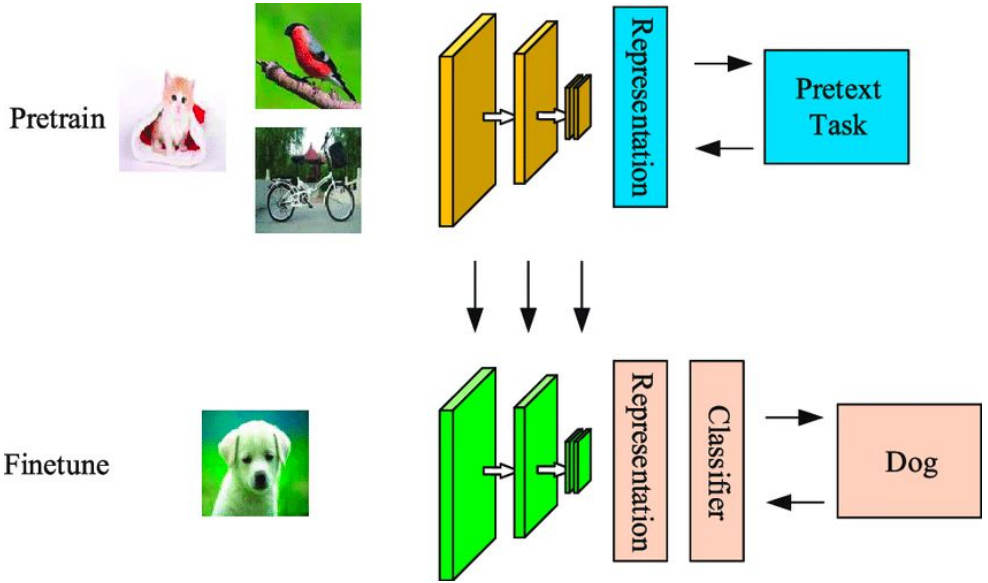
Self-supervised feature extraction for ball detection in nao robots

Can Lin, **Marco** Zimmatore,
Marco Volpini, **Penelope** Malaspina

Fundamentals of Data Science
Sapienza University of Rome

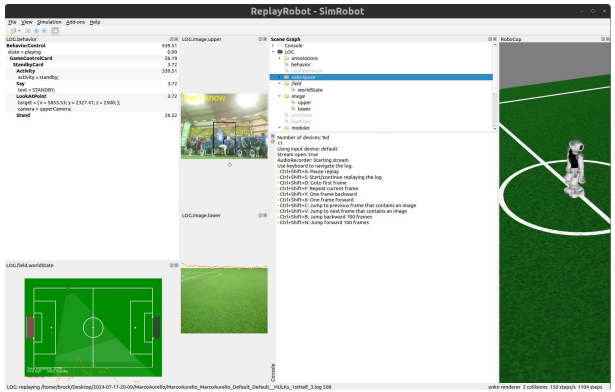
WHAT IS SELF-SUPERVISED LEARNING???

Self-supervised learning is a machine learning paradigm where models learn from unlabeled data by solving pretext tasks that generate supervisory signals from the data itself.

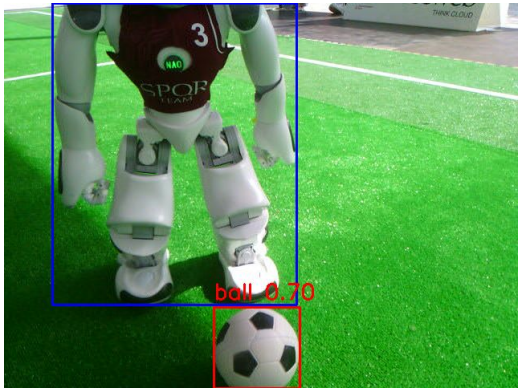


DATASET CREATION

The process for creating our dataset involved the following steps:



Retrieving Data: extracting images from the robots' logs using a specialized tool.

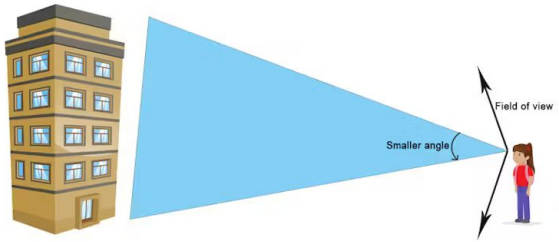


Initial Labeling: using YOLO-World to generate bounding boxes around the ball and robots.

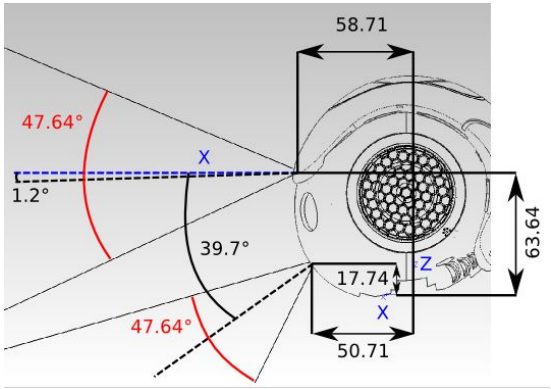
Manual Annotation: reviewing and refining the labels using LabelImg for accuracy.

PATCH EXTRACTOR

To accommodate the NAO robot's limited computational power, we focus on processing 32x32 patches. However, scaling the entire image would result in significant feature loss. Instead, we extract patches of varying scales—32x32, 64x64, and 128x128—ensuring the ball is properly represented at different distances in the image.

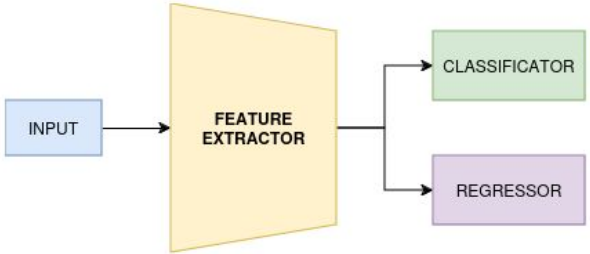


MOTHERBOARD		
CPU	CPU processor	ATOM E3845
	Cache memory	2 MB
	Clock speed	1.91 GHz
RAM	4 GB DDR3	
Flash memory	32 GB eMMC	



PATCHES EXAMPLE

ORIGINAL MODEL



Circular DIoU Loss (DIoU):

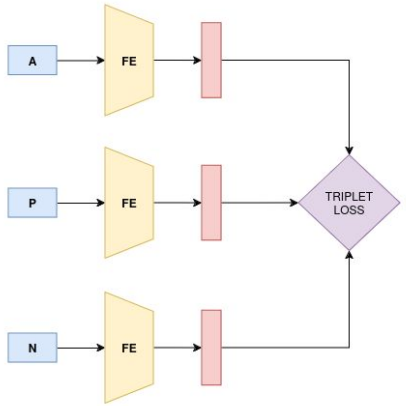
$$\mathrm{DIoU} = 1 - \frac{A_{\mathrm{int}}}{A_{\mathrm{union}}} + D$$

$$\mathrm{BCEWithLogits}(z, y) = \max(z, 0) - z \cdot y + \log\left(1 + e^{-|z|}\right)$$

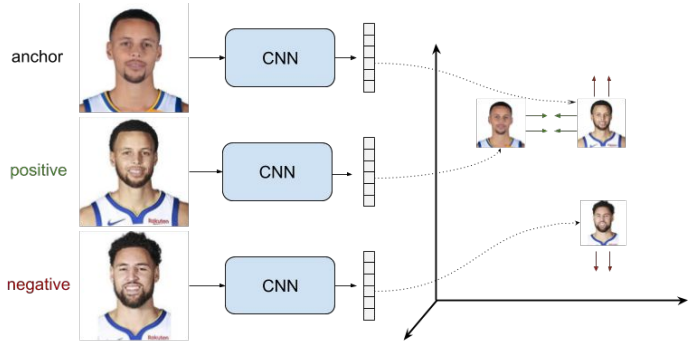
```
You, 24 hours ago | 1 author (You)
class BallPerceptor(nn.Module):
    def __init__(self, backbone_size = 9):
        super(BallPerceptor, self).__init__()

        self.conv1 = DepthwiseSeparableConv(1, 8, stride=1)
        self.conv2 = DepthwiseSeparableConv(8, 16, stride=1)
        self.conv3 = DepthwiseSeparableConv(16, 32, stride=1)

        backbone_layers = []
        for _ in range(backbone_size):
            backbone_layers.append(DepthwiseSeparableConv(32, 32, stride=1,use_residual=True))

        self.backbone = nn.Sequential(*backbone_layers)

        self.pool = nn.MaxPool2d(2, 2)

        self.classifier = nn.Sequential(
            nn.Linear(32 * 4 * 4, 128),
            nn.ReLU(),
            nn.Linear(128, 32),
            nn.ReLU(),
            nn.Linear(32, 1),
        )
        self.segmenter = nn.Sequential(
            nn.Linear(32 * 4 * 4, 512),
            nn.ReLU(),
            nn.Linear(512, 128),
            nn.ReLU(),
            nn.Linear(128, 32),
            nn.ReLU(),
            nn.Linear(32, 3), # x, y, r
        )

    def forward(self, x):
        x = self.pool(self.conv1(x))
        x = self.pool(self.conv2(x))
        x = self.pool(self.conv3(x))

        x = self.backbone(x)

        feature_map = x.view(-1, 32 * 4 * 4)

        classification = self.classifier(feature_map) if self.training else torch.sigmoid(self.classifier(feature_map))
        segmentation = self.segmenter(feature_map)
        return classification, segmentation
```

TRIPLET LOSS MODEL

Triplet loss is a metric learning approach that trains a model to embed similar samples closer together and dissimilar ones farther apart in feature space. It uses triplets of anchor, positive, and negative samples, minimizing the distance between the anchor and positive while maximizing the distance from the negative.



$$L = \max\left(d(a,p) - d(a,n) + \alpha, 0\right)$$



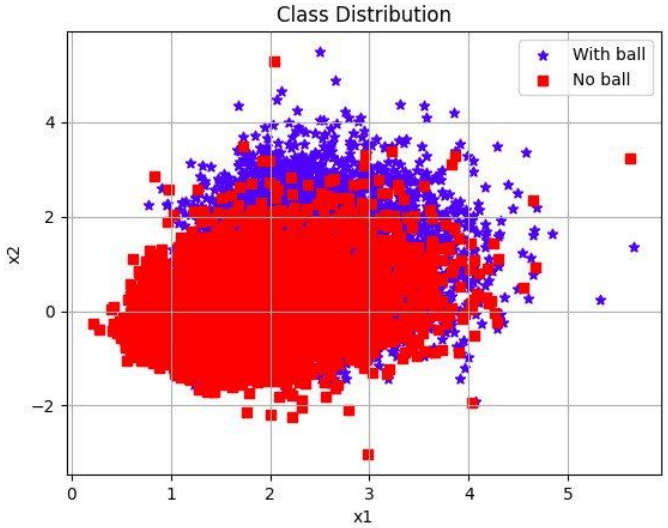
TRIPLET MINING

Moderate Triplets

$$\mathcal{L}_{\text{BH}}(\theta; X) = \sum_{i=1}^{P} \sum_{a=1}^{K} \Big[m + \overbrace{\max_{p=1\ldots K} D\left(f_\theta(x_a^i), f_\theta(x_p^i)\right)}^{\text{hardest positive}} - \overbrace{\min_{\substack{j=1\ldots P \\ n=1\ldots K \\ j \neq i}} D\left(f_\theta(x_a^i), f_\theta(x_n^j)\right)}^{\text{hardest negative}} \Big]_+ , \quad (5)$$

$\overbrace{}^{\text{all anchors}}$ over $\sum_{i=1}^{P} \sum_{a=1}^{K}$

In Defense of the Triplet Loss for Person Re-Identification, *by Hermans, Beyer and Leibe*

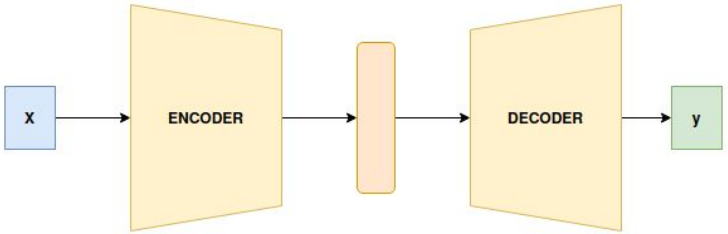



```
from src.TripletBallPatchesDataset import TripletBallPatchesDataset
from torch.utils.data import random_split, DataLoader

triplet_dataset = TripletBallPatchesDataset(dataset,model,k=10, batch_size=256)
✓ 135m 4.5s

100%|██████████| 1929/1929 [2:14:10<00:00,  4.17s/it]
```

Very computationally expensive!

COLORIZATION

The **colorization problem** focuses on transforming grayscale images into fully colored ones by inferring and applying plausible colors based on image content and context.



Main inspiration: ***Colorful Image Colorization*** *by Zhang*

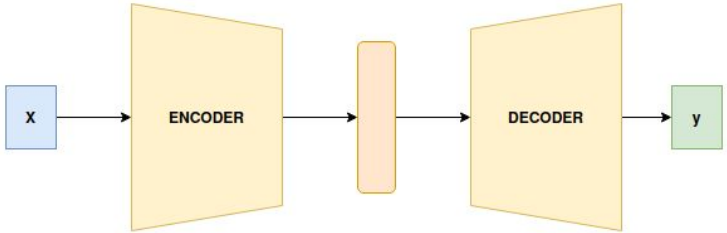
We used the **MSE loss function** to train the model

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

Test Example

EDGE DETECTION

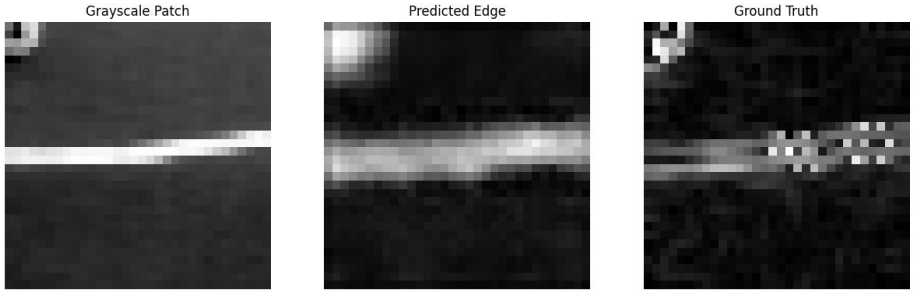
The **edge detection** focuses on detecting the boundaries within an image by identifying sharp transitions between areas of different colors or intensities, with the goal of extracting the structures and shapes present in the image.



We used the **MSE loss function** to train the model

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

Test Example



RESULTS

Model	Accuracy	F1	Loss
Triplet_hard	96.21%	90.34%	**1.760**
Colorization	96.49%	91.07%	1.7954
Triplet_soft	94.78%	87.39%	1.802
Base_SPQR	94.74%	86.32%	1.8071
Edge	to test	to test	to test

Using an RTX 3050 GPU, training the model for 5 epochs took approximately 10 hours to complete.

References

Papers:

- Colorful Image Colorization by Richard Zhang, Phillip Isola, Alexei A. Efros

- Defense of the Triplet Loss for Person Re-Identification by Hermans, Beyer, Leibe

Links:

- Robocup Standard platform league: https://spl.robocup.org/
- *SPQR Team: http://spqr.diag.uniroma1.it/*

Thank you for the attention!

LINKEDIN

SPQR TEAM

INSTAGRAM