

SUPER-RISOLUZIONE DI IMMAGINI SATELLITARI GEOLOGICHE

COMPUTATIONAL
IMAGING

LEONARDO VORABBI, DAVIDE DE ROSA, CARLOTTA NUNZIATI



INTRODUZIONE

Immagina di dover valutare i danni dopo una frana avendo a disposizione immagini satellitari a 10 metri di risoluzione

Satellitari → perdita di dettagli cruciali

Aeree → costose e meno frequenti



INTRODUZIONE

Immagina di dover valutare i danni dopo una frana avendo a disposizione immagini satellitari a 10 metri di risoluzione

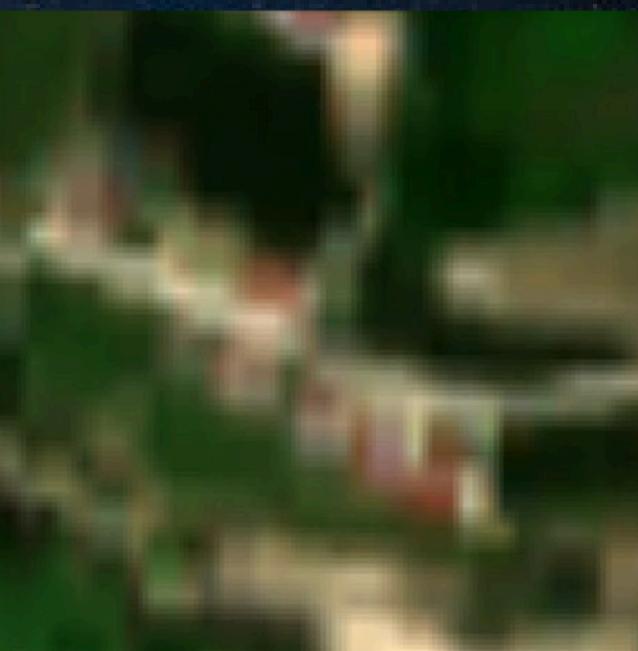
Soluzione?

Super-risoluzione guidata da deep learning

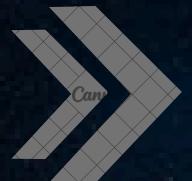
COS'È LA SUPER-RISOLUZIONE

Formalizzazione del problema

- Input: immagine a bassa risoluzione (Sentinel-2, 10 m)
- Output: immagine ad alta risoluzione (target: aerea, 2 m)
- Obiettivo: ricostruire dettagli persi



Input



Output

LA SFIDA

Ricostruire immagini ad alta risoluzione a partire da input a bassa risoluzione mantenendo realismo, coerenza geometrica e qualità visiva

Compromesso tipico

Alta copertura (es. satelliti)

vs.

Alta risoluzione (es. aerei)

OBIETTIVI DEL PROGETTO

Trasformare immagini multispettrali Sentinel-2 in
aeree ad alta risoluzione

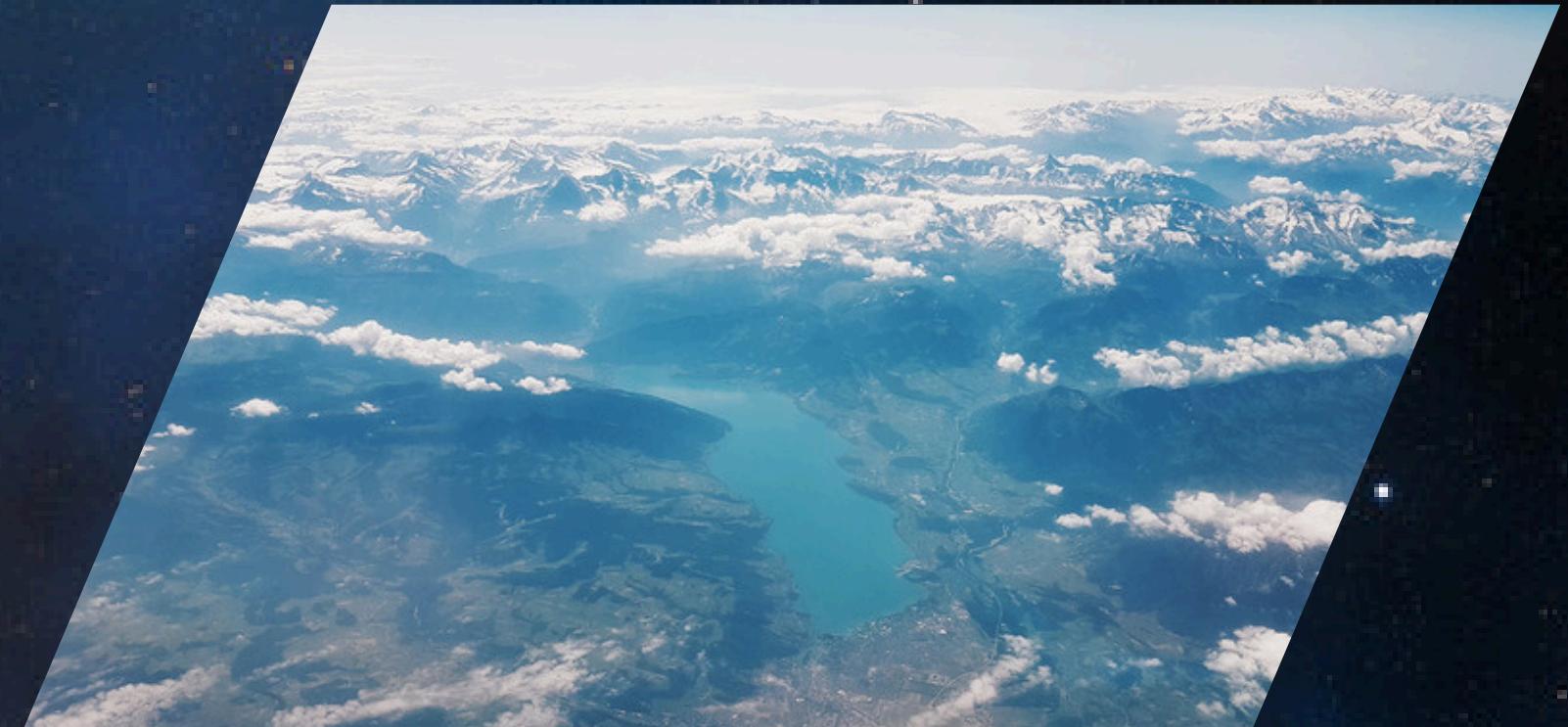
Coppie di immagini (Sentinel-Aerea)



Rete neurale che impara la mappatura

CONTESTO APPLICATIVO

Dove le aeree non arrivano, possiamo generarle



- Monitoraggio di frane e dissesti
- Agricoltura di precisione
- Gestione post-evento (alluvioni, incendi)
- Urbanistica e consumo di suolo



IMPATTO ATTESO

Super-risoluzione → Informazioni più fruibili

- Decisioni più rapide
- Analisi più accurate
- Meno costi di rilievo
- Copertura spaziale più ampia





FLUSSO DI LAVORO

Dal dato grezzo alla predizione

1. Download immagini Sentinel & aeree
2. Generazione patch 128x128
3. Preprocessing: normalizzazione, augmentation
4. Training modello (Residual UNet / VIT)
5. Predizione e valutazione



PUNTI CRITICI TECNICI

- Gestione bordi e NaN nelle patch
- Gestione batch e memoria GPU
- Esclusione dataset “Brisighella” per limiti computazionali

IL DATASET

Immagini utilizzate

- Sentinel-2: pre e post evento (10m)
- Aerea AGEA / CGR del 2020 e 2023 (2m)
- Area studio: 3 comuni emiliani
 - Predappio
 - **Casola Valsenio**
 - Modigliana



PATCH & GENERATORE

Generazione Patch

- Accoppiamento
- Ritaglio patch 128×128
- Filtro su bordi neri / NaN
- Normalizzazione
- Augmentation (flip, rotazioni)
- Divisione dataset

PATCH & GENERATORE

Accoppiamento

```
sentinel_list = [
    "/content/drive/MyDrive/ColabContent/data/PREDAPPIO/Sentinel/Sentinel2_post_Predappio_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/PREDAPPIO/Sentinel/Sentinel2_pre_Predappio_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/CASOLA_VALSENIO/Sentinel/Sentinel2_post_Casola_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/CASOLA_VALSENIO/Sentinel/Sentinel2_pre_Casola_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/MODIGLIANA/Sentinel/Sentinel2_post_modigliana_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/MODIGLIANA/Sentinel/Sentinel2_pre_Modigliana_2m.tif"
]

aerial_list = [
    "/content/drive/MyDrive/ColabContent/data/PREDAPPIO/Aerial/PREDAPPIO_cgr_2023_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/PREDAPPIO/Aerial/PREDAPPIO_ägea_2020_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/CASOLA_VALSENIO/Aerial/CASOLA_VALSENIO_cgr_2023_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/CASOLA_VALSENIO/Aerial/CASOLA_VALSENIO_ägea_2020_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/MODIGLIANA/Aerial/MODIGLIANA_cgr_2023_2m.tif",
    "/content/drive/MyDrive/ColabContent/data/MODIGLIANA/Aerial/MODIGLIANA_ägea_2020_2m.tif"
]
```

PATCH & GENERATORE

Ritaglio patch 128x128

```
# Estrai patch a griglia
patches = self._extract_patches(sentinel, aerial)
```



PATCH & GENERATORE

Filtro su bordi neri / NaN

```
# Scarta patch contenenti valori non finiti
if not np.isfinite(sp).all() or not np.isfinite(ap).all():
    continue

valid_ratio = np.count_nonzero(sp) / sp.size
if valid_ratio < 0.1:
    continue
```

Mantiene le patch con meno del 10% di bordo

PATCH & GENERATORE

Normalizzazione

```
if self.normalize:  
    sp = (sp - sp.mean()) / (sp.std() + 1e-8)
```

- Centra i dati
- Uniformità delle scale
- Stabilità numerica

$$\text{Patch Normalizzata} = \frac{\text{Patch} - \text{Media}}{\text{Deviazione Standard} + 1e-8}$$

PATCH & GENERATORE

Augmentation

- **Flip verticale** al 50%
- **Flip orizzontale** al 50%
- **Rotazione di 90°** al 50%

```
if self.augment:  
    # Flip verticale  
    if random.random() < 0.5:  
        sentinel = np.flip(sentinel, axis=1)  
        aerial = np.flip(aerial, axis=1)  
  
    # Flip orizzontale  
    if random.random() < 0.5:  
        sentinel = np.flip(sentinel, axis=2)  
        aerial = np.flip(aerial, axis=2)  
  
    # Rotazione 90°  
    if random.random() < 0.5:  
        sentinel = np.rot90(sentinel, k=1, axes=(1, 2))  
        aerial = np.rot90(aerial, k=1, axes=(1, 2))
```

PATCH & GENERATORE

Divisione del dataset

```
# Split fisso per immagini (non per patch)
indices = list(range(n_imgs))
random.seed(42)
random.shuffle(indices)

# Assegna 1 immagine al test, 1 alla val, il resto al train
test_idx = indices[-1:]
val_idx = indices[-2:-1]
train_idx = indices[:-2]

# Costruisci i path per ogni split
train_s = [sentinel_paths[i] for i in train_idx]
train_a = [aerial_paths[i] for i in train_idx]

val_s = [sentinel_paths[i] for i in val_idx]
val_a = [aerial_paths[i] for i in val_idx]

test_s = [sentinel_paths[i] for i in test_idx]
test_a = [aerial_paths[i] for i in test_idx]
```

- **Train set:** 4 immagini
- **Val set:** 1 immagine
- **Test set:** 1 immagine

METRICHE DI VALUTAZIONE

- **MSE** – Errore quadratico medio
- **SSIM** – Similarità strutturale
- **PSNR** – Rapporto segnale/rumore

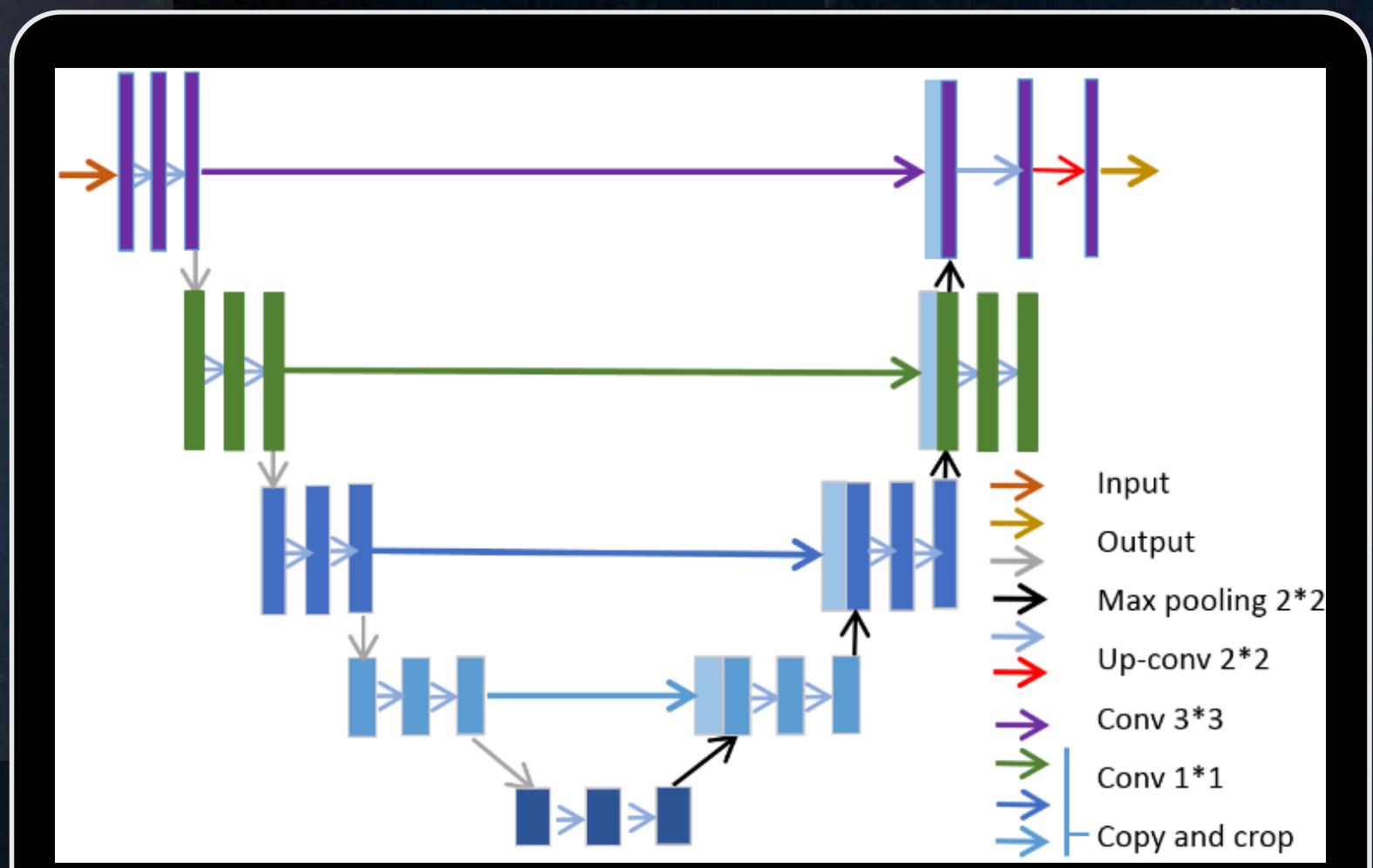


ADDESTRAMENTO DEI MODELLI

UNET CLASSICA

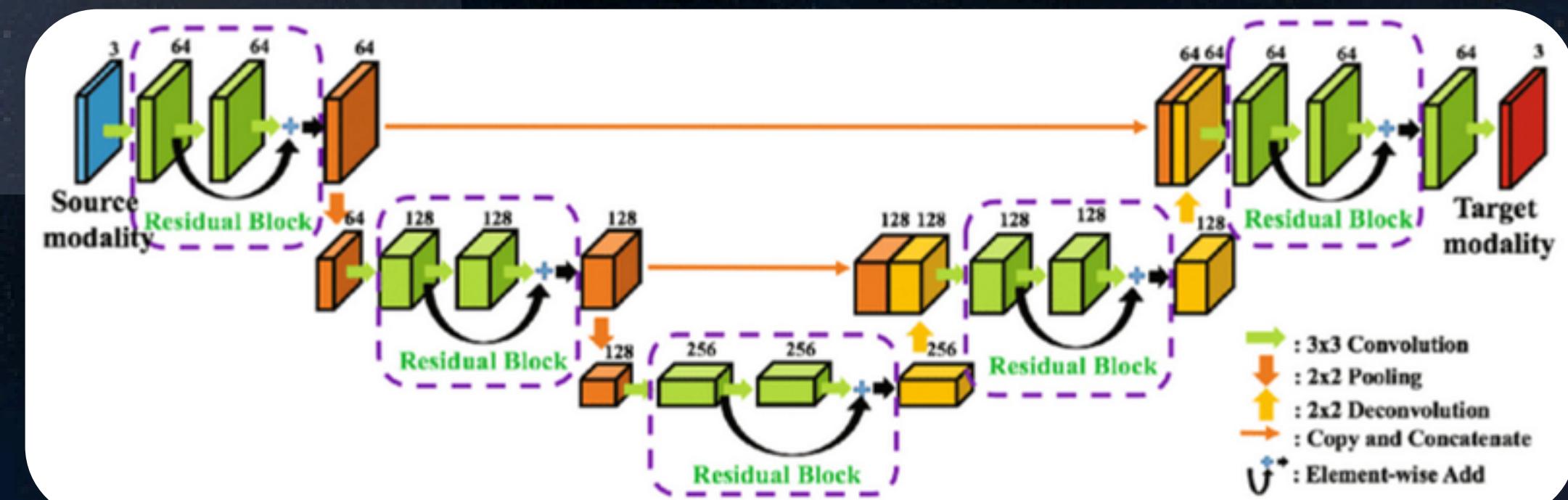
Architettura base

- Modello per segmentazione e ricostruzione di immagini
- Architettura a "U":
 - Encoder
 - Decoder
- Connessioni di Skip
- Vantaggi:
 - Alta precisione nella segmentazione
 - Efficiente anche con pochi dati



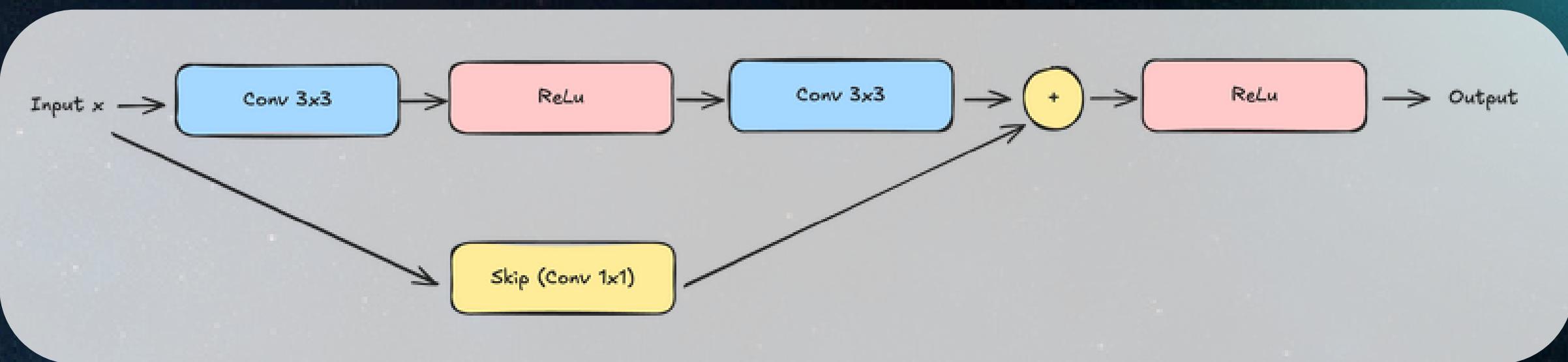
UNET RESIDUALE

- Unisce la struttura encoder-decoder di U-Net con l'apprendimento dei residui di ResNet.
- Connessioni Residuali
- Vantaggi:
 - Maggiore efficienza
 - Convergenza più rapida grazie alle connessioni residue.
 - Miglioramento delle prestazioni



UNET RESIDUALE

Implementazione – Blocco residuale



- **Blocco base del modello:** due convoluzioni 3×3 con ReLU.
- Somma con uno **skip connection** per facilitare il flusso del gradiente

UNET RESIDUALE

Implementazione - Encoder e Bottleneck

```
class UNet(nn.Module):
    """
    UNet con blocchi residui e skip esterni additivi (shortcut connections).
    Architettura: Encoder → Bottleneck → Decoder → Output.
    """
    def __init__(self, in_channels=3, out_channels=3):
        super().__init__()

        # Encoder
        self.enc1 = ResidualUNetBlock(in_channels, 64)
        self.pool1 = nn.MaxPool2d(2)

        self.enc2 = ResidualUNetBlock(64, 128)
        self.pool2 = nn.MaxPool2d(2)

        # Bottleneck
        self.bottleneck = ResidualUNetBlock(128, 256)
```

- **Encoder** estrae feature a bassa risoluzione riducendo progressivamente le dimensioni spaziali
- **MaxPool** dimezza risoluzione dopo ogni blocco
- **Bottleneck** è il punto più profondo della rete, con feature ricche e compatte

UNET RESIDUALE

Implementazione - Decoder e Skip Connections

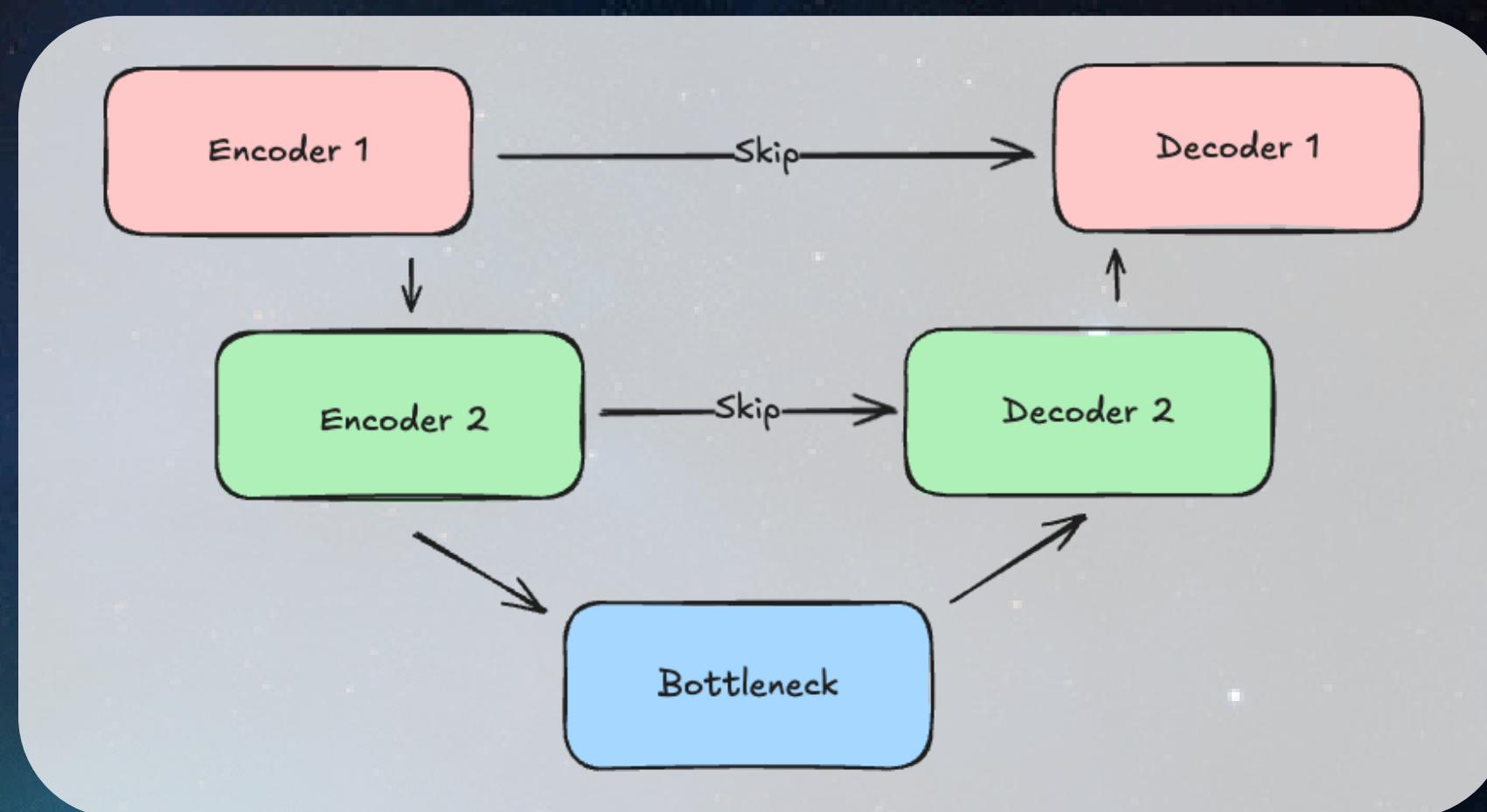
- **Decoder** ricostruisce l'immagine aumentando la risoluzione tramite upsampling.
- Le **skip connections**: aiutano a recuperare dettagli persi nel downsampling.

```
# Decoder
self.up2 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)
self.dec2 = ResidualUNetBlock(256, 128)
self.skip2 = nn.Conv2d(128, 128, kernel_size=1)

self.up1 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
self.dec1 = ResidualUNetBlock(128, 64)
self.skip1 = nn.Conv2d(64, 64, kernel_size=1)
```

UNET RESIDUALE

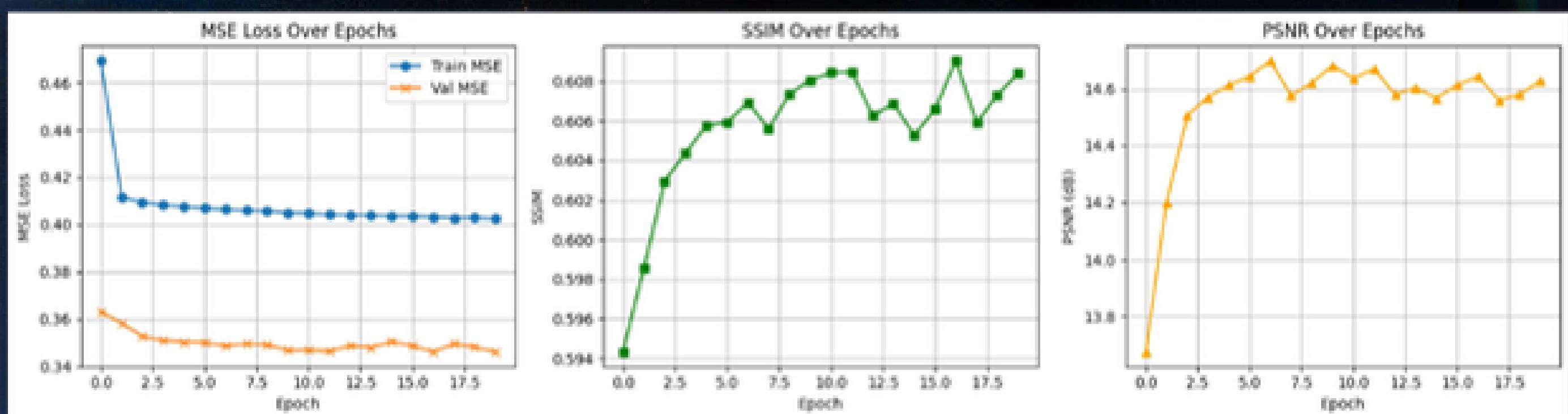
Implementazione - Architettura completa



- L'immagine attraversa **encoder** → **bottleneck** → **decoder**
- Il **layer finale** (Conv 1×1) proietta ai canali desiderati

UNET RESIDUALE

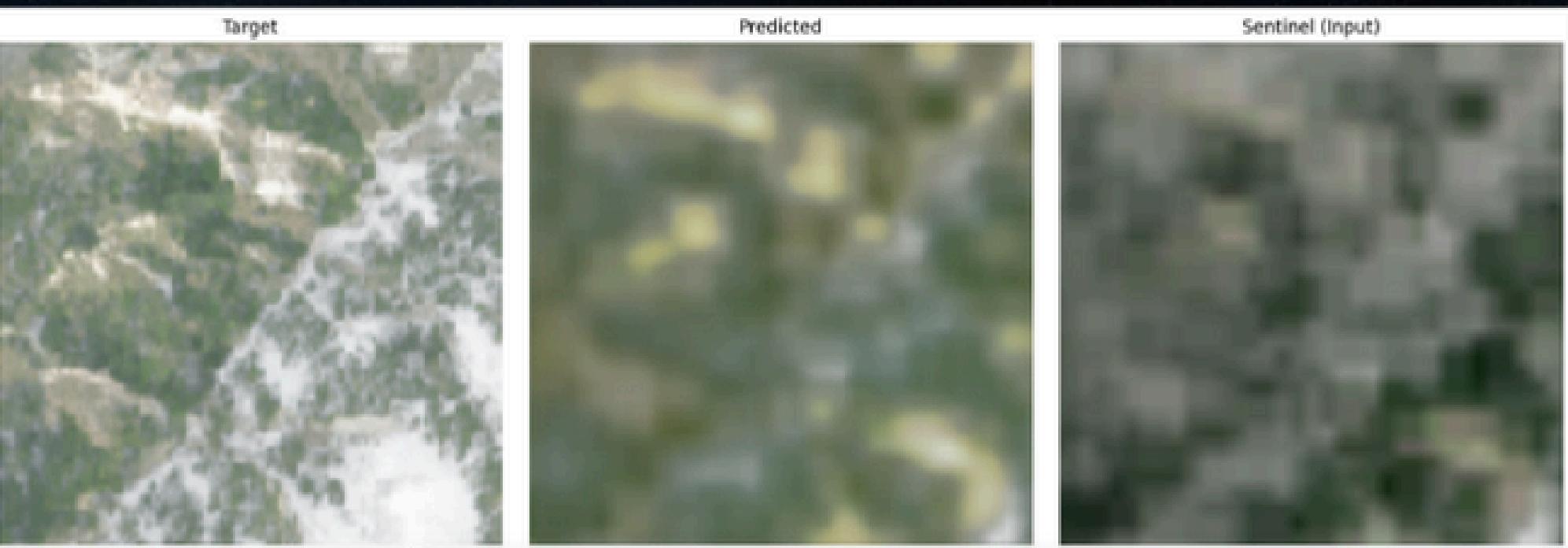
Risultati



Test Loss (MSE): 0.4495 | SSIM: 0.5142 | PSNR: 11.93 dB

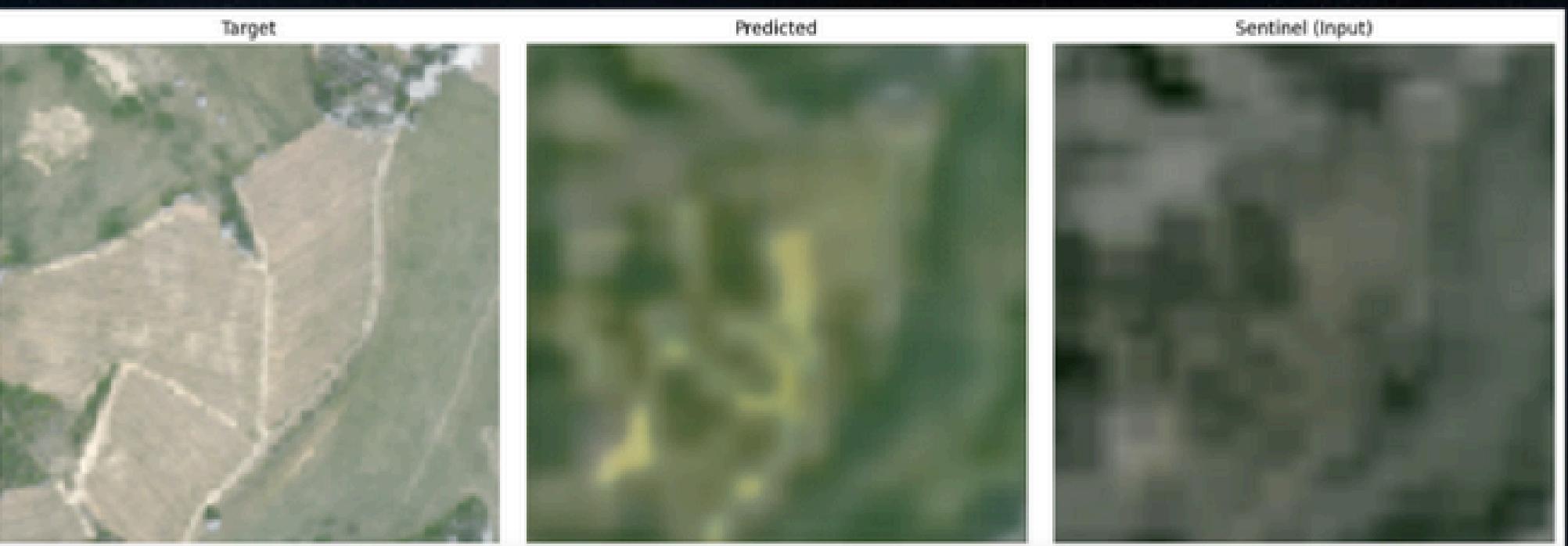
UNET RESIDUALE

Altri output



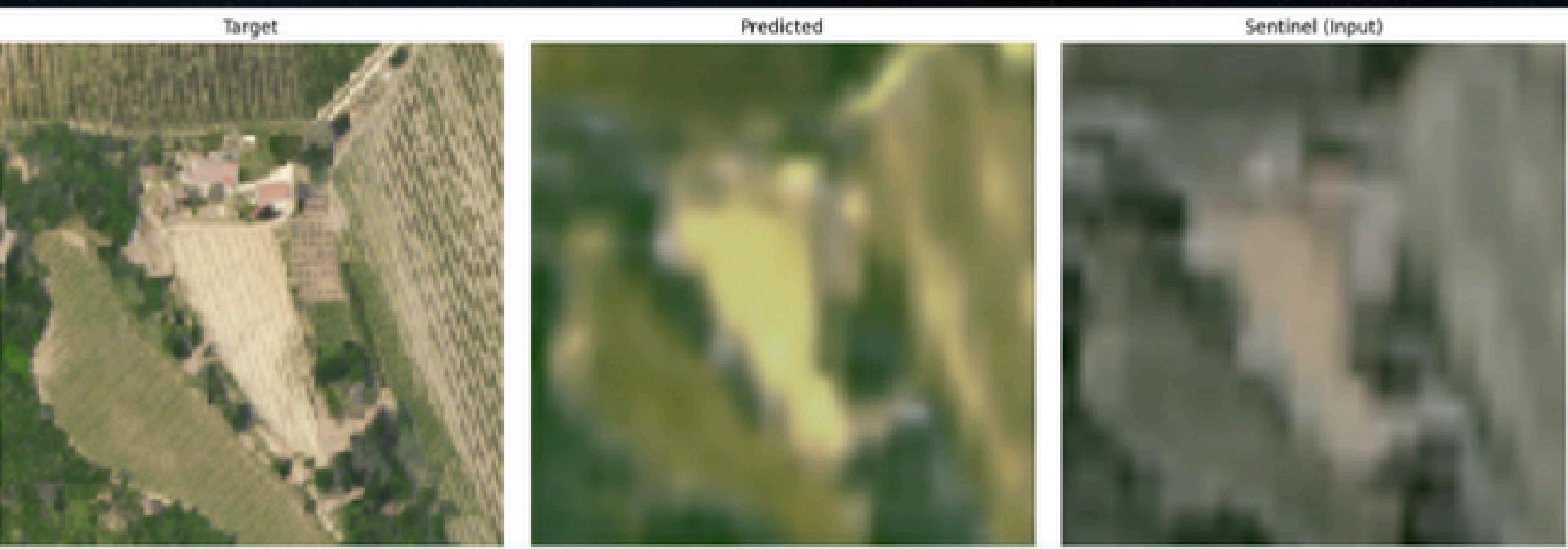
UNET RESIDUALE

Altri output



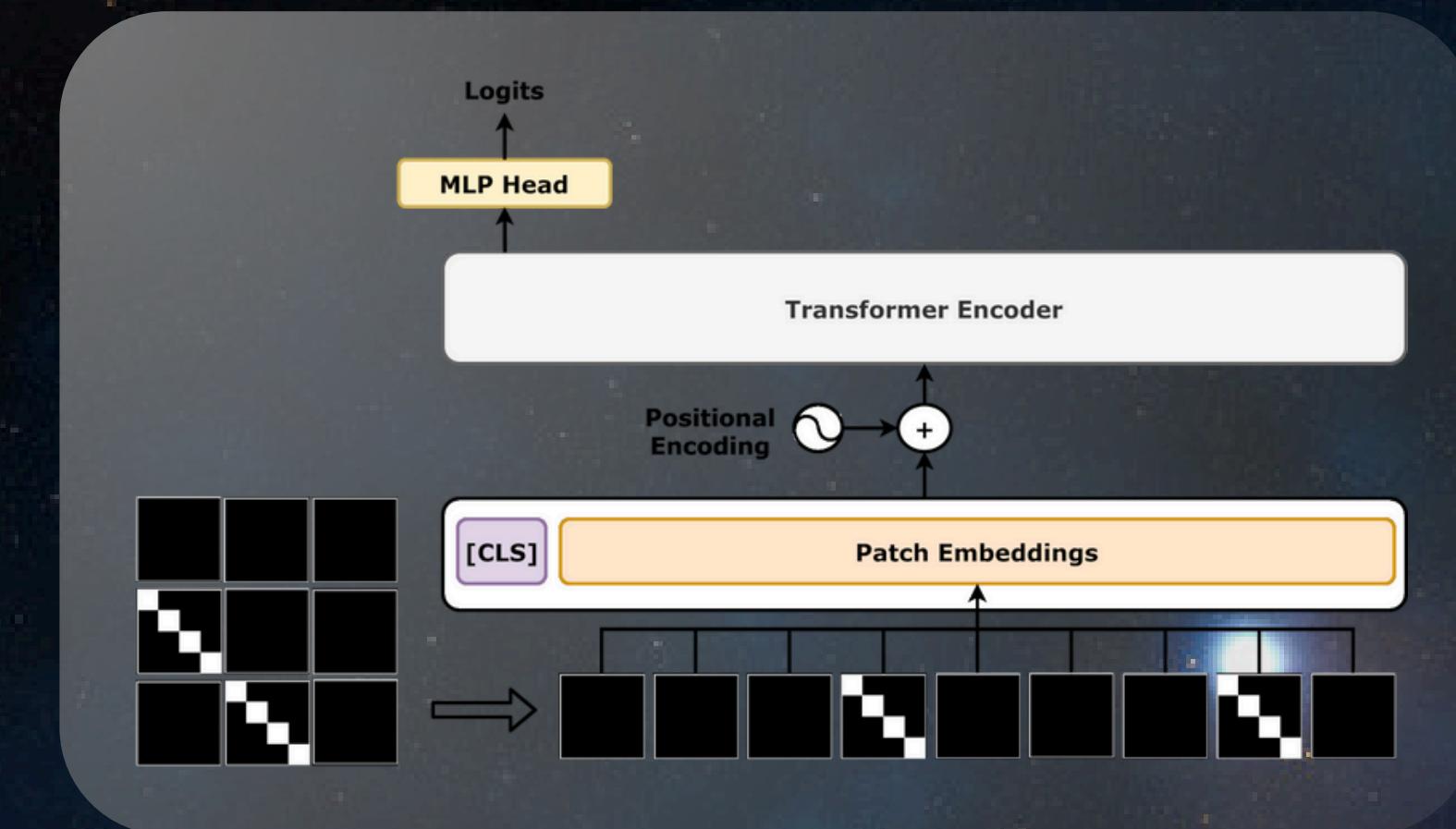
UNET RESIDUALE

Altri output



VIT – VISION TRANSFORMER

- **Suddivisione in patch**
- **Embedding e Posizione**: ogni patch è trasformata in un vettore.
- **Transformer Encoder**: Applica self-attention per catturare relazioni locali e globali.



VIT – VISION TRANSFORMER

Implementazione – Input e Patch embedding

```
class ViTRegressor(nn.Module):
    def __init__(self, in_channels=3, out_channels=3, image_size=128, patch_size=8, dim=512, depth=6, heads=8, mlp_dim=1024):
        super().__init__()

        assert image_size % patch_size == 0,
        self.patch_size = patch_size
        self.image_size = image_size
        num_patches = (image_size // patch_size) ** 2
        patch_dim = in_channels * patch_size * patch_size

        # Embedding: usa Conv2d per estrarre i patch embeddings
        self.proj = nn.Conv2d(in_channels, dim, kernel_size=patch_size, stride=patch_size)
```

- L'immagine è suddivisa in **patch 8×8**
- Ogni patch viene trasformata in un vettore tramite **convoluzione**
- Otteniamo una sequenza di **patch embeddings**

VIT – VISION TRANSFORMER

Implementazione – Positional Embedding

```
# Positional embeddings
self.pos_embedding = nn.Parameter(torch.randn(1, num_patches, dim))
```

- I transformer non hanno nozione di posizione spaziale
- Si aggiungono embedding posizionali appresi a ciascun patch embedding

VIT – VISION TRANSFORMER

Implementazione – Transformer Encoder

```
# Transformer Encoder
self.transformer = nn.TransformerEncoder(
    nn.TransformerEncoderLayer(d_model=dim, nhead=heads, dim_feedforward=mlp_dim, batch_first=True),
    num_layers=depth
)
```

- Sequenza passata a una serie di transformer encoder layers
- Ogni layer usa self-attention per combinare info locali e globali

VIT – VISION TRANSFORMER

Implementazione – Output Projection e Convoluzione

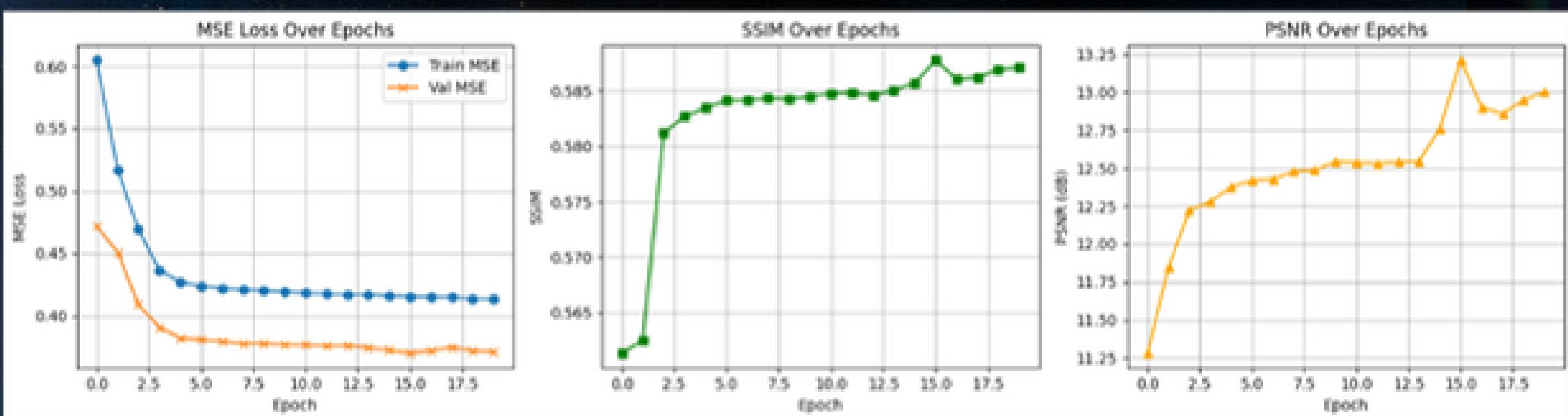
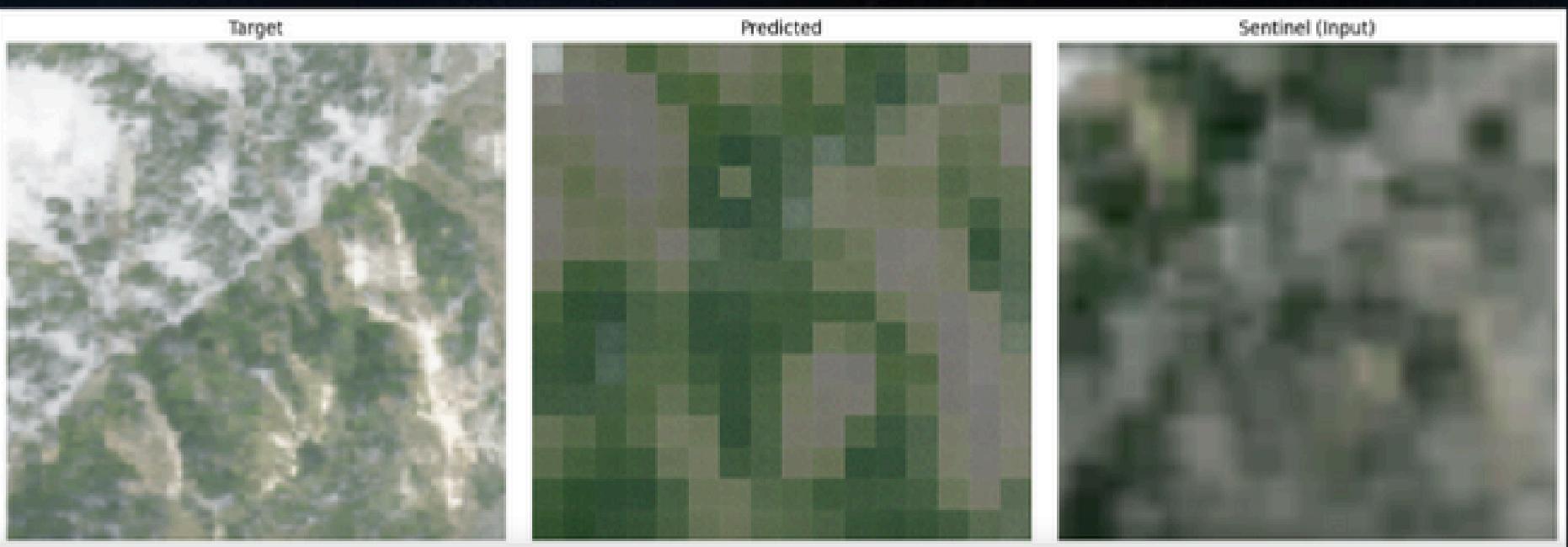
```
# Output projection
self.output_proj = nn.Linear(dim, patch_dim)

# Conv finale per proiettare sui canali finali
self.final_conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)
```

- Ogni embedding viene trasformato di nuovo in patch
- Convoluzione finale per ottenere i canali di output desiderati
- Le patch ricostruite vengono riassemmiate per ricreare l'immagine

VIT – VISION TRANSFORMER

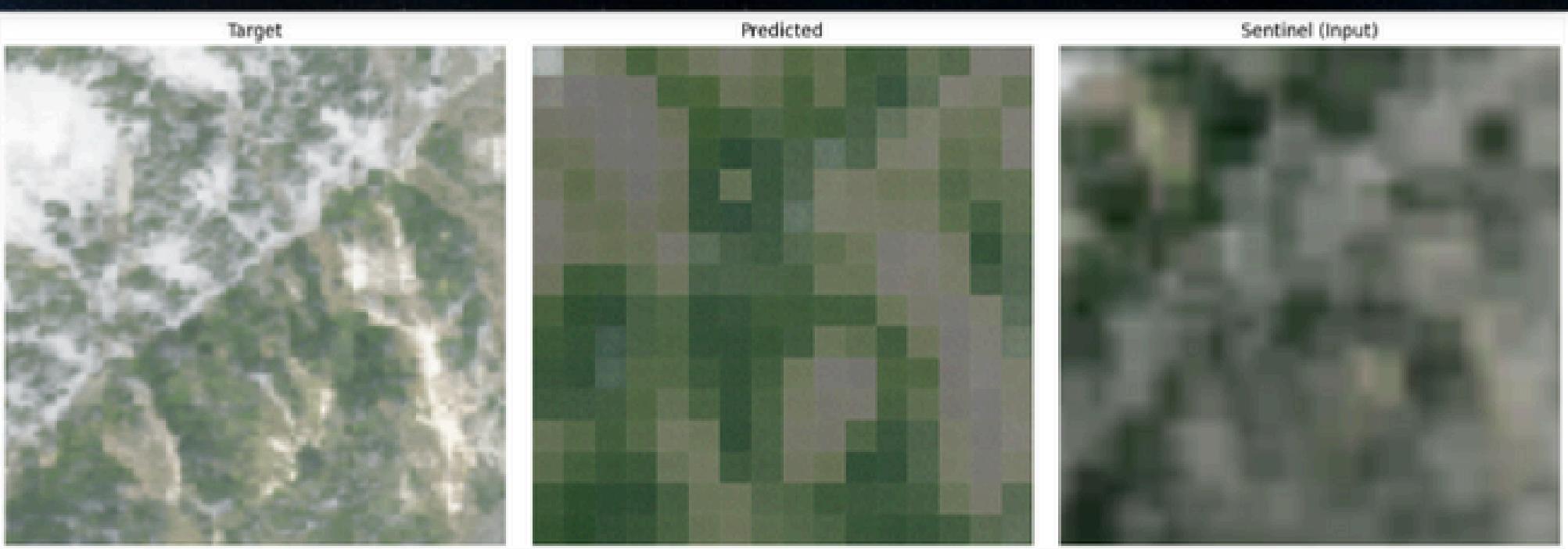
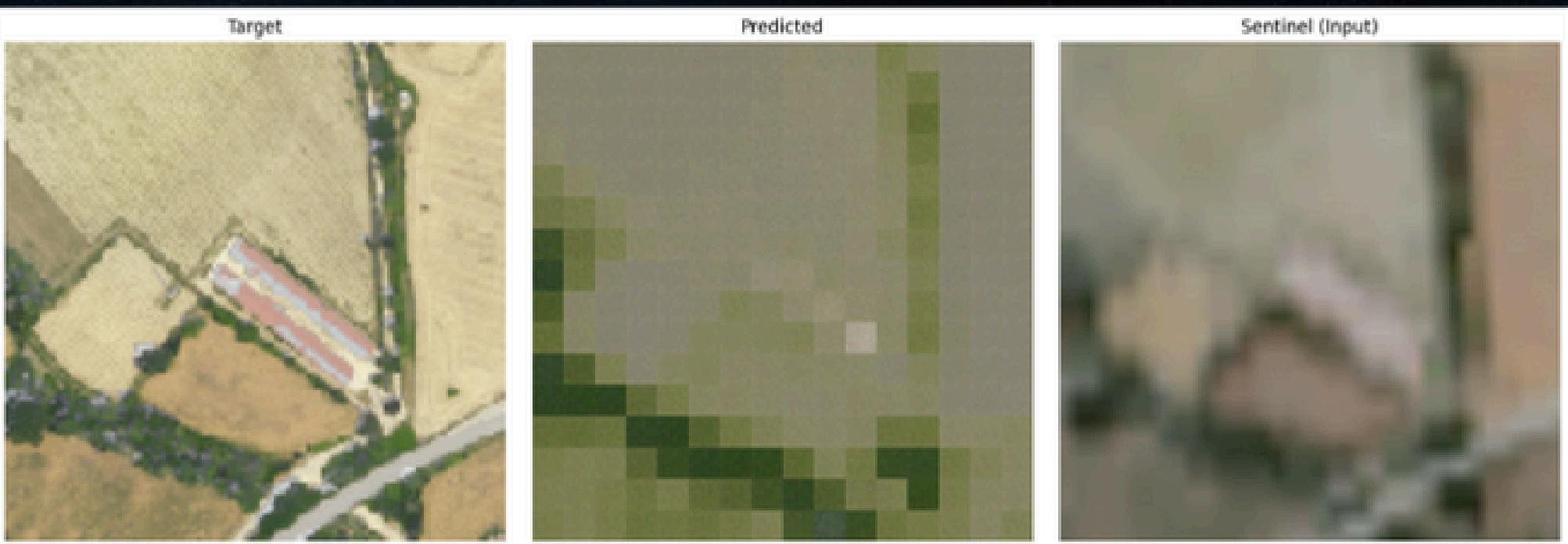
Risultati



Test Loss (MSE): 0.4498 | SSIM: 0.5154 | PSNR: 11.60 dB

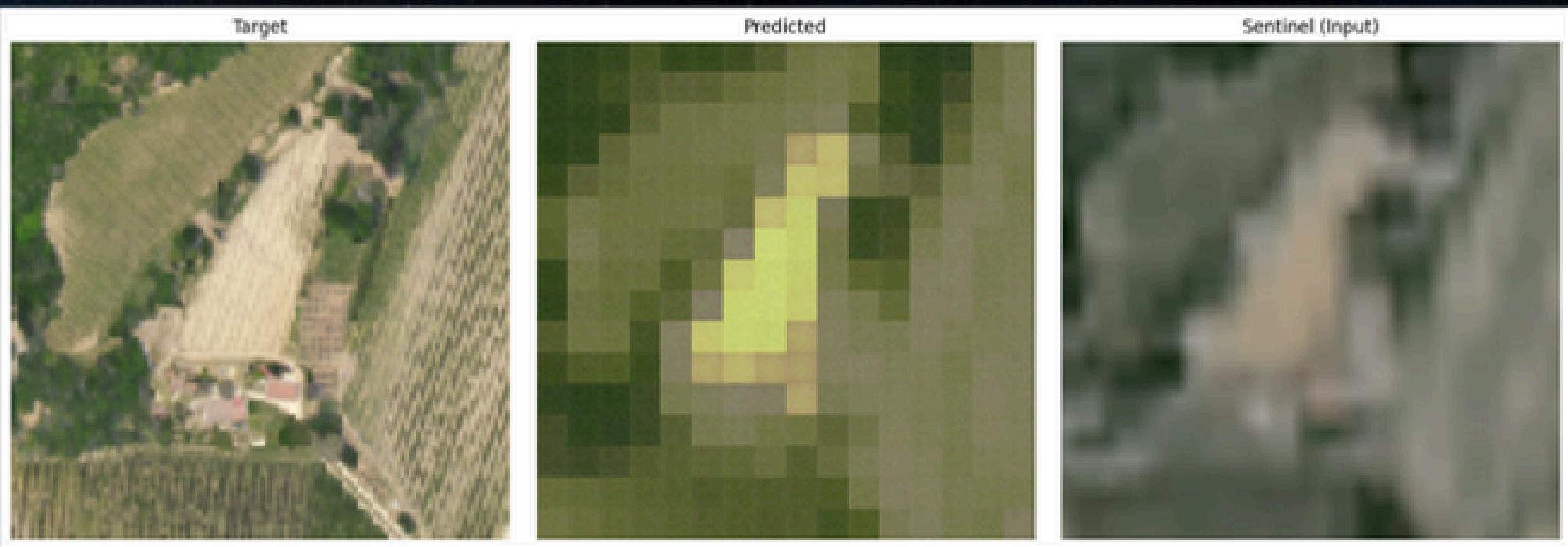
VIT – VISION TRANSFORMER

Altri output



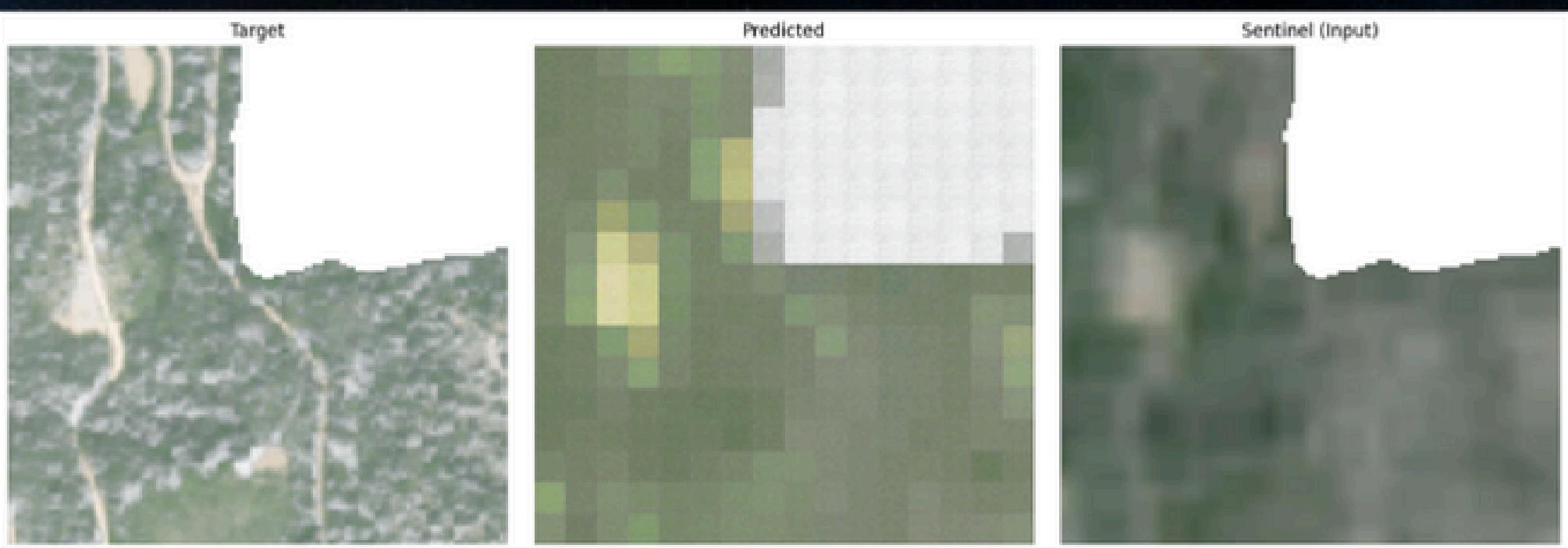
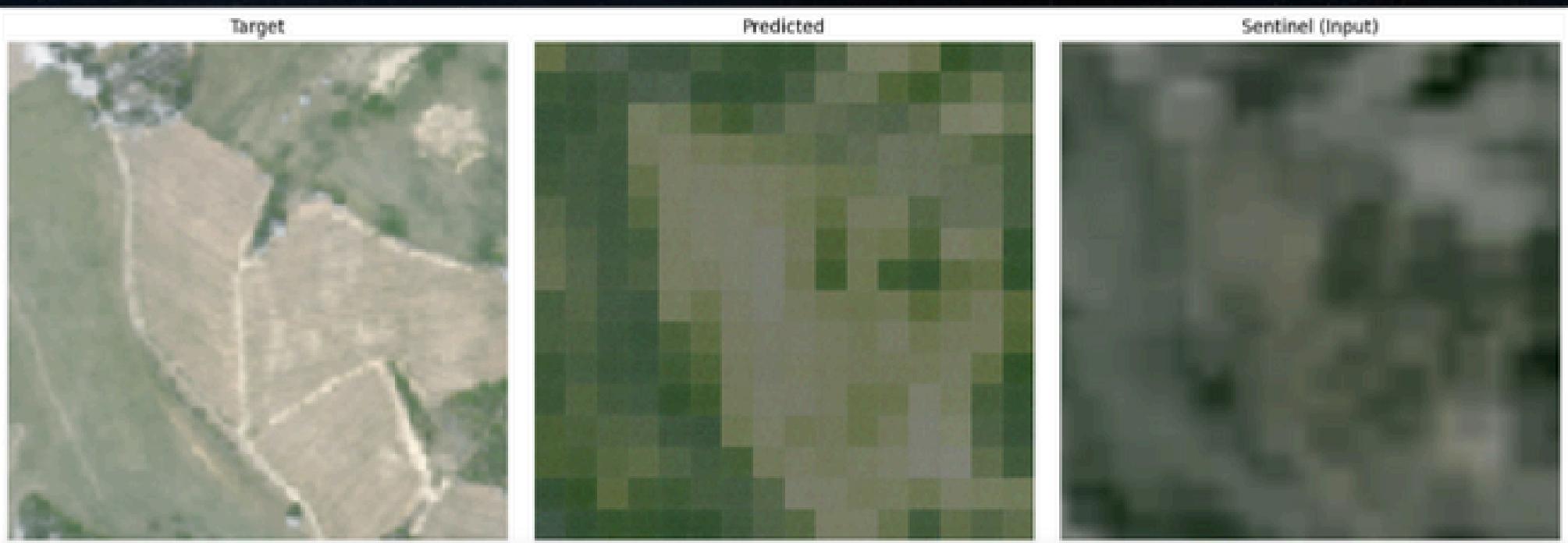
VIT – VISION TRANSFORMER

Altri output



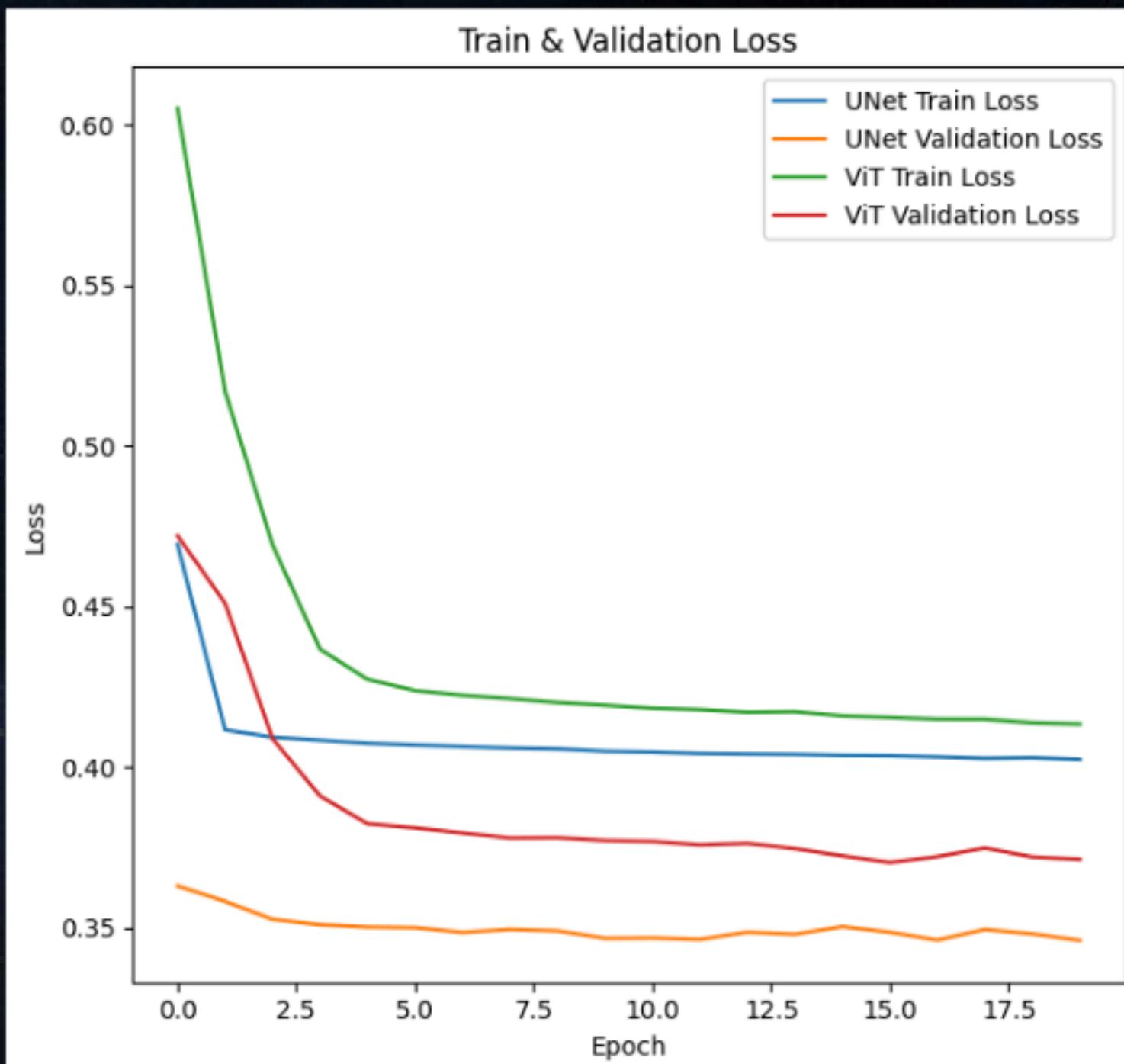
VIT – VISION TRANSFORMER

Altri output



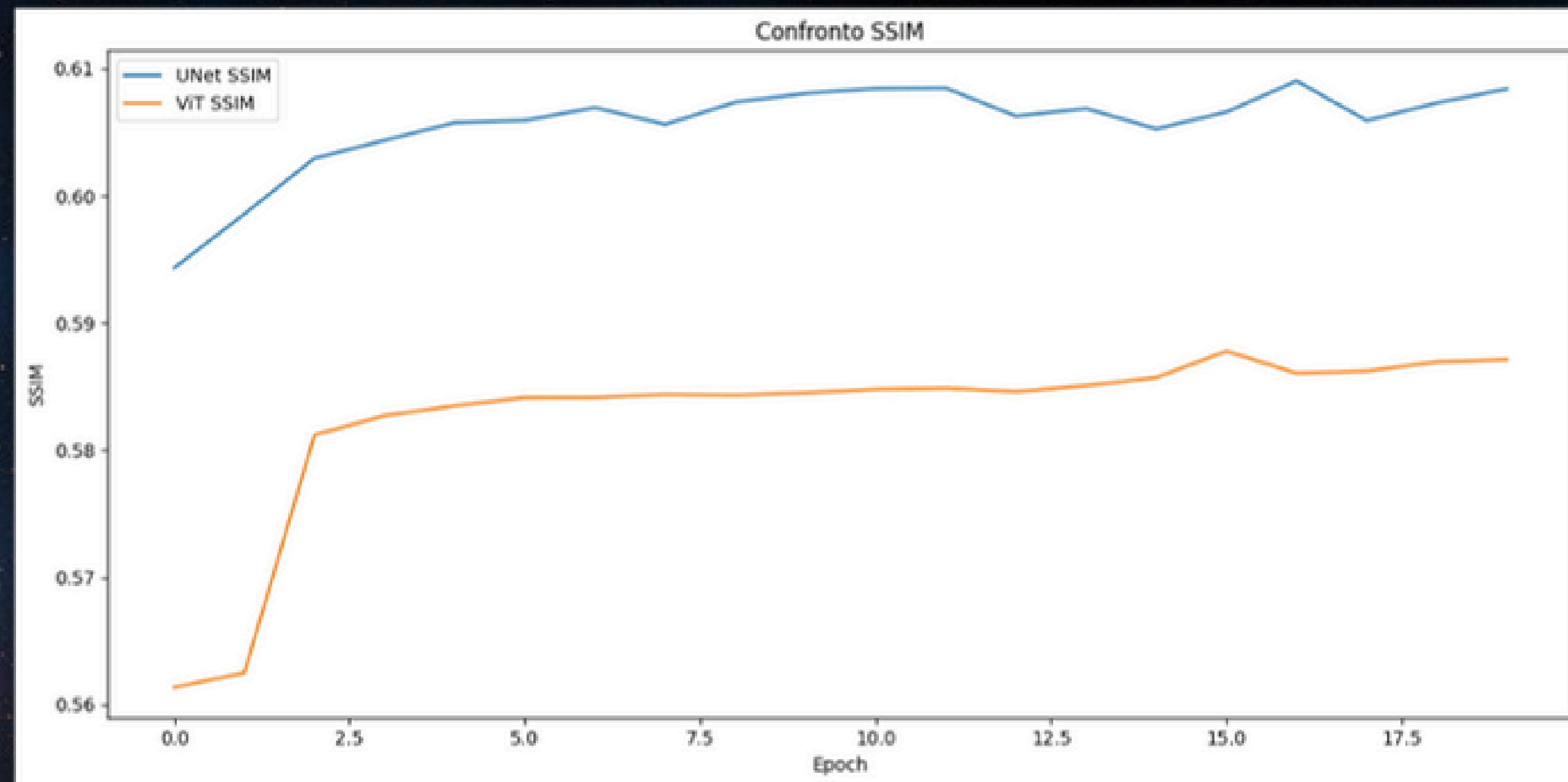
METRICHE FINALI

LOSS a confronto



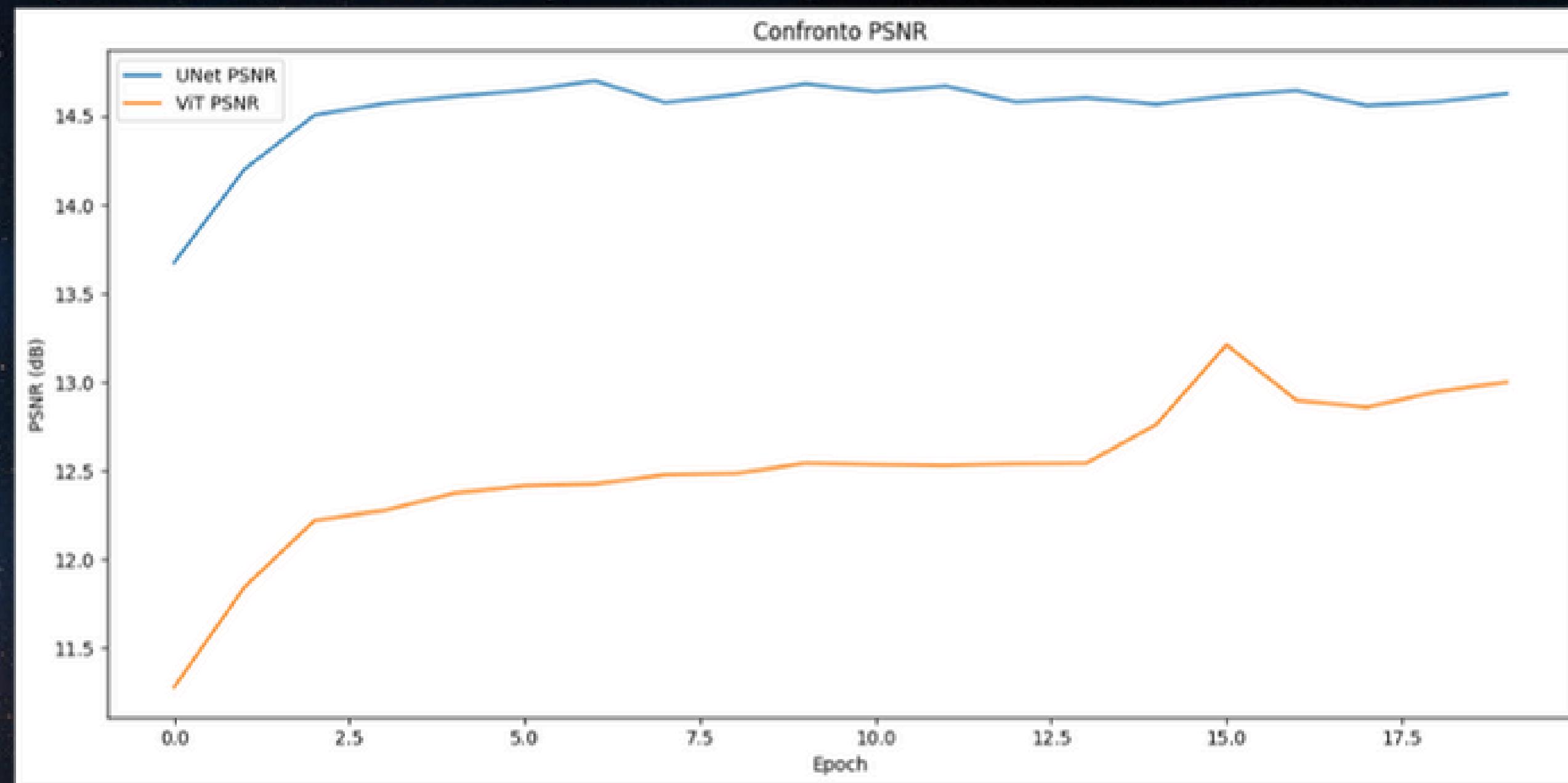
METRICHE FINALI

SSIM a confronto



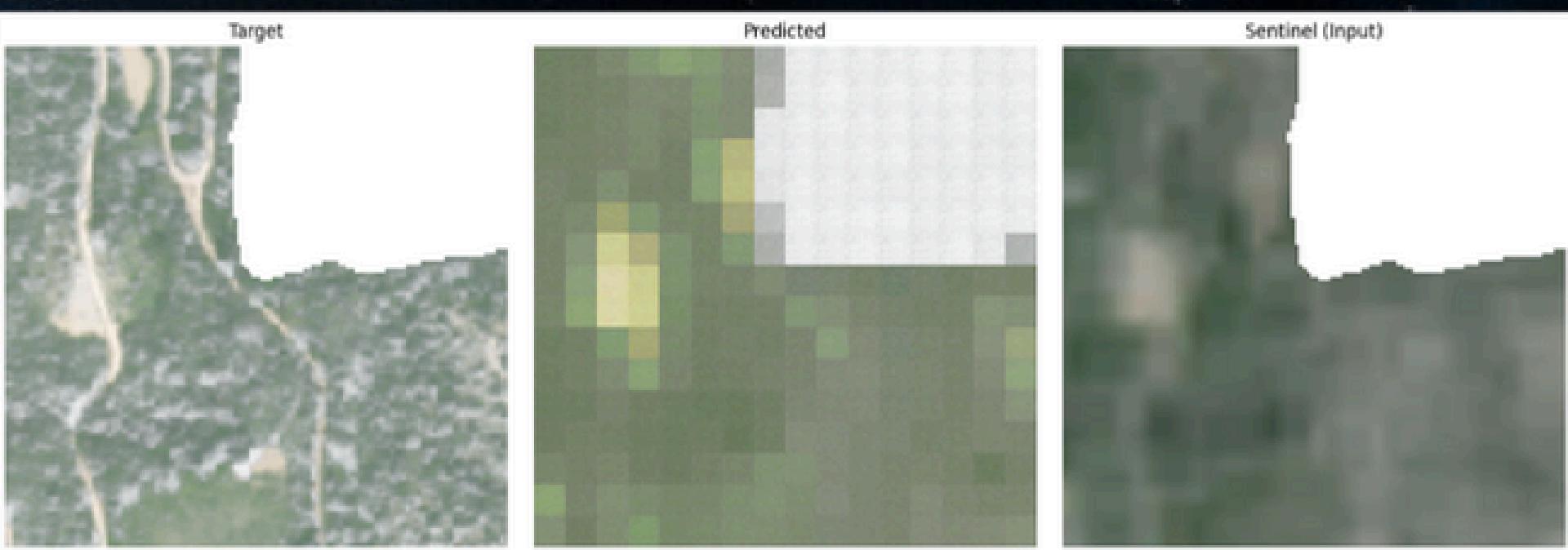
METRICHE FINALI

PSNR a confronto



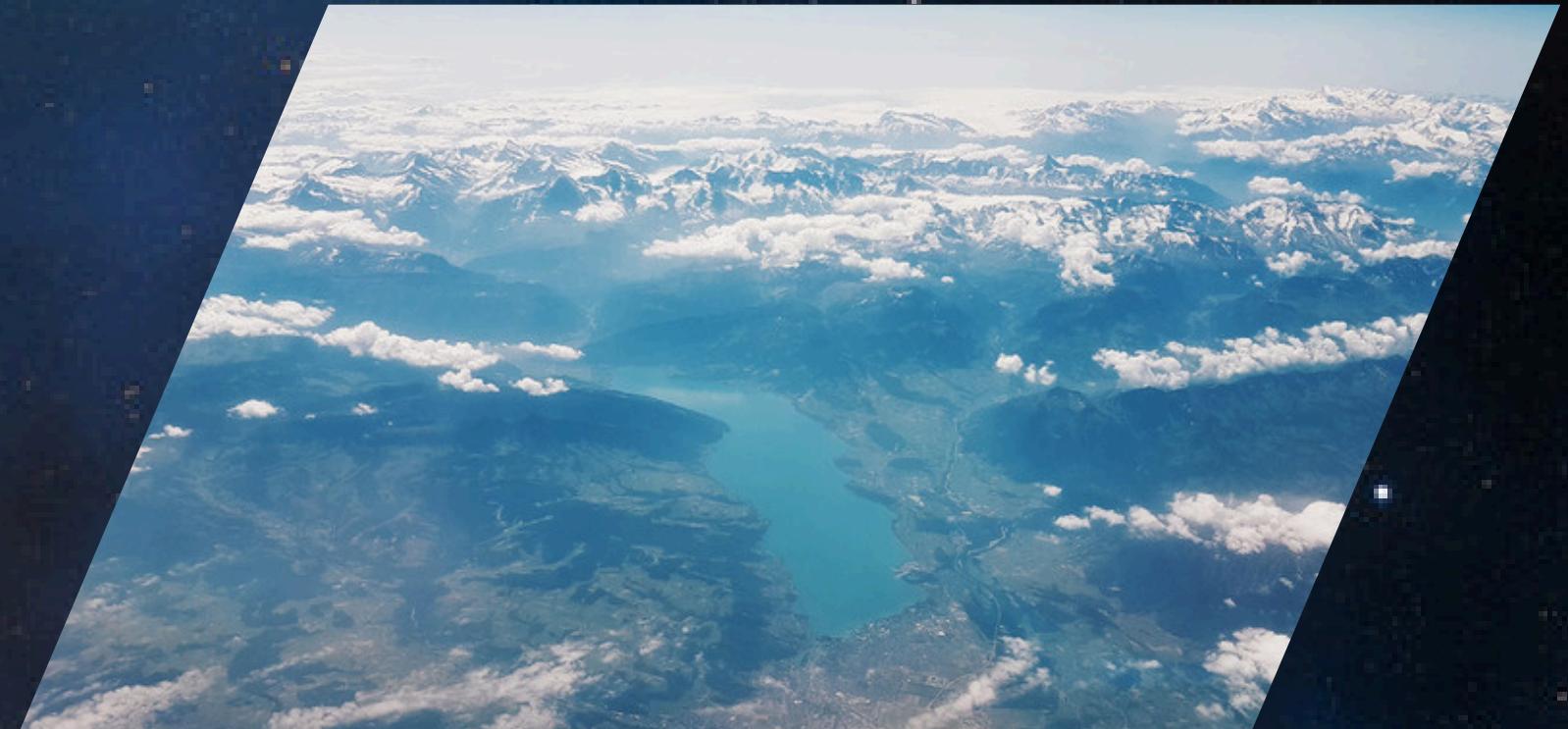
RISULTATI VISIVI

a confronto



CONCLUSIONI

- Risultati non ottimali → dataset ridotto
(data augmentation aiuta questo aspetto)
- UNet visivamente migliore di ViT



Sviluppi futuri

- Dataset migliore, più ampio
- Utilizzo di modelli ibridi (UNet - ViT)





**GRAZIE
DELL'ATTENZIONE**

BIBLIOGRAFIA

- [1] A. Sharifi and M. M. Safari, "Enhancing the Spatial Resolution of Sentinel-2 Images Through Super-Resolution Using Transformer-Based Deep-Learning Models," in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 18, pp. 4805–4820, 2025, doi: 10.1109/JSTARS.2025.3526260.
- [2] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *arXiv preprint arXiv:1505.04597*, May 2015. [Online]. Available: <https://arxiv.org/abs/1505.04597>
- [3] Z. Zhang, Q. Liu and Y. Wang, "Road Extraction by Deep Residual U-Net," in *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, May 2018, doi: 10.1109/LGRS.2018.2802944.
- [4] Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv* 2020, arXiv:2010.11929.