Finanziato dall'Unione europea
NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

AstroClass

Cristina Martellini, Cristiano Andolfi, Daniele Bertillo, Tiziano Pagliaroli, Riccardo Panciroli
Fuko Srl

Spoke 3 Progetti Bandi a Cascata, 29/05/ 2025

www.fuko.srl

Team Overview

Company core business

Project Overview

Main Goal:

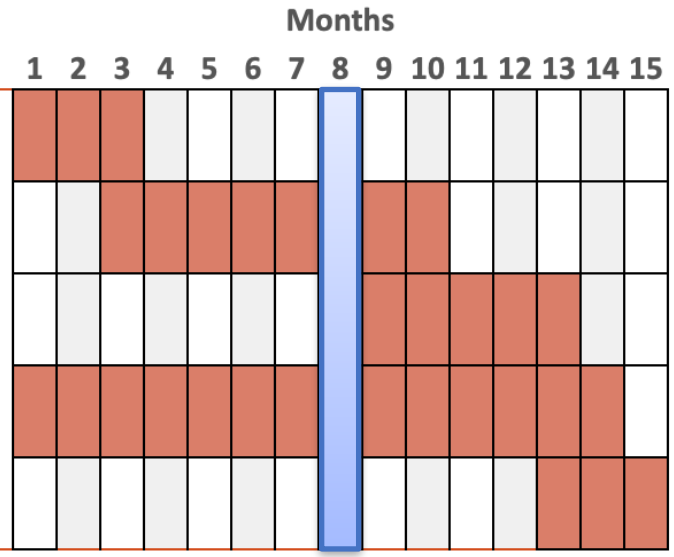
1. Automate extraction of some characteristic features of astrophysical galaxy clusters through ML techniques
2. The project addresses the challenge of extracting meaningful 3D physical profiles from a limited dataset of X-ray and radial distribution of the signal at different frequencies.

- We will present the project's framework
- We will discuss how we handle the reduced size of the dataset and issues coming from it
- State of the art of the project

TimeScale

Project Plan ASTROCLASS

Activity	Start-plan	Duration-plan	Start-effective	Duration-effective	Percentage completion
WP 01: Design SW	1	3	1	3	100%
WP 02: Implementation NN	3	10	1	10	80%
WP 03: Validation NN	9	13	5	13	30%
WP 04: Interchange platform	1	14	7	14	10%
WP 05: Dissemination	13	15	7	15	0%



Main Objectives:

1. The project was reviewed
2. We reviewed the database and aligned the input parameters for our

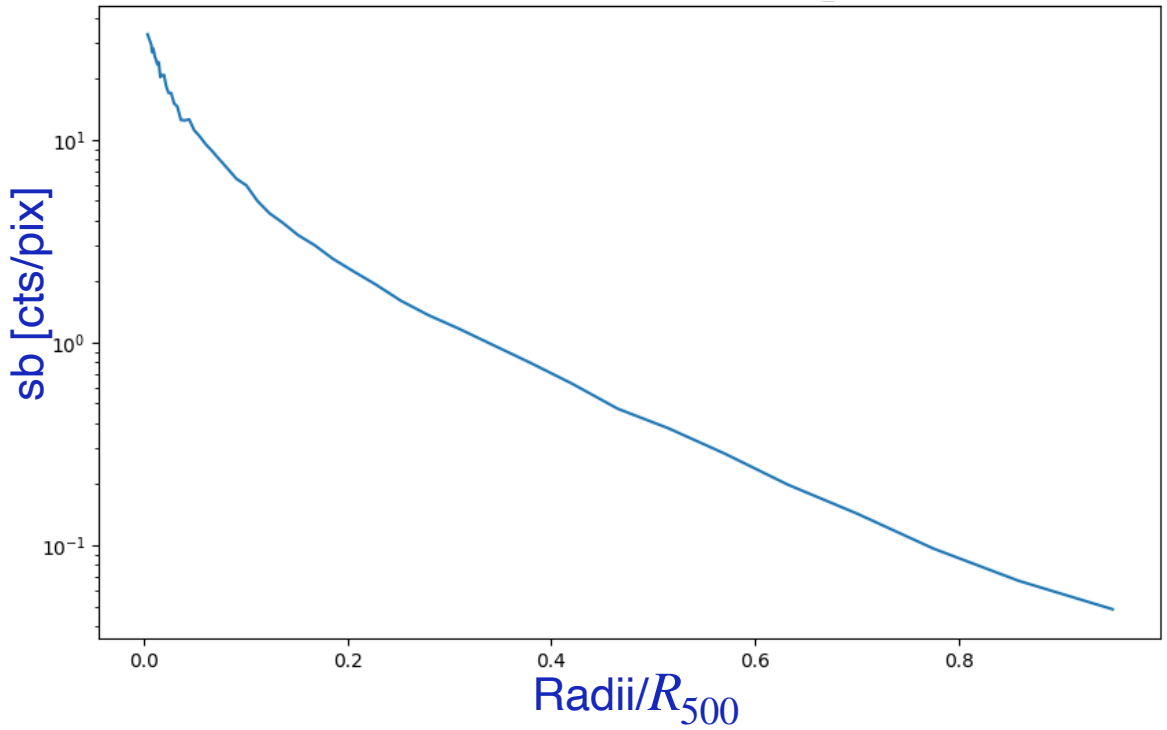
ML algorithm :

a. Surface brightness in X
b. Spectroscopic temperature
c. Six frequency signal profiles

Defined outputs :

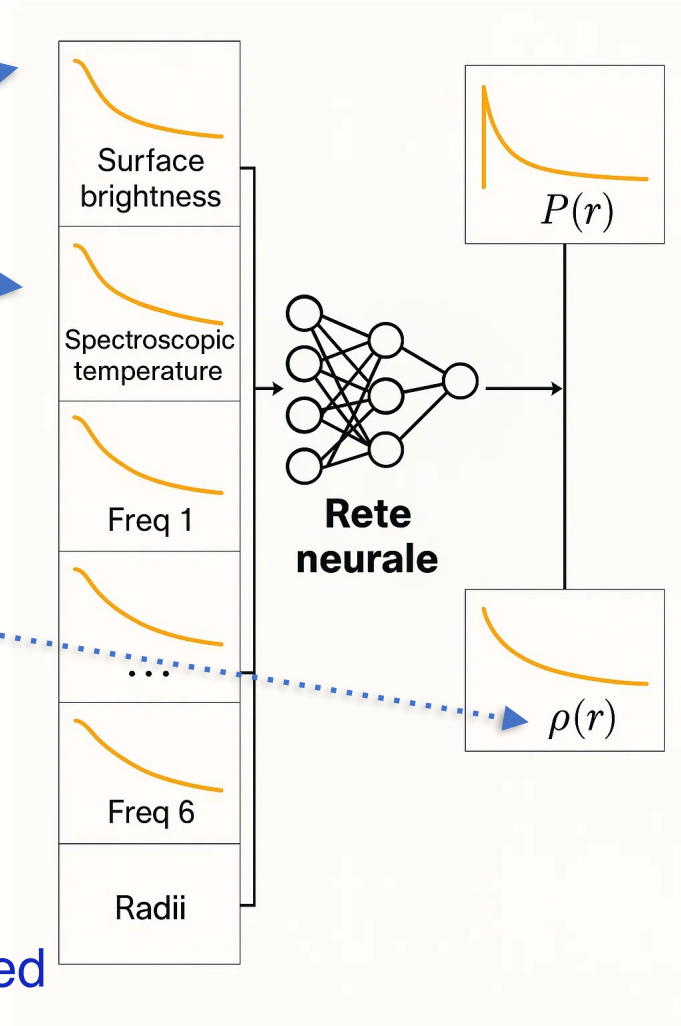
a. 3D Density profiles
b. 3D Pressure profiles
c. 3D Temperature profiles

Brightness Surface profile



Dataset Overview

- **120 galaxy clusters in total**
  - **For each cluster : - 9** radial input observables



- **1** physical target profile output (at the moment)

Each profile : 1D distribution over ~ 50-100 radial points

Challenges:

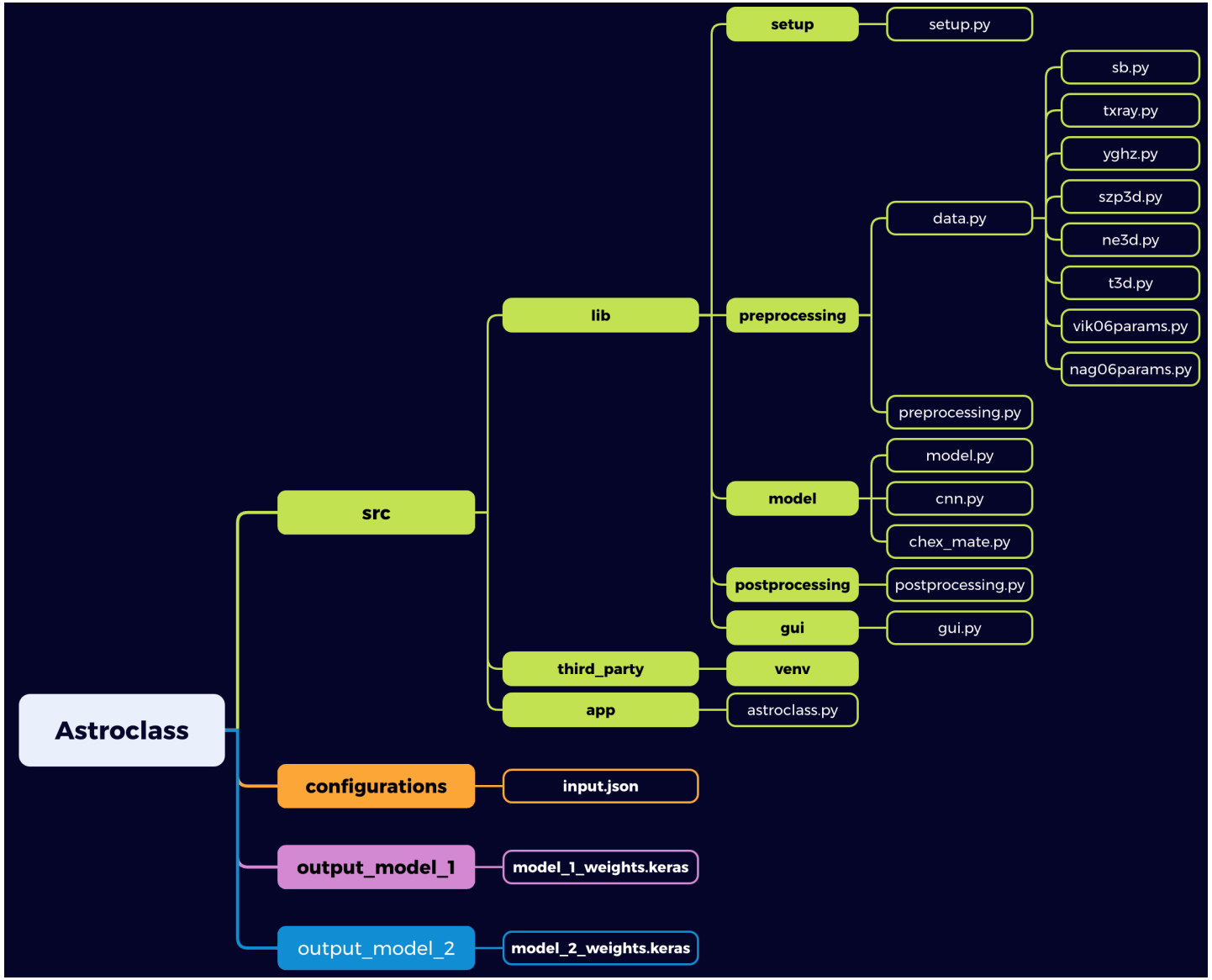
- Limited data sample → risk of overfitting
- Profiles heterogeneity → normalisation and interpolation required
- Need a robust yet compact NN architecture

Next to add T(r) profile

Architecture of the **AstroClass App**

The organisation is set within different directories and main files

- **Src**
  - **Libraries**
  - **Third parties**
  - **Application**
- **Preprocessing**
- **Model**
- **Postprocessing**
- **GUI**
- **Configurations**
- **Output1**
- **Output2**



Architecture of the **AstroClass App**

Input_File.json as an input file with main path where to find the data

The input variables :

- **Surface brightness**
- **Spectroscopy temperature**
- **Six frequencies Planck priors**

The output variables :

- **3d pressure profile**
- **3d density profile**
- **3d temperature profile**

```
"Preprocessing": {
    "Correlations":         "off",
    "Cleaning":             "interpolate",
    "Transformation":       "Standard",
    "Randomize":            "off",
    "Data Blocks":          "on",
    "Resampling":           {
      "Mode":                "on",
      "Input list":          ["sb", "Txray", "y100GHz", "y143GHz",
                              "y217GHz", "y353GHz", "y545GHz", "y857GHz"],
      "Output list":         ["szp3d","ne3d","t3d"]
    }

},
```

input_file.json

Model ML: modular code organised into functional layers

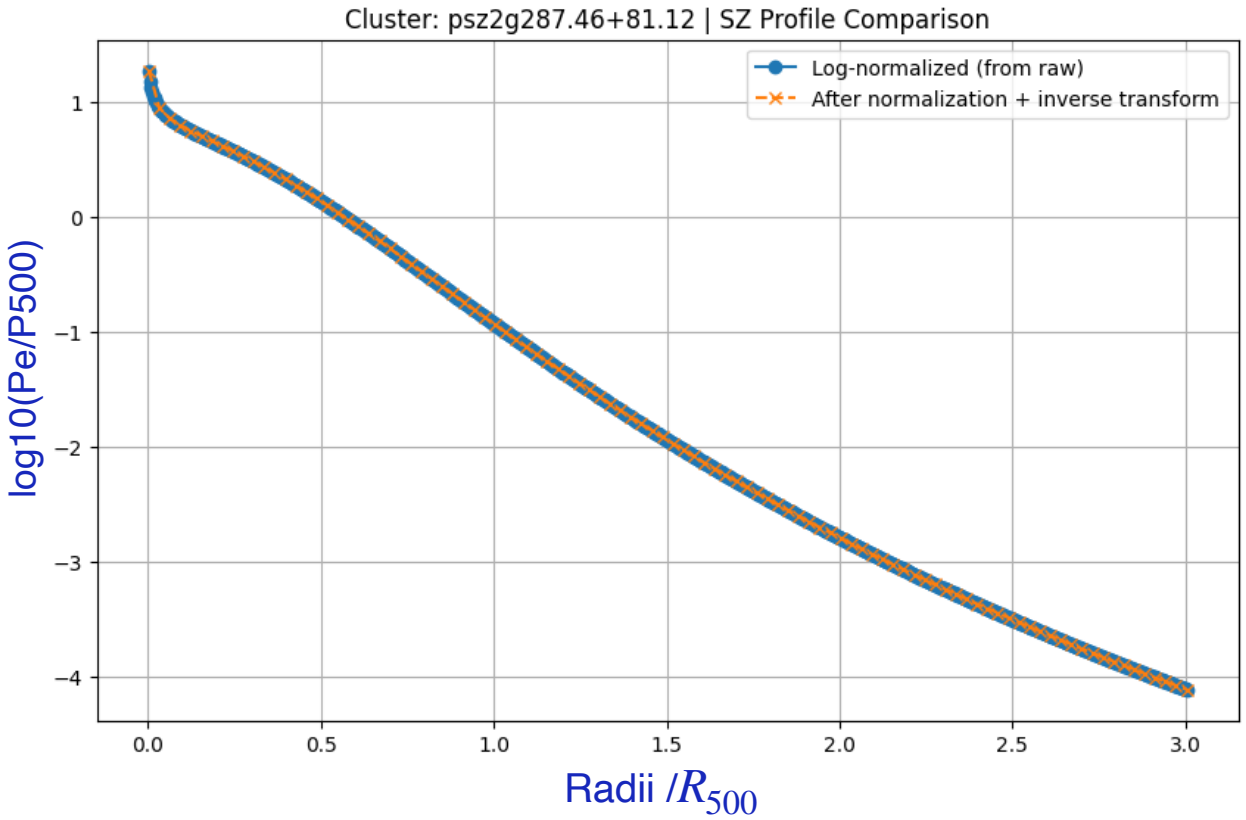
- prediction for pressure ✓
- Completely parametrisable

> 200 configurations tested to find optimal Hyperparameters configuration

Data checking

- We carefully checked that the preprocessing wouldn't change the physical shape of the data

✓ In **blue** Log-normalised pressure profile
✓ In **orange** the same distribution after the preprocessing



Training and results

Datasample : - 70% training sample
- 20% validation sample
- 10 % testing sample

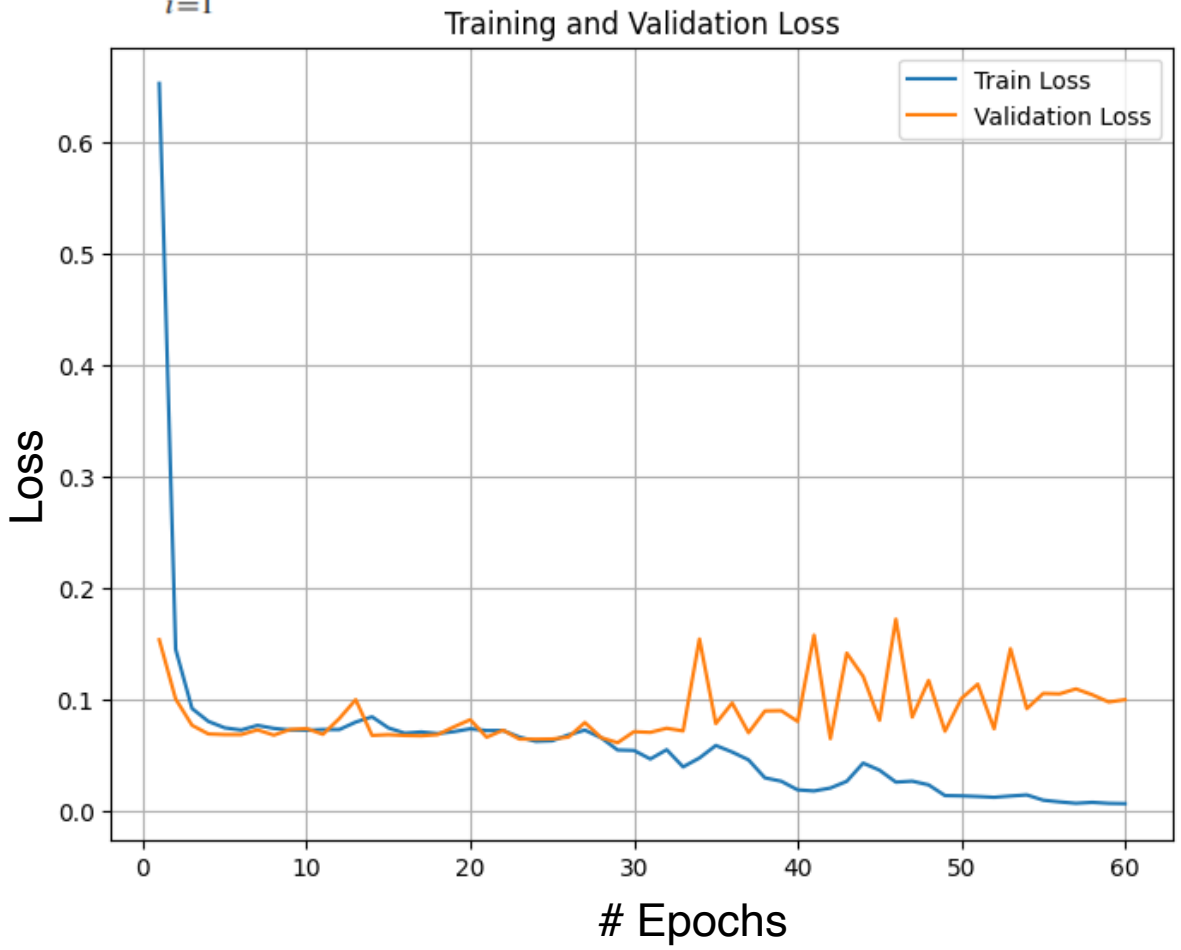
> 200 configurations on the hyper parameters to find best fit

Best Fit result:

```
====================================
Running experiment with hyperparameters:
        encoder_channels: [16, 32, 64, 128]
        latent_dim: 512
        decoder_channels: [32, 32]
        num_epochs: 60
        batch_size: 8
        dropout_rate: 0.0
====================================
```

Loss function is the mean absolute error

$$\mathrm{MAE}(z, y) = \frac{1}{n}\sum_{i=1}^{n} | y_i - z_i |$$



Results on First Observable: 3D pressure profile

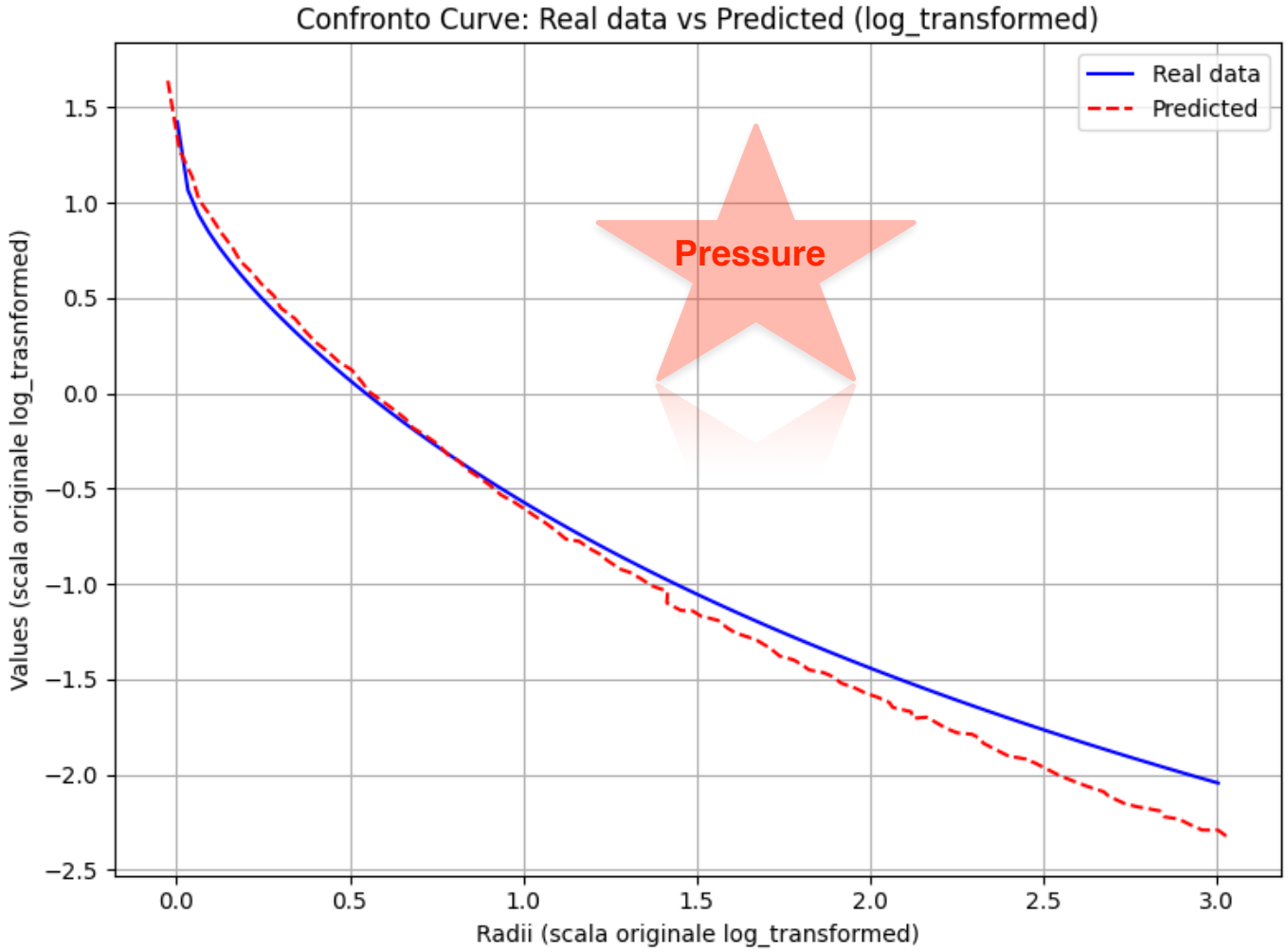
Metrics results :

$$|R_{true} - R_{pred}| = -0.0019872128 \text{ (mean)}$$

$$|Pe_{true} - Pe_{pred}| = -0.074650355 \text{ (mean)}$$

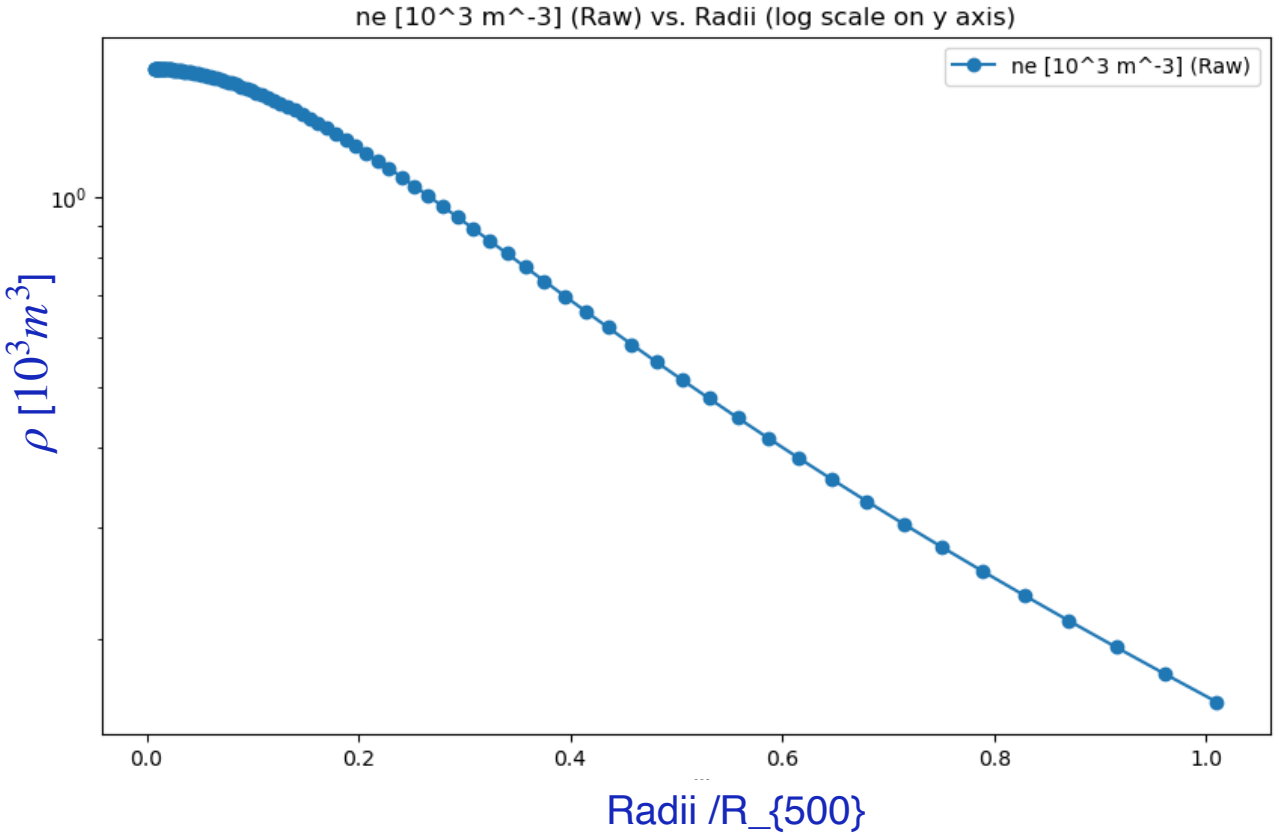
$$(|R_{true} - R_{pred}| + |Pe_{true} - Pe_{pred}|) = 0.142071932554245$$

MSE radii = 0.000226
RMSE radii = 0.015021
MSE values = 0.022373
RMSE values = 0.149576



Second Observable started : 3D density profile

- We made the same plan, checking the distribution for a first training loop on the second output, i.e. the **density** profile



Training and results

Datasample : - 70% training sample
- 20% validation sample
- 10 % testing sample

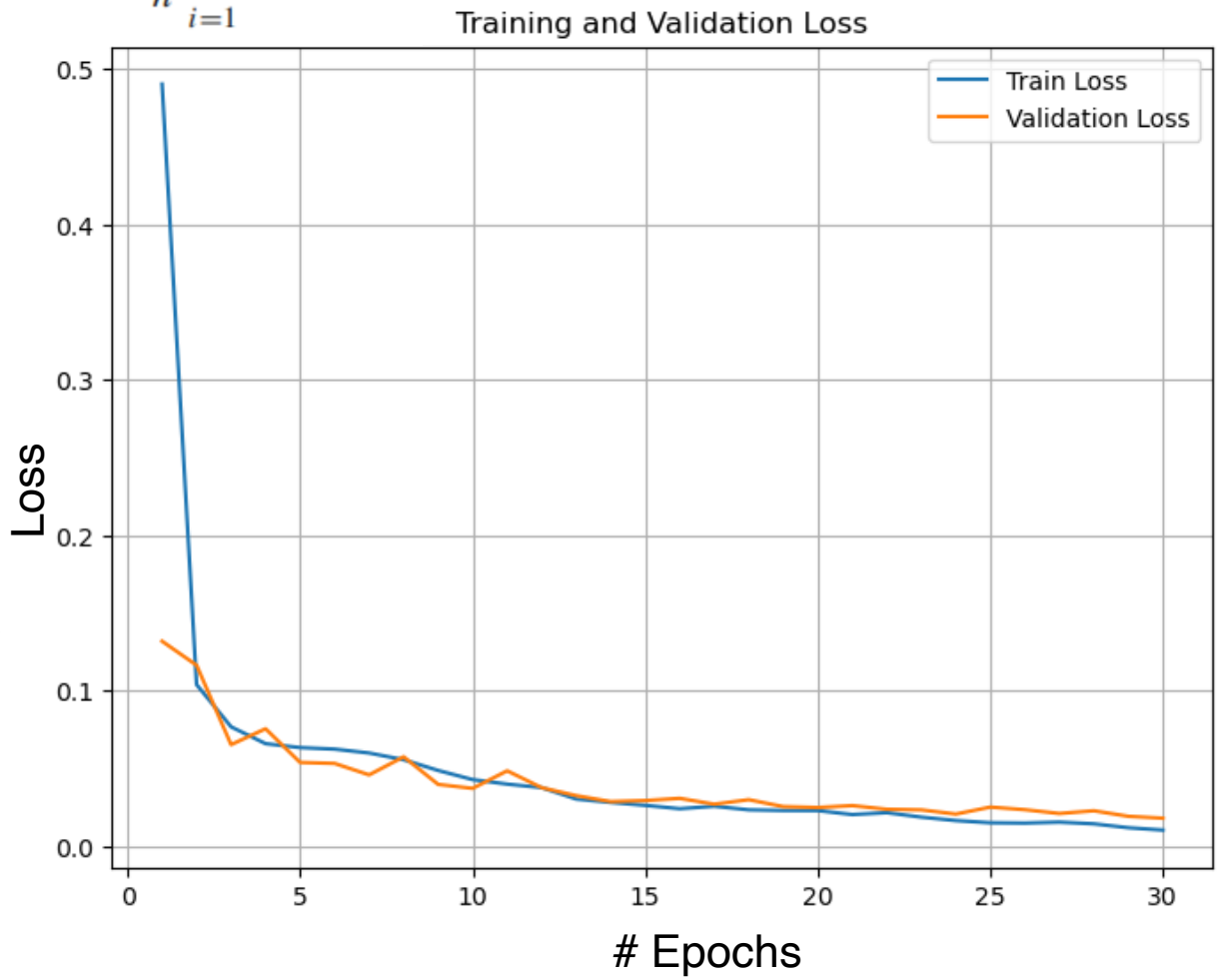
> 200 configurations on the hyper parameters to find best fit

Best Fit result:

```
====================================
Running experiment with hyperparameters:
        encoder_channels: [16, 32, 64]
        latent_dim: 512
        decoder_channels: [16, 16]
        num_epochs: 30
        batch_size: 4
        dropout_rate: 0.0
====================================
```

Loss function is the mean absolute error

$$\mathrm{MAE}(z, y) = \frac{1}{n}\sum_{i=1}^{n} | y_i - z_i |$$



Results on Second Observable: 3D density profile

Metrics results :

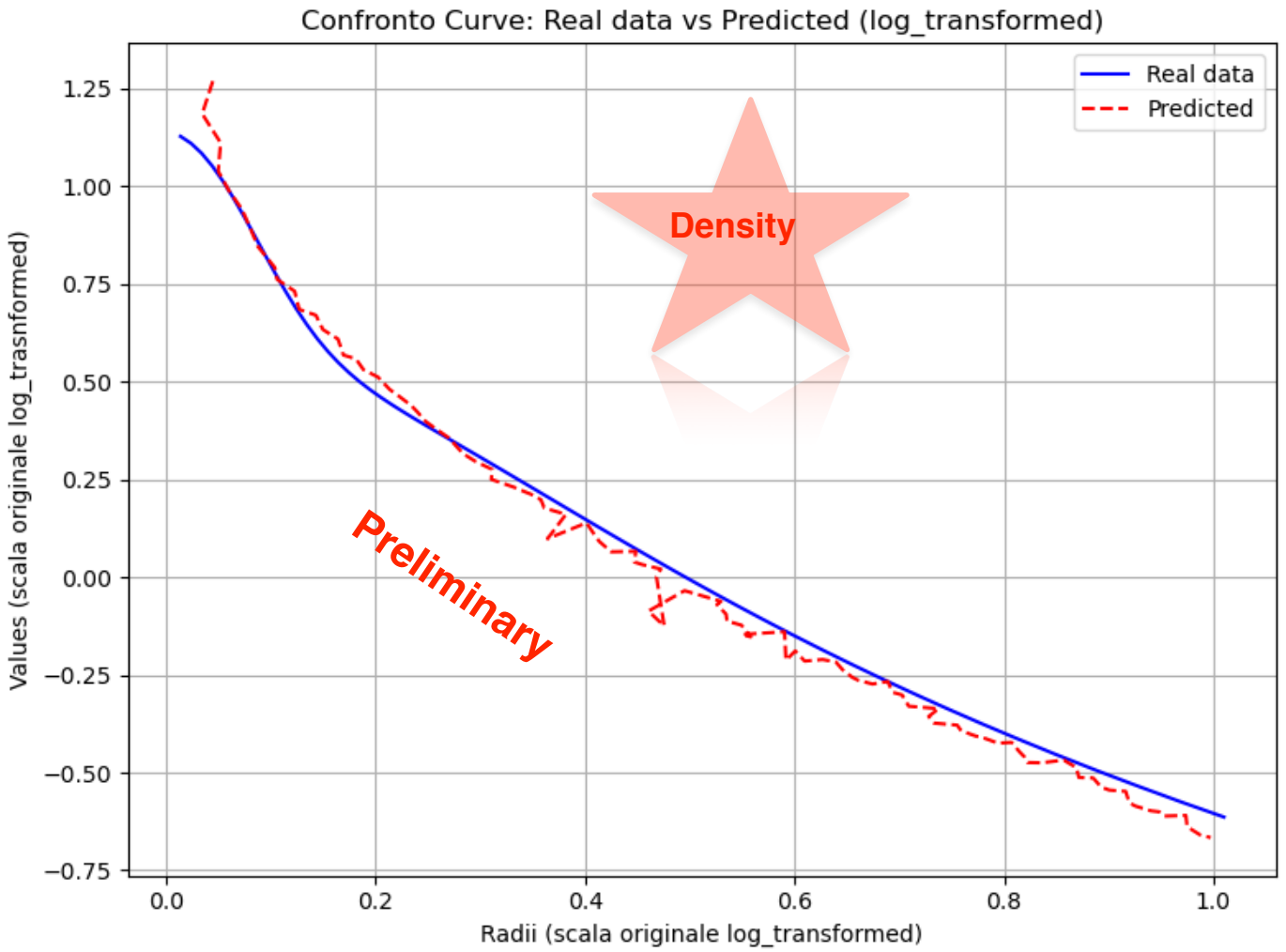
$$|R_{true} - R_{pred}| = -0.0017778769 \text{ (mean)}$$

$$|\rho_{true} - \rho_{pred}| = -0.019408487 \text{ (mean)}$$

$$(|R_{true} - R_{pred}| + |\rho_{true} - \rho_{pred}|) = 0.03741067275404 93$$

MSE radii = 0.0002
MSE values = 0.0112

Made on test set



Conclusions

- Pressure profile prediction 
- Density profile prediction 
- Refining the first two predictions and uncertainties and unify the code (In Progress)
- Working on the third output feature (Temperature) **most challenging one** (In progress)
- Finalising the end-to-end interface and platform (In Progress)
- Once the project is finalised we will work on dissemination (Paper, conferences..)

Thank you for your attention

BACK UP SLIDES

Input variables

```python
import os
import sys
import traceback
from   src.lib.setup.setup                   import Setup
from   src.lib.preprocessing.preprocessing   import PreProcessing
from   src.lib.models.model                  import Model
from   src.lib.postprocessing.postprocessing import PostProcessing
from   src.lib.gui.gui                       import Gui

class Astroclass:
    def __init__(self, config_filename):
        """
        App inizialize.

        Parameters
        ----------
        config_filename : str
            config json file path.
        """
        try:
            self.setup              = Setup(config_file=config_filename)
            if self.setup.args.mode == "train":
                self.data           = PreProcessing(setup=self.setup)
                self.model          = Model(setup=self.setup, data=self.data)
                self.postprocessing = PostProcessing(setup=self.setup, data=self.data, model=self.model.model)
            elif self.setup.args.mode == "gui":
                self.postprocessing = Gui(setup=self.setup)
        except Exception as e:
            print(f"App error: {e}")
            traceback.print_exc()
            sys.exit(1)

if __name__ == "__main__":
    json_path  = os.path.join('configurations', "chex_model.json")
    app        = Astroclass(config_filename=json_path)
```

Sviluppo Algoritmo

Lo sviluppo dell'algoritmo sarebbe dovuto iniziare a partire da questo mese

Siamo riusciti in realtà già a sviluppare un primo scheletro funzionante già ad oggi

Data preparation per la rete

```
input_to_include = ['sb', 'txray', 'y100GHz', 'y143GHz', 'y217GHz', 'y353GHz', 'y5
output_to_include = ['SZP3d']
for cluster_key, cluster_data in data_ml_profiles.items():
    cluster_input = []

    for key in cluster_data:
        if is_input and key not in input_to_include:
            continue

        if not is_input and key not in output_to_include:
            continue

        # Carica i dati e seleziona le colonne (supponendo che siano tabelle o array numpy)
        data = np.array(cluster_data[key])  # Converti in array numpy

        # Seleziona la prima e la quarta colonna
        radii = data[:, 0]  # Prima colonna
        values = data[:, 3]  # Quarta colonna

        # Interpolazione per garantire una dimensione uniforme
        if is_input:
            feature = np.interp(
                np.linspace(radii.min(), radii.max(), input_output_shape[0] * input_output_shape[1]),
                radii, values
            ).reshape(input_output_shape)  # Reshape a (50, 50)
        else:
            feature = np.interp(
                np.linspace(radii.min(), radii.max(), output_output_shape[0] * output_output_shape[1]),
                radii, values
            ).reshape(output_output_shape)  # Reshape a (1000, 1000)
        cluster_input.append(feature)

    if cluster_input:
        # Stack dei canali
        input_data.append(np.stack(cluster_input, axis=-1))  # Dimensione: (50, 50, n_features)
```

Model

1. Encoder e decoder layers

```python
class EncoderDecoder(nn.Module):
    def __init__(self, input_channels=8, output_channels=1):
        super(EncoderDecoder, self).__init__()

        # Encoder layers
        self.encoder_conv1 = nn.Conv2d(input_channels, 32, kernel_size=3, padding=1)
        self.encoder_pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.encoder_conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.encoder_pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.encoder_conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.encoder_pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Latent space
        self.flatten = nn.Flatten()
        self.fc_latent = nn.Linear(128 * 6 * 6, 1000)
        self.fc_decode = nn.Linear(1000, 128 * 6 * 6)

        # Decoder layers with more upsampling
        self.decoder_reshape = lambda x: x.view(-1, 128, 6, 6)

        self.decoder_deconv1 = nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1,
        output_padding=1)
        self.decoder_deconv2 = nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1,
        output_padding=1)
        self.decoder_deconv3 = nn.ConvTranspose2d(32, 16, kernel_size=3, stride=2, padding=1,
        output_padding=1)

        # Additional upsampling to reach 1000x1000
        self.upsample = nn.Upsample(size=(1000, 1000), mode='bilinear', align_corners=False)

        self.decoder_output = nn.Conv2d(16, output_channels, kernel_size=3, padding=1)
```

Model

- Activation functions and pooling

```python
def forward(self, x):
    # Encoder
    x = F.relu(self.encoder_conv1(x))
    x = self.encoder_pool1(x)
    x = F.relu(self.encoder_conv2(x))
    x = self.encoder_pool2(x)
    x = F.relu(self.encoder_conv3(x))
    x = self.encoder_pool3(x)

    x = self.flatten(x)
    x = F.relu(self.fc_latent(x))
    x = F.relu(self.fc_decode(x))
    x = self.decoder_reshape(x)

    # Decoder
    x = F.relu(self.decoder_deconv1(x))
    x = F.relu(self.decoder_deconv2(x))
    x = F.relu(self.decoder_deconv3(x))

    x = self.decoder_output(x)

    # Upsample to match label size
    x = self.upsample(x)

    return x
```

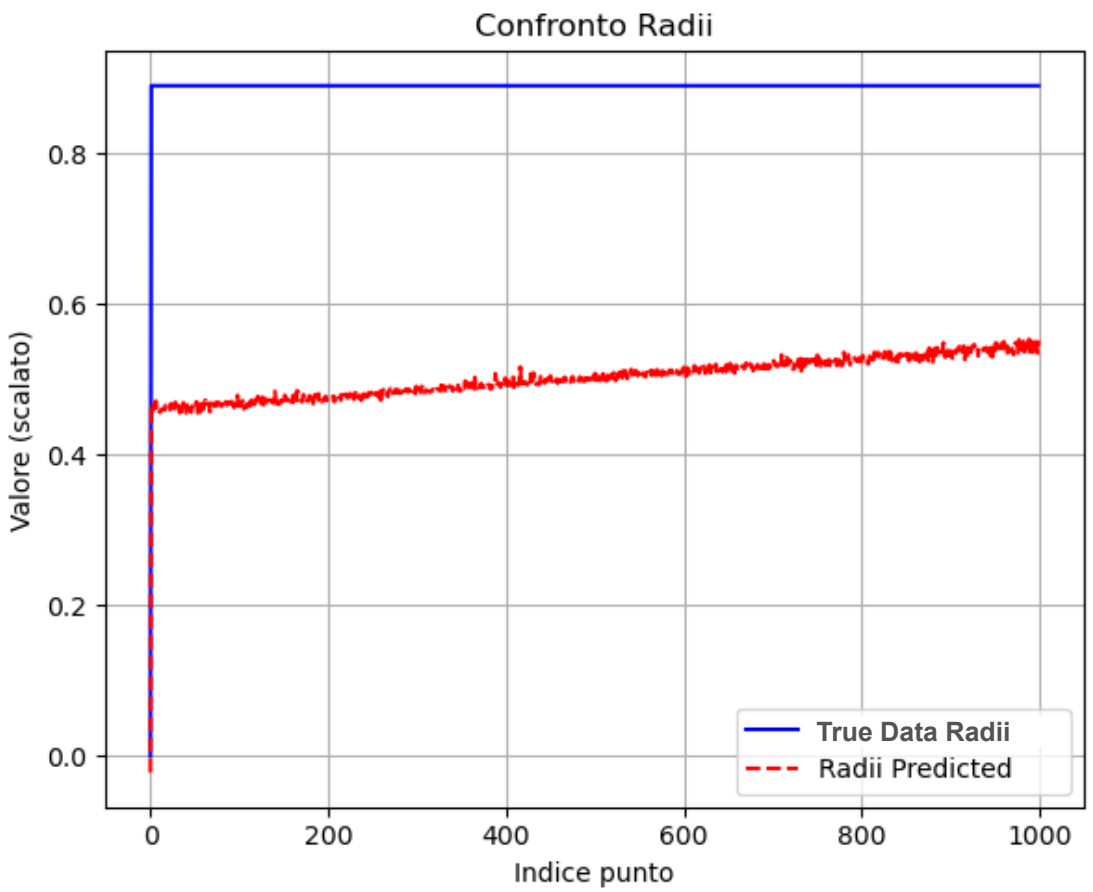
Second step

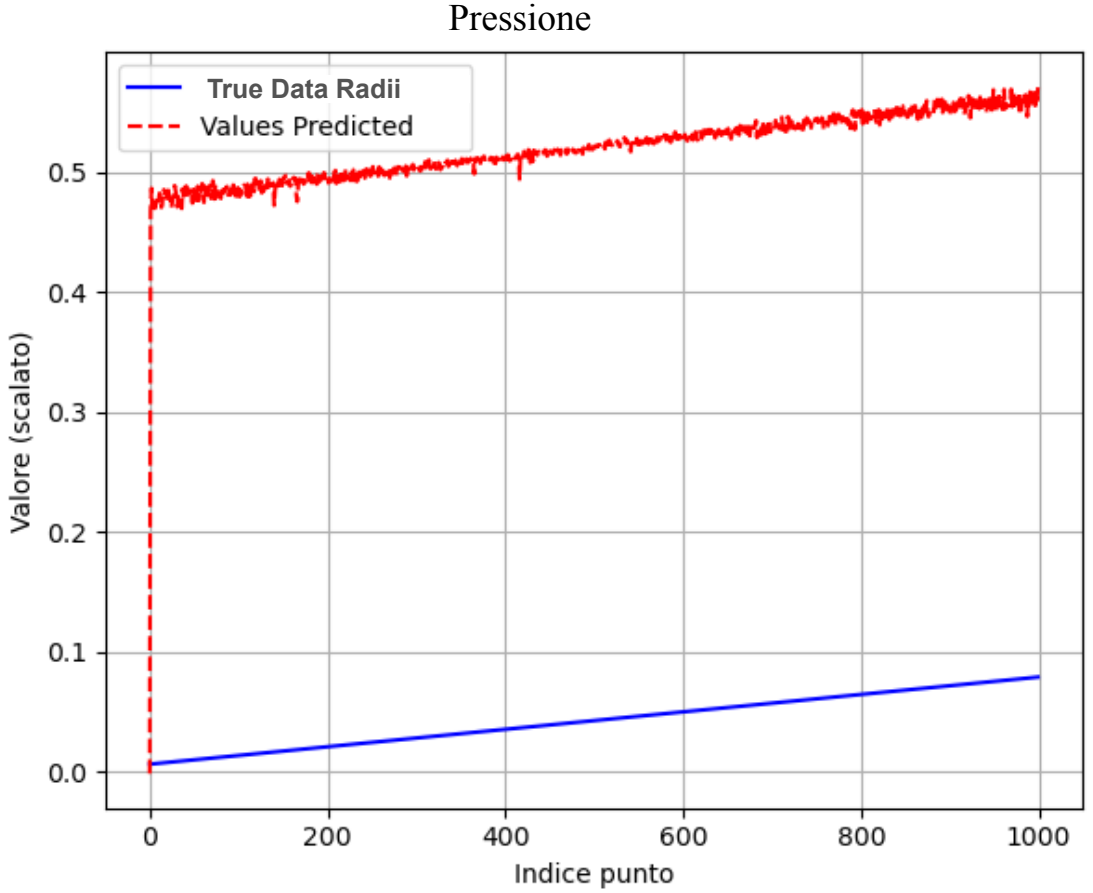
Dataset:

- ~~118~~ 117 galaxy profiles
- distanze normalizzate
- 9 input features
  - Radii
  - Values of files: sb, ..
- 2 output features (2 vectors of 100 points)
  - Radii
  - Temperature / Pression / Density

```python
class EncoderDecoderSimplified(nn.Module):
    def __init__(self, input_length=50, input_width=1, input_features=9,
                 output_length=100, output_width=1, output_features=2,
                 latent_dim=512):  # smaller latent space than 2048
        super(EncoderDecoderSimplified, self).__init__()
        self.input_length = input_length

        # --- Encoder ---
        # Reshape from (B, 50, 1, 9) -> (B, 9, 50)
        # Using two Conv1d layers with fewer channels.
        self.encoder_conv1 = nn.Conv1d(input_width * input_features, 32, kernel_size=3, padding=1)
        self.encoder_conv2 = nn.Conv1d(32, 64, kernel_size=3, padding=1)

        # Flatten and project to a smaller latent space.
        self.flatten = nn.Flatten()
        latent_input_size = 64 * input_length  # Length remains unchanged due to padding.
        self.fc_latent = nn.Linear(latent_input_size, latent_dim)

        # --- Decoder ---
        # Expand the latent space back to a feature map suitable for convolution.
        self.fc_decode = nn.Linear(latent_dim, output_length * output_width * 16)

        # Reshape to (B, 16, output_length, output_width)
        self.decoder_reshape = lambda x: x.view(-1, 16, output_length, output_width)

        # Final conv layer to produce the desired output features.
        self.final_conv = nn.Conv2d(16, output_features, kernel_size=1)
```

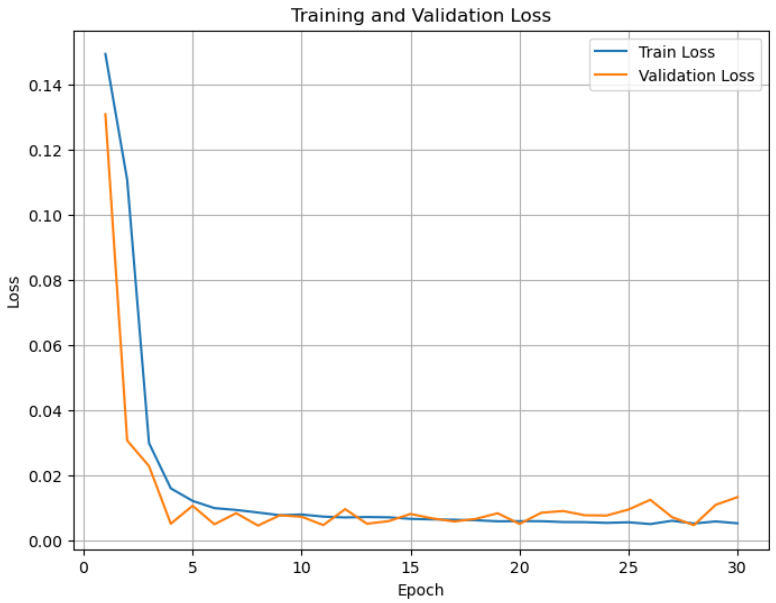
Just a recap : First attempt

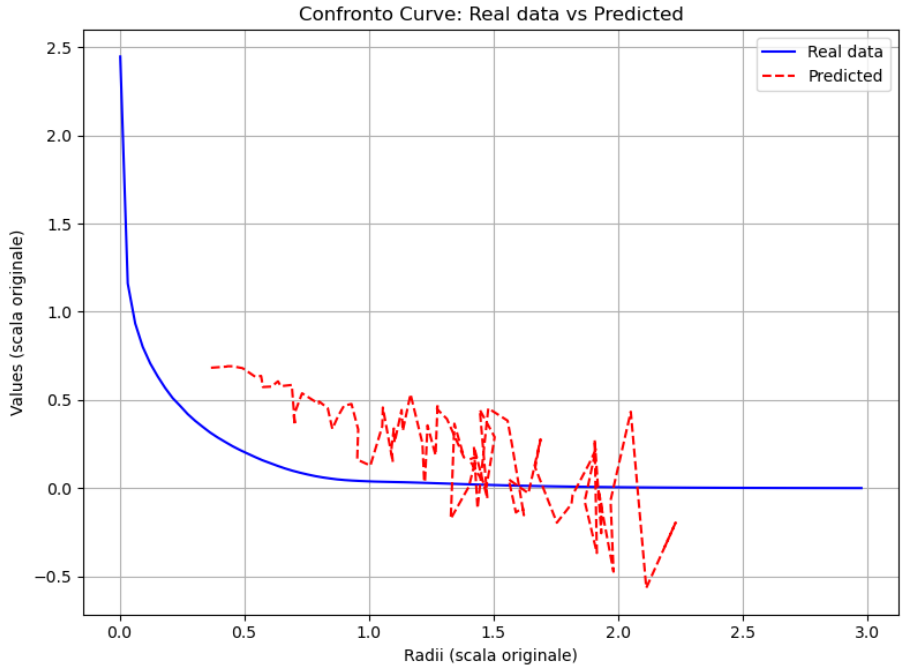




Second step- Test ⟶ Overfitting

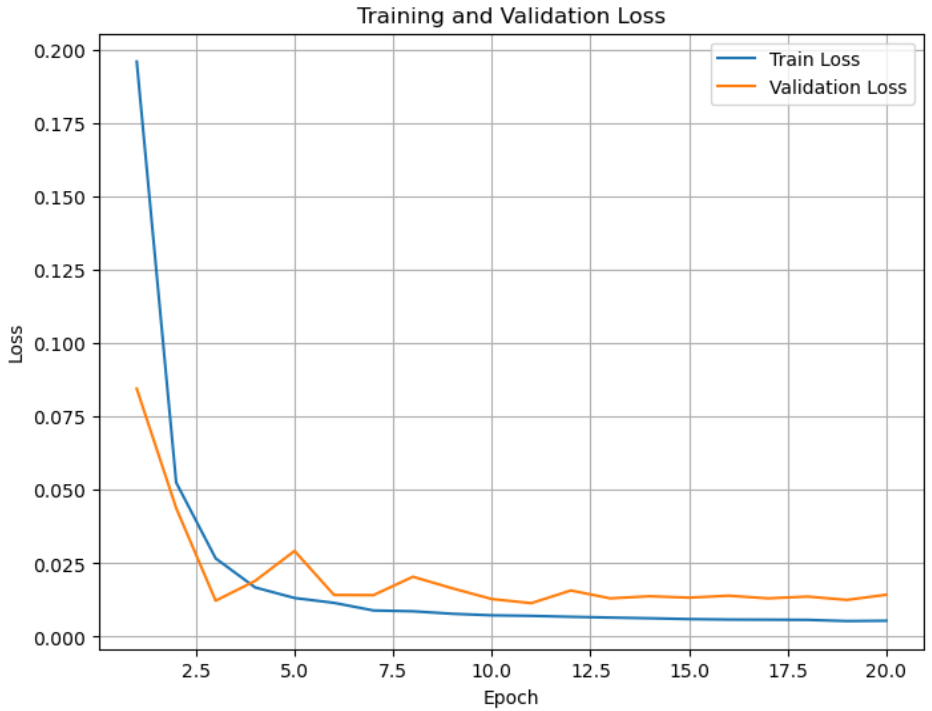
- `encoder_channels: [8, 16, 32, 64, 128]` **`latent_dim: 1024`**
  `decoder_channels: [8, 16, 32, 64]` **`num_epochs: 30`**
  `batch_size: 16 dropout_rate: 0.2`

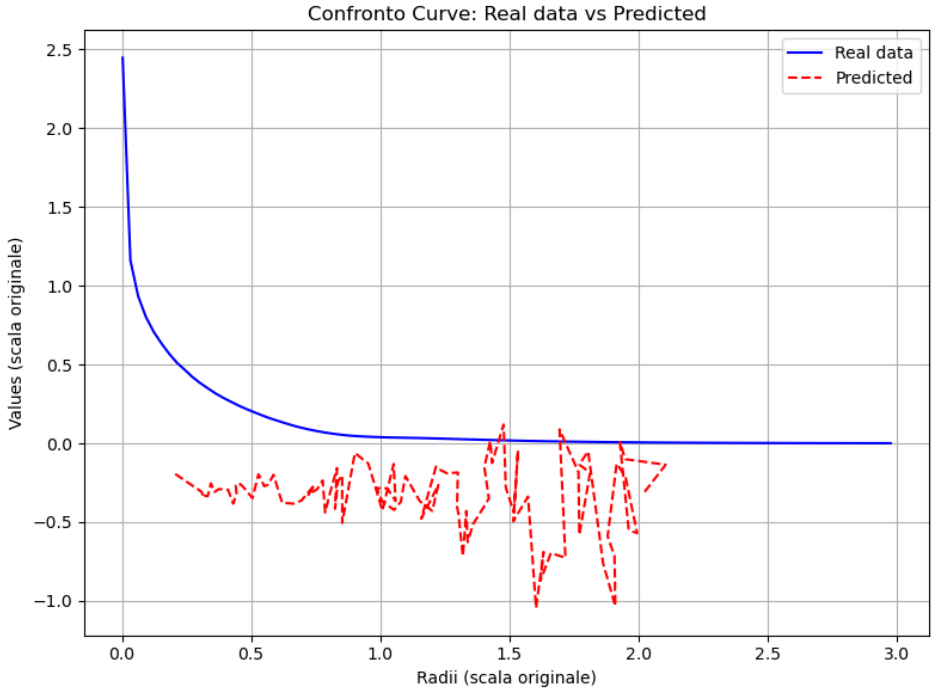




Second step- Test 2 ⟶ Overfitting

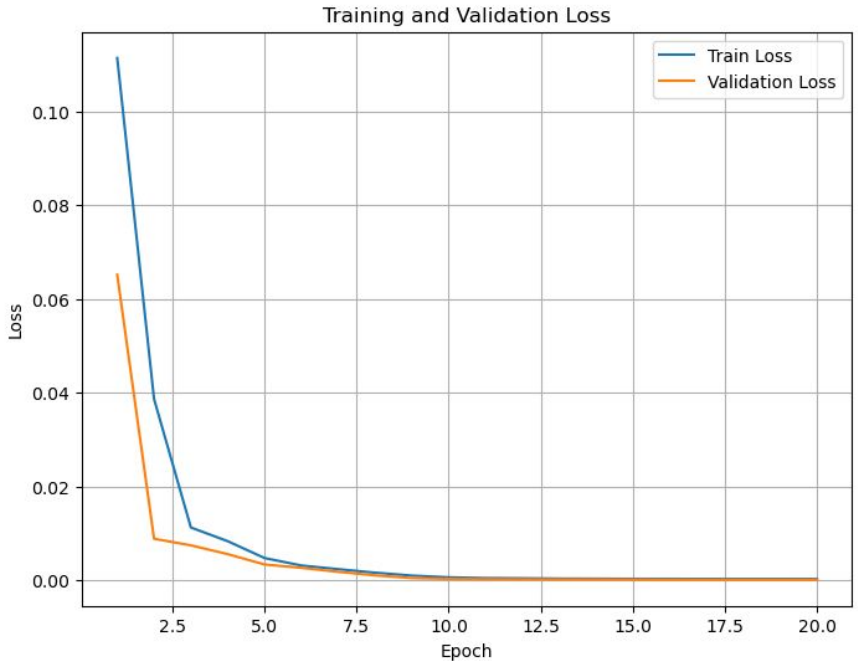
- `encoder_channels: [8, 16, 32, 64, 128]` **`latent_dim: 2048`**
  `decoder_channels: [8, 16, 32, 64]` **`num_epochs: 20`**
  `batch_size: 16 dropout_rate: 0.2`

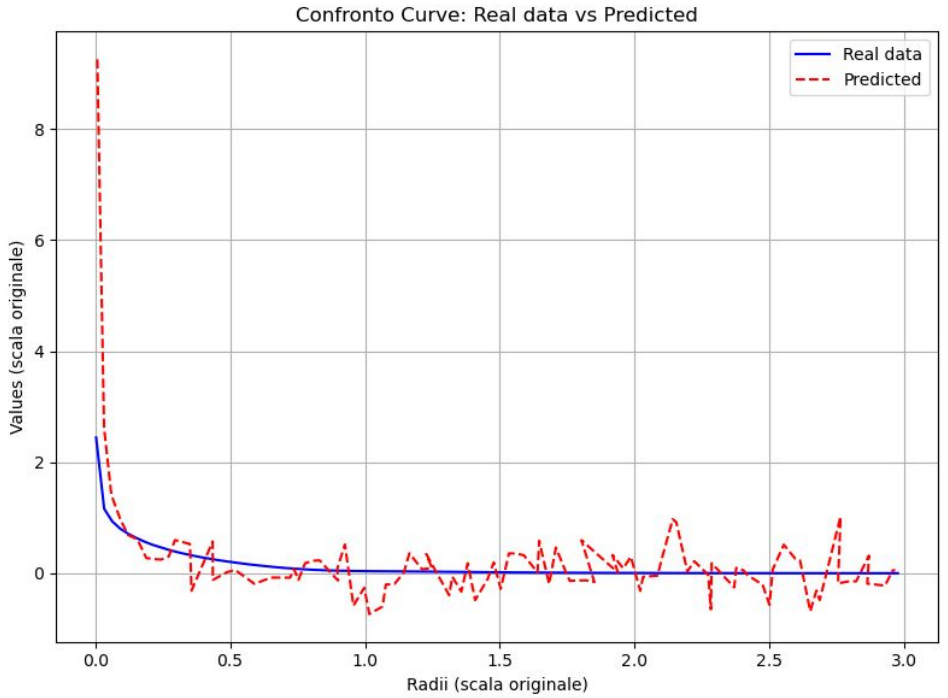




Second step- Test 3 ⟶ Reducing overfitting

- `encoder_channels: [8, 16, 32, 64, 128] latent_dim: 2048 decoder_channels: [8, 16, 32, 64] num_epochs: 20 batch_size: 16dropout_rate: 0.`





Next step : Code structure

Parametri specifici della rete

Encoder:

- 3 layers con canali (16,32,64) → Performance generalmente buone
- Diminuendo layer e canali → Performance peggiorano!

Balance : aumentando i layers e aumentando il numero di iterazioni (epoche) si ottengono alcuni miglioramenti

Latent space (rappresentazione compressa dei dati):

- Se è troppo piccolo non aiuta le performance → In particolare sulla feature spaziale

```
self.encoder_convs = nn.ModuleList()
self.encoder_dropouts = nn.ModuleList()
in_channels = self.input_width * self.input_features  # ad esempio: 1 * 9 = 9
for out_channels in encoder_channels:
    self.encoder_convs.append(nn.Conv1d(in_channels, out_channels, kernel_size=3, padding=1))
    self.encoder_dropouts.append(nn.Dropout(self.dropout_rate))
    in_channels = out_channels
self.final_encoder_channels = in_channels
```

Next step : Code structure

Parametri specifici della rete

Encoder:

- 3 layers con canali (16,32,64) → Performance generalmente buone
- Diminuendo layer e canali → Performance peggiorano!

Balance : aumentando i layers e aumentando il numero di iterazioni (epoche) si ottengono alcuni miglioramenti

Latent space (rappresentazione compressa dei dati):

- Se è troppo piccolo non aiuta le performance → In particolare sulla feature spaziale

```
self.encoder_convs = nn.ModuleList()
self.encoder_dropouts = nn.ModuleList()
in_channels = self.input_width * self.input_features  # ad esempio: 1 * 9 = 9
for out_channels in encoder_channels:
    self.encoder_convs.append(nn.Conv1d(in_channels, out_channels, kernel_size=3, padding=1))
    self.encoder_dropouts.append(nn.Dropout(self.dropout_rate))
    in_channels = out_channels
self.final_encoder_channels = in_channels
```

Next step : Code structure

```
self.flatten = nn.Flatten()
self.fc_latent = nn.Linear(self.final_encoder_channels * self.input_length, latent_dim)
self.latent_dropout = nn.Dropout(self.dropout_rate)
```

Latent space (rappresentazione compressa dei dati):

- Se è troppo piccolo non aiuta le performance → In particolare sulla feature spaziale

```
# Decoder: espansione del vettore latente a una mappa di feature
self.fc_decode = nn.Linear(latent_dim, decoder_channels[0] * self.output_length * self.output_width)
self.decode_dropout = nn.Dropout(self.dropout_rate)

self.decoder_convs = nn.ModuleList()
self.decoder_dropouts = nn.ModuleList()
for i in range(1, len(decoder_channels)):
    self.decoder_convs.append(nn.Conv2d(decoder_channels[i-1], decoder_channels[i],
                                        kernel_size=3, padding=1))
    self.decoder_dropouts.append(nn.Dropout(self.dropout_rate))
self.final_decoder_channels = decoder_channels[-1]
# Convoluzione finale per ottenere il numero di output features desiderato.
self.final_conv = nn.Conv2d(self.final_decoder_channels, self.output_features, kernel_size=1)
```