Meteorological Super-Resolution vzzs Wind Representations

Gruppo 21
Marzia De Maina, Matteo Galiazzo, Federica Santisi

Alma Mater Studiorum - Università di Bologna

Dipartimento di Informatica - Scienza e Ingegneria (DISI)

Prof. **Fabio Merizzi**

Purpose

The purpose of the project is to study the impact of different wind representations in the context of super-resolution.

Problem explanation and theory recap

[1]

Datasets and Preprocessing

Datasets and Preprocessing

For the purpose of this project, the wind field representations are derived from the ERA5 and VHR–REA datasets.

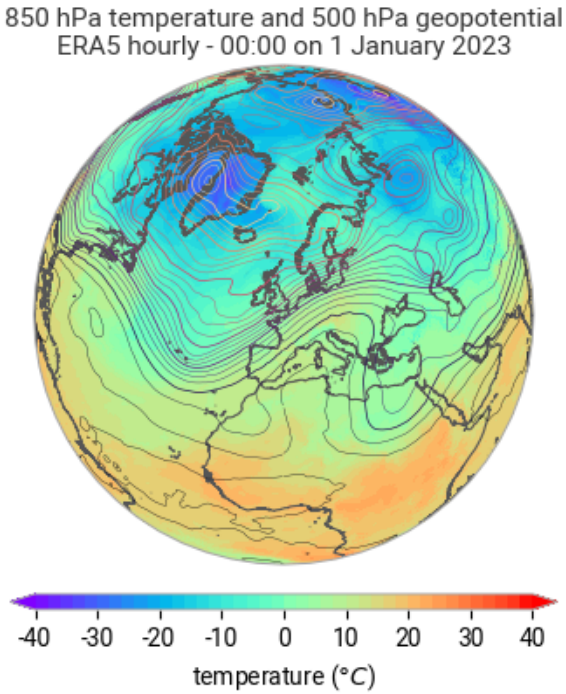
- **ERA5** is a global reanalysis dataset developed by ECMWF, providing hourly data since 1950 at a 0.25° (˜31 km) resolution.
- **VHR–REA** is a downscaled regional reanalysis for Italy based on the COSMO model, with a resolution of 2.2 km.

Both datasets are aligned temporally (06, 12, 18, 00 UTC) and spatially through reprojection.

ERA5 acts as the low-resolution input; VHR–REA is used as the high-resolution target for training.

Datasets and Preprocessing

- **ERA5** is the fifth-generation global reanalysis dataset produced by ECMWF under the Copernicus Climate Change Service (C3S). It replaces the previous ERA-Interim product, offering higher accuracy, finer temporal and spatial resolution (0.25°), and consistent uncertainty estimates.
- Based on a 4D-Var data assimilation scheme, ERA5 integrates a vast range of observational data, including satellite, radiosonde, buoy, and aircraft measurements. It covers the atmosphere, land surface, and ocean waves with hourly updates and is widely used for climate studies, energy forecasting, and risk assessment.
- In this project, we focus on the surface wind fields, specifically the `u10` and `v10` components, which serve as low-resolution inputs for the super-resolution model.



Datasets and Preprocessing

VHR–REA (Very High Resolution ReAnalysis) is a dataset produced by CMCC by dynamically downscaling ERA5 with the COSMO regional model. The dataset has 2.2 km resolution, explicitly resolving convection processes important in complex terrain.
This high-resolution data supports regional applications such as hydrological forecasting, renewable energy planning, and urban climatology.

- Target resolution for training a super-resolution model.
- Data span: one full year, 4 times daily over Northern Italy.

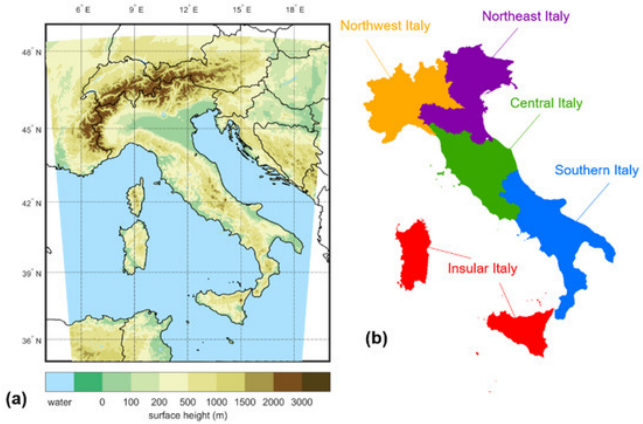


Figura: *

VHR–REA spatial coverage over Northern Italy

Dataset and Preprocessing

Spatial Alignment & Patch Extraction

- Center crop to 224×224 pixel patches
- Temporal synchronization verification
- Channel-wise data stacking (u10, v10 components)
- Custom PyTorch Dataset implementation

Data Pipeline Architecture

- ItalyWeatherDataset class for paired data loading
- Automatic temporal alignment checking
- Flexible normalizer integration
- Memory-efficient batch processing

Quality Assurance

- Index bounds verification
- Consistent tensor dimensions
- Error handling for misaligned datasets

```
def extract_region(self, data_tensor):
    patch_h, patch_w = (224, 224)
    _, h, w = data_tensor.shape
    x = (w - patch_w) // 2
    y = (h - patch_h) // 2
    return data_tensor[:,
                       y:y+patch_h,
                       x:x+patch_w]

def __getitem__(self, idx):
    # Process ERA5 data
    era_sample_slice = self.
           era5_dataset.isel(
        valid_time=idx)
    era_data_arrays = [
        era_sample_slice[var].values
        for var in self.ERA5_VARIABLES
    ]
    era_stacked = np.stack(
           era_data_arrays, axis=0)
    era_tensor = torch.from_numpy(
           era_stacked).float()
    era_tensor = self.extract_region(
           era_tensor)

    # Apply normalization if available
    if self.era5_normalizer is not
           None:
        era_tensor = self.
              era5_normalizer.
              normalize(era_tensor)
```

Normalization & Coordinate Transformation

MinMax Normalization Strategy

- Training-based statistics computation
- Per-channel normalization across spatial dimensions
- Feature range scaling to [0, 1]
- Epsilon handling for division-by-zero cases
- Serializable normalizer objects

Coordinate System Transformation

- **Option 1**: Cartesian coordinates (u, v)
- **Option 2**: Polar coordinates (magnitude, direction)
- $magnitude = \sqrt{u^2 + v^2}$
- $direction = 180 + \frac{180}{\pi} \arctan 2(-u, -v) \bmod 360$

Dataset Management

- 80-20 train-test split with fixed seed (42)
- Separate normalization preservation

```
class MinMaxNormalizer:
    def compute_stats(self, data_tensor):
        # Min/max per channel across (N,H,W)
              dims
        self.min_val = data_tensor.amin(
            dim=(0, 2, 3), keepdim=True)
        self.max_val = data_tensor.amax(
            dim=(0, 2, 3), keepdim=True)

        # Handle division by zero
        diff = self.max_val - self.min_val
        epsilon = 1e-7
        self.max_val[diff < epsilon] = \
            self.min_val[diff < epsilon] +
                  epsilon

    def normalize(self, x):
        return (x - self.min_val) / \
               (self.max_val - self.min_val)

# Coordinate transformation to polar
if COORDINATES == "1":
    u_squared = tensor[0, :, :]**2
    v_squared = tensor[1, :, :]**2
    magnitude = torch.sqrt(u_squared + v_squared
          )
    direction = (180 + (180/math.pi) *
                 torch.atan2(-u_squared, -
                       v_squared)) % 360
    tensor = torch.stack([magnitude, direction],
           axis=0)
```

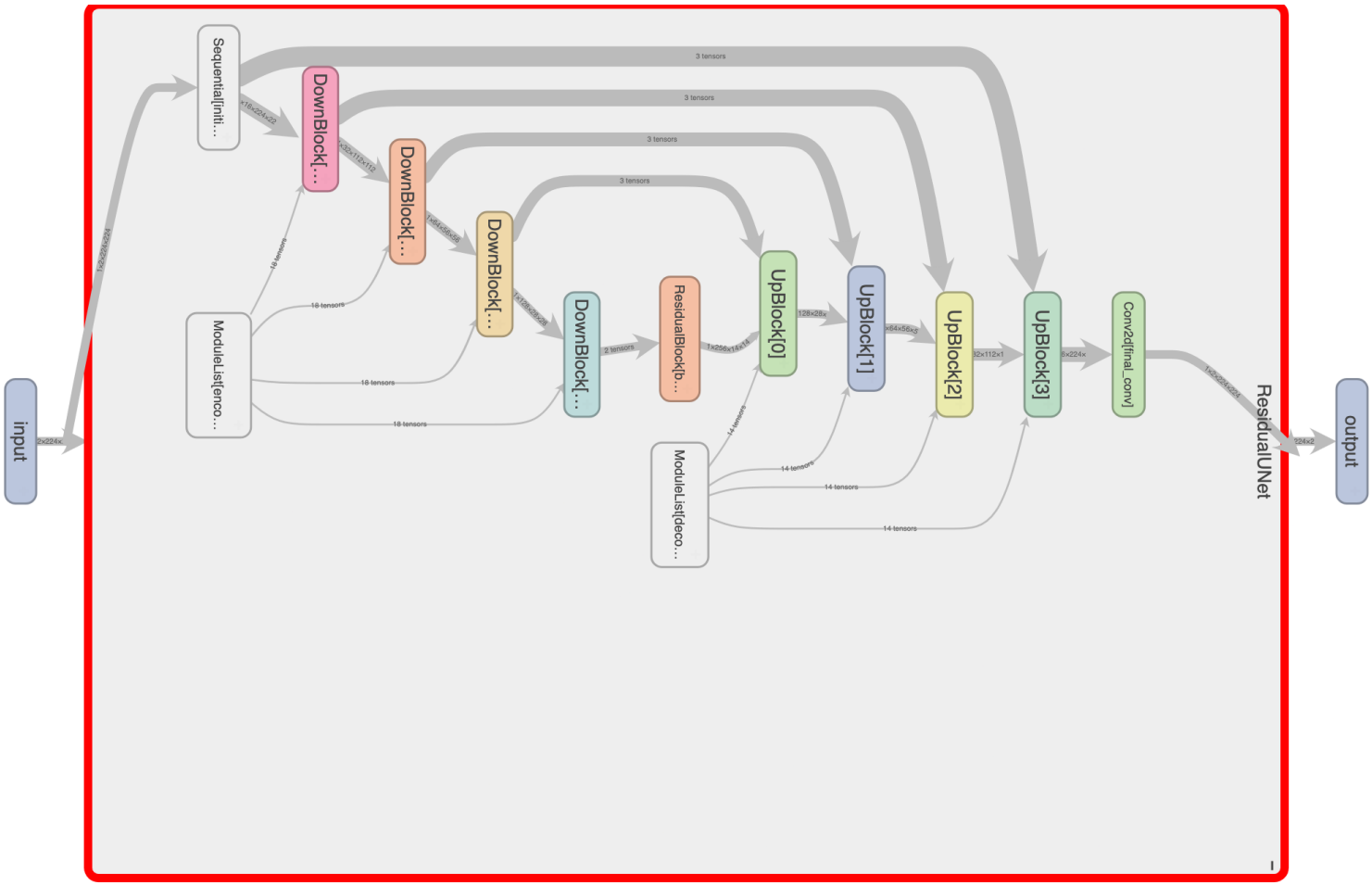
Model and hyperparameters - Introduction

Our model is based on the U-Net architecture, extended with residual connections for better gradient flow and stability. The input has 2 channels (u, v) and the output also has the same 2 channels representing the high-resolution version of the same fields. The key building blocks are:

- **Residual Block**: it's the core module of the architecture. It has two paths
  - One that applies convolutions and normalization.
  - One that does nothing (the residual application).

  The model learns how much to use either path.
- **Downsampling Block**: reduces the spatial resolution and increases the feature size. It uses stride 2 to downsample and includes a residual block.
- **Upsampling Block**: these are the mirror of down blocks. They use bilinear upsampling to increase the spatial dimension instead of transposed convolutions. Then, we concatenate the skip connections from the encoder and use a 1×1 convolution to reduce the number of channels after concatenations.

Model and hyperparameters - ResUNet



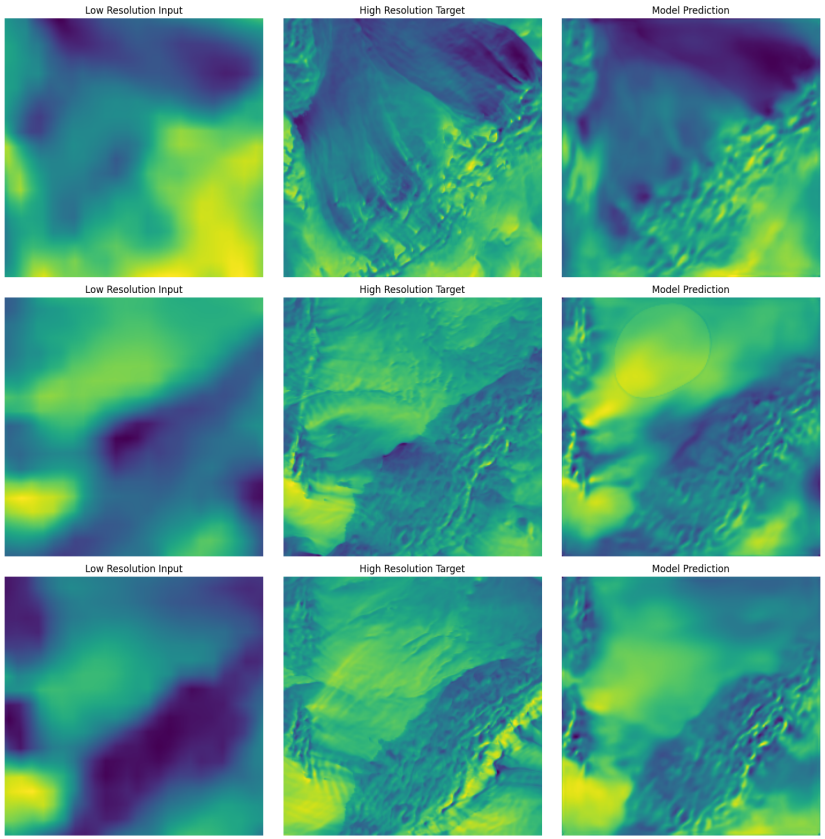
Model and hyperparameters - Transposed vs Bilinear

Network Type	Parameters Count	Training Time	Test Loss	Test SSIM
Transposed Conv	13,041,922 (13.0 M)	54:03 16.22s/it	0.197755	0.709096
Bilinear Interp	3,607,586 (3.6 M)	08:04 2.42s/it	0.171656	0.749828

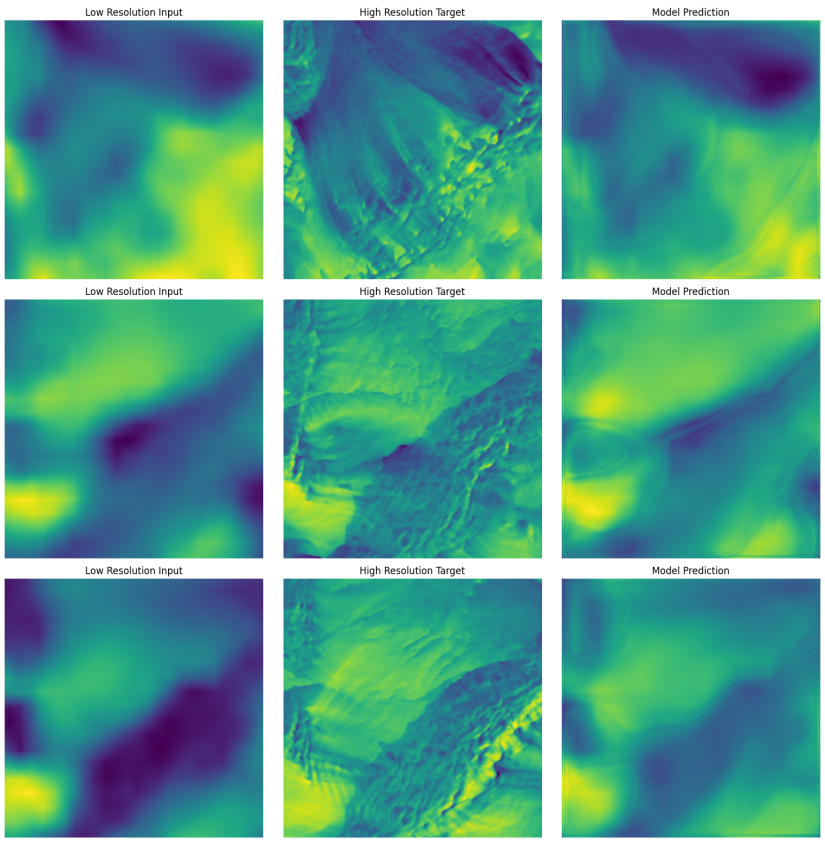
Tabella: Comparison of bilinear interpolation vs transposed convoltuion.

Model and hyperparameters - Loss

Even the best model couldn't get decent results with the suggested MSE loss, so we adopted a combined loss approach



L1 + SSIM loss. SSIM: 0.7556



MSE loss. SSIM: 0.7022

Model and hyperparameters - Loss

```
class L1SSIMLoss(nn.Module):
    def __init__(self, alpha=0.85, ssim_window_size=11, ssim_data_range=1.0,
         ssim_channel=1):
        super(L1SSIMLoss, self).__init__()
        self.alpha = alpha
        self.l1_loss = nn.L1Loss() # Mean Absolute Error
        self.ssim_loss_fn = SSIMLoss(window_size=ssim_window_size, data_range=
             ssim_data_range, channel=ssim_channel)

    def forward(self, y_pred, y_true):
        ssim_val_loss = self.ssim_loss_fn(y_pred, y_true)
        l1_val_loss = self.l1_loss(y_pred, y_true)

        combined_loss = self.alpha * ssim_val_loss + (1 - self.alpha) *
             l1_val_loss
        return combined_loss
```

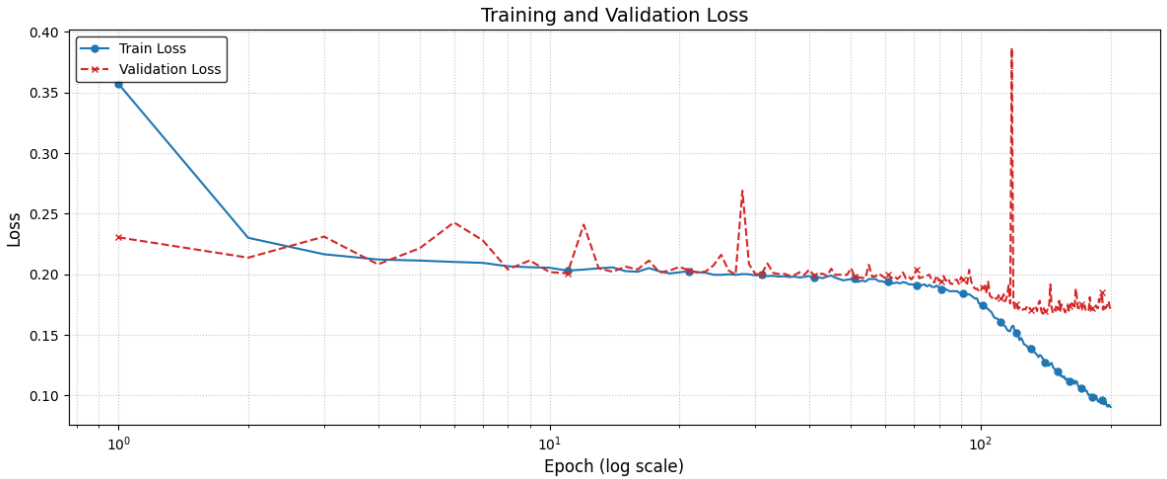
Model and hyperparameters - Hyperparameters

Channels	Epochs	Loss	Batch Size	LR	LR Scheduler	Weight Decay	Test Loss
[32, 64, 128]	200	L1 + SSIM	8	1.00E-03	NO	1.00E-05	0.19090
[16, 32, 64, 128]	200	L1 + SSIM	8	1.00E-03	NO	1.00E-05	0.17700
[16, 32, 64, 128, 256]	300	L1 + SSIM	8	1.00E-03	NO	1.00E-05	**0.16768**
[16, 32, 64, 128, 256]	300	MSE + SSIM	8	1.00E-03	NO	1.00E-05	<u>0.15464</u>
[16, 32, 64, 128, 256]	300	MSE	8	1.00E-03	NO	1.00E-05	0.39491
[16, 32, 64, 128, 256, 512]	300	L1 + SSIM	8	1.00E-03	NO	1.00E-05	0.19207
[16, 32, 64, 128, 256, 512]	300	L1 + SSIM	8	1.00E-03	YES	1.00E-05	0.19207

Tabella: Performance comparison of different network configurations on the local datasets. Best and runner-up models are highlighted in bold and underline, respectively.

Local Training - Loss and SSIM curves

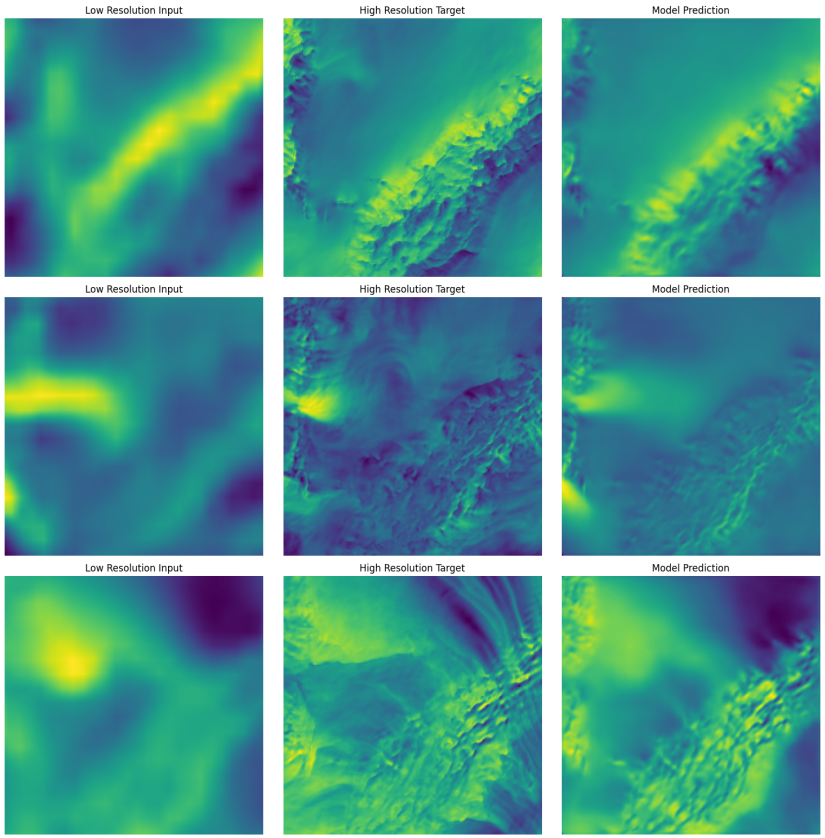
Training curves for the best model on local data (loss and SSIM):



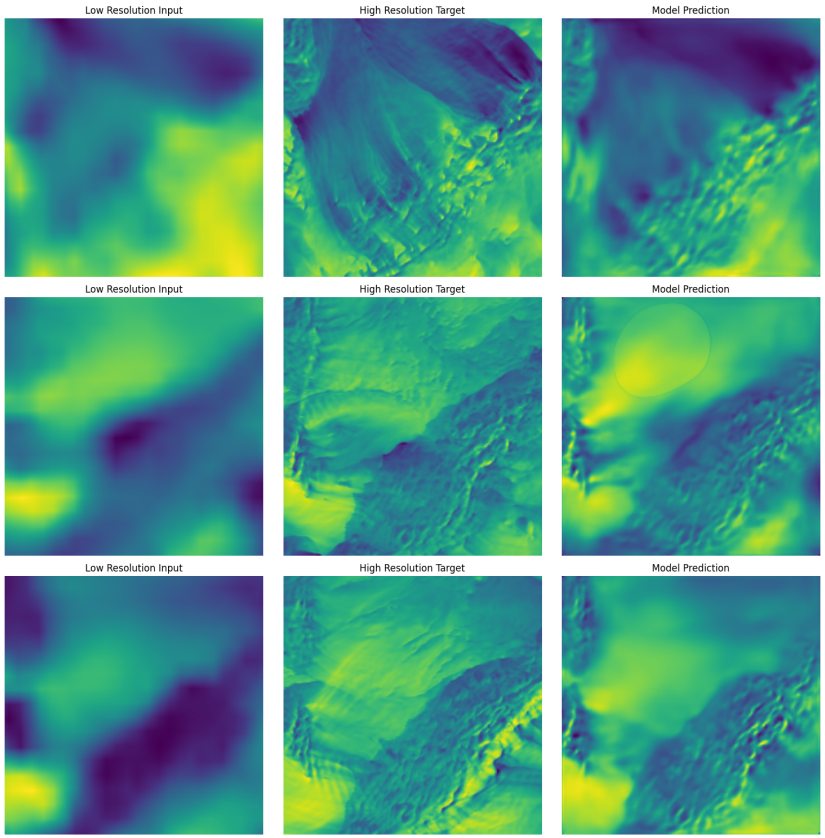
Loss curve

SSIM curve

Colab Training - Difference with Local data



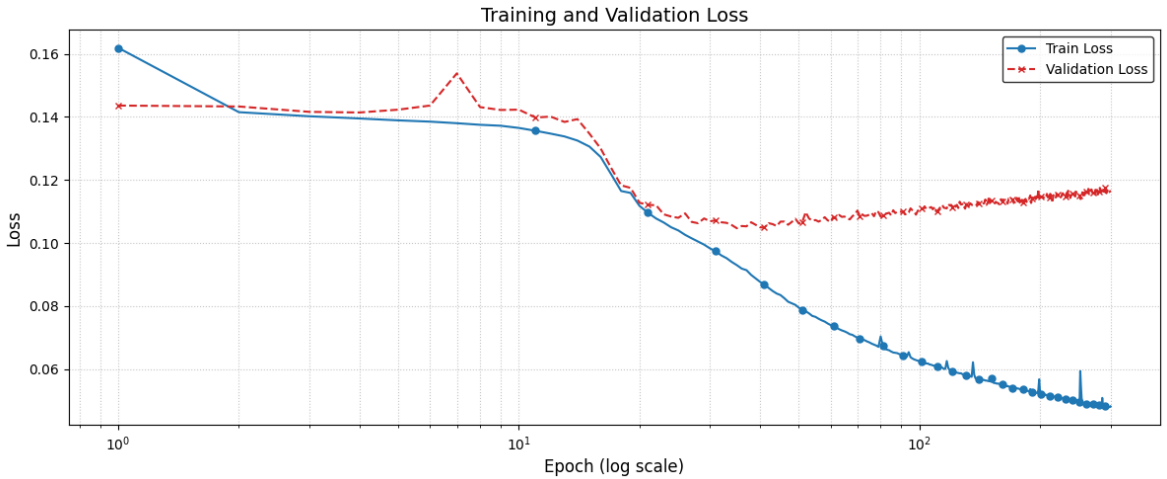
Colab data. SSIM: 0.84483



Local data. SSIM: 0.7556

Colab Training - Loss and SSIM curves

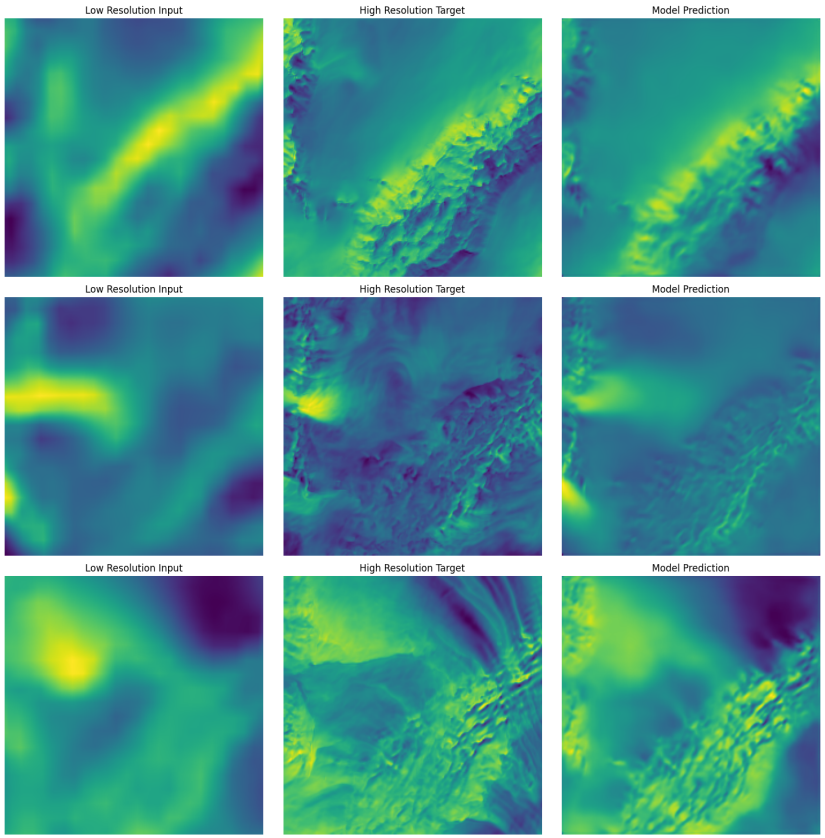
Training curves for the best model on colab data (loss and SSIM):



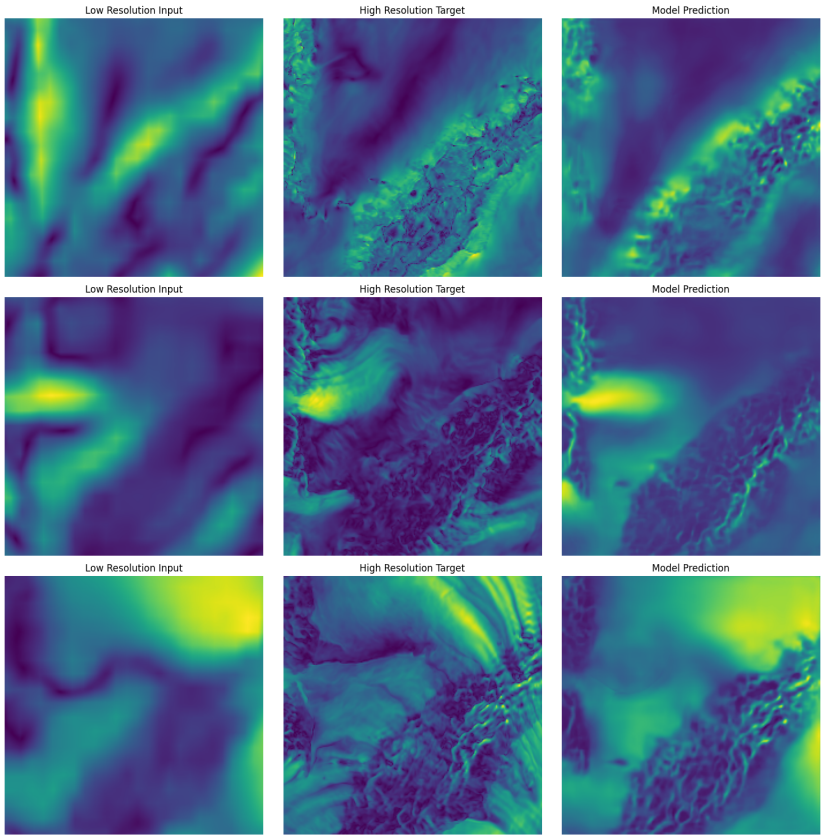
Loss curve

SSIM curve

Colab Training - Vectorial data vs Directional data



Vector data. SSIM: 0.84483



Direction data. SSIM: 0.48154

Results

Final Considerations

References

[1] Fabio Merizzi, Andrea Asperti e Stefano Colamonaco. "Wind speed super-resolution and validation: from ERA5 to CERRA via diffusion models". In: *Neural Computing and Applications* 36.34 (2024), pp. 21899–21921.