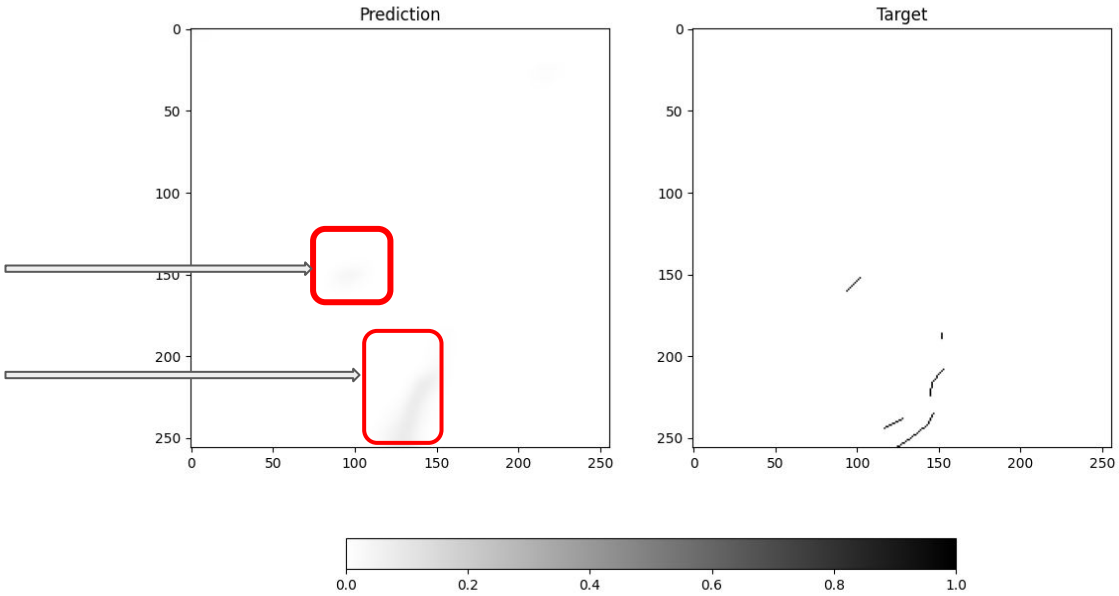
Hyper-paramètres :

Learning Rate = 0.0001
Batch sizes = 128
Max Epochs = 10
Fonction de loss = BCELoss
Optimiseur = AdamW

4. Premiers résultats

Résultat à la 9ème époque

Probabilité non nulle mais très faible



Le total loss est bonne mais c'est seulement dû à la bonne reconnaissance des vrais négatifs

5. Améliorations possibles

Problème majeur :
trop peu de pixels positifs
(0,27%)
⇒ modèle apprend bien à dire négatif mais pas positif

Solution 1 :
faire plus d'images plus petites (64*64)

Toute base d'entraînement à refaire

Solution 2 :
augmenter l'épaisseur des traits d'annotation des sargasses de 1 à 3

facile à implémenter, mais risque d'apprentissage sur faux positifs

Solution 3 :
augmenter le seuil minimum de pourcentage de sargasse

tri des images quasi vides, impact sur la taille du jeu de données

Solution 4 :
pondérer le learning-rate pour les vrais positifs par rapport aux vrais négatifs

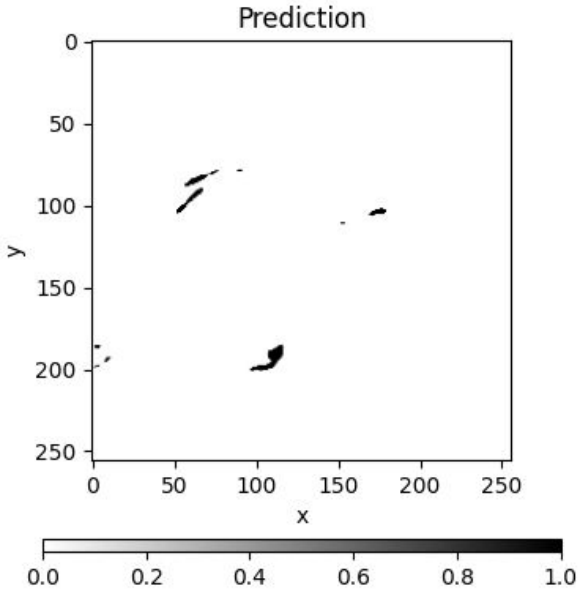
Difficile à implémenter

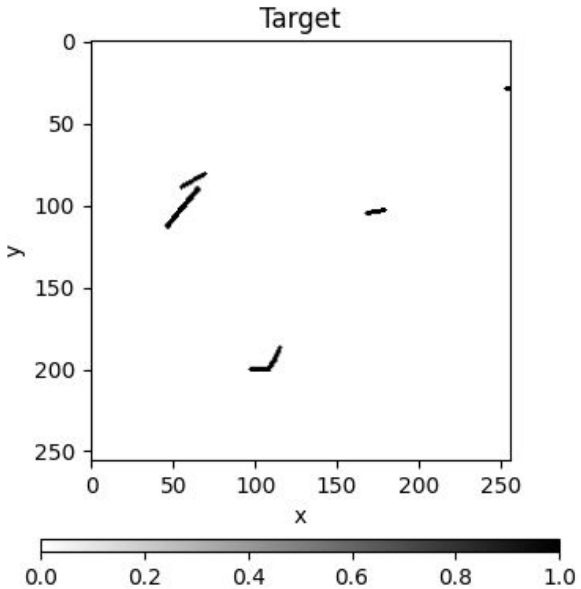
4. Seconds résultats

Améliorations implémentée :

Solution 2 :
augmenter l'épaisseur des traits d'annotation des sargasses de 1 à 3

⇒ +de recouvrement de sargasse par image, force le modèle à s'entraîner sur plus de pixels positifs





Plan

1. Introduction
2. Méthodologie
    a. Création de la base de données
    b. Réseau de neurones
3. Résultats
4. Conclusion
5. Bibliographie

4. Conclusion

Possibilité de calculer des métriques complémentaires :

F1-score ⇒ *Mesure l'équilibre entre éviter les faux positifs et ne pas rater les vrais positifs*

IoU ⇒ Mesure le recouvrement entre la prédiction et la vérité

Possibilité de calculer des métriques complémentaires :

F1-score ⇒ *Mesure l'équilibre entre éviter les faux positifs et ne pas rater les vrais positifs*

IoU ⇒ Mesure le recouvrement entre la prédiction et la vérité

23/01/2026

Merci pour votre attention

Florent PUY & Joris SAINT-GENÈS, IENM3

Plan

1. Introduction
2. Méthodologie
    a. Création de la base de données
    b. Réseau de neurones
3. Résultats
4. Conclusion
5. Bibliographie

Bibliographie :

- Wang, M., & Hu, C. (2021). Satellite remote sensing of pelagic Sargassum macroalgae: The power of high resolution and deep learning. Remote Sensing of Environment, 264, 112631.

- Arellano-Verdejo, J., Lazcano-Hernandez, H. E., & Cabanillas-Terán, N. (2019). ERISNet : deep neural network for Sargassum detection along the coastline of the Mexican Caribbean. PeerJ, 7, e6842.

Annexe

```python
def __init__(
    self,
    in_channels: int = 3,
    out_channels: int = 1,
    input_shape: Optional[tuple[int, int]] = None,):
    super().__init__()

    self.relu = nn.ReLU(inplace=True)
    self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

    # Encoder
    self.conv1 = nn.Conv2d(in_channels, 64, kernel_size=3, padding=1)
    self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)

    self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
    self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=1)

    self.conv5 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
    self.conv6 = nn.Conv2d(256, 256, kernel_size=3, padding=1)

    self.conv7 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
    self.conv8 = nn.Conv2d(512, 512, kernel_size=3, padding=1)

    self.conv9 = nn.Conv2d(512, 1024, kernel_size=3, padding=1)
    self.conv10 = nn.Conv2d(1024, 1024, kernel_size=3, padding=1)
    self.conv11 = nn.Conv2d(1024, 1024, kernel_size=3, padding=1)

    # Decoder (upsampling)
    self.upconv1 = nn.ConvTranspose2d(1024, 1024, kernel_size=2, stride=2)
    self.conv12 = nn.Conv2d(1024, 512, kernel_size=3, padding=1)
    self.conv13 = nn.Conv2d(512, 512, kernel_size=3, padding=1)

    self.upconv2 = nn.ConvTranspose2d(512, 512, kernel_size=2, stride=2)
    self.conv14 = nn.Conv2d(512, 256, kernel_size=3, padding=1)
    self.conv15 = nn.Conv2d(256, 256, kernel_size=3, padding=1)

    self.upconv3 = nn.ConvTranspose2d(256, 256, kernel_size=2, stride=2)
    self.conv16 = nn.Conv2d(256, 128, kernel_size=3, padding=1)
    self.conv17 = nn.Conv2d(128, 128, kernel_size=3, padding=1)

    self.upconv4 = nn.ConvTranspose2d(128, 128, kernel_size=2, stride=2)
    self.conv18 = nn.Conv2d(128, 64, kernel_size=3, padding=1)
    self.conv19 = nn.Conv2d(64, 64, kernel_size=3, padding=1)

    # Final 1x1 conv to produce logits
    self.final_conv = nn.Conv2d(64, out_channels, kernel_size=1)
    self.sigmoid = nn.Sigmoid()
```

```python
def forward(self, x: Tensor) -> Tensor:
    # Encoder
    x = self.relu(self.conv1(x))
    x = self.relu(self.conv2(x))
    x = self.maxpool(x)

    x = self.relu(self.conv3(x))
    x = self.relu(self.conv4(x))
    x = self.maxpool(x)

    x = self.relu(self.conv5(x))
    x = self.relu(self.conv6(x))
    x = self.maxpool(x)

    x = self.relu(self.conv7(x))
    x = self.relu(self.conv8(x))
    x = self.maxpool(x)

    x = self.relu(self.conv9(x))
    x = self.relu(self.conv10(x))
    x = self.relu(self.conv11(x))

    # Decoder
    x = self.upconv1(x)
    x = self.relu(self.conv12(x))
    x = self.relu(self.conv13(x))

    x = self.upconv2(x)
    x = self.relu(self.conv14(x))
    x = self.relu(self.conv15(x))

    x = self.upconv3(x)
    x = self.relu(self.conv16(x))
    x = self.relu(self.conv17(x))

    x = self.upconv4(x)
    x = self.relu(self.conv18(x))
    x = self.relu(self.conv19(x))

    x = self.final_conv(x)
    x = self.sigmoid(x)

    return x
```