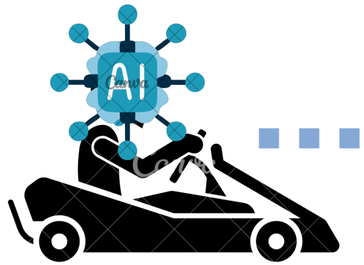
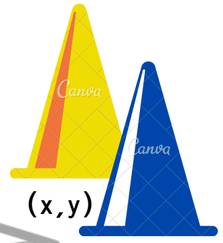
CONES DETECTION & BEV RECONSTRUCTION



Project created by Elena Azzi and Arianna Cella

Introduction

In our project we focused on the creation of a neural network with the aim of reconstructing the roadway starting from the position of the blue and yellow cones in world coordinates that delimit its boundaries.



Initial Input

Some files containing the (x, y) coordinates of yellow and blue cones collected from a given frame.

Goal

Reconstruction of roadway boundaries and the centerline

Project Development Pipeline

Dataset Preprocessing

- *Feature selection* to remove unclear or erroneous frames and applying transformations to standardize the data.
- D*ata augmentation* step was then conducted to enrich the dataset and enhance the model's robustness.
- C*onvertion into a grid format* to serve as input for the neural network.

Neural Network Model Development

Various neural network models were developed and tested to identify the most suitable architecture for solving the specific problem.

Performance Evaluation and Results Visualization

This step aimed to evaluate the model's effectiveness in reconstructing the road and localizing the cones accurately.

Dataset Overview

blue_cones.dat yellow_cones.dat

Input

They contain the coordinates of blue and yellow cones, where each line represents the (x, y) position of a cone in the BEV frame.

blue_curve.dat yellow_curve.dat center_curve.dat

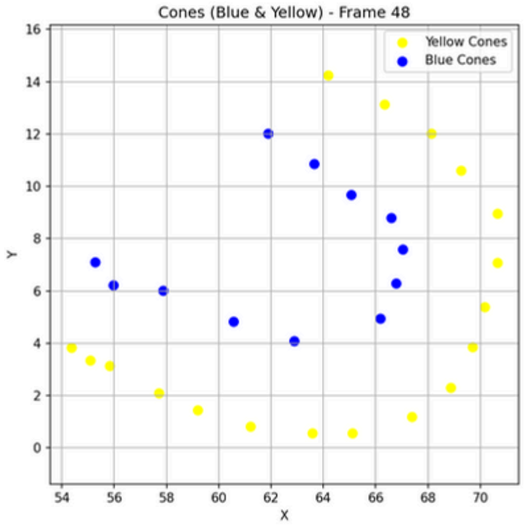
Ground truth

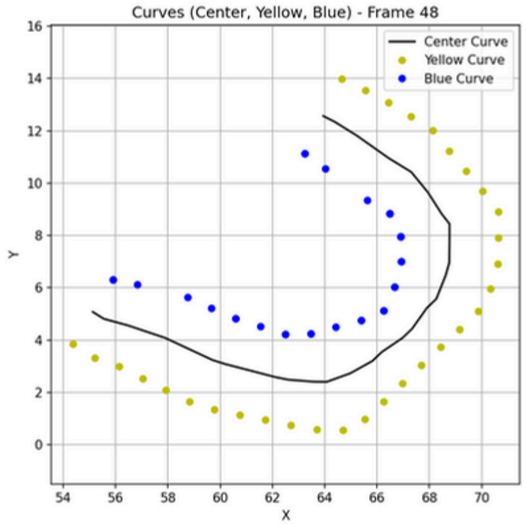
The curve files represent the paths of yellow and blue road curves, as well as the centerline of the road, which are important for lane detection.

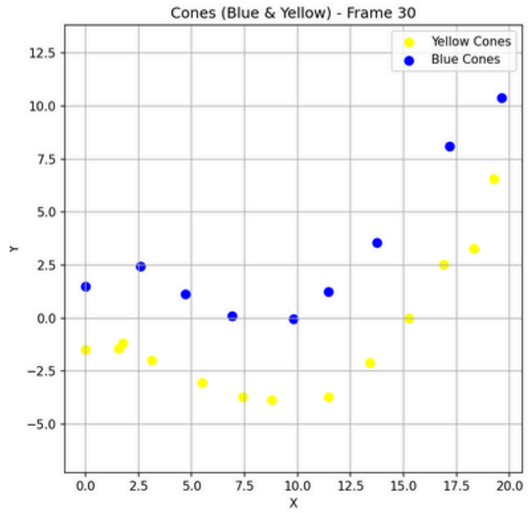
The data is filtered by frame ID to process each frame individually

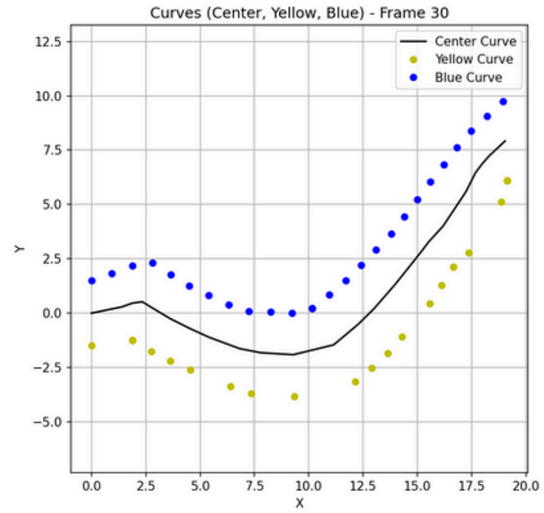
First dataset

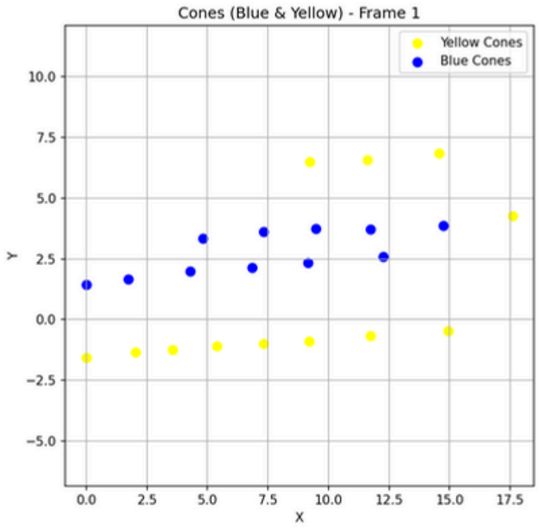
We received several sequences of frames containing cone and curve data.

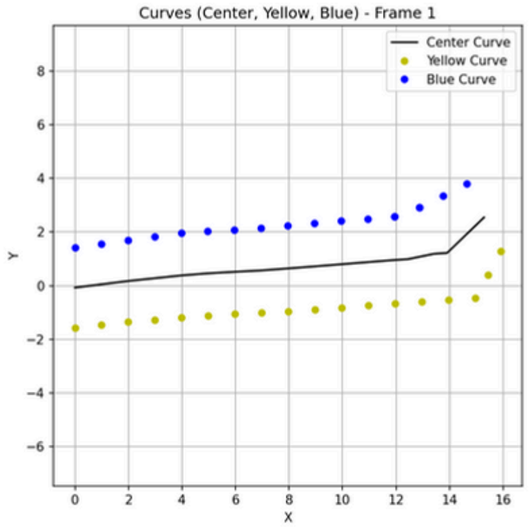


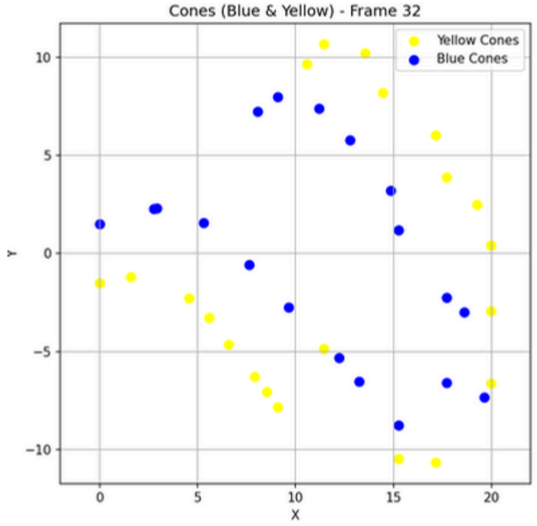


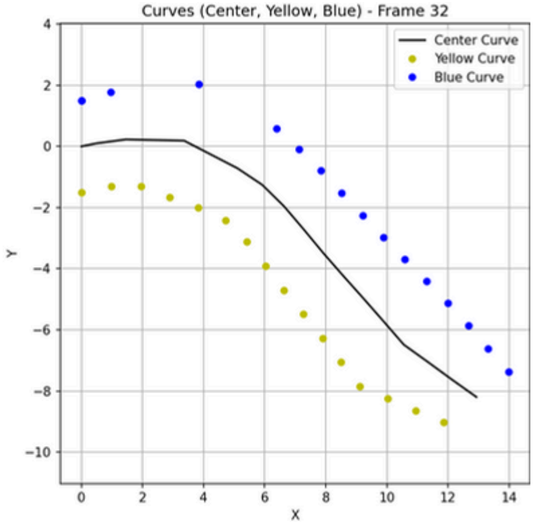




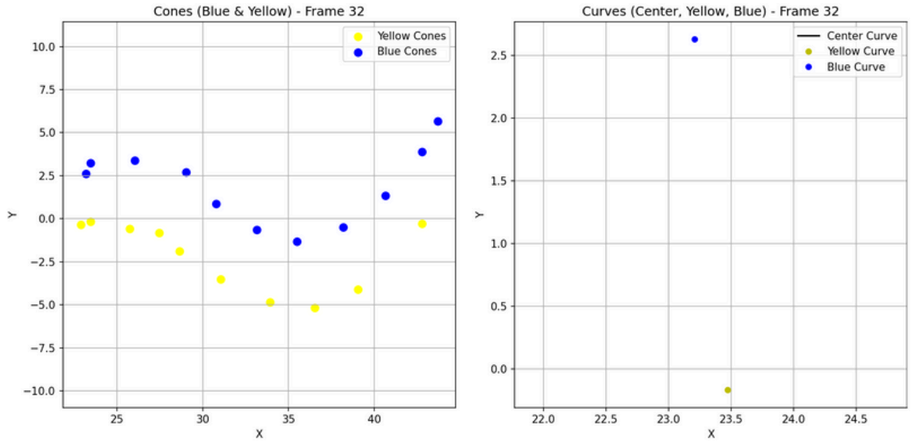




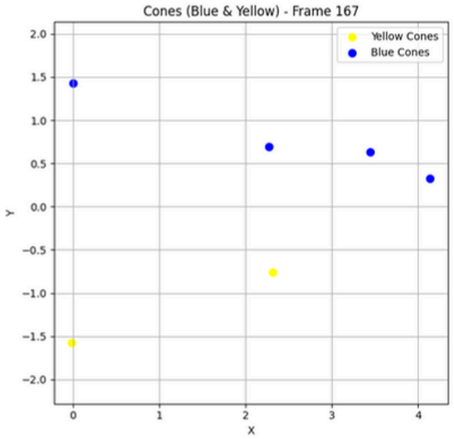


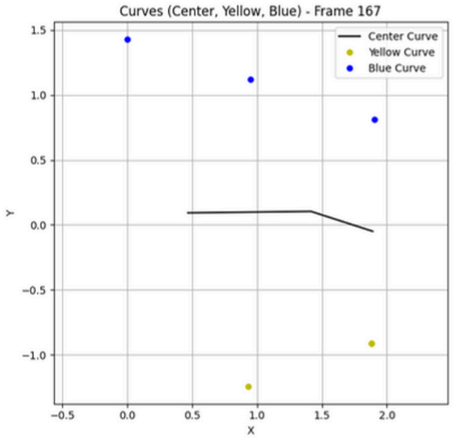


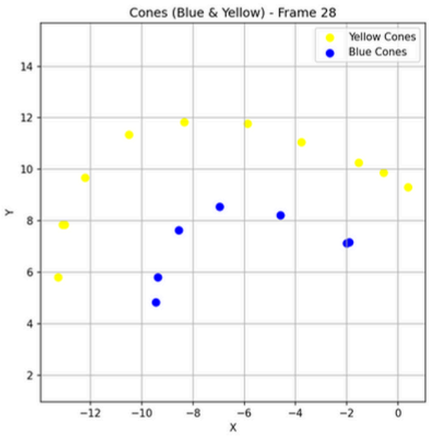
Problems:

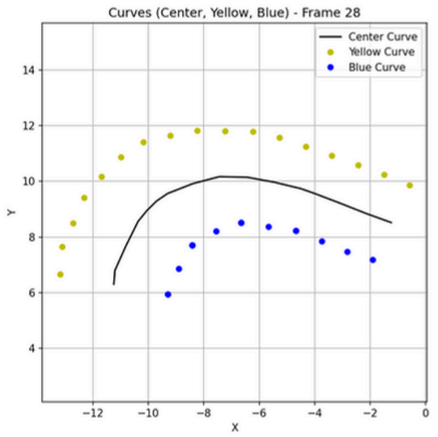

Cones (Blue & Yellow) - Frame 32
Yellow Cones
Blue Cones
X
Y

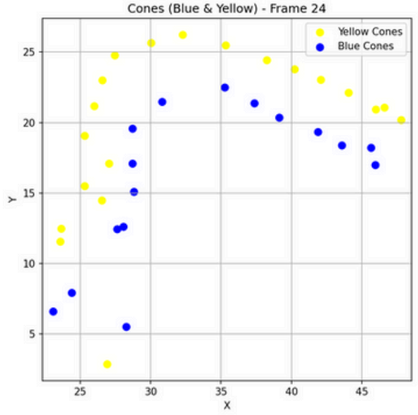
Curves (Center, Yellow, Blue) - Frame 32
Center Curve
Yellow Curve
Blue Curve
X
Y

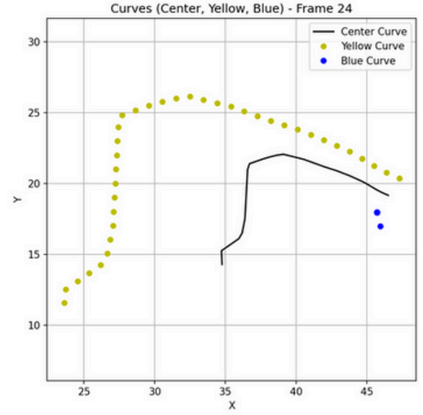

Cones (Blue & Yellow) - Frame 167
Yellow Cones
Blue Cones
X
Y

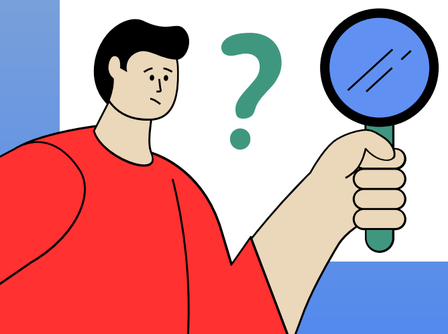

Curves (Center, Yellow, Blue) - Frame 167
Center Curve
Yellow Curve
Blue Curve
X
Y


Cones (Blue & Yellow) - Frame 28
Yellow Cones
Blue Cones
X
Y

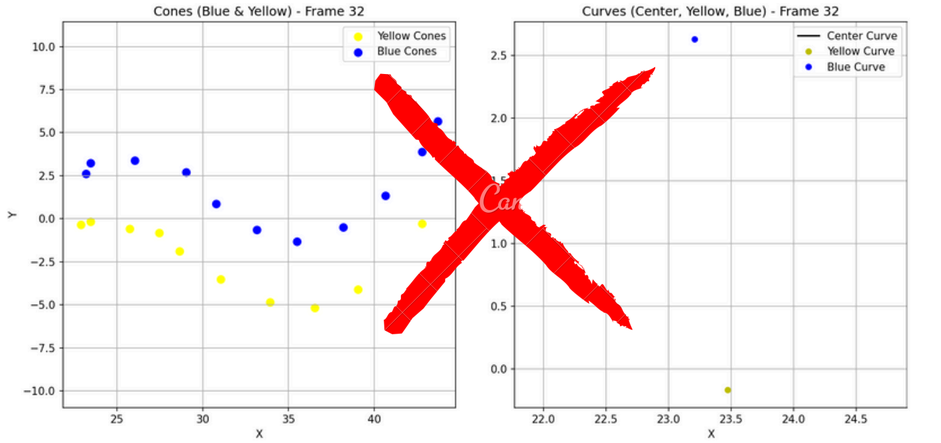

Curves (Center, Yellow, Blue) - Frame 28
Center Curve
Yellow Curve
Blue Curve
X
Y

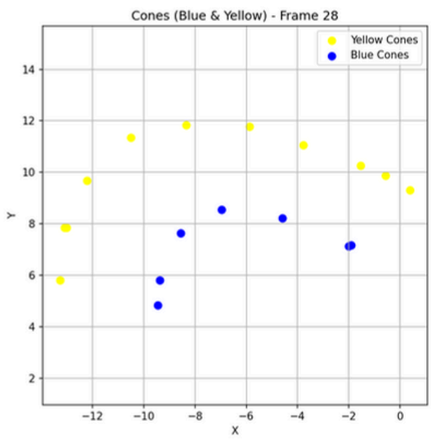

Cones (Blue & Yellow) - Frame 24
Yellow Cones
Blue Cones
X
Y

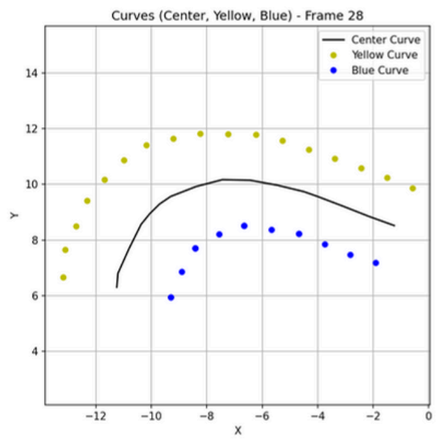

Curves (Center, Yellow, Blue) - Frame 24
Center Curve
Yellow Curve
Blue Curve
X
Y

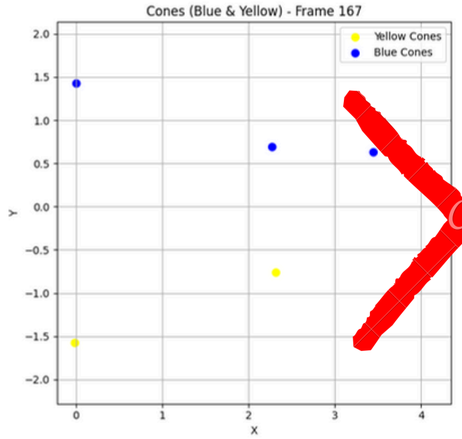


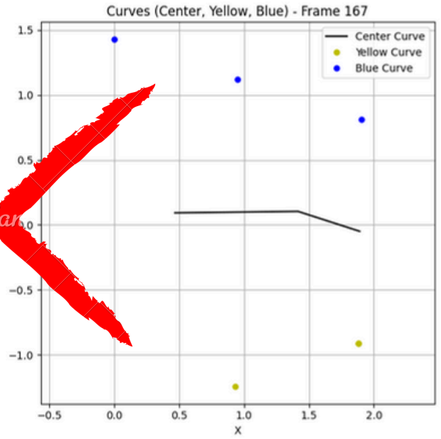
Problems:

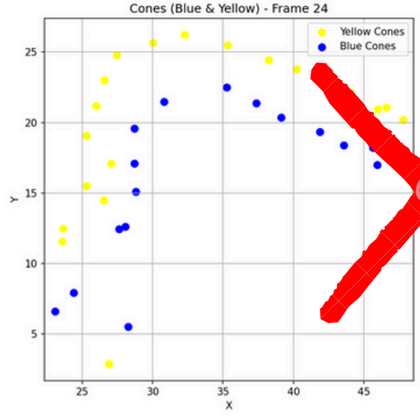


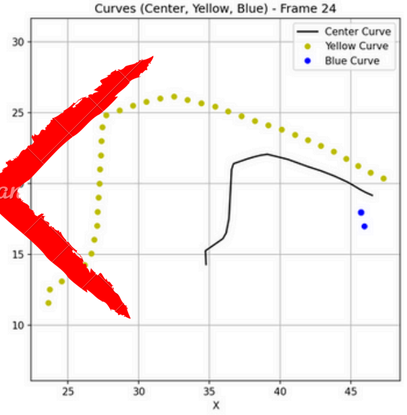


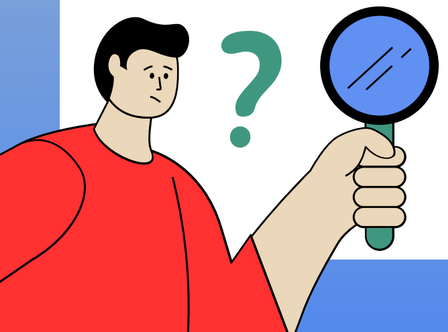












 # total frames - # discarded frames = 340

(1243) **(903)**

28%

Traslation process

```python
# Trova dove inizia la center line
dx=-center_curve_frame[0][0]
dy=-center_curve_frame[0][1]

# Traslazione di tutti i punti
center_curve_frame[:, 0] += dx
center_curve_frame[:, 1] += dy
yellow_cones_frame[:, 0] += dx
yellow_cones_frame[:, 1] += dy
blue_cones_frame[:, 0] += dx
blue_cones_frame[:, 1] += dy
yellow_curve_frame[:, 0] += dx
yellow_curve_frame[:, 1] += dy
blue_curve_frame[:, 0] += dx
blue_curve_frame[:, 1] += dy

if(center_curve_frame[0][0]>center_curve_frame[1][0]):
    # stiamo andando da destra verso sinistra quindi
    # Flip
    center_curve_frame[:, 0] *=-1
    yellow_cones_frame[:, 0] *=-1
    blue_cones_frame[:, 0] *=-1
    yellow_curve_frame[:, 0] *=-1
    blue_curve_frame[:, 0] *=-1

    # cambiare colore
    blue_cones_frame, yellow_cones_frame =yellow_cones_frame, blue_cones_frame
    blue_curve_frame, yellow_curve_frame =yellow_curve_frame, blue_curve_frame
```

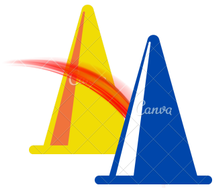
It standardizes the position of the data points, ensuring consistency and alignment across the dataset.

1. The translation vector, based on the first centerline point, shifts all cone and curve data to align with the origin (0, 0).

$$\begin{cases} dx = x0center \\ dy = y0center \end{cases}$$

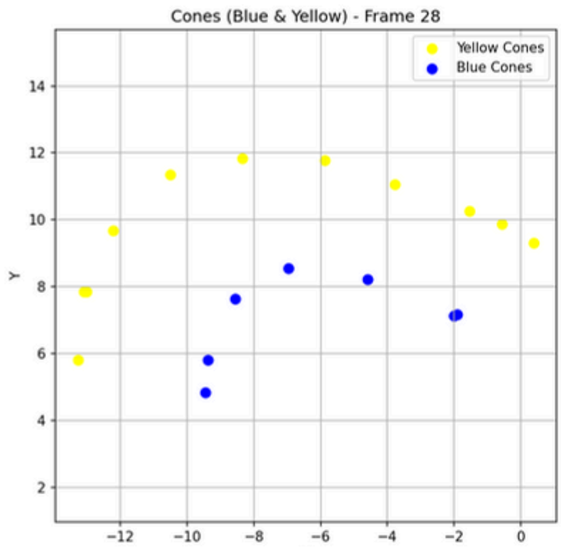
$$(x , y) \longrightarrow (x - dx, y - dy)$$

2. If the roadway has negative coordinates relative to the x-axis, we invert all x-coordinates by multiplying them by -1 and swap the cone colors accordingly.

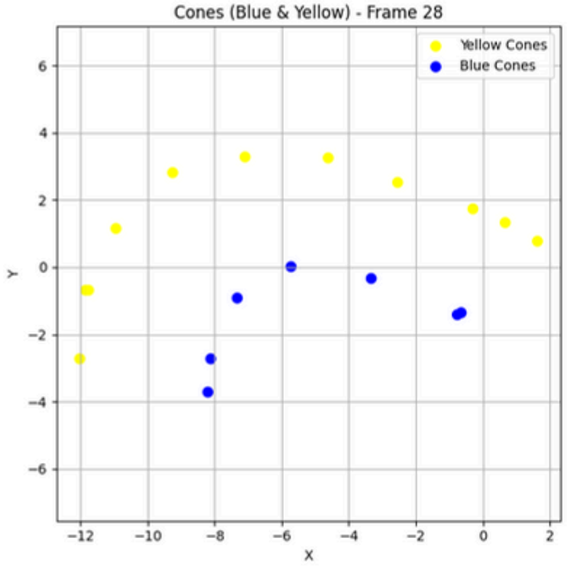


$$(x*(-1) , y)$$

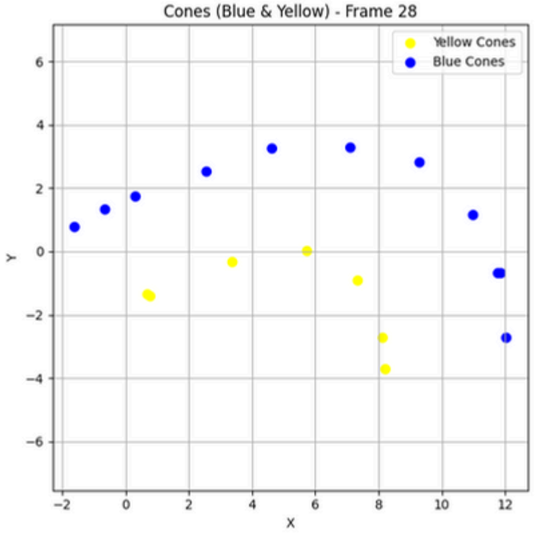
Traslation & flip process

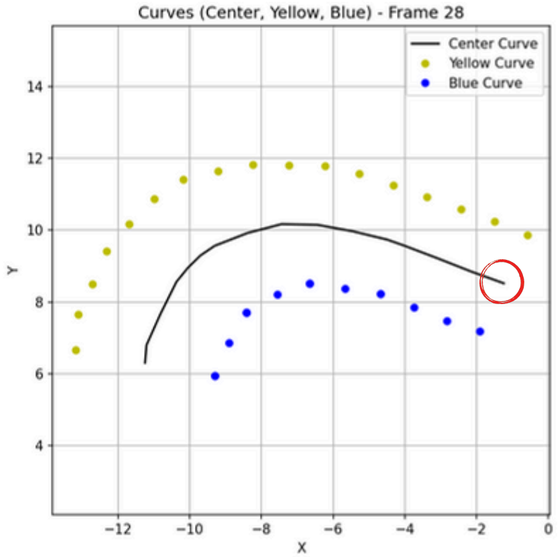


1

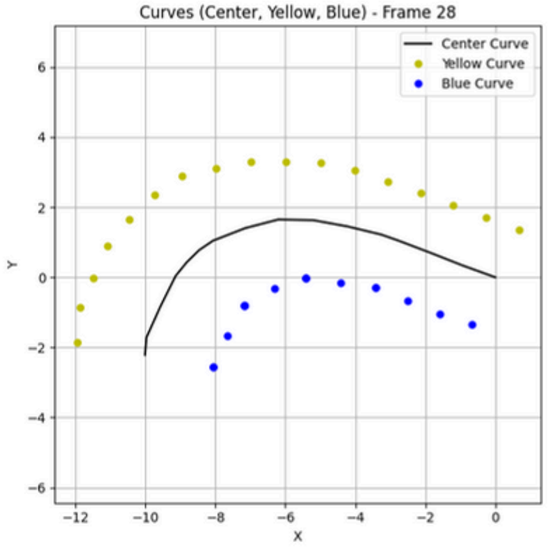


2

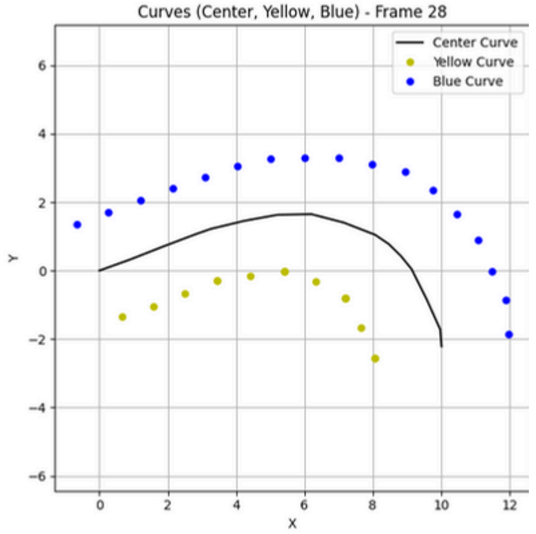




1



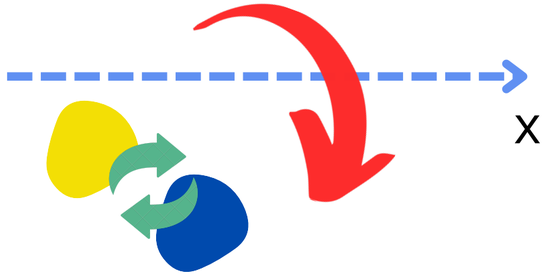
2



Data augmentation

Rotation

Flip

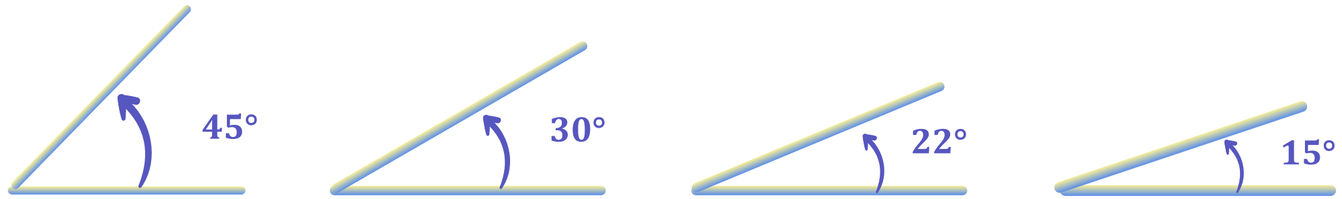


Rotation

The 2D rotation matrix is a 2×2 square matrix that rotates a vector in a Cartesian plane by an angle θ, defined as:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

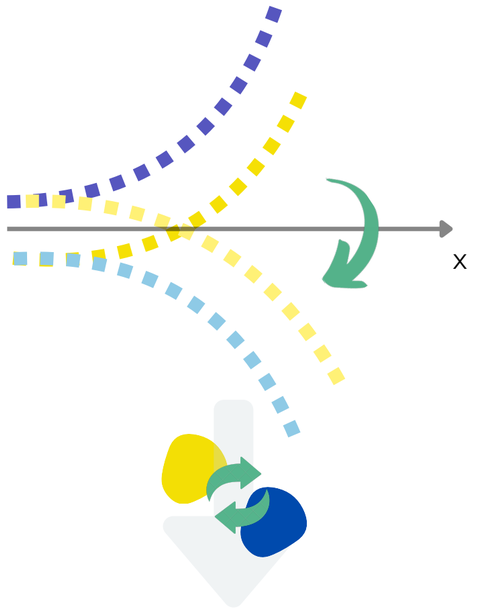
Applying the rotation matrix to a vector $[x,y]^T$, the result obtained is a new vector rotated by θ around the origin, that preserving its distance from the origin.



By generating new angles, we enrich the dataset with diverse perspectives, enabling **more robust** model training and improved generalization to unseen scenarios.

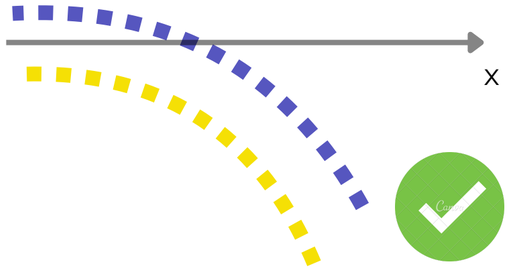
Flip

After performing data augmentation by applying various rotations to the dataset, we implemented a transformation that flips the entire dataset along the x-axis.



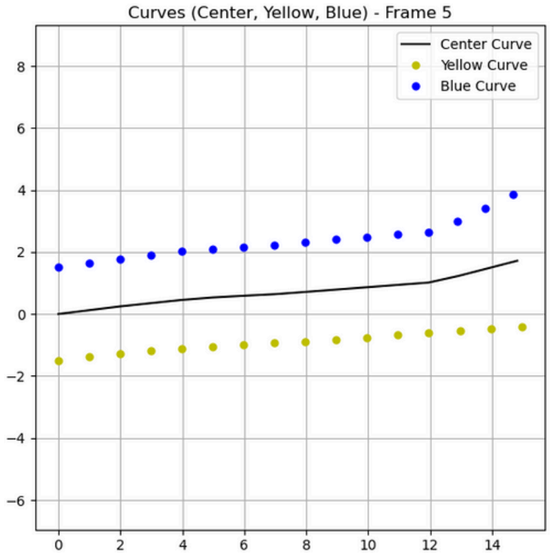
This operation mirrors the positions of all data points vertically, creating a symmetric counterpart for each original frame in the dataset.

In addition to flipping, we inverted the colors of the cones.

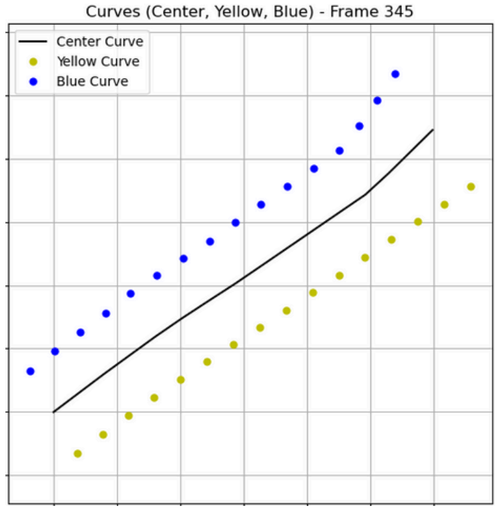


This step ensures that the newly generated frames not only maintain spatial symmetry, but also accurately reflect the roles and positions of cones in a real-world scenario.

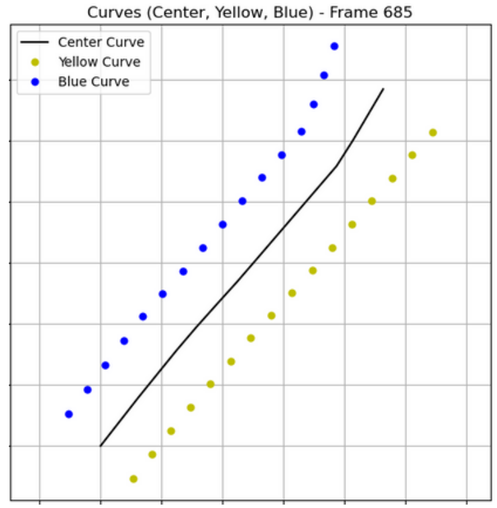
Let’s rotate…



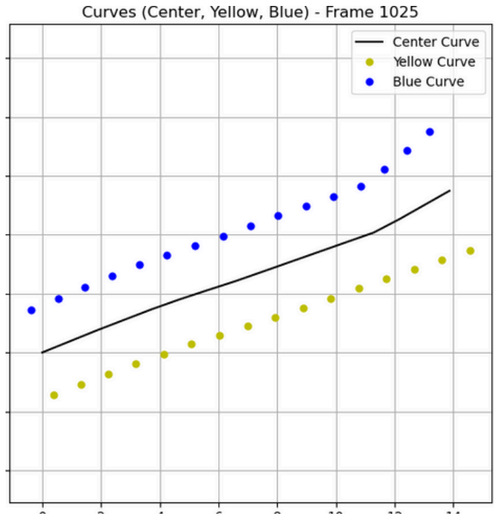
Original



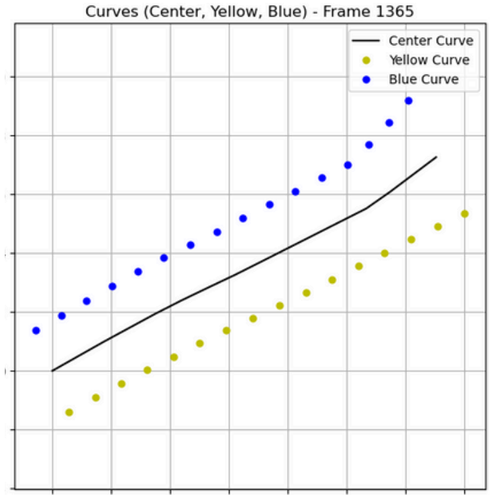
π/6 = 30°



π/4 = 45°



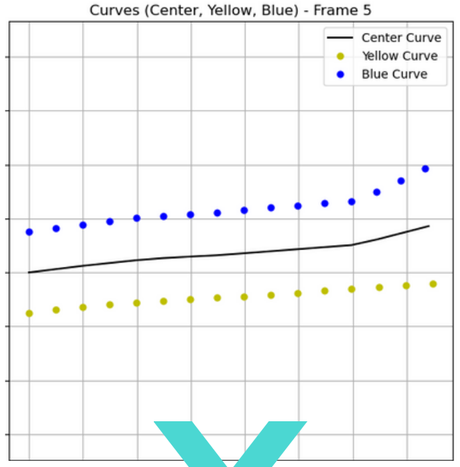
π/12 = 15°

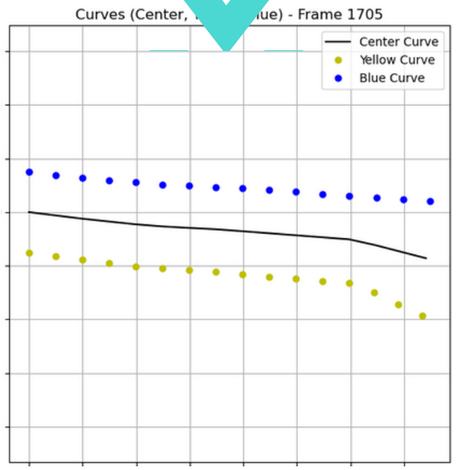


π/8 = 22°

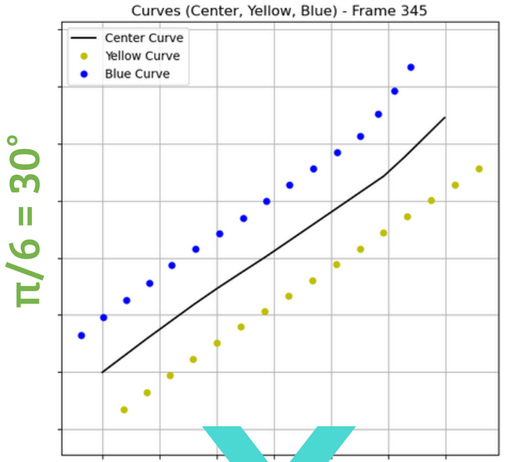
...and flip!

Original

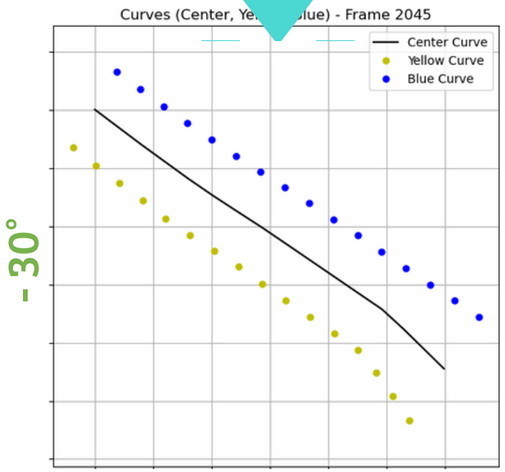




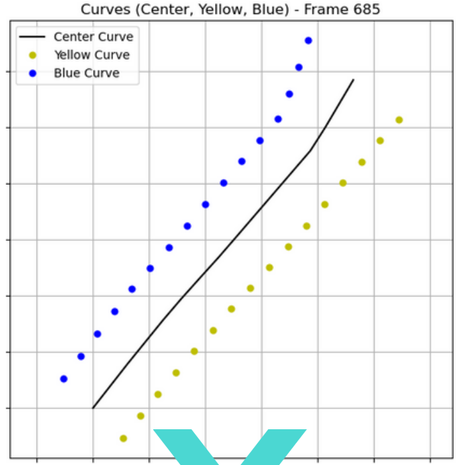
π/6 = 30°



- 30°

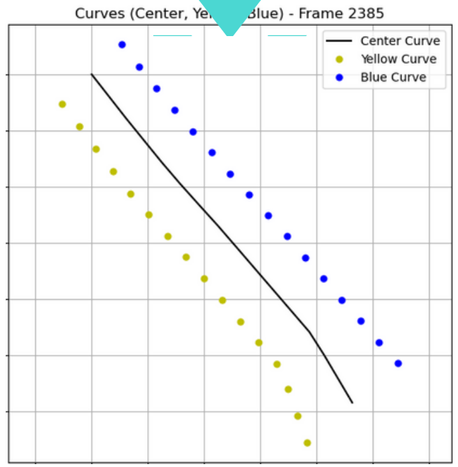


π/4 = 45°



(...and so on)

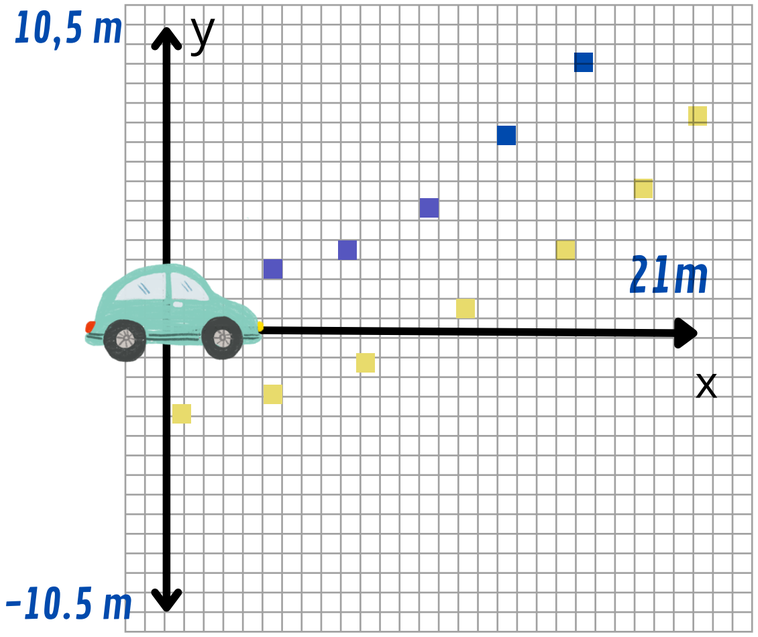
-45°



Transformation to Grid

We decided to create two grids for each individual frame

- The first grid combines the positions of the **blue cones** and **yellow cones**
- The second grid contains the data for the **blue curve**, **yellow curve**, and **center curve**, representing the trajectory and boundaries.



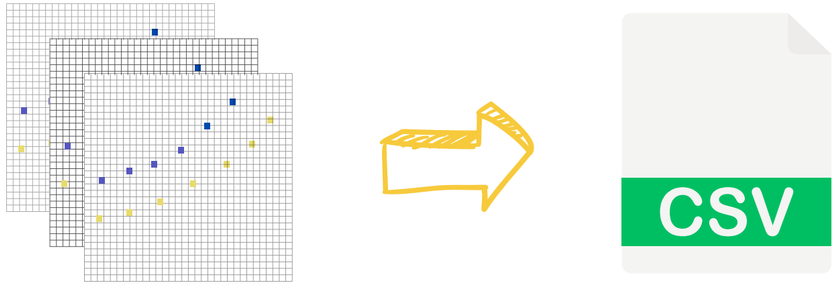
Grid resolution = 0.3 m
Grid size = 70 pixels
Horizon = 21 m

$$x_{grid} = \frac{x}{grid\_resolution}$$

$$y_{grid} = \frac{10.5 - y}{grid\_resolution}$$

Grid to CSV

The data defined through the grid was transcribed into a CSV file, where each row represents a frame from the dataset and the cells contain the values of the flattened grid.

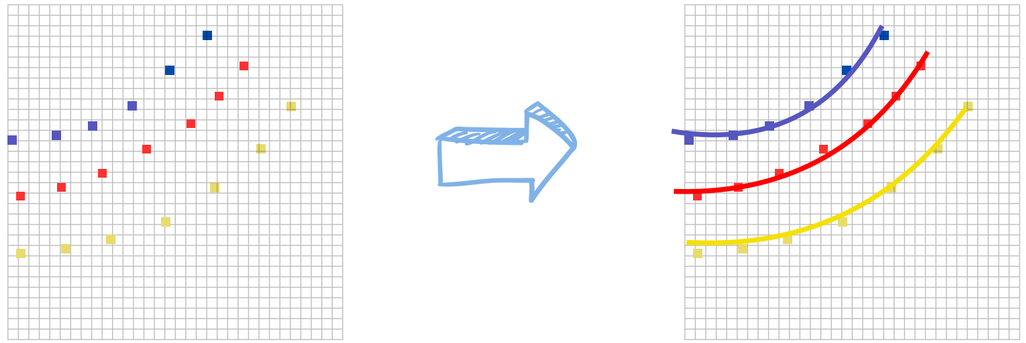


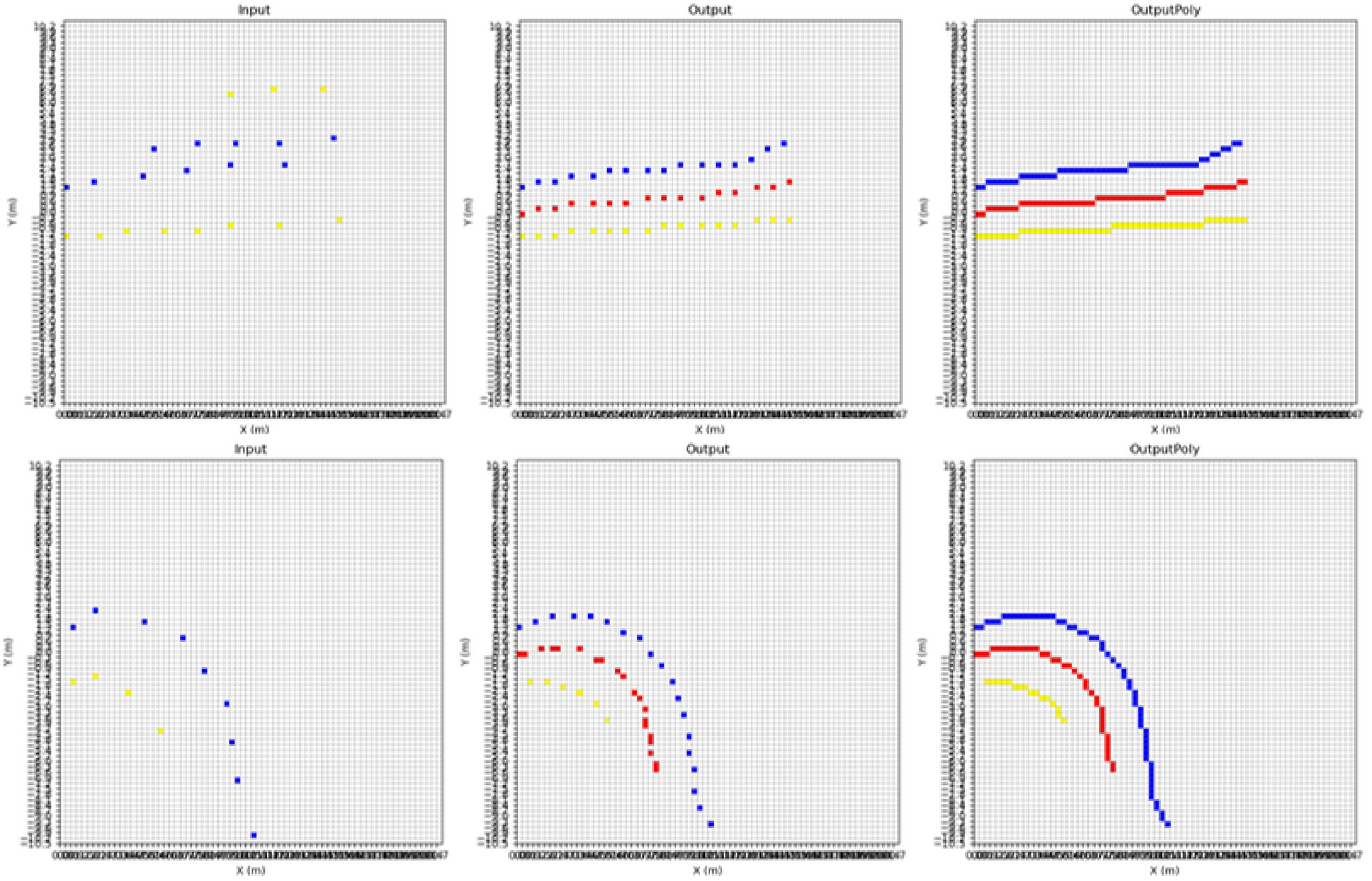

```python
def draw_curve(grid, curve_data, curve_value, thickness=1):
    points = []
    for cone in curve_data:
        x, y = cone
        grid_coords = map_to_grid(x, y, grid_size, grid_resolution)
        if grid_coords is not None:
            points.append(grid_coords)

    if points:
        points = np.array(points, dtype=np.int32)
        # Disegna la curva con polylines
        cv2.polylines(grid, [points], isClosed=False, color=curve_value, thickness=thickness)
```

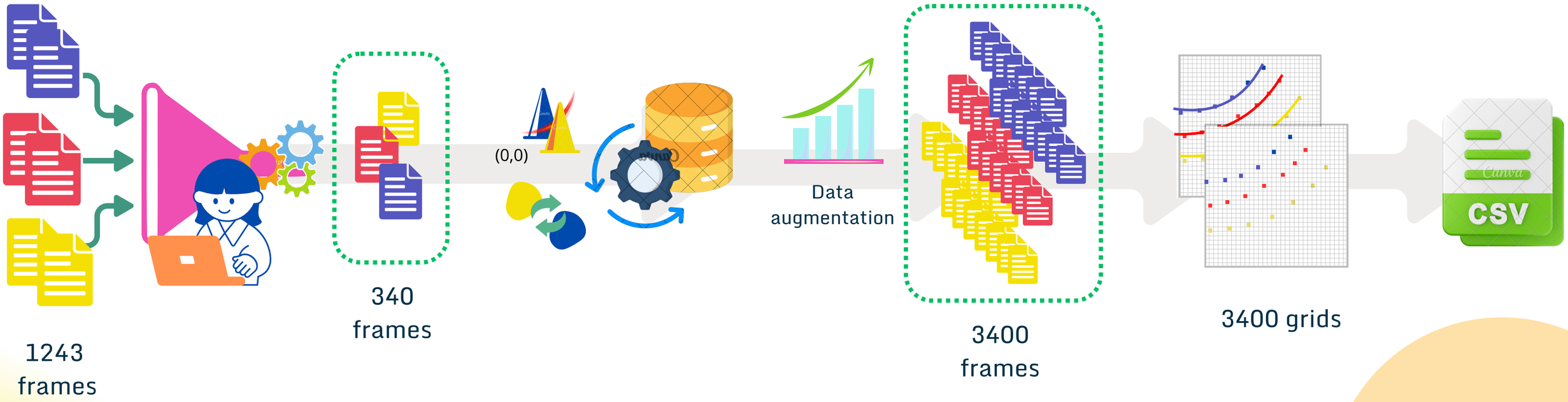
Polyline for curves

To represent the curves in the data (blue, yellow, and center), we used the ***polylines function*** from OpenCV.




Input
Output
OutputPoly
Y (m)
X (m)
Input
Output
OutputPoly
Y (m)
X (m)

Data Processing Workflow Summary



1243
frames

340
frames

3400
frames

3400 grids

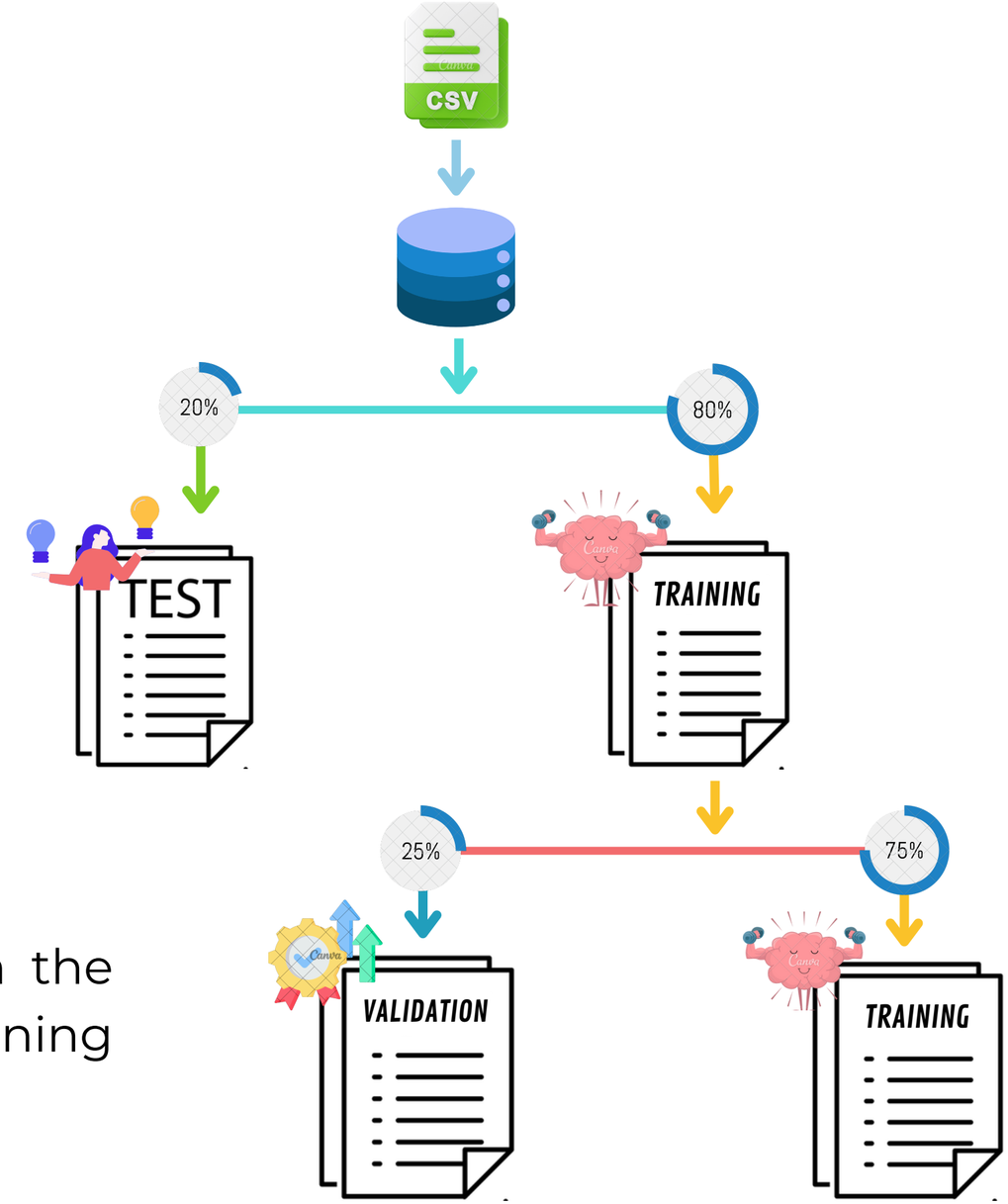
Dataset split

From the **CSV file**, we extract the data representing our grids and divide them as follows:

Training set, used to train the network

Test set, used to evaluate the performance of the network

Additionally, a **validation set** is extracted from the training set to monitor performance during training and fine-tune the model.



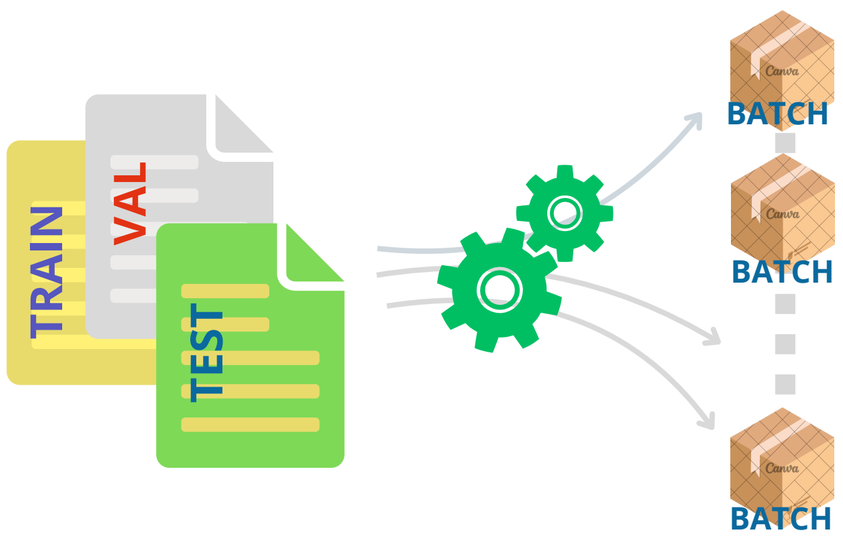
Pytorch and tensor transformation

PyTorch is an open-source machine learning library widely used for developing and training deep learning models. It offers an extensive ecosystem of tools and libraries that simplify tasks such as data preprocessing, model building, and deployment.

Preprocessing and transformation of the grid data into tensors

RESHAPE INTO 70X70 MATRICES

CONVERSION INTO THREE-CHANNEL TENSORS



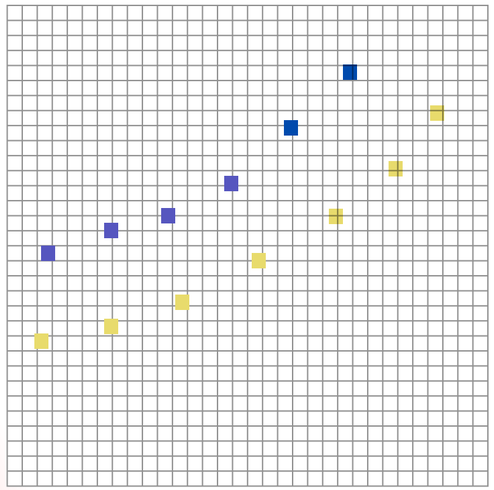
Then, these tensors were split into training, validation, and test sets, and organized into PyTorch DataLoader objects to facilitate batch processing and efficient data loading during model training.

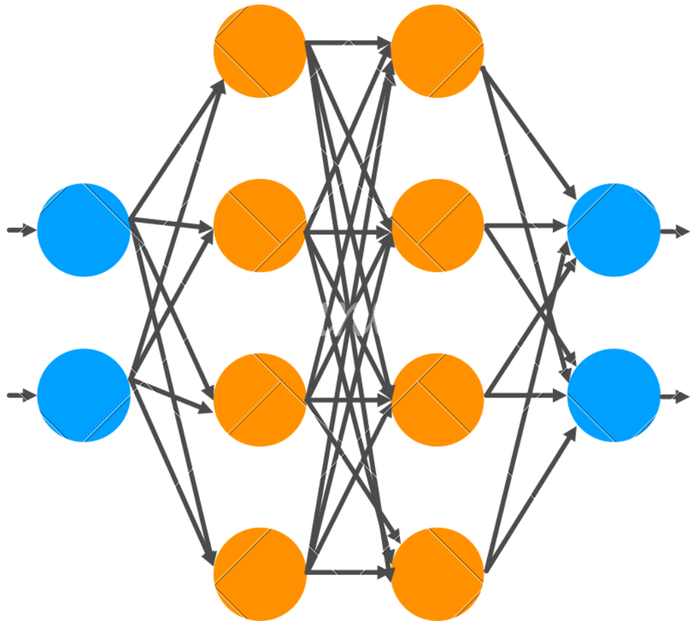
Types of Networks Evaluated

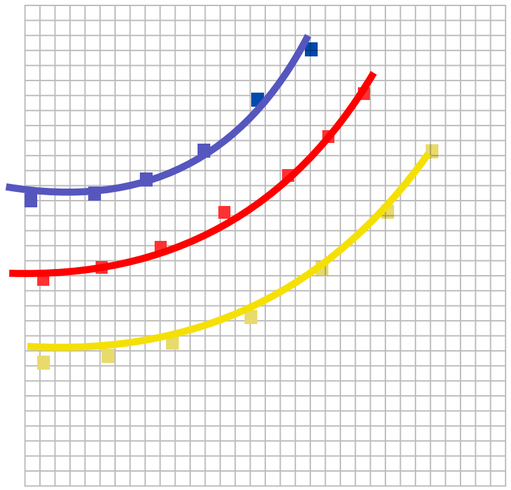
1. Convolutional Network
2. Encoder-Decoder Network
3. UNet Network

WHY?

The input size must be equal to the output size.

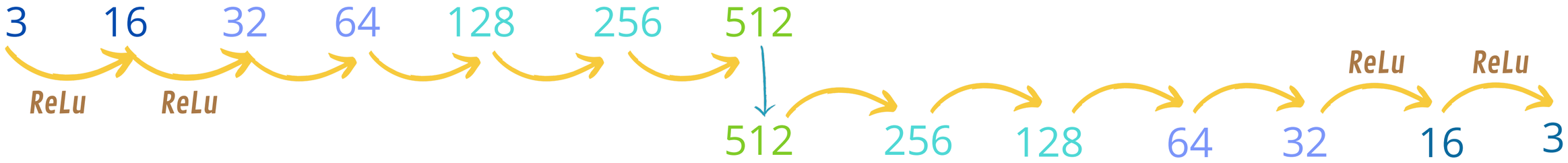






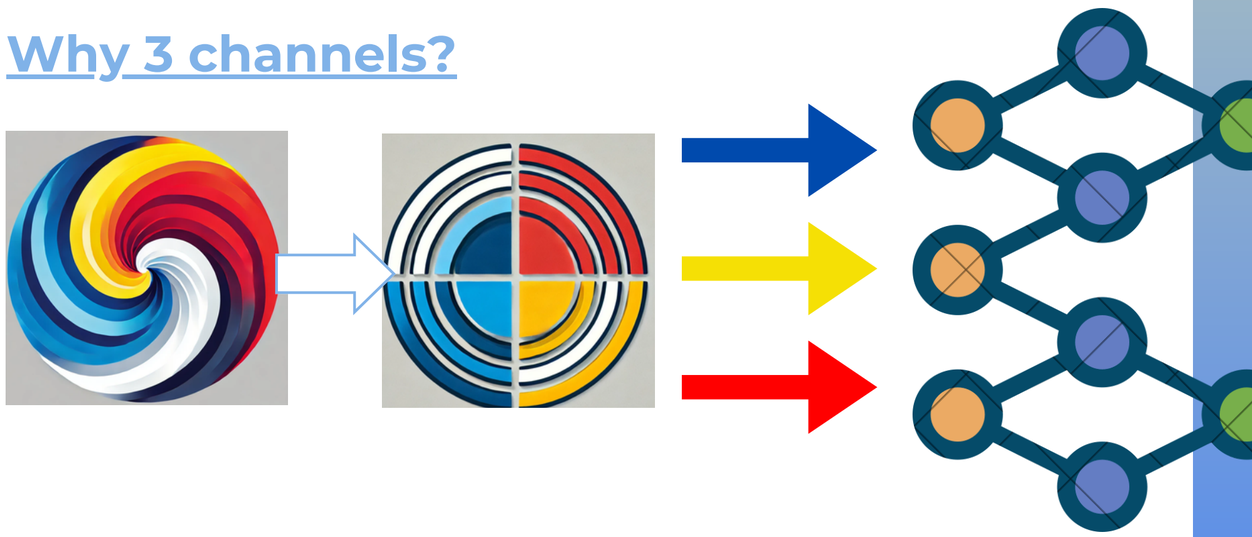
Convolutional Network

The network has 13 2D convolutions (nn.Conv2d), each with **kernel_size=3** and **padding=1**

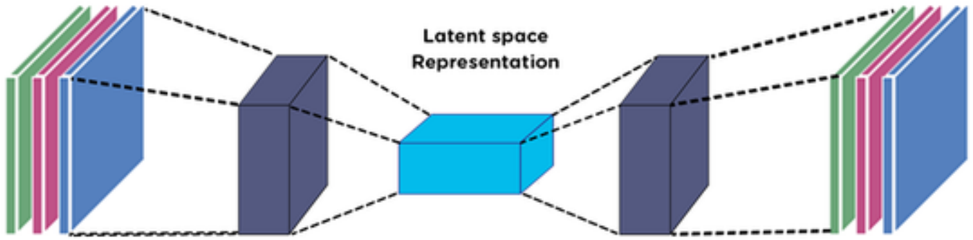


After each convolution (except the last one), a ReLU activation function is applied.
It introduces non-linearity, enabling the network to learn more complex functions.

Why 3 channels?

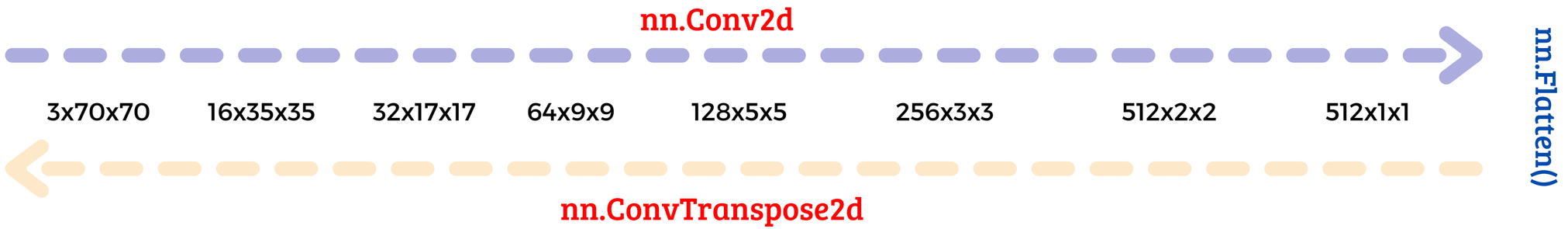


2 Encoder Decoder



It takes a grid and first “compresses” it into a smaller, more meaningful representation. It then “rebuilds” the grid from that summary, restoring it to the original size.
This approach allows for both efficient data handling and the ability to recreate the main visual content from a compressed form.

Encoder: thanks to the convolutions (**Conv2d**), the number of channels increases, allowing the extraction of increasingly abstract and complex features from the grid.



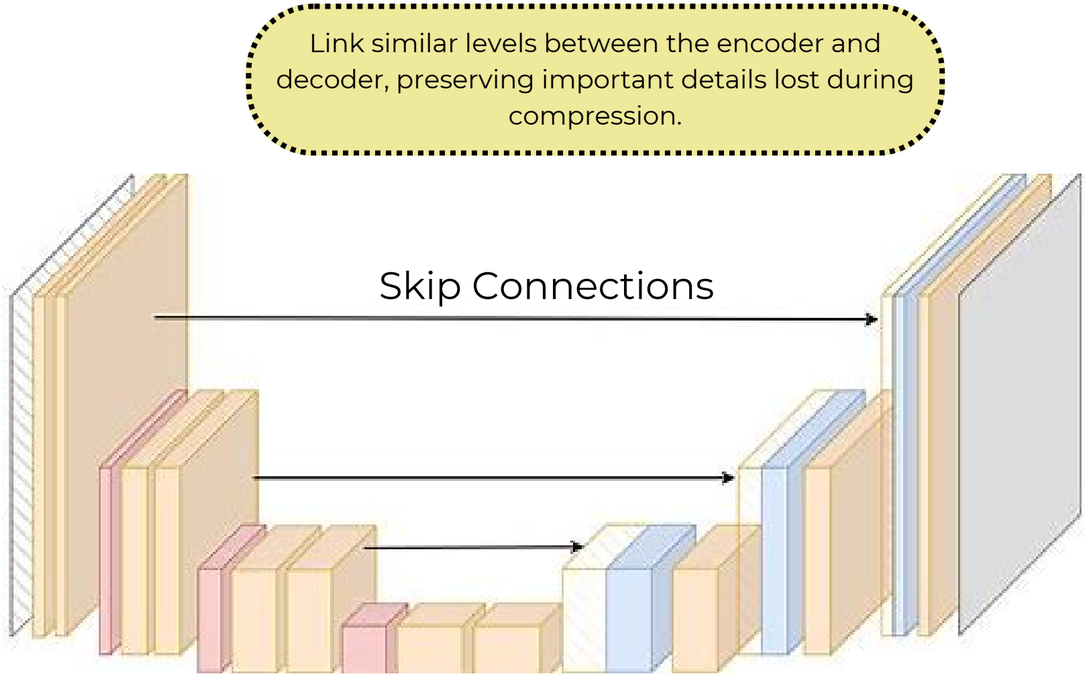
Decoder: the number of channels is reduced using transposed convolutions (**ConvTranspose2d**), resulting in a grid with the same number of channels as the input.

3 U-Net Network

It has a "U" shape with two paths:

- **Encoder** (contraction path): compresses information through convolutional and pooling layers, reducing the grid size and extracting relevant features.
- **Decoder** (expansion path): reconstructs the grid and, thanks to skip connections, receives information directly from the corresponding encoder levels.

During the contraction path, the input image is progressively reduced in height and width while increasing the number of channels, allowing the network to capture high-level features.



- Leverages a **smaller amount of data.**
- Maintains **speed** and **accuracy**.
- Spoiler: it achieves the **best results**. 🤫

We are ready for training!

Mean Squared Error (MSE) Loss

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

This value measures the difference between the predicted value and the true value. It penalizes large errors more heavily due to the squaring operation.
It produces a non-negative value (0 is perfect).

ADAM optimizer

Adam is an optimization algorithm that combines ideas from RMSProp and SGD con Momentum, using first and second moment estimates of gradients to adapt the learning rate for each parameter.

Other parameters:

Min Channels: 3
Max Channels: 512
Seed: 42
Patience: 3
Batch size: 32

Let's compare the results!

MODEL	Final training loss	Final validation loss	Test loss	Epoch stop
CONVOLUTIONAL NETWORK	0.0039	0.0039	0.0040	25
ENCODER DECODER NETWORK	0.0048	0.0054	0.0055	44
UNET	0.0024	0.0035	0.0036	27

Test metrics

In addition to the Mean Squared Error (MSE), we have also introduced other metrics to evaluate the results on the test set.

Mean Absolute Difference

$$\mathrm{MAD} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

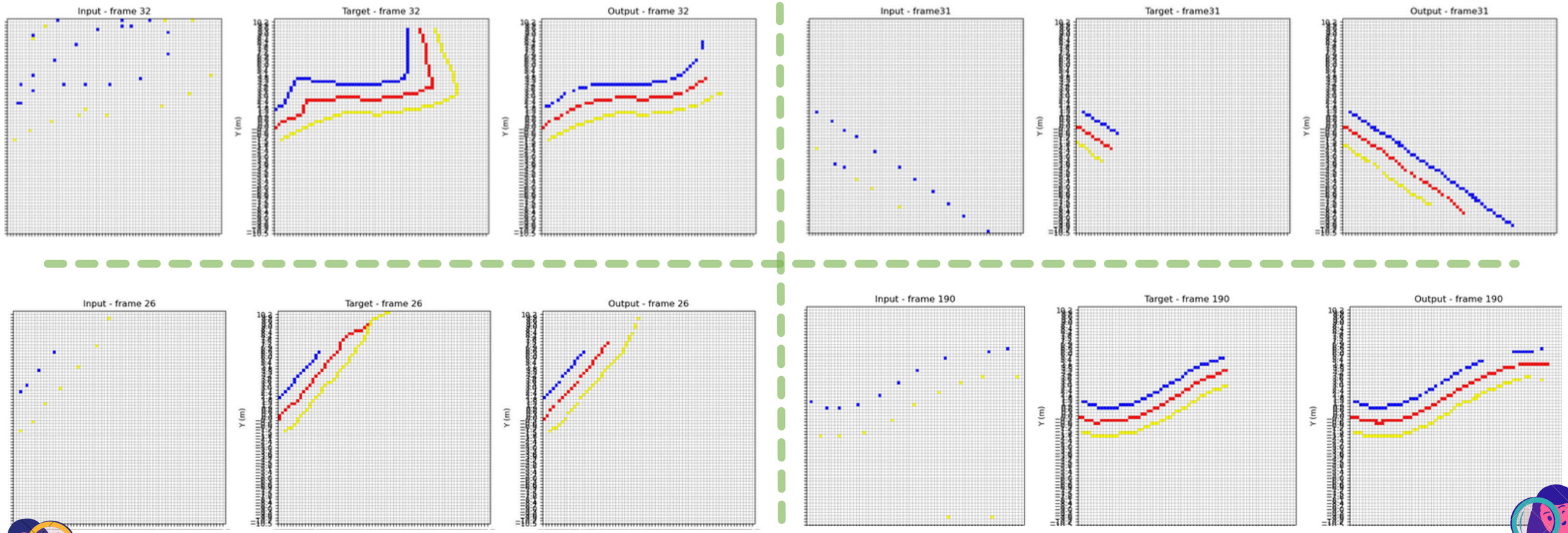
The MAD measures the average of the absolute values of the differences between the predicted values and the actual values.

Category-specific Recall

$$Recall = \frac{True\ Positive}{Total\ Positive} \times 100$$

Percentage of how well a certain category has been correctly classified.
To better evaluate the model's behavior, we exclude zeros, which, being numerically dominant, would have distorted the overall results.

This is why visual comparison remains the best metric!



Our model sometimes gets results that are better than the target output...
that's why classic evaluation metrics are not entirely reliable

Test Phase

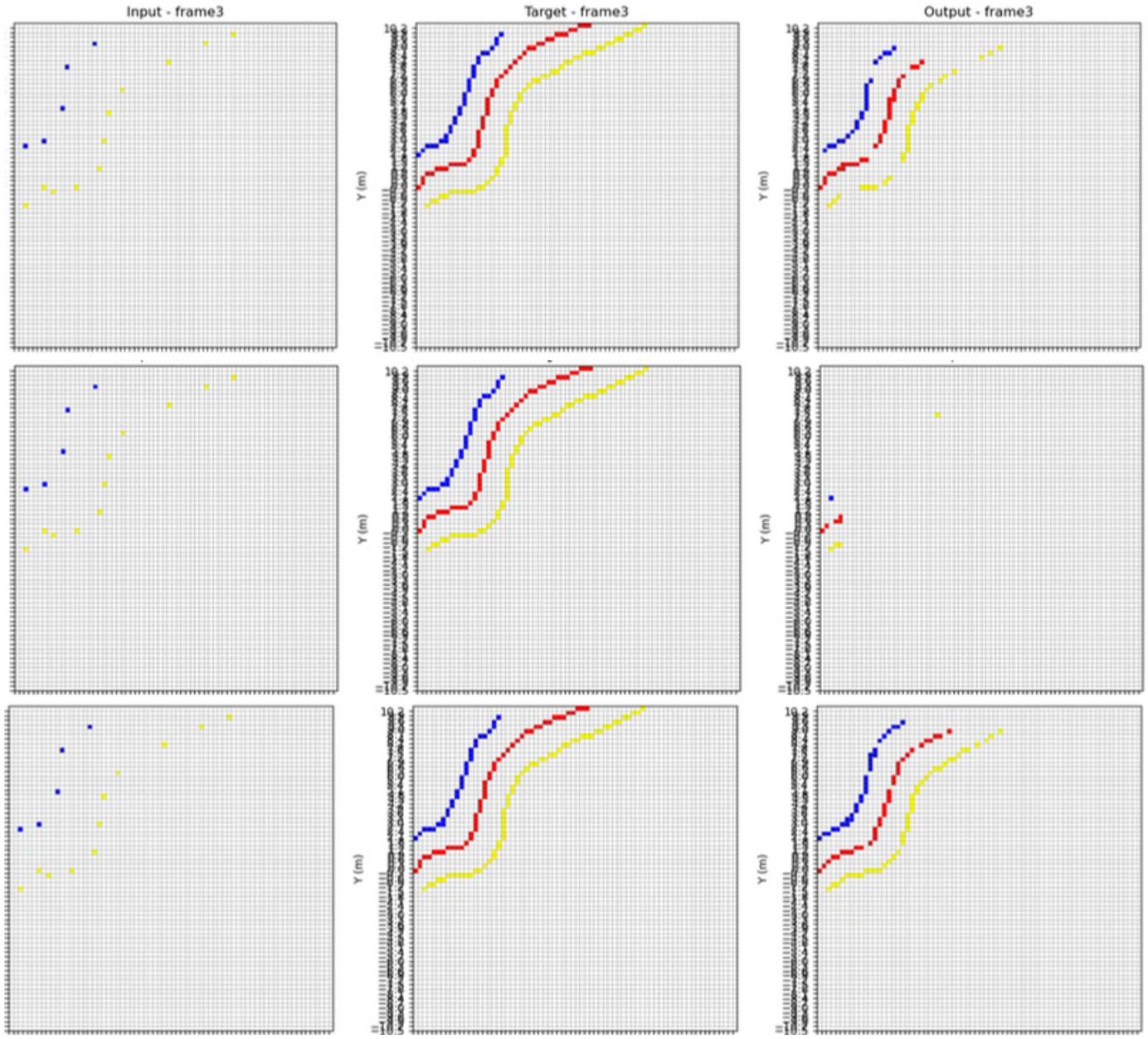
MODEL	MSE	MAD	RECALL BLUE	RECALL YELLOW	RECALL RED
CONVOLUTIONAL NETWORK	0.0124	0.0138	59.1743	56.7067	61.2069
ENCODER DECODER NETWORK	0.0158	0.0174	26.6000	26.8024	36.4410
UNET	0.0108	0.0120	69.6783	69.0913	65.3776

Comparison between the Nets

Convolutional Network

Encoder Decoder Network

UNet Network

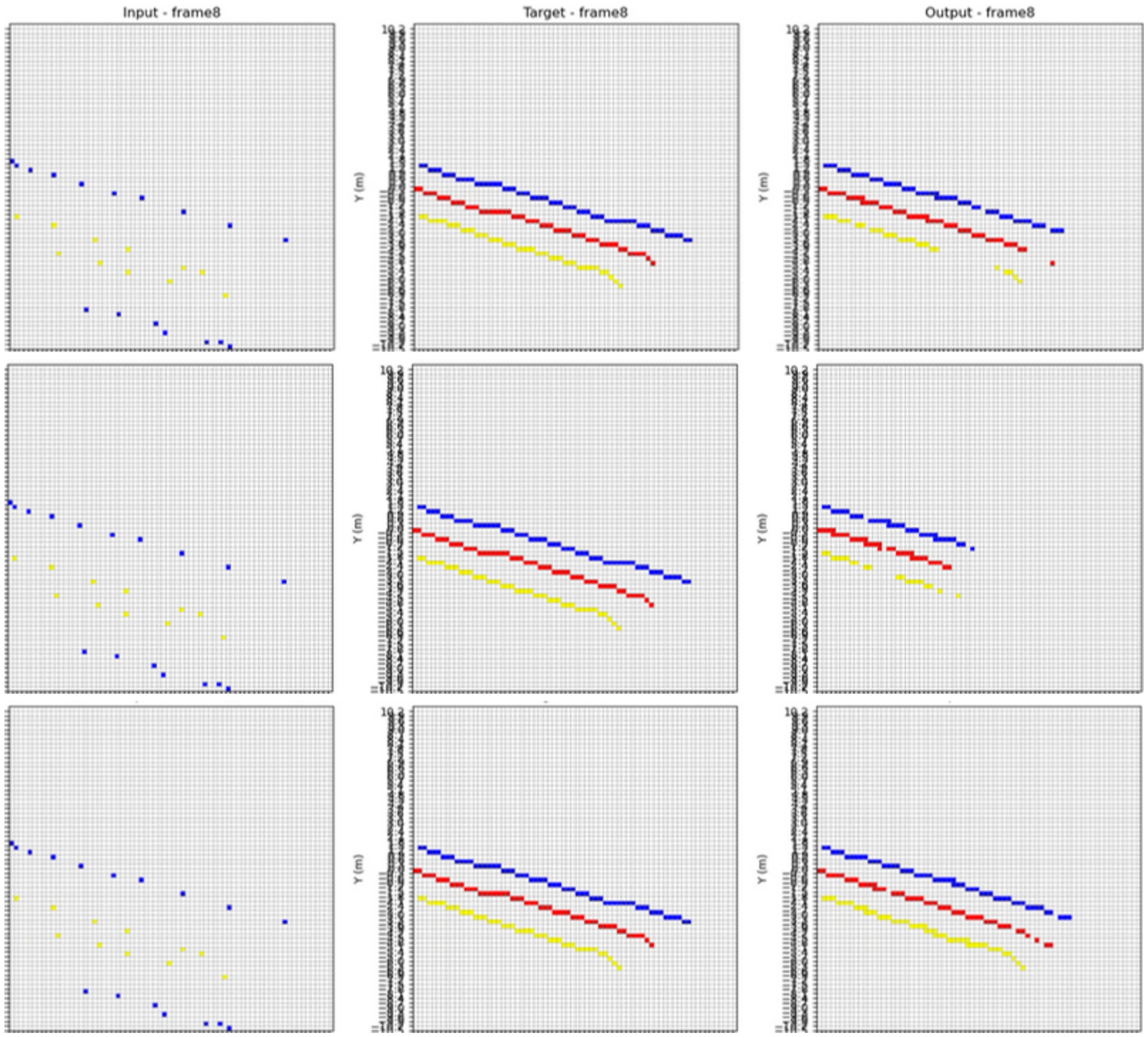


Comparison between the Nets

Convolutional Network

Encoder Decoder Network

UNet Network

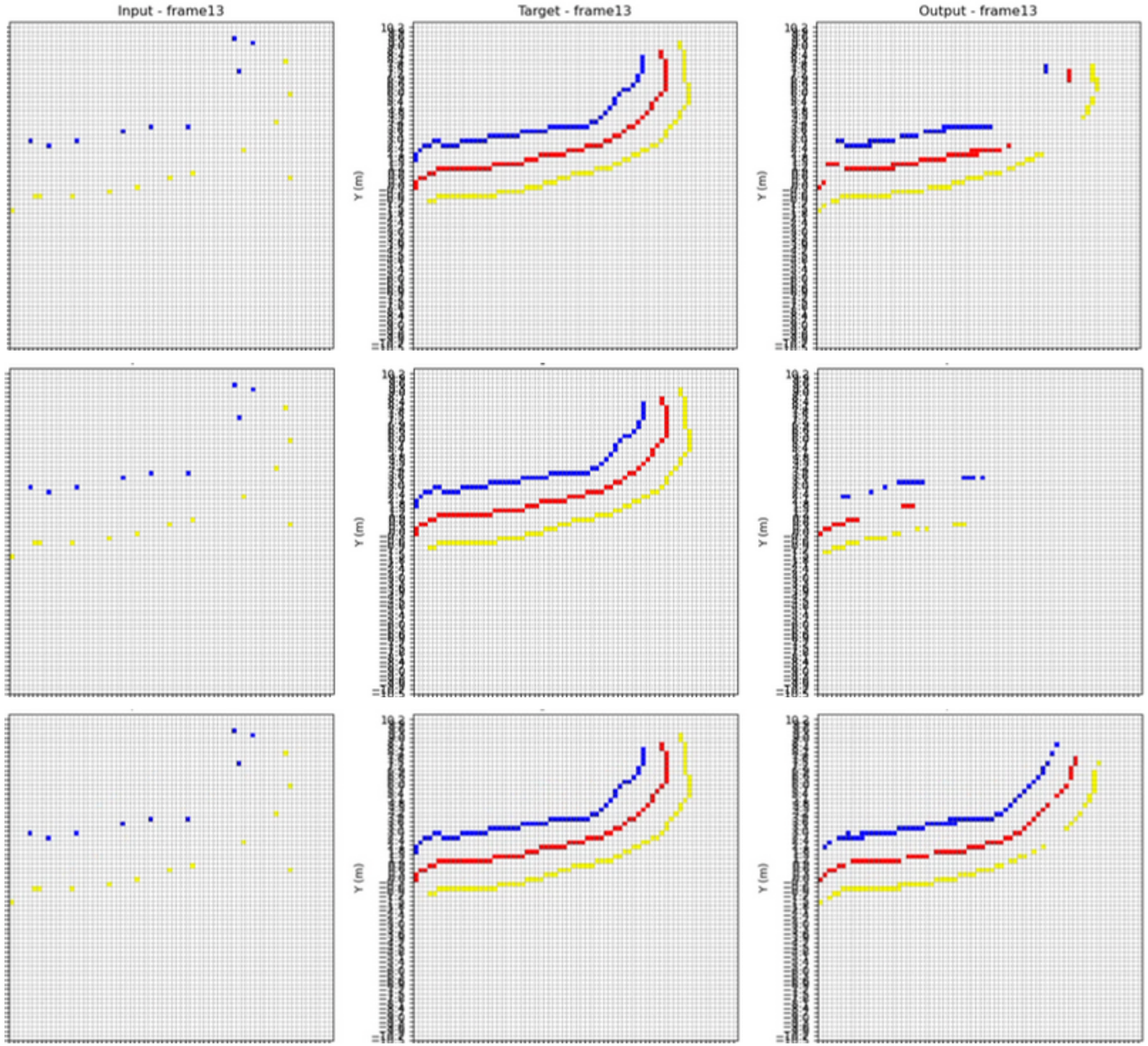


Comparison between the Nets

Convolutional Network

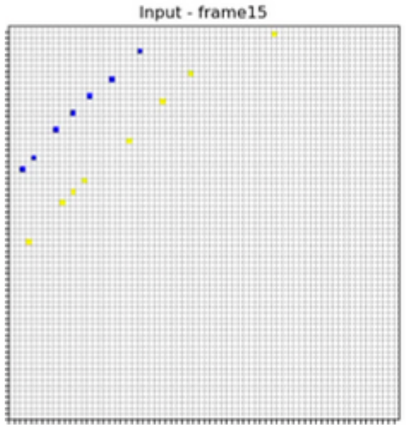
Encoder Decoder Network

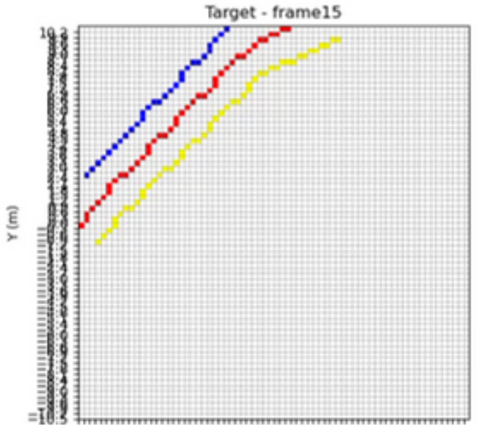
UNet Network

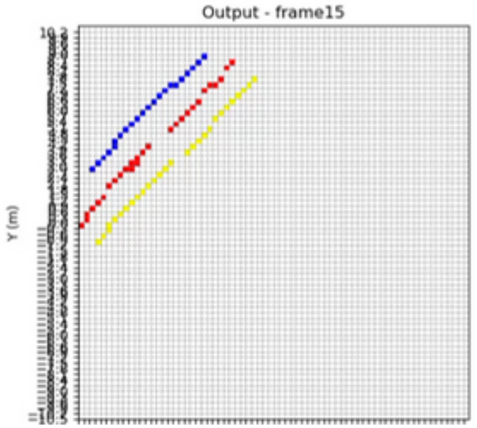


Comparison between the Nets

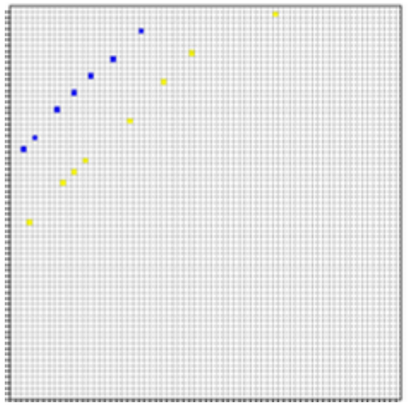
Convolutional Network

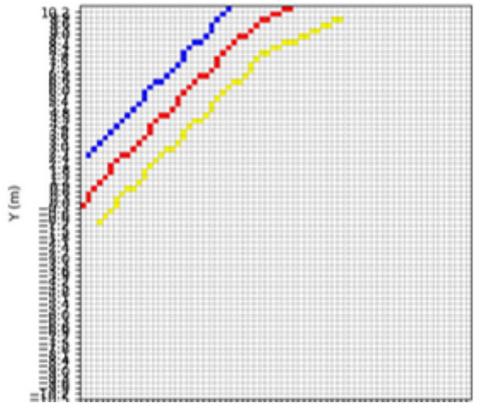


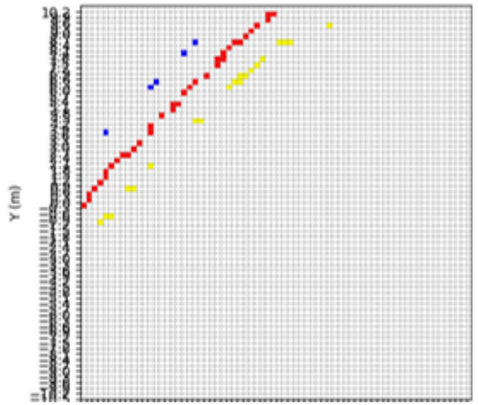




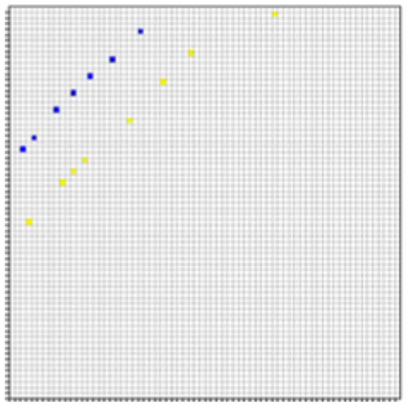
Encoder Decoder Network

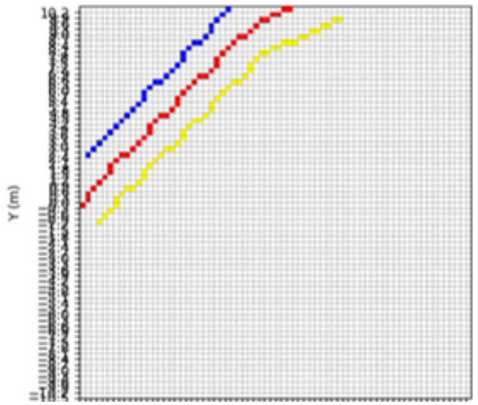


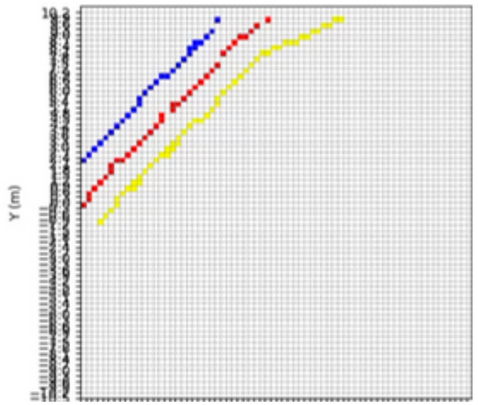




UNet Network





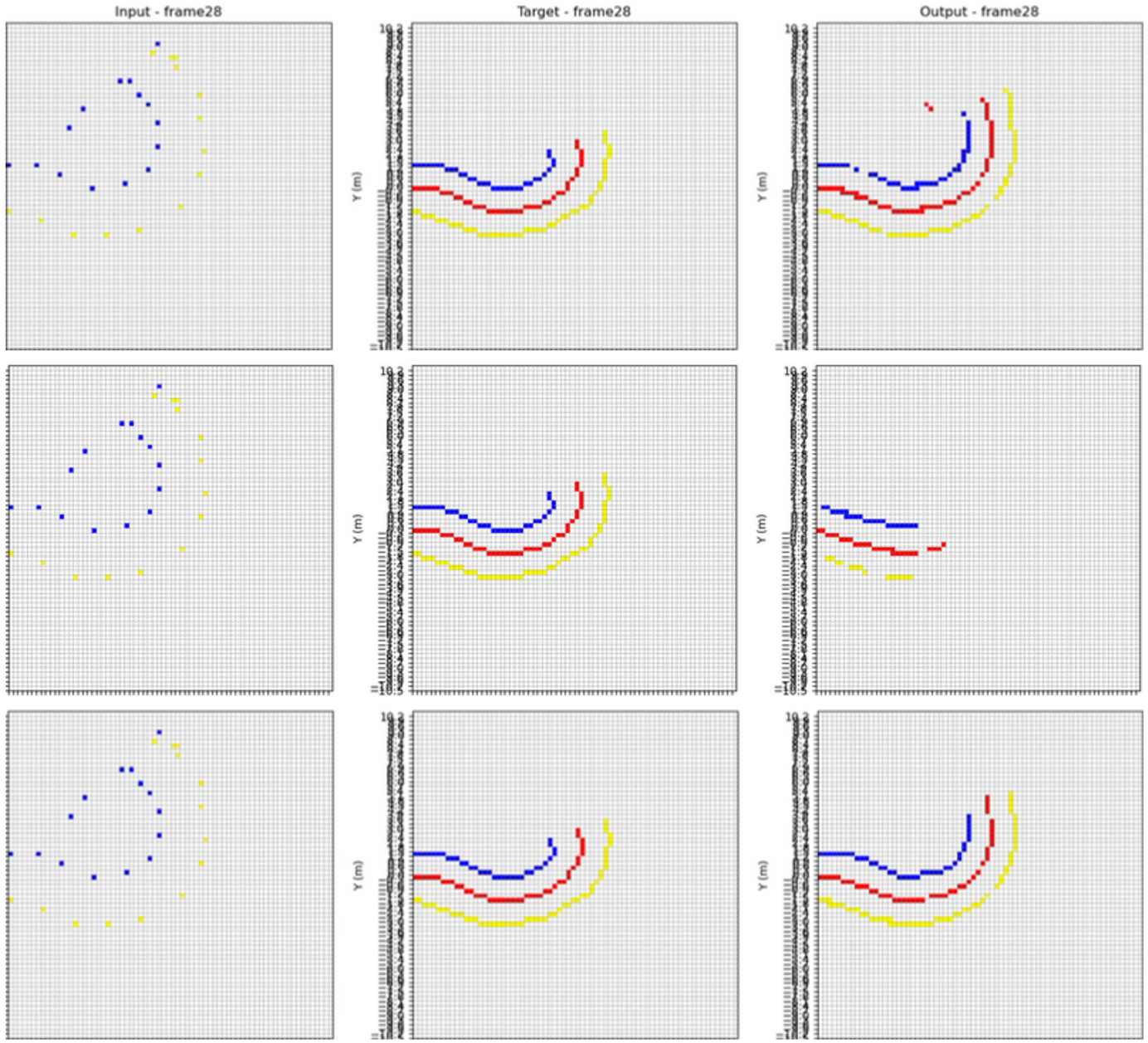


Comparison between the Nets

Convolutional Network

Encoder Decoder Network

UNet Network



Test Phase

MODEL	MSE	MAD	RECALL BLUE	RECALL YELLOW	RECALL RED
CONVOLUTIONAL NETWORK	0.0124	0.0138	59.1743	56.7067	61.2069
ENCODER DECODER NETWORK	0.0156	0.0174	26.6000	26.8024	36.4410
UNET	0.0108	0.0120	69.6783	69.0913	65.3776

UNET is the best network in all aspects !

U-Net network tests with parameter variations

In the analysis of the U-Net network performance, we conducted a series of tests by varying key parameters to evaluate their impact on learning capacity and prediction quality.

What have we changed?

- **Patience**: from 3 to undefined
- **Loss**: from MSE to L1 smooth
- **Batch size**: from 32 to 16
- **Number for max channels**: 512 to 1024
- **Seed**

Changes of patience

Patience is a parameter of early stopping that determines how many epochs without improvements can be tolerated before halting the training, thus preventing overfitting.

	UNET PATIENCE = 3	UNET PATIENCE = 10	WITHOUT PATIENCE
FINAL TRAINING LOSS	0.0024	0.0016	0.0002
FINAL VALIDATION LOSS	0.0035	0.0035	0.0039
TEST LOSS	0.0036	0.0036	0.0036
EPOCH STOP	27	35	75

Worse training loss values but generalize well on validation and test data, avoiding overfitting.

Overfitting

Lower training loss

Increase validation loss

Changes of Batch size

The **batch size** is the number of samples processed by the model in a single iteration during training.

- A smaller batch size improves generalization
- A larger batch size stabilizes training, but may reduce generalization ability.

	unet 32	unet 16
FINAL TRAINING LOSS	0.0024	0.0026
FINAL VALIDATION LOSS	0.0035	0.0034
TEST LOSS	0.0036	0.0035
EPOCH STOP	27	21
MSE	0.0108	0.0104
MAD	0.0120	0.0115
RECALL BLUE	69.6783	68.9216
RECALL YELLOW	69.0913	70.0410
RECALL RED	65.3776	68.0199

Changes of Loss

	UNET WITH MSE LOSS	UNET WITH L1-SMOOTH
FINAL TRAINING LOSS	0.0024	0.0018
FINAL VALIDATION LOSS	0.0035	0.0020
TEST LOSS	0.0036	0.0019
EPOCH STOP	27	14
MSE	0.0108	0.0117
MAD	0.0120	0.0130
RECALL BLUE	69.6783	55.0180
RECALL YELLOW	69.0913	58.9957
RECALL RED	65.3776	68.0683

Smooth L1 Loss is a loss function that combines the advantages of MSE (Mean Squared Error) and MAD (Mean Absolute Difference):

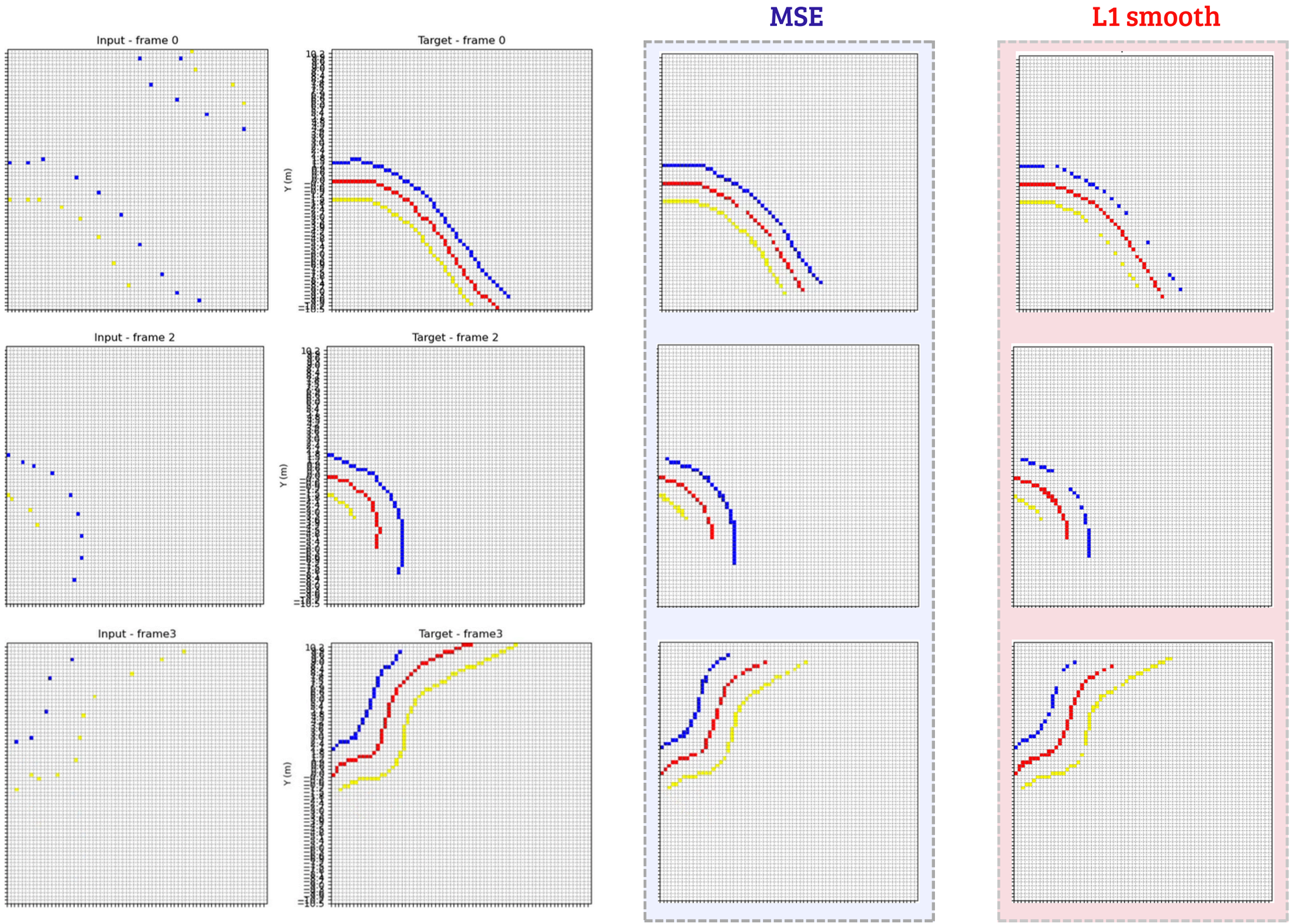
- For small errors, it behaves like MSE, ensuring stability.
- For large errors, it behaves like MAD, reducing the impact of outliers.

$$\text{Smooth L1 Loss}(x) = \begin{cases} 0.5 \cdot (y_i - \hat{y}_i)^2 & \text{se } |y_i - \hat{y}_i| < \beta \\ \beta \cdot |y_i - \hat{y}_i| - 0.5 \cdot \beta^2 & \text{altrimenti} \end{cases}$$

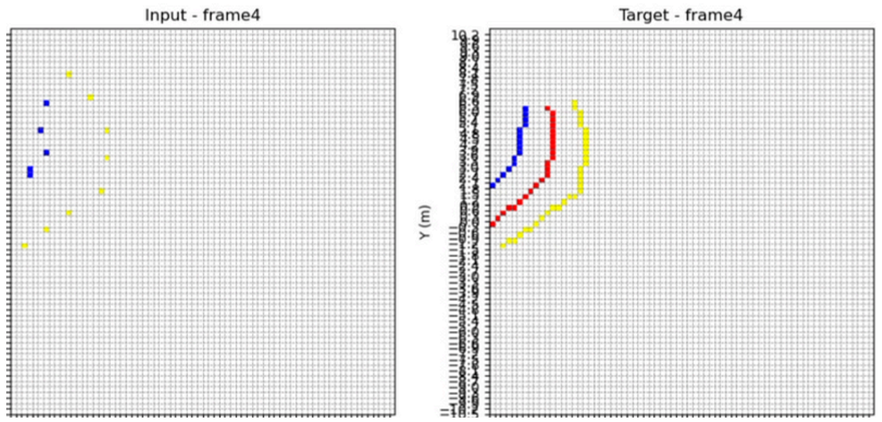
L1-Smooth offers faster convergence and lower losses, but it may sacrifice recall for some categories.

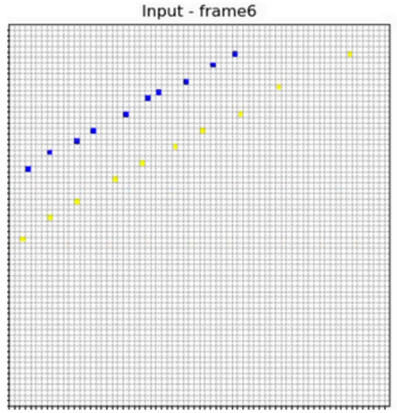
Let's compare visually!

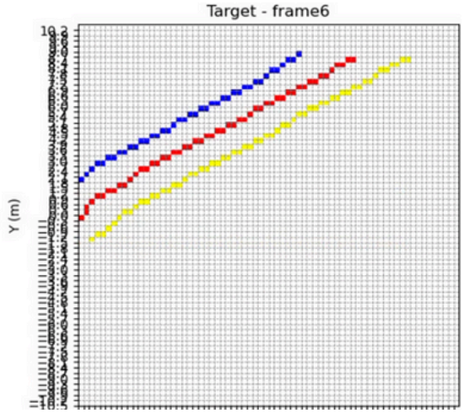
Comparison between the Nets

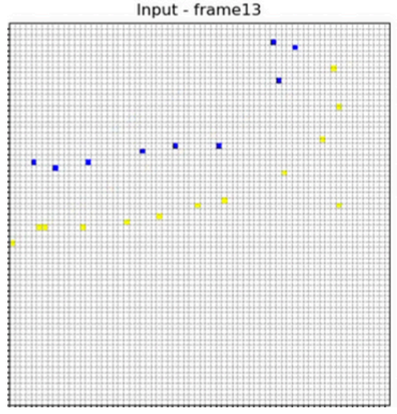


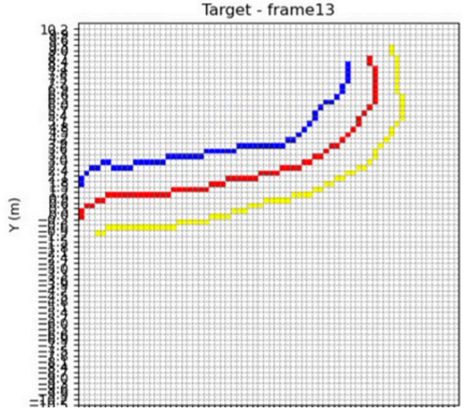
Comparison between the Nets



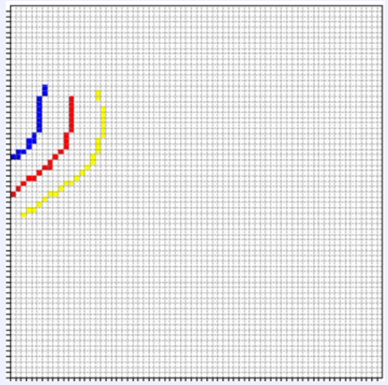


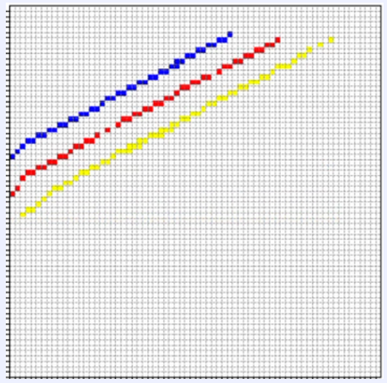


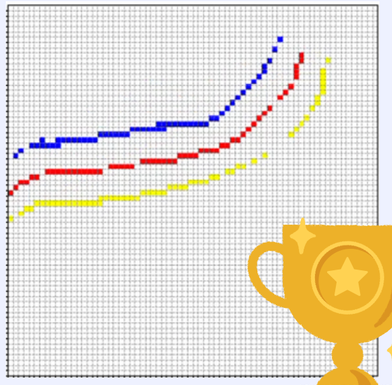




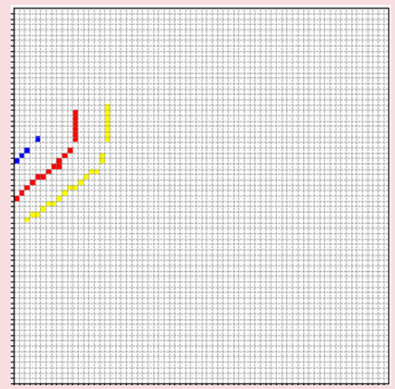
MSE

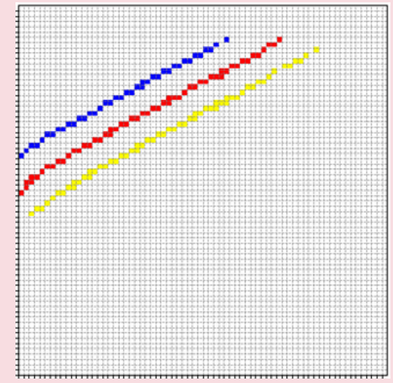


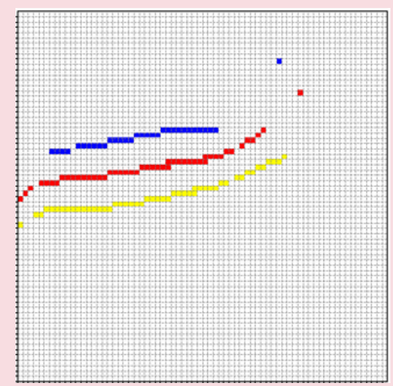




L1 smooth







Changes of Number for max channels:

	unet 512	unet 1024
FINAL TRAINING LOSS	0.0024	0.0031
FINAL VALIDATION LOSS	0.0035	0.0035
TEST LOSS	0.0036	0.0036
EPOCH STOP	27	24
MSE	0.0108	0.0109
MAD	0.0120	0.0121
RECALL BLUE	69.6783	66.0140
RECALL YELLOW	69.0913	69.3015
RECALL RED	65.3776	67.3898

Channels in a convolutional neural network represent different features or filters that the model learns.

Increasing the number of channels allows the model to capture more details and improve performance, but it also requires more memory and computation, **increasing the risk of overfitting** and slowing down training.

Changes of Seed

	seed 42	seed 12	seed 27	seed 79	seed 61
FINAL TRAINING LOSS	0.0024	0.0025	0.0037	0.0028	0.0025
FINAL VALIDATION LOSS	0.0035	0.0035	0.0040	0.0035	0.0037
TEST LOSS	0.0036	0.0035	0.0040	0.0035	0.0034
EPOCH STOP	27	28	14	22	26
MSE	0.0108	0.0102	0.0120	0.0105	0.0101
MAD	0.0120	0.0114	0.0134	0.0117	0.0112
RECALL BLUE	69.6783	66.9696	54.5777	64.0399	65.0581
RECALL YELLOW	69.0913	70.3615	58.9845	62.4012	68.8109
RECALL RED	65.3776	68.0475	59.4370	67.1488	66.8384

Changing the **seed** allows you to test the variability in the model's performance and assess its robustness.

If the performance is stable across different seeds, it means the model is generalizing well.
If it varies significantly, you may need to take action to improve generalization.

Test made with:

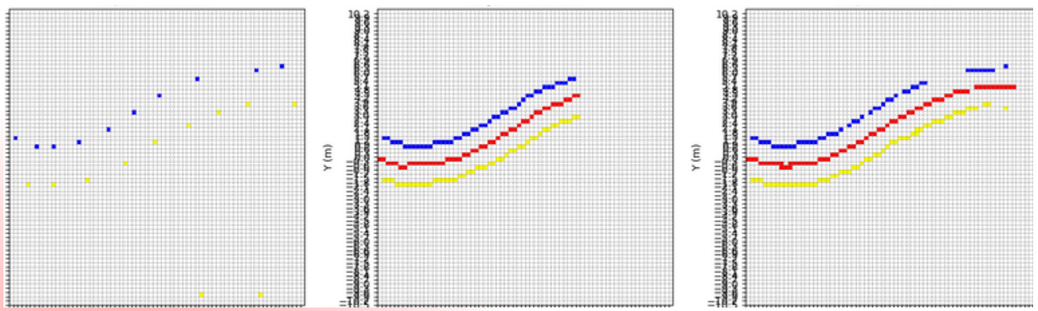
- Loss: MSE
- Max channels: 512
- Batch size: 32
- Patience: 3

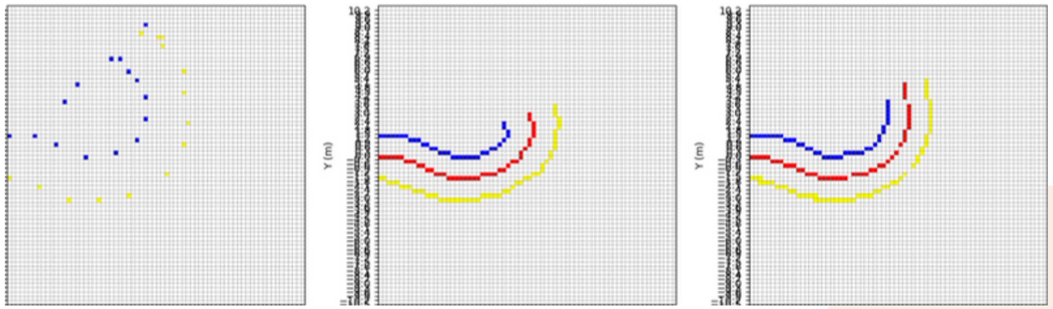
Average of the best combination

Loss: **MSE**
Max channels: **512**
Batch size: **32**
Patience: **3**

Final training loss	Final validation loss	Test loss	Epoch stop
0.0027	0.0036	0.0036	24

MSE	MAD	RECALL BLUE	RECALL YELLOW	RECALL RED
0.0107	0.0109	64 %	66%	65.5 %





Conclusions

Limits

Few data with some mistakes

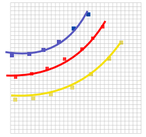
Current metrics are unrealistic; visual evaluation is the most concrete, but subjective.

Improvements

Make the dataset larger, more diverse and of higher quality.

New network architectures and new parameter combinations.

More objective and reliable evaluation metrics, reducing the reliance on visual analysis.



Add the polyline to fill the gap in the output.