Université Paris Cité

Adaptation des modèles de fondation

M1 Info TER 2025

ALLAIN Yanis, AISSAOUI Amine

Lien du projet : TER-DOFA-Adaptation-de-modele-de-fondation

Encadrant: LOBRY Sylvain

Contents

1 Introduction

1.1 Contexte général

Les modèles de fondation ont transformé l'apprentissage profond en permettant d'extraire des représentations générales à partir de grands volumes de données. Reposant sur des architectures, comme les Transformers, ces modèles sont à l'origine de percées majeures dans des domaines comme le traitement du langage naturel (GPT, BERT) ou la génération d'images (DALL-E).

En télédétection, la transposition de ces approches reste un domaine émergent. L'imagerie satellite présente des défis spécifiques : diversité des capteurs (optique, radar, multispectral...), résolutions spatiales variées, complexité des scènes, ou encore rareté des annotations. Cette hétérogénéité rend difficile la conception de modèles véritablement universels, capables de s'adapter à de nouvelles modalités sans réentraînement complet.

Des travaux récents comme SatlasPretrain[1] ont tenté de répondre à ces enjeux en développant des modèles capables de traiter plusieurs types d'imagerie. Cependant, ces approches restent limitées dès qu'elles rencontrent des modalités non vues à l'entraînement. Pour surmonter cette barrière, Xiong et al. ont proposé le modèle DOFA (Neural Plasticity-Inspired Multimodal Foundation Model for Earth Observation)[2], conçu pour ajuster dynamiquement les poids d'un modèle de fondation. Cette approche ouvre la voie à une meilleure généralisation sans nécessiter un réentraînement exhaustif.

1.2 Objectifs du projet

Ce projet vise à évaluer le potentiel d'adaptation d'un modèle de fondation en télédétection à une nouvelle tâche spécifique : la détection d'objets. Pour cela, trois axes de travail sont proposés.

Premièrement, une phase de familiarisation avec l'architecture et le fonctionnement du modèle DOFA permettra de comprendre en quoi il propose une solution au problème de généralisation à des modalités non vues lors de l'entraînement. Cette phase implique une étude approfondie du code source, de la documentation technique ainsi que des jeux de données exploités, permettant de s'initier dans les fondements méthodologiques et théoriques des modèles de fondation et de l'apprentissage profond.

Deuxièmement, une phase de reproduction des résultats obtenus par les auteurs sera conduite sur un jeu de données afin de vérifier notre bonne compréhension du modèle et de mettre en œuvre de manière concrète les principes fondamentaux de l'apprentissage profond étudiés.

Enfin, nous essaierons d'adapter le modèle à une tâche pour laquelle il n'a pas été initialement conçu : la détection d'objets.

2 Etat de l'art

2.1 Modèles de fondation en général

Les modèles de fondation se caractérisent par leur capacité à apprendre des représentations génériques à partir de grandes quantités de données. L'objectif est d'obtenir un modèle pré-entraîné capable de s'adapter efficacement à diverses tâches spécifiques (downstream tasks). Ces modèles sont souvent basés sur l'architecture Transformer, introduite en 2017, connue pour son mécanisme d'attention[3].

Cette méthodologie a donné naissance à des architectures emblématiques comme GPT de OpenAI, capable de générer du texte , BERT de Google qui lui aussi est capable de générer du texte en langage naturel, ou encore DALL-E, qui transpose ces principes à la génération d'images à partir de descriptions textuelles. Tous ces modèles ont en commun un entraînement auto-supervisé, c'est-à-dire sans étiquettes explicites, en utilisant des tâches comme le masquage de mots ou la prédiction de tokens.

La phase de fine-tuning consiste ensuite à adapter ces modèles pré-entraînés à des tâches précises (ex. : classification, segmentation), avec un nombre de paramètres souvent réduit à entraîner.

2.2 Applications en télédétection

Spécificités de la télédétection

La télédétection se distingue des autres domaines par ses spécificités techniques : imagerie multispectrale et hyperspectrale, données radar, variation des résolutions spatiales et temporelles, etc. Ces données sont issues de capteurs hétérogènes, et souffrent de contraintes telles que la couverture nuageuse, les angles de prise de vue, ou encore les variations saisonnières. Par conséquent, les modèles de fondation classiques ne peuvent pas être appliqués directement sans modifications.

SatlasPretrain[1]

Les auteurs du projet SatlasPretrain[1] ont conçu un jeu de données à grande échelle, combinant des images aériennes à très haute résolution (NAIP) et des images multispectrales provenant du satellite Sentinel-2. Cela permet de couvrir une large gamme de scènes et de modalités. Le modèle SatlasNet, entraîné sur ce jeu de données, a été évalué sur plusieurs tâches de vision par ordinateur telles que la classification, la segmentation sémantique ou la détection de changements, et a montré des performances supérieures à celles des approches concurrentes.

	Types	Classes	Labels	Pixels	km^2
SatlasPretrain	7	137	302222K	17003B	21320K
UCM [57]	1	21	2K	1B	1K
BigEarthNet [51]	1	43	1750K	9B	850K
AID [56]	1	30	10K	4B	14K
Million-AID [39]	1	51	37K	4B	18K
RESISC45 [19]	1	45	32K	2B	10K
FMoW [22]	1	63	417K	437B	1748K
DOTA [55]	1	19	99K	9B	38K
iSAID [58]	1	15	355K	9B	38K

Figure 1: Comparaison de SATLASPRETRAIN[1] avec les jeux de données existants en télédétection (K = milliers, B = milliards). Types correspond au nombre de types de labels et km^2 à la superficie couverte.

SatlasPretrain[1] se distingue par son large nombre de 137 classes et sa couverture de 21 320 km^2, avec plus de 17 milliards de pixels et 302 milliards de labels, ce qui en fait un ensemble de données particulièrement vaste. En comparaison, des ensembles comme UCM et BigEarthNet[4] sont plus petits, avec moins de classes et une couverture géographique réduite. D'autres ensembles comme FMoW et DOTA présentent également un nombre important de classes, mais couvrent des superficies et des quantités de pixels plus spécifiques.

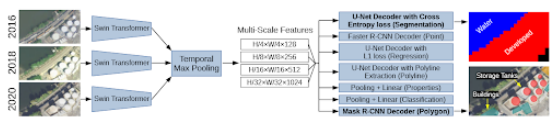


Figure 2: Architecture du modèle SATLASNET. Une tête distincte est utilisée pour prédire les sorties correspondant à chaque type de label. Avec illustration des exemples de sorties provenant de deux de ces têtes (segmentation et polygone).

Method	High-Resolution NAIP Images						Low-Resolution Sentinel-2 Images						
	Seg ↑	Reg ↓	Pt ↑	Pgon ↑	Pline ↑	Prop ↑	Seg ↑	Reg ↓	Pt ↑	Pgon ↑	Pline ↑	Prop ↑	Cls ↑
PSPNet (ResNext-101) [59]	77.8	15.0	-	-	53.2	-	62.1	16.2	-	-	30.7	-	-
LinkNet (ResNext-101) [15]	77.3	12.9	-	-	61.0	-	61.1	14.1	-	-	41.4	-	-
DeepLabv3 (ResNext-101) [16]	80.1	10.6	-	-	59.8	-	61.8	13.9	-	-	44.7	-	-
ResNet-50 [32]	-	-	-	-	-	87.6	-	-	-	-	-	70.3	97
ViT-Large [26]	-	-	-	-	-	78.1	-	-	-	-	-	66.9	99
Swin-Base [38]	-	-	-	-	-	87.1	-	-	-	-	-	69.4	99
Mask R-CNN (ResNet-50) [31]	-	-	27.6	30.4	-	-	-	-	22.0	12.3	-	-	-
Mask R-CNN (Swin-Base) [31]	-	-	30.4	31.5	-	-	-	-	25.6	15.2	-	-	-
ISTR [34]	-	-	2.0	4.9	-	-	-	-	1.2	1.4	-	-	-
SatlasNet (single-image, per-type)	79.4	8.3	28.0	30.4	61.5	86.6	64.8	9.3	25.7	14.8	42.5	67.5	99
SatlasNet (single-image, joint)	74.5	7.4	28.0	31.1	60.9	87.3	55.8	10.6	22.0	10.3	45.5	73.8	99
SatlasNet (single-image, fine-tuned)	79.8	**7.2**	32.3	33.0	**62.4**	**89.5**	65.3	9.0	27.4	16.3	45.9	**80.0**	99
SatlasNet (multi-image, per-type)	79.4	8.2	25.8	27.5	59.2	77.3	67.2	10.5	31.9	19.0	48.1	67.1	99
SatlasNet (multi-image, joint)	79.2	7.8	31.2	33.8	53.6	87.8	66.7	8.5	31.5	19.5	41.9	78.8	99
SatlasNet (multi-image, fine-tuned)	**81.0**	7.6	**33.2**	**34.1**	61.1	89.2	**69.7**	**7.8**	**32.0**	**20.2**	**50.4**	**80.0**	99

Figure 3: Résultats sur l'ensemble de test de SATLASPRETRAIN[1] pour les modes d'image haute et basse résolution. Les résultats sont détaillés par type de label : segmentation (Seg), régression (Reg), points (Pt), polygones (Pgon), polylignes (Pline), propriétés (Prop) et classification (Cls). L'erreur absolue est indiquée pour la régression (plus elle est faible, mieux c'est), tandis que la précision est affichée pour les autres (plus elle est élevée, mieux c'est).

Les modèles classiques de segmentation (PSPNet, LinkNet, DeepLabv3) affichent des scores solides en segmentation (Seg), avec DeepLabv3 atteignant 80.1% sur les images haute résolution. Cependant, leurs performances déclinent en basse résolution. En régression (Reg), SatlasNet surpasse les autres, avec une erreur absolue inférieure (7.2 contre 10.6 pour le meilleur modèle classique).

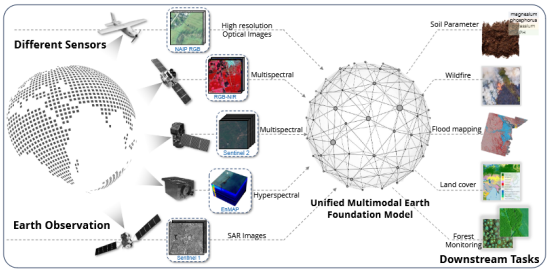
En détection de points (Pt), de polygones (Pgon) et de polylignes (Pline), les méthodes comme Mask R-CNN ou ISTR restent limitées, tandis que SatlasNet offre les meilleures performances. SatlasNet atteint par exemple 33.2% en détection de points sur NAIP, contre 30.4% pour Mask R-CNN (Swin-Base).

Enfin, pour la classification (Cls) et la prédiction de propriétés (Prop), les variantes de SatlasNet dominent également, atteignant jusqu'à 99% de précision en classification. Ces résultats soulignent la robustesse et la polyvalence de SatlasNet face à la diversité des tâches et des résolutions.

Cependant, SatlasPretrain[1] souffre d'une limitation importante : son manque de flexibilité face à des modalités non rencontrées lors de l'entraînement. En d'autres termes, le modèle ne généralise pas bien lorsqu'il est exposé à des types de capteurs différents ou à des résolutions inédites.

DOFA : un modèle inspiré de la plasticité neuronale[2]

Pour dépasser ces limites, Xiong et al. (2024) ont proposé DOFA[2] , un modèle de fondation explicitement conçu pour gérer la diversité des capteurs et des modalités en télédétection en adaptant les poids aux caractéristiques d'un capteur particulier.



(a)

Figure 4: DOFA[2]

Contrairement aux architectures statiques, DOFA[2] ajuste ses paramètres en fonction des modalités des données, ce qui renforce sa capacité à généraliser sur des images issues de capteurs hétérogènes tels que NAIP ou Sentinel-2. Son architecture repose sur la combinaison d'une génération dynamique des poids, d'une projection spécifique à chaque modalité, et d'un transformeur partagé, qui assure l'intégration des informations de différentes sources.

2.3 La détection d'objets dans l'imagerie satellite

La détection d'objets est une tâche fondamentale de la vision par ordinateur, qui consiste à localiser et classer les objets présents dans une image. Dans le domaine de la télédétection, elle est cruciale pour des applications comme la surveillance urbaine, le suivi des cultures ou la gestion des catastrophes naturelles.

Parmi les approches courantes de détection d'objet, des modèles comme Faster R-CNN privilégient une approche en deux étapes : une première phase identifie des régions candidates dans l'image, puis une seconde les classe et affine leur localisation. Ces modèles, bien que précis, peuvent s'avérer lourds pour des images satellitaires de grande taille. À l'inverse, des approches comme YOLO traitent l'image en une seule passe. Enfin, des architectures récentes exploitent l'attention globale pour relier les objets à leur contexte[3], évitant ainsi les ancrages manuels et améliorant la cohérence des prédictions. Ces méthodes nécessitent des ressources importantes et peinent parfois à généraliser face à la diversité des résolutions ou des perturbations propres aux données satellitaires.

Les particularités de l'imagerie satellite rendent difficile la détection précise d'éléments qui peuvent être de très petite taille, partiellement masqués etc. De plus, la diversité des types d'images disponibles (optiques, radar, infrarouges) nécessite généralement des adaptations spécifiques pour chaque source de données.

On veut donc explorer comment DOFA[2] peut être adapté efficacement à la tâche spécifique de détection d'objets en imagerie satellite, tout en conservant sa capacité à traiter différentes sources de données

3 Méthode développée

3.1 Familiarisation avec le modèle

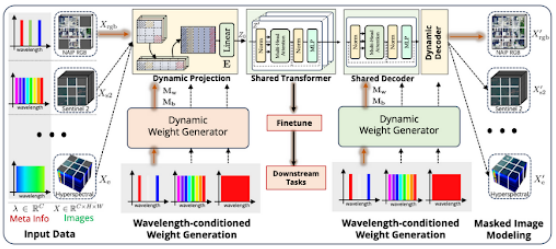


Figure 5: Architecture générale de DOFA[2]

DOFA[2] est conçue pour traiter des données issues de capteurs multiples. Son objectif est de produire des représentations communes à partir de différentes modalités, tout en tenant compte,par exemple, de leurs longueurs d'onde associées.

Encodage dynamique des patchs conditionné par la longueur d'onde

A L'entrée du modèle, chaque image est divisée en petites régions fixes (patch) qui sont ensuite projetées dans un espace de caractéristiques via un patch embedding. Contrairement aux approches standards, DOFA[2] adapte dynamiquement cette projection en fonction de la longueur d'onde de chaque canal spectral. Cette adaptation permet au modèle de capturer les spécificités de chaque modalité.

Génération dynamique des poids

DOFA[2] intègre un hyperréseau qui prend en entrée des métadonnées telles que la longueur d'onde et génère dynamiquement les poids utilisés dans le modèle principal. Ainsi il produit des poids optimisés en fonction de la modalité observée. Ce mécanisme permet au modèle de s'adapter automatiquement à chaque type de données sans devoir être ré-entraîné pour chaque nouvelle source.

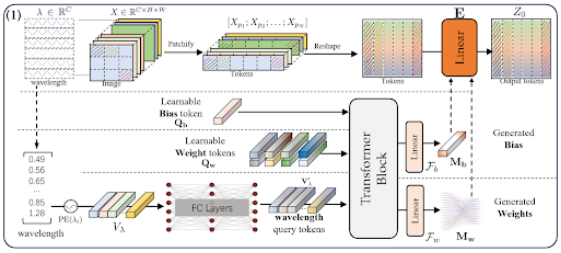


Figure 6: Générateur de poids dynamiques

Ce schéma illustre comment les poids dynamiques sont générés en fonction des longueurs d'onde des bandes spectrales :

- L'image multispectrale X est découpée en patchs et transformée en tokens.
- les longueurs d'onde λ sont utilisées pour guider la génération des poids.
- Des tokens de biais et de poids apprenables sont injectés dans un transformer.
- Un réseau de couches entièrement connectées (FC layers) transforme les longueurs d'onde en query tokens pour conditionner l'apprentissage.
- Finalement, le transformer produit des poids et biais dynamiques (Mw et Mb) utilisés pour adapter la projection dynamique.

Transformer partagé et représentation globale

Une fois les patchs encodés, ils sont traités par un Transformer partagé qui applique un mécanisme d'auto-attention pour faire interagir tous les patchs entre eux[3]. Cela permet à chaque patch de prendre en compte l'ensemble du contexte spatial de l'image.

Pré-entraînement multimodal et continu

DOFA[2] est pré-entraîné sur des données issues de Sentinel-1, Sentinel-2, NAIP, EnMAP, Gaofen, etc. Le modèle apprend à reconstruire les images via une tâche d'auto-supervision, ce qui favorise l'apprentissage de représentations robustes entre modalités. Ce pré-entraînement est progressif : les données sont introduites de manière croissante pour permettre au modèle de s'adapter continuellement sans oublier ce qu'il a déjà appris.

Distillation des connaissances

Pour accélérer l'entraînement et réduire la consommation de ressources, DOFA[2] utilise la distillation des connaissances. Il s'appuie sur un modèle enseignant pré-entraîné et aligne ses représentations avec celles du modèle étudiant. Ce transfert permet au modèle d'exploiter les informations déjà acquises sur des modalités plus simples, facilitant ainsi la convergence.

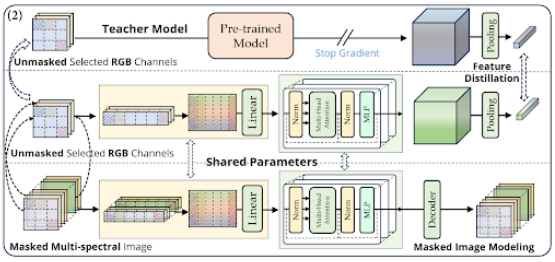


Figure 7: Distillation des connaissances et reconstruction des images masquées

Ce schéma illustre un processus d'entraînement progressif qui inclut une distillation des connaissances et une reconstruction des images masquées :

- Un modèle enseignant pré-entraîné est utilisé pour guider l'apprentissage d'un modèle étudiant.
- Une extraction des canaux RGB est utilisée pour aligner les représentations entre les deux modèles.
- Le modèle étudiant partage les mêmes paramètres et s'entraîne sur une image multispectrale masquée.
- Deux objectifs d'entraînement sont utilisés :
  - Feature Distillation : Alignement des représentations intermédiaires avec celles du modèle enseignant.
  - Masked Image Modeling : Reconstruction des parties masquées de l'image pour l'apprentissage des représentations.

Applications aux downstream tasks

Une fois pré-entraîné, DOFA[2] peut être adapté à différentes tâches en aval telles que la classification ou la segmentation. Pour la classification, un token spécifique est ajouté pour décrire globalement l'image et prédire une étiquette. Pour la segmentation, les représentations sont décodées via des modules produisant une carte où chaque pixel est associé à une classe.

3.2 Reproduction des résultats pour une tâche de classification

On veut reproduire les résultats suivants pour EuroSAT[5]: 92,2% de précision avec un backbone ViT-B, et 93,8% avec ViT-L sur une tâche de classification.

Method	Backbone	m-bigearthnet	m-forestnet	m-brick-kiln	m-pv4ger	m-so2sat	m-eurosat
Fully Trained	ViT-S	66.0	53.8	98.1	97.6	57.5	97.3
Fully Trained	SwinV2-T	70.0	58.0	98.7	98.0	56.1	97.4
Fully Trained	ConvNext-B	69.1	56.8	98.9	98.0	58.1	97.7
rand. init.	ViT-B	52.9	41.5	84.5	91.3	38.3	85.7
MAE.Single [43]	ViT-B	63.6	-	88.9	92.2	50.0	89.0
OFA-Net [42]	ViT-B	65.0	-	94.7	93.2	49.4	91.9
SatMAE [24]	ViT-B	62.1	-	93.9	-	46.9	86.4
Scale-MAE [21]	ViT-L	-	-	-	96.9	-	-
GFM [20]	Swin-B	-	-	-	96.8	-	-
Cross-Scale MAE [22]	ViT-B	-	-	-	93.1	-	-
FG-MAE [23]	ViT-B	63.0	-	94.7	-	51.4	87.0
CROMA [36]	ViT-B	**67.4**	-	91.0	-	49.2	90.1
DOFA	ViT-B	63.8	45.3	94.7	96.9	52.1	92.2
DOFA	ViT-L	64.4	**47.4**	**95.1**	**97.3**	**59.3**	**93.8**

Figure 8: Résultats du linear probing sur six tâches de classification. Les absences de résultats sont dues à l'incapacité du modèle à s'adapter à ce domaine.

Les résultats montrent que DOFA[2] se positionne comme une bonne méthode parmi les modèles fondation récents pour la classification en télédétection. Avec un score de 92.2% sur EuroSAT[5] en configuration ViT-B, DOFA[2] dépasse d'autres approches de type MAE (par exemple SatMAE à 86.4% ou FG-MAE à 87.0%).

Sur l'ensemble m-So2Sat, DOFA[2] ViT-L atteint 59.3%, soit la meilleure performance enregistrée sur cette tâche dans le tableau, devant ConvNext-B (58.1%) et largement devant MAE (50.0%) ou CROMA (49.2%).

Sur d'autres jeux comme ForestNet ou BigEarthNet, les performances de DOFA[2] restent correctes mais inférieures aux meilleurs modèles entièrement entraînés (ex. : 47.4% contre 58.0% pour SwinV2-T sur ForestNet).

Pour reproduire les résultats de DOFA[2] pour une tâche de classification sur EuroSAT[5], on a dans un premier temps chargé le modèle pré-entraîné DOFA. On l'adapte à la classification sur EuroSAT en remplaçant la tête de classification par une couche linéaire adaptée aux 10 classes de ce jeu de données. Seule cette nouvelle tête est entraînée, selon l'approche du linear probing, tandis que le reste du modèle reste figé.

Les images Sentinel-2, composées de 13 bandes spectrales, nécessitent un prétraitement spécifique. Elles sont normalisées à l'aide des moyennes et écarts-types calculés sur l'ensemble d'entraînement. Un vecteur de longueurs d'onde est fourni comme entrée supplémentaire au modèle. Chaque image subit une série de transformations incluant un recadrage aléatoire, un redimensionnement à 224×224[6], une normalisation, puis une conversion en tenseur.

Les données sont réparties en trois ensembles (entraînement, validation et test). Une classe personnalisée, Sentinel2Dataset, est utilisée pour charger les fichiers TIFF, appliquer les transformations et lier chaque image à son étiquette.

L'entraînement du modèle s'appuie sur une fonction de perte adaptée à la classification, couplée à un optimiseur standard pour l'apprentissage supervisé. Seuls les paramètres de la tête de classification sont ajustés, tandis que le reste du modèle pré-entraîné reste figé. Pour limiter le surapprentissage, une régularisation de type weight decay est appliquée lors de l'optimisation. À chaque époque, on évalue aussi l'ensemble de validation pour suivre la convergence.

Enfin, les performances du modèle sont évaluées sur l'ensemble de test. Une matrice de confusion est générée pour visualiser les erreurs de classification, et plusieurs métriques standards sont calculées : accuracy, précision, rappel, F1-score, erreur absolue moyenne (MAE) et erreur quadratique moyenne (MSE).

On a aussi testé le dropout dont l'objectif est de réduire le surapprentissage, en désactivant temporairement une partie des neurones, le modèle apprend des représentations plus stables et généralise mieux sur les données de validation et de test.

Cette modification est particulièrement utile dans notre configuration, où seule la tête est entraînée (linear probing), et où le risque de surajustement est accentué par le faible nombre de paramètres mis à jour.

3.3 Adaptation à la détection d'objets

Pour adapter DOFA[2] à la détection d'objet, on veut rajouter au réseau une tête de détection DETR.
DETR aborde la détection d'objets comme une tâche de prédiction d'un ensemble, où il génère directement un ensemble fixe de propositions de détection. Chaque prédiction comprend une classe d'objet et une boîte englobante. Pour entraîner le modèle, une fonction de perte globale basée sur le Bipartite Matching est utilisée, ce qui permet des prédictions uniques en associant chaque prédiction à une vérité terrain correspondante.
L'architecture de DETR se compose de trois composants principaux :

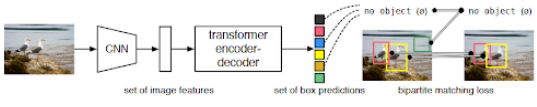


Figure 9: Architecture DETR[7]

1. Backbone CNN : Un réseau de neurones convolutionnel (par exemple, ResNet-50 ou ResNet-101) est utilisé pour extraire une représentation des caractéristiques de l'image.

2. Transformer encoder-decoder : Les caractéristiques extraites sont transformées en une séquence et enrichies avec des encodages positionnels, puis passées à un transformer encoder-decoder. Le decoder reçoit un petit ensemble fixe de requêtes d'objet apprises, qui sont utilisés pour produire des prédictions.

3. Réseau de neurones feed-forward (FFN) : Chaque sortie du decoder est passée à un FFN qui prédit une détection (classe et boîte englobante).

Pour adapter DOFA[2] à la détection d'objet, on veut donc l'utiliser à la place du CNN de la figure comme extracteur de caractéristiques figé, c'est-à-dire que ses poids ne sont pas mis à jour pendant l'apprentissage.

Pour permettre cette intégration, une classe personnalisée, DOFAFeatureExtractor, a été conçue afin d'extraire les représentations intermédiaires produites par les blocs Transformer de DOFA[2]. Ces sorties, sous forme de séquences de patches, sont ensuite transformées par une classe FakeBackbone qui projette ces vecteurs dans un espace à 2048 dimensions compatible avec l'entrée attendue par DETR[7], puis les remodèle en cartes de caractéristiques bidimensionnelles, simulant ainsi la sortie d'un backbone convolutionnel[6].

Le backbone d'origine de DETR[7] est désactivé, et seuls les modules du Transformer et les têtes de détection (classification et régression de boîtes englobantes) sont chargés à partir du modèle préentraîné. Les données d'entrées proviennent du jeu de données DIOR, qui contient des annotations de boîtes horizontales (HBB) et orientées (OBB) sur 20 classes d'objets. Les images sont redimensionnées et normalisées, tandis que les annotations sont mises à l'échelle.

4 Étude expérimentale

4.1 Résultats de l'inférence sur une tâche de classification

Pour reproduire les résultats obtenus par les auteurs, nous avons choisi d'utiliser le jeu de données EuroSAT[5].

Le jeu de données EuroSAT[5] est basé sur des images satellites issues du capteur Sentinel-2, couvrant 13 bandes spectrales. Il contient un total de 27 000 images labellisées et géoréférencées, réparties en 10 classes cibles : Annual Crop, Forest, Herbaceous Vegetation, Highway, Industrial Buildings, Pasture, Permanent Crop, Residential Buildings, River, Sea & Lake. Les images sont fournies sous forme de GeoTIFFs à 13 canaux.

Concernant les paramètres d'entraînement, seule la tête de classification du modèle DOFA[2] est ajustée, suivant une approche de linear probing, tandis que les poids des couches précédentes restent gelés. Pour cette phase, nous avons utilisé la fonction de perte CrossEntropyLoss, adaptée aux tâches de classification multiclasse. Elle permet de mesurer l'écart entre les prédictions du modèle et les étiquettes réelles, en pénalisant plus fortement les mauvaises classifications.

Afin d'évaluer l'influence des choix d'optimisation, de nombreuses configurations ont été explorées en amont, avant de retenir celles présentées ci-dessous. L'objectif était d'identifier les combinaisons de paramètres les plus stables et performantes pour notre tâche de classification.

Parmi les configurations retenues (avec les meilleurs résultats), l'optimiseur SGD a été utilisé avec un taux d'apprentissage de 1e-3, un momentum de 0.9 et un weight decay de 5e-4 pendant 25 epochs. En parallèle, une autre phase d'entraînement a été menée avec l'optimiseur Adam (lr = 1e-2, weight decay = 1e-5), également sur 26 epochs. Une troisième configuration utilise un dropout à 50% dans la tête de classification, combiné à Adam (lr = 1e-3, weight decay = 1e-4), et a été entraînée sur 25 epochs, dans le but de réduire le surapprentissage.

La taille des batchs était fixée à 64 pour l'entraînement et à 128 pour la validation et le test. Des courbes de perte ont été analysées pour suivre la dynamique d'apprentissage et s'assurer de la bonne convergence du modèle. Ces tests ont permis de comparer l'impact des différents réglages sur la stabilité et la performance de la tête de classification.

Pour SGD (lr = 1e-3, momentum = 0.9, weight decay = 5e-4, 25 epochs):

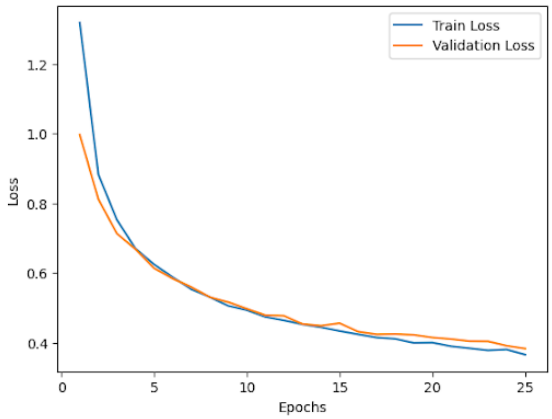


Figure 10: Courbe de loss SGD

Paramètre	Valeur
Époque	25 / 25
Train Loss	0.364775
Train Accuracy	0.8983
Validation Loss	0.382473
Validation Accuracy	0.8931

Table 1: Résultats de l'entraînement à l'époque 25

La perte diminue de façon continue, aussi bien sur l'ensemble d'entraînement que de validation. L'accuracy augmente de manière stable, atteignant près de 90% . Cela indique un apprentissage efficace, avec une bonne capacité de généralisation du modèle. L'absence de divergence entre les courbes d'entraînement et de validation suggère également que le modèle ne souffre pas de surapprentissage,

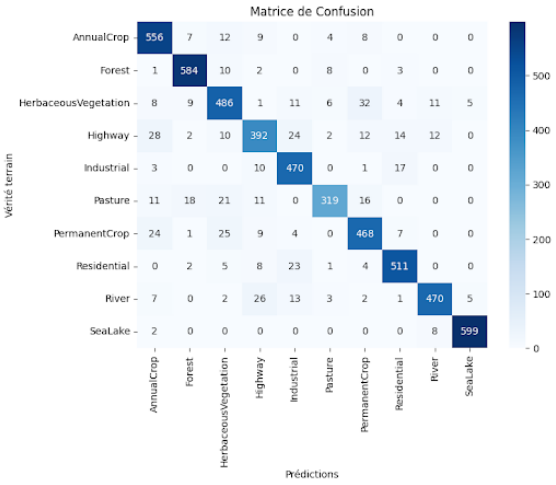


Figure 11: Matrice de confusion SGD

Pour Adam (lr = 1e-3, weight decay = 1e-4,26 epochs):

Métrique	Valeur
Accuracy	89.91%
Precision	0.8997
Recall	0.8991
F1 Score	0.8986
Mean Absolute Error (MAE)	0.3365
Mean Squared Error (MSE)	1.4176

Table 2: Résumé des performances du modèle

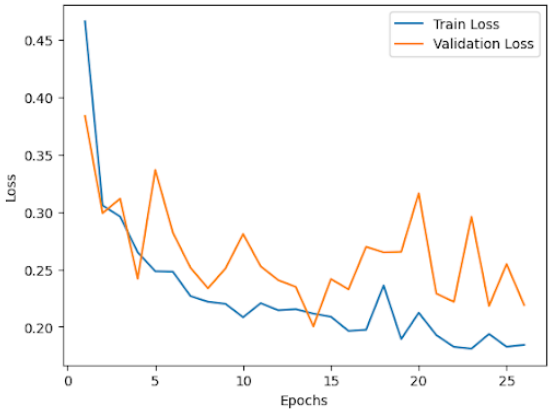


Figure 12: Courbe de loss Adam

Paramètre	Valeur
Époque	26 / 26
Train Loss	0.184260
Train Accuracy	0.9472
Validation Loss	0.218900
Validation Accuracy	0.9428

Table 3: Résultats de l'entraînement à l'époque 26

Les pertes d'entraînement et de validation baissent nettement pendant les premières époques. Mais à partir de la 6ème époque, la perte de validation commence à fluctuer, tandis que la perte d'entraînement continue de diminuer. Cela montre que le modèle surapprend : il devient trop spécialisé sur les données d'entraînement et perd en capacité à généraliser. Comparé à l'entraînement avec SGD, qui progresse plus lentement mais reste plus stable, Adam converge plus vite mais peut manquer de régularité. Pour éviter ce surapprentissage, on peut utiliser des techniques comme le dropout.

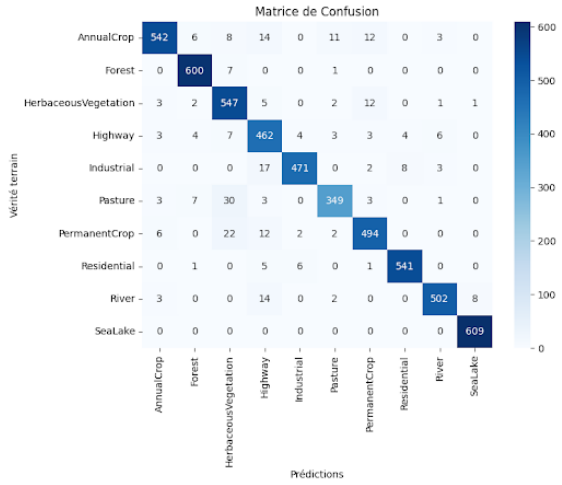


Figure 13: Matrice de confusion Adam

Métrique	Valeur
Accuracy	94.76%
Precision	0.9489
Recall	0.9476
F1 Score	0.9477
Mean Absolute Error (MAE)	0.1696
Mean Squared Error (MSE)	0.6959

Table 4: Résumé des performances du modèle

Pour Adam avec dropout (lr = 1e-3, weight decay = 1e-4, 25 epochs):

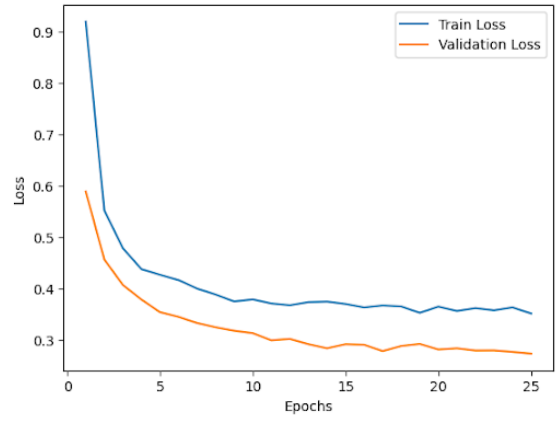


Figure 14: Courbe de loss Adam avec dropout

Paramètre	Valeur
Époque	25 / 25
Train Loss	0.351570
Train Accuracy	0.8889
Validation Loss	0.273362
Validation Accuracy	0.9248

Table 5: Résultats de l'entraînement à l'époque 25

L'entraînement montre une évolution plus stable et efficace que précédemment. Dès les premières époques, la perte d'entraînement et la perte de validation diminuent rapidement. Contrairement au cas précédent, la perte de validation reste stable et continue à baisser globalement tout au long des 25 époques, avec peu de fluctuations.

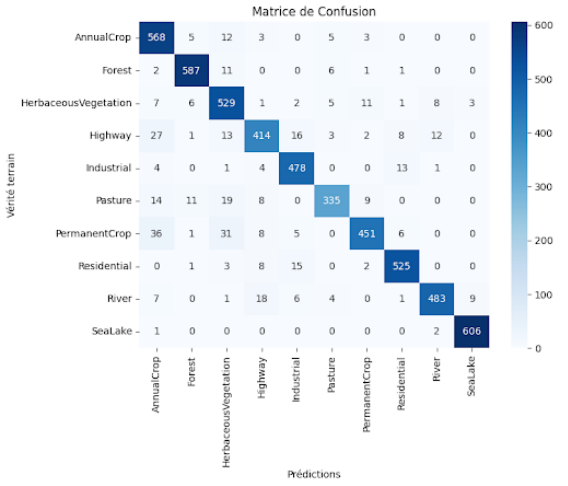


Figure 15: Matrice de confusion Adam avec dropout

Métrique	Valeur
Accuracy	91.76%
Precision	0.9185
Recall	0.9176
F1 Score	0.9172
Mean Absolute Error (MAE)	0.2802
Mean Squared Error (MSE)	1.2131

Table 6: Résumé des performances du modèle

Adam sans dropout donne les meilleurs résultats, avec une accuracy de 94.76% . SGD est moins efficace, avec une accuracy de 89.91%, dû à un apprentissage plus lent. Ajouter du dropout à Adam améliore la régularisation : les performances sont meilleures que SGD (91.76%) mais un peu moins bonnes qu'Adam seul. Cela indique que le modèle de base ne surapprenait pas beaucoup, et qu'Adam sans dropout reste le plus performant dans ce cas.

4.2 Expérimentation et résultats de l'adaptation à la détection

Le jeu de données DIOR[8] (Dataset for Object Detection in Optical Remote sensing images) a été introduit dans l'article Object Detection in Optical Remote Sensing Images: A Survey and A New Benchmark dans le but de proposer une référence pour la détection d'objets avec les images satellites. Il se distingue par sa grande diversité

géographique et sémantique, avec plus de 23 000 images haute résolution couvrant différentes régions du globe. Chaque image est annotée avec des boîtes englobantes (bounding boxes) pour 20 classes d'objets telles que véhicules, avions, ponts, bâtiments résidentiels etc. Le dataset contient au total plus de 190 000 annotations, avec des variations importantes en termes d'échelle, d'orientation et de densité des objets. DIOR permet donc de tester les méthodes de détection face aux défis spécifiques de l'imagerie satellitaire, comme la variation d'angle de vue, le chevauchement des objets, ou encore les arrière-plans complexes.Il suit la répartition suivante: 25% entraînement,25% validation et 50% test.

Pour adapter le modèle DOFA[2] à la détection d'objets, notre première approche a consisté à utiliser un modèle DETR[7] préentraîné, notamment la version disponible sous facebook/detr-resnet-50. Nous avons gelé les poids de DOFA pour l'utiliser comme simple extracteur de caractéristiques, et nous avons ensuite essayé de transmettre ces dernières directement dans l'encodeur de DETR. Cependant, cette approche s'est révélée inadéquate.

En effet, ce contournement court-circuite la logique interne de DETR[7] : le modèle parent s'attend toujours à recevoir des pixel-values (tenseurs qui représentent les images d'entrées prétraitées), à les faire passer par son propre backbone (par défaut, ResNet), et à produire lui-même les entrées nécessaires pour son Transformer.

Un autre point critique concerne la structure de sortie du backbone. Le backbone de DETR[7] ne renvoie pas uniquement des features, mais aussi un masque spatial (pixel-mask), qui indique quelles régions de l'image sont valides ou non. Dans notre tentative initiale, nous avons essayé de projeter les features de DOFA[2] (de dimension [B, N, 768]) via une simple couche linéaire, sans les formater en une structure spatiale classique ([B, C, H, W]). En conséquence, le masque ne pouvait pas être utilisé correctement.

Face à ces limitations, nous avons réorienté notre stratégie. Plutôt que d'essayer de connecter manuellement les sorties de DOFA[2] à l'encodeur/décodeur de DETR[7], nous avons choisi de mimer le comportement du backbone ResNet d'origine. Pour cela, nous avons conçu notre propre backbone utilisant les caractéristiques extraites par DOFA et capable de reproduire les mêmes sorties (features + pixel mask) que ResNet, dans le bon format. Cela nous a permis de désactiver complètement le backbone d'origine et de le remplacer par notre propre pipeline basé sur DOFA, tout en conservant l'encodeur/décodeur préentraînés du modèle DETR. Cette approche assure la compatibilité au niveau de la structure et permet une meilleure intégration de DOFA dans l'architecture de détection.

Du fait des limitations de temps de GPU avec Google Colab, nous n'avons pu entraîner notre réseau que pour 24 epochs.

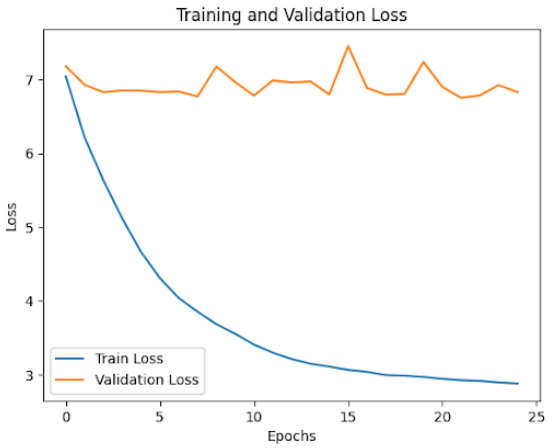


Figure 16: Courbe de perte (training et validation) pour l'adaptation de DOFA[2] à la détection d'objet

La courbe de perte de l'ensemble d'entraînement baisse progressivement, ce qui montre que le modèle apprend bien avec les données disponibles, même si elles sont peu nombreuses. En revanche, la perte de celui de validation reste stable autour de 7, avec des variations, ce qui signifie que le modèle ne généralise pas bien. Cela peut venir du fait qu'il y a trop peu de données d'entraînements seulement 25%) ou de la façon dont on extrait les features avec DOFA[2].

Pour évaluer les performances de notre modèle, on utilise les métriques mAP (mean Average Precision) et AP (Average Precision). L'AP mesure la précision moyenne du modèle pour détecter correctement une seule classe d'objet, en tenant compte à la fois de la précision (le taux de bonnes détections parmi toutes les détections faites) et du rappel (le taux d'objets correctement détectés parmi tous les objets présents). La mAP correspond à la moyenne des AP calculées sur toutes les classes d'objets du jeu de données, offrant ainsi une mesure globale de la qualité du modèle. Ces métriques reflètent la capacité du modèle à localiser précisément les objets, mais aussi à limiter les fausses détections, ce qui est très important pour une bonne détection d'objets.

Nous avons obtenu les résultats suivants pour toutes les classes de DIOR:

Métrique / Classe	Valeur (%)
mAP@0.5	0.00001645
mAP@0.5:0.95	0.00000245
airplane	0.00000000
airport	0.00000000
baseballfield	0.00000000
basketballcourt	0.00000000
bridge	0.00000000
chimney	0.00000000
dam	0.00000000
expresswayservicearea	0.00000000
expresswaytollstation	0.00000000
golffield	0.00000000
groundtrackfield	0.00000000
harbor	0.00000000
overpass	0.00000000
ship	0.00001856
stadium	0.00000000
storagetank	0.00000000
tenniscourt	0.00000000
trainstation	0.00000000
vehicle	0.00002176
windmill	0.00000877

Table 7: Résultats de détection par classe et scores mAP

Voici quelques exemples de prédiction (dans la majorité du temps, on ne détecte rien ou de très petites boites englobantes)

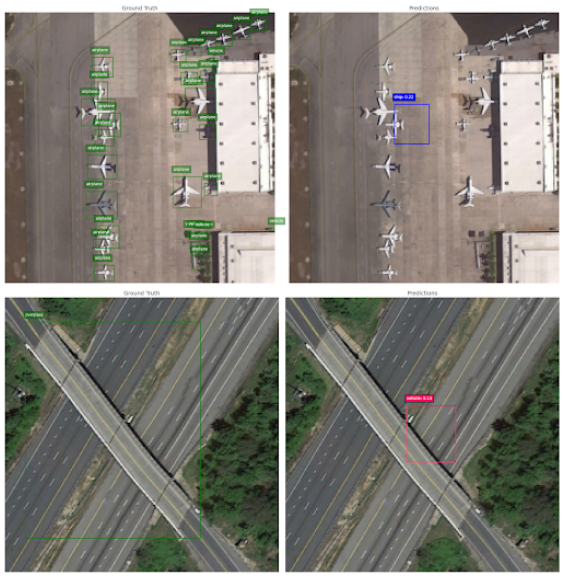


Figure 17: Exemple de prédiction avec le nouveau réseau

Les résultats présentés dans le tableau montrent des performances solides des modèles classiques de détection d'objets, avec des mAP comprises entre environ 57% (YOLOv3) et 66,1% (RetinaNet et PANet). Ces valeurs indiquent que ces modèles détectent et localisent efficacement les objets.

Modèle	Backbone	mAP(%)
RetinaNet	ResNet-101	66.1
Mask R-CNN, FPN	ResNet-50	65.7
Faster R-CNN, FPN	ResNet-101	65.1
PANet	ResNet-101	66.1
YOLOv3	Darknet-53	57.1

Table 8: Performances des modèles de détection d'objets selon le Backbone et le mAP

En revanche, nos propres résultats sont extrêmement faibles, avec un mAP@0.5 quasiment nul (0.00001645%) et un mAP@0.5:0.95 encore plus bas (0.00000245%). De plus, les précisions par classe sont toutes nulles ou proches de zéro, excepté pour quelques classes comme "ship" (0.00001856%) et "vehicle" (0.00002176%), qui restent néanmoins insignifiantes comparées aux références.

Cette énorme différence de performance souligne plusieurs problèmes possibles : un entraînement insuffisant, des données d'entraînement trop peu nombreuses, ou encore une mauvaise généralisation du modèle sur notre jeu de données. Comparé aux modèles établis, notre modèle semble incapable de détecter les objets correctement, ce qui montre la nécessité d'améliorer l'adaptation de DOFA[2] à la détection d'objets.

Pour mieux comprendre pourquoi notre modèle ne fonctionne pas, on pourrait créer une version très réduite du jeu de test, avec seulement 1% des données. Le but est de faire overfitter le modèle, c'est-à-dire qu'il apprenne parfaitement ce petit jeu. Si le modèle réussit, cela voudra dire qu'il peut apprendre, et que le problème vient peut-être de l'entraînement.

En revanche, si même avec ce mini jeu de test le modèle échoue à détecter correctement, cela indiquerait que le problème est plus profond, lié à l'adaptation du modèle DOFA[2] à la tâche de détection d'objets. Ce test permettrait donc de mieux cibler les axes d'amélioration à privilégier.

Parmi les différentes pistes d'amélioration à envisager, on peut par exemple avoir une augmentation de données plus robuste. Cela inclut des transformations géométriques (retournements, rotations, zooms) et des variations de couleur (luminosité, contraste, saturation), tout en veillant

à ajuster correctement les boîtes englobantes.

Ensuite, la gestion des boîtes orientées (OBB pour Oriented Bounding Box) est un point critique : si les objets du dataset DIOR peuvent être inclinés, convertir les OBB en boîtes horizontales (HBB) peut provoquer une perte d'information, notamment en englobant des zones inutiles ou en rendant les détections moins précises. Nous avons effectué cette conversion principalement pour des raisons de simplicité et de compatibilité avec l'architecture actuelle du modèle. Il peut alors être préférable de traiter directement les OBB.

Un autre point clé concerne la résolution et la représentation multi-échelle des images. DOFA[2], en tant que ViT, produit des caractéristiques à une seule échelle, ce qui peut éventuellement limiter sa capacité à détecter des objets de tailles variées. Il serait pertinent de modifier notre backbone pour extraire des caractéristiques à différentes résolutions.

En résumé, nos résultats actuels montrent que l'adaptation de DOFA[2] à la détection d'objets reste perfectible, et plusieurs pistes d'amélioration doivent encore être explorées pour atteindre des performances satisfaisantes.

5 Conclusion et perspectives

Ce projet s'est déroulé en trois phases. D'abord, une familiarisation avec l'architecture de DOFA[2] a permis de comprendre sa capacité à gérer l'hétérogénéité des capteurs. Ensuite, la reproduction des performances de classification sur EuroSAT[5], avec 94.76% d'accuracy, a confirmé la bonne compréhension du modèle et des notions de base en apprentissage profond.

L'adaptation de DOFA[2] à la détection d'objets avec DETR[7] sur le dataset DIOR a soulevé plusieurs défis liés à l'architecture et aux différences de format entre les deux modèles. Malgré des résultats encore faibles, cette première tentative a permis d'identifier les limites actuelles, d'ouvrir plusieurs pistes d'amélioration.

En effet l'adaptation de DOFA[2] à la détection d'objets sur DIOR[8] a donné des résultats très faibles, avec un mAP@0.5 presque nul (0,00001645%). Les précisions par classe sont aussi très basses.

Pour mieux comprendre le problème, on propose d'entraîner le modèle sur une petite partie du jeu de test (1%). Si le modèle réussit à surapprendre ce petit jeu, cela indiquerait un problème au niveau de l'entraînement. Sinon, cela montrerait que DOFA[2] n'est pas bien adapté à la détection d'objets, et qu'il faut revoir notre pipeline.

Plusieurs améliorations sont à envisager :

- Une augmentation de données plus solide, avec des transformations géométriques (rotations, retournements, zooms) et des variations de couleur.
- Une meilleure gestion des boîtes orientées (OBB)
- Une représentation multi-échelle en s'inspirant, par exemple, des Feature Pyramid Networks (FPN) pour extraire des caractéristiques à plusieurs résolutions.

Ces pistes doivent être explorées pour améliorer les performances de DOFA[2] en détection d'objets.

References

[1] F. Bastani *et al.*, "Satlaspretrain: A Large-scale Dataset for Remote Sensing Image Understanding," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[2] Z. Xiong, "Neural Plasticity-Inspired Multimodal Foundation Model for Earth Observation," *arXiv preprint arXiv:2403.15356*, 2024.

[3] A. Vaswani *et al.*, "Attention is All You Need," in *Advances in Neural Information Processing Systems*, 2017.

[4] G. Sumbul *et al.*, "BigEarthNet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding," in *IGARSS 2019 - IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2019.

[5] P. Helber *et al.*, "Introducing EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2019.

[6] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv preprint arXiv:2010.11929*, 2020.

[7] N. Carion *et al.*, "End-to-End Object Detection with Transformers," in *European Conference on Computer Vision (ECCV)*, 2020.

[8] G. Cheng *et al.*, "Object Detection in Optical Remote Sensing Images: A Survey and A New Benchmark," *ISPRS Journal of Photogrammetry and Remote Sensing*, 2020.

Annexes

c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
Airplane	Airport	Baseball field	Basketball court	Bridge	Chimney	Dam	Expressway service area	Expressway toll station	Golf course
c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
Ground track field	Harbor	Overpass	Ship	Stadium	Storage tank	Tennis court	Train station	Vehicle	Wind mill

	Backbone	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20	mAP
R-CNN	VGG16	35.6	43.0	53.8	62.3	15.6	53.7	33.7	50.2	33.5	50.1	49.3	39.5	30.9	9.1	60.8	18.0	54.0	36.1	9.1	16.4	37.7
RICNN	VGG16	39.1	61.0	60.1	66.3	25.3	63.3	41.1	51.7	36.6	55.9	58.9	43.5	39.0	9.1	61.1	19.1	63.5	46.1	11.4	31.5	44.2
RICAOD	VGG16	42.2	69.7	62.0	79.0	27.7	68.9	50.1	60.5	49.3	64.4	65.3	42.3	46.8	11.7	53.5	24.5	70.3	53.3	20.4	56.2	50.9
RIFD-CNN	VGG16	56.6	53.2	**79.9**	69.0	29.0	71.5	63.1	69.0	56.0	68.9	62.4	**51.2**	51.1	31.7	**73.6**	41.5	79.5	40.1	28.5	46.9	56.1
Faster R-CNN	VGG16	53.6	49.3	78.8	66.2	28.0	70.9	62.3	69.0	55.2	68.0	56.9	50.2	50.1	27.7	73.0	39.8	75.2	38.6	23.6	45.4	54.1
SSD	VGG16	59.5	72.7	72.4	75.7	29.7	65.8	56.6	63.5	53.1	65.3	68.6	49.4	48.1	59.2	61.0	46.6	76.3	55.1	27.4	65.7	58.6
YOLOv3	Darknet-53	**72.2**	29.2	74.0	78.6	31.2	69.7	26.9	48.6	54.4	31.1	61.1	44.9	49.7	**87.4**	70.6	**68.7**	**87.3**	29.4	**48.3**	78.7	57.1
Faster RCNN with FPN	ResNet-50	54.1	71.4	63.3	81.0	42.6	72.5	57.5	68.7	62.1	73.1	76.5	42.8	56.0	71.8	57.0	53.5	81.2	53.0	43.1	80.9	63.1
	ResNet-101	54.0	74.5	63.3	80.7	44.8	72.5	60.0	75.6	62.3	76.0	76.8	46.4	57.2	71.8	68.3	53.8	81.1	59.5	43.1	81.2	65.1
Mask-RCNN with FPN	ResNet-50	53.8	72.3	63.2	81.0	38.7	72.6	55.9	71.6	67.0	73.0	75.8	44.2	56.5	71.9	58.6	53.6	81.1	54.0	43.1	81.1	63.5
	ResNet-101	53.9	76.6	63.2	80.9	40.2	72.5	60.4	76.3	62.5	76.0	75.9	46.5	57.4	71.8	68.3	53.7	81.0	**62.3**	43.0	81.0	65.2
RetinaNet	ResNet-50	53.7	77.3	69.0	81.3	44.1	72.3	62.5	76.2	66.0	77.7	74.2	50.7	59.6	71.2	69.3	44.8	81.3	54.2	45.1	83.4	65.7
	ResNet-101	53.3	77.0	69.3	**85.0**	44.1	73.2	62.4	78.6	62.8	78.6	76.6	49.9	59.6	71.1	68.4	45.8	81.3	55.2	44.4	85.5	**66.1**
PANet	ResNet-50	61.9	70.4	71.0	80.4	38.9	72.5	56.6	68.4	60.0	69.0	74.6	41.6	55.8	71.7	72.9	62.3	81.2	54.6	48.2	**86.7**	63.8
	ResNet-101	60.2	72.0	70.6	80.5	43.6	72.3	61.4	72.1	66.7	72.0	73.4	45.3	56.9	71.7	70.4	62.0	80.9	57.0	47.2	84.5	**66.1**
CornerNet	Hourglass-104	58.8	**84.2**	72.0	80.8	**46.4**	**75.3**	**64.3**	**81.6**	**76.3**	**79.5**	**79.5**	26.1	**60.6**	37.6	70.7	45.2	84.0	57.1	43.0	75.9	64.9

Figure 18: mAP de 12 méthodes représentatives sur l'ensemble de test DIOR proposé. Les meilleures précisions moyennes (AP) pour chaque catégorie d'objet sont indiquées en gras[8]

```python
class DOFAFeatureExtractor(nn.Module):
    def __init__(self, dofa_model):
        super().__init__()
        self.dofa = dofa_model

    def forward(self, pixel_values, wavelengths=None):
        patch_embed_output = self.dofa.patch_embed(pixel_values, wvs=
            wavelengths)
        # Si l'encodage retourne un tuple, on s lectionne le premier    lment
            (les features)
        if isinstance(patch_embed_output, tuple):
            features = patch_embed_output[0]
        else:
            features = patch_embed_output
        # Normalisation des features avec la couche fully-connected de
            normalisation du mod le DOFA
        features = self.dofa.fc_norm(features)

        # Passage des features    travers chaque bloc du mod le DOFA
        for block in self.dofa.blocks:
            features = block(features)

        return features
```

Listing 1: Extraction de caractéristiques avec DOFA

```python
class DetrWithDOFA(nn.Module):
    def __init__(self, feature_extractor):
        super().__init__()
        self.feature_extractor = feature_extractor

       # Charger la config sans backbone
```

```
        detr_config = DetrConfig.from_pretrained("facebook/detr-resnet-50")
        detr_config.use_pretrained_backbone = False
        detr_config.num_labels = 20
        self.detr_model = DetrForObjectDetection(detr_config)

        # Charger uniquement les poids utiles (transformer)
        pretrained = DetrForObjectDetection.from_pretrained("facebook/detr-
            resnet-50")
        filtered_weights = {
            k: v for k, v in pretrained.state_dict().items()
            if "transformer" in k
        }
        self.detr_model.load_state_dict(filtered_weights, strict=False)
        #  Extraire la dimension des features de DOFA
        with torch.no_grad():
            dummy = torch.randn(1, 3, 224, 224).to(device)
            dummy_wvs = torch.tensor([0.665, 0.56, 0.49], dtype=torch.float32).
                to(device)
            feats = self.feature_extractor(dummy, dummy_wvs)  # [1, N, D]
            self.feature_dim = feats.shape[2]  # D

        #  Remplacer le backbone ResNet par Backbone perso bas  sur DOFA
        self.detr_model.model.backbone = FakeBackbone(self.feature_extractor,
            self.feature_dim)

    def forward(self, images, targets=None):
        # Appel direct    DetrForObjectDetection, les features sont g r es
            par FakeBackbone
        return self.detr_model(pixel_values=images, labels=targets)
```

Listing 2: Inclusion de dofa dans l'architecture DETR

```
class FakeBackbone(nn.Module):
    def __init__(self, feature_extractor, feature_dim):
        super().__init__()
        self.feature_extractor = feature_extractor
        self.proj = nn.Linear(feature_dim, 2048)  #  match les attentes de DETR

    def forward(self, pixel_values, *args, **kwargs):
        batch_size = pixel_values.shape[0]
        device = pixel_values.device
        wavelengths = torch.tensor([0.665, 0.56, 0.49], dtype=torch.float32).to
            (device)

        # DOFA features [B, N, D]
        features = self.feature_extractor(pixel_values, wavelengths)

        # Projection     2048 canaux
        features = self.proj(features)  # [B, N, 2048]

        # Reshape pour format image-like [B, 2048, H, W]
        h = w = int(features.shape[1] ** 0.5)
        features = features.permute(0, 2, 1).contiguous()      # [B, 2048, N]
        features = features.view(batch_size, 2048, h, w)       # [B, 2048, H, W]

        mask = torch.zeros((batch_size, h, w), dtype=torch.bool, device=device)

        # Cr er un dummy object_queries_list pour satisfaire DETR
        object_queries_list = [torch.zeros((features.shape[0], 256, 1), device=
            device)]

        # Retour format attendu : (features_list, object_queries_list)
        return [(features, mask)], object_queries_list
```

Listing 3: Backbone DETR-like

```python
class DOFAFeatureExtractor(nn.Module):
    def __init__(self, dofa_model):
        super().__init__()
        self.dofa = dofa_model

    def forward(self, pixel_values, wavelengths=None):
        patch_embed_output = self.dofa.patch_embed(pixel_values, wvs=
            wavelengths)
        # Si l'encodage retourne un tuple, on s lectionne le premier   lment
             (les features)
        if isinstance(patch_embed_output, tuple):
            features = patch_embed_output[0]
        else:
            features = patch_embed_output
        # Normalisation des features avec la couche fully-connected de
            normalisation du mod le DOFA
        features = self.dofa.fc_norm(features)

        # Passage des features    travers chaque bloc du mod le DOFA
        for block in self.dofa.blocks:
            features = block(features)

        return features
```

Listing 4: Extraction de caractéristiques avec DOFA

```python
class DetrWithDOFA(nn.Module):
    def __init__(self, feature_extractor):
        super().__init__()
        self.feature_extractor = feature_extractor

       # Charger la config sans backbone
        detr_config = DetrConfig.from_pretrained("facebook/detr-resnet-50")
        detr_config.use_pretrained_backbone = False
        detr_config.num_labels = 20
        self.detr_model = DetrForObjectDetection(detr_config)

        # Charger uniquement les poids utiles (transformer)
        pretrained = DetrForObjectDetection.from_pretrained("facebook/detr-
            resnet-50")
        filtered_weights = {
            k: v for k, v in pretrained.state_dict().items()
            if "transformer" in k
        }
        self.detr_model.load_state_dict(filtered_weights, strict=False)
        #  Extraire la dimension des features de DOFA
        with torch.no_grad():
            dummy = torch.randn(1, 3, 224, 224).to(device)
            dummy_wvs = torch.tensor([0.665, 0.56, 0.49], dtype=torch.float32).
                to(device)
            feats = self.feature_extractor(dummy, dummy_wvs)  # [1, N, D]
            self.feature_dim = feats.shape[2]  # D

        #  Remplacer le backbone ResNet par Backbone perso bas   sur DOFA
        self.detr_model.model.backbone = FakeBackbone(self.feature_extractor,
            self.feature_dim)

    def forward(self, images, targets=None):
        # Appel direct    DetrForObjectDetection, les features sont g r es
            par FakeBackbone
```

```
        return self.detr_model(pixel_values=images, labels=targets)
```

Listing 5: Inclusion de dofa dans l'architecture DETR

```
class FakeBackbone(nn.Module):
    def __init__(self, feature_extractor, feature_dim):
        super().__init__()
        self.feature_extractor = feature_extractor
        self.proj = nn.Linear(feature_dim, 2048)  #  match les attentes de DETR

    def forward(self, pixel_values, *args, **kwargs):
        batch_size = pixel_values.shape[0]
        device = pixel_values.device
        wavelengths = torch.tensor([0.665, 0.56, 0.49], dtype=torch.float32).to
            (device)

        # DOFA features [B, N, D]
        features = self.feature_extractor(pixel_values, wavelengths)

        # Projection     2048 canaux
        features = self.proj(features)  # [B, N, 2048]

        # Reshape pour format image-like [B, 2048, H, W]
        h = w = int(features.shape[1] ** 0.5)
        features = features.permute(0, 2, 1).contiguous()     # [B, 2048, N]
        features = features.view(batch_size, 2048, h, w)      # [B, 2048, H, W]

        mask = torch.zeros((batch_size, h, w), dtype=torch.bool, device=device)

        # Cr er un dummy object_queries_list pour satisfaire DETR
        object_queries_list = [torch.zeros((features.shape[0], 256, 1), device=
            device)]

        # Retour format attendu : (features_list, object_queries_list)
        return [(features, mask)], object_queries_list
```

Listing 6: Backbone DETR-like