Self-supervised feature extraction for ball detection in NAO robots

Can Lin, Marco Zimmatore, Marco Volpini , Penelope Malaspina
Department of Ingegneria informatica, automatica e gestionale "Antonio Ruberti"
University of Rome "La Sapienza"

***Abstract*—Ball detection is crucial for Nao robots in the RoboCup Standard Platform League (SPL), as accurate tracking significantly impacts performance. However, labeling training data for neural networks is resource intensive. To address this, we leverage self-supervised learning (SSL) to train models on unlabeled data, enabling the extraction of meaningful features. These features are fine-tuned for ball detection, reducing dependence on labeled datasets while maintaining accuracy. Our approach enhances ball tracking under various conditions, improving robot performance in matches, and showcasing SSL's potential in robotic vision systems.**

***Index Terms*—Deep Learning, Robotics, Self-supervised Learning**

I. Introduction

RoboCup, the world's largest robotics competition, challenges researchers to push the boundaries of autonomous systems. A key milestone is the RoboCup 2050 goal: creating a team of autonomous humanoid robots capable of defeating the FIFA World Cup champions. Among the critical tasks in this challenge, robust ball detection is essential for effective gameplay.

This paper focuses on leveraging pretext tasks that solve auxiliary problems to extract meaningful features for a robust ball extractor. By designing multiple pretext tasks, we aim to enhance the model's ability to detect the ball in dynamic and complex soccer scenarios, contributing to the broader vision of autonomous robotics in RoboCup.

II. Related work

Self-supervised learning (SSL) has gained traction in robotics and computer vision, particularly for tasks like obstacle detection and visual navigation. Techniques such as contrastive learning and pretext tasks have been applied to improve feature extraction in settings with little labeled data. In robotics, SSL has been used to enhance perception systems, enabling robots to better detect obstacles and navigate their environments autonomously. These methods could be applied to improve ball detection in Nao robots, allowing the model to learn useful features from a variety of tasks without relying on large labeled datasets.

III. Pretext tasks for Self-supervised learning

A. Triplet Loss

Triplet loss is a widely used loss function in metric learning, particularly for tasks such as face recognition, image retrieval, and more. The primary objective of triplet loss is to learn an embedding space where the distance between an anchor and a positive sample (a sample of the same class as the anchor) is minimized, while the distance between the anchor and a negative sample (a sample of a different class) is maximized. The loss function is defined as:

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

Where:

- $d(a,p)$ represents the distance between the anchor a and the positive sample p,
- $d(a,n)$ represents the distance between the anchor a and the negative sample n,
- α is a margin that ensures the negative pair is sufficiently farther from the anchor than the positive pair by at least the margin α.

This loss function encourages the model to learn embeddings such that similar samples are close to each other, while dissimilar samples are pushed further apart in the embedding space. The margin α prevents trivial solutions where the distances between all samples become arbitrarily small.

The goal of applying triplet loss in our context is to improve the model's ability to extract important features of the ball by using it as the anchor. This approach draws inspiration from the work of Hermans, Beyer, and Leibe (2017) in their paper, *"In Defense of the Triplet Loss for Person Re-Identification"* [1].

B. Colorization

The **colorization problem** in computer vision refers to the task of assigning realistic colors to *grayscale* images, transforming them into plausible full-color images. It is a form of image-to-image translation where the goal is to predict the missing chrominance information while maintaining the structure and semantics of the original grayscale input.
The choice to use the Colorization Problem as a pretext task for self-supervised learning is inspired by the work of few Berkeley researchers [3] where they built a Colorization model trained on the *ImageNet* dataset.

C. Edge detector

To enhance the model's ability to detect the ball in football field images, we introduced a pretext task centered on edge detection. This technique identifies the boundaries or edges

of objects by detecting discontinuities in image intensity. The goal is to highlight regions where there is a sudden change in intensity.

In our case, the task focuses on detecting the contours of objects like the ball and robots. This is particularly useful because the green football field creates a strong contrast with the ball, making its edges more prominent. This contrast enhances the model's accuracy in detecting the ball, even in dynamic or cluttered environments, by effectively isolating it from the background for more precise detection.

IV. Dataset creation

A. Retrieving logs from Nao Robots

To extract data from the Nao robots, we utilized a specialized tool from the SPQR Nao 2024 github repository [2]. This tool streamlined the process of accessing logs stored on multiple robots, allowing us to efficiently retrieve all the recorded images. The logs contained video data captured by the robots during various tasks. To manage and preprocess this data, we applied a frame-sampling strategy. Specifically, for each video, we extracted every 20th frame to reduce redundancy while retaining sufficient temporal diversity.

B. Yolo-world

To construct the dataset, we selected images containing the ball and robots from the patches we had. This was achieved through a labelling procedure, enabling the precise identification and annotation of the regions of interest within each image. For the first part of labelling, we utilized a model called YOLO-World, pre-trained on large-scale datasets, including detection, grounding, and image-text datasets. This model generated bounding boxes around our objects of interest, providing a preliminary identification of the ball and robots within the patches.

C. Manual annotation

We observed that while the YOLO-World model provided a strong starting point by generating bounding boxes for the ball and robots, its predictions were not always accurate. Specifically, there were instances of false positives—where the model identified objects as balls or robots incorrectly—and false negatives—where it failed to detect the objects of interest present in the image patches. These errors highlighted the need for a thorough manual verification and refinement process. To address this, we used LabelImg, an open-source annotation tool, to review and correct the bounding boxes proposed by the model. After loading the YOLO-generated annotations into the tool, we examined and refined them, manually adjusted bounding boxes, adding annotations for missed objects, ensuring accurate labeling of the ball and robots and thereby improving the dataset's reliability and suitability.

D. Patch Extractor

To optimize computational efficiency on the Nao robot (NAO V5 & V4, equipped with an ATOM Z530 1.6 GHz CPU and 1 GB RAM), we implemented a patch extraction technique. This approach focuses on processing smaller image patches rather than the entire image, as the robot's hardware limitations make it more feasible to predict on patches.

We extract patches of varying sizes to accommodate different ball sizes, as the ball's size changes depending on its distance from the robot. When the image contains a ball, we prioritize cropping patches that contain the ball, ensuring a higher probability of capturing the ball's presence. On the other hand, when the image does not contain a ball, we aim to select hard negative patches by calculating the edge percentage within the patch. This helps to avoid patches with little information, such as those containing only the green football field, which are irrelevant for the task at hand.

V. Models implementations

A. Base Model

1) Architecture: The base model **BallPerceptor** is structured as follows:

- **Depthwise Separable Convolutions:** Three initial convolution layers with 8, 16, and 32 output channels respectively. These layers use a kernel size of 3x3 with a stride of 1 and padding of 1.
- **Backbone:** A sequence of 9 Depthwise Separable Convolutions with 32 output channels each, some of which use residual connections to improve feature flow.
- **Max Pooling:** A 2x2 max-pooling layer is applied after each convolutional layer to reduce the spatial dimensions.
- **Classifier:** A fully connected network that outputs a scalar value representing the likelihood that the image contains a ball.
- **Segmenter:** A fully connected network that outputs the ball's x-coordinate, y-coordinate, and radius.

Depthwise Separable Convolution (DSC) is a factorized version of the standard convolution operation. In traditional convolutions, each input channel is convolved with each filter in the convolutional layer, resulting in a large number of parameters. DSC splits this operation into two smaller steps:

1) **Depthwise Convolution:** Each input channel is convolved with its own set of filters (one filter per input channel).
2) **Pointwise Convolution:** A 1x1 convolution is then applied to combine the output of the depthwise convolution across all channels.

The key advantage of DSC is that it significantly reduces the number of parameters and computation, which is especially useful in lightweight models for mobile and embedded systems. In the case of the **BallPerceptor** model, DSCs allow for efficient feature extraction while maintaining a small memory footprint.

2) Loss Function: The model employs different loss functions for the regression and classification heads.

For the regression head, we use the Circular DiIoU loss, defined as follows:

$$L_{\text{Circular\_DIoU}} = 1 - \frac{A_{\text{intersection}}}{A_{\text{union}}} + D \tag{1}$$

Where:

- $A_{\text{intersection}}$ is the intersection area between the predicted and ground truth circles,
- A_{union} is the union area of the predicted and ground truth circles,
- D is the distance penalty term that is calculated based on the center distance and the radius of the circles.

For the classification head, we use Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss), which is formulated as:

$$L_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\sigma(x_i)) + (1-y_i)\log(1-\sigma(x_i))\right] \tag{2}$$

Where:

- x_i is the predicted value for the i-th sample,
- y_i is the true label for the i-th sample,
- $\sigma(x_i)$ is the sigmoid function applied to the predicted logits.

For the optimizer, we use the AdamW optimizer, which is a variant of the Adam optimizer with weight decay. The AdamW update rule is formulated as follows:

$$\theta_{t+1} = \theta_t - \eta\left(\frac{\nabla_\theta J(\theta_t)}{\sqrt{v_t}+\epsilon} + \lambda\theta_t\right) \tag{3}$$

Where:

- θ_t represents the parameters at time step t,
- $\nabla_\theta J(\theta_t)$ is the gradient of the objective function,
- v_t is the moving average of the squared gradients,
- λ is the weight decay factor (penalty on large weights),
- ϵ is a small value to avoid division by zero.

AdamW helps prevent overfitting and improves convergence by decoupling weight decay from the optimization steps.

B. *Triplet loss model*

*1) **Triplet soft dataset**:* Before building the model, we first had to generate combinations of anchors, positives, and negatives for training. To achieve this, we created a dataset where, for a given value of k, each positive sample (patch containing the ball) is treated as an anchor. For each anchor, k positive samples (other patches with the ball) and random negative samples (patches without the ball) are selected. This process generates $P \times K$ triplets, where P is the number of positive samples and K is the number of negative samples selected for each anchor.

*2) **Triplet mining**:* To obtain meaningful triplets, we first created a Triplet Loss Soft Model using a dataset where triplets were sampled without hard mining. We then developed a Triplet Loss Hard Model by employing a triplet mining strategy inspired by the *Batch Hard* method [1]. Negatives were divided into batches, and for each combination of an anchor (a) and k positives (p), the hardest negative (n^*) was selected as the one minimizing $d(a,n) + d(p,n)$, where $d(x,y) = \|f(x) - f(y)\|_2$ is the Euclidean distance. We opted for actual Euclidean distance instead of the squared version to reduce feature collapse, as noted in [1]. The soft and hard models were thus distinguished by their triplet sampling approach, with the hard model leveraging a more robust mining technique.

*3) **Architecture**:* The **TripletBallPerceptor** model utilizes Depthwise Separable Convolutions (DSC) to extract features from input patches. It consists of three initial convolutional layers (**Conv1**, **Conv2**, **Conv3**) followed by several backbone layers (controlled by **backbone_size**) to further refine the features. The final feature map, of size $32 \times 4 \times 4$, is flattened and used to compute distances between positive and negative triplets. This is where the distances between the anchor (patch with ball), positive (similar patch with ball), and negative (patch without ball) are computed for triplet loss.

C. *Colorization model*

The model architecture is based on the **Ball Perceptor** model already cited. After the *Feature Extractor* components (**BackBone** + 3 *DepthwiseSeparableConvolutional* Layers), the Colorization model replaces the *Segmenter* and *Classifier* parts with a decoder:

- **Decoder**: formed by 3 deconvolution (trasposed convolutional) layers that take reduce the dimensions of the input using also ReLu activations each time to introduce non-linearity to the output. In the end, we use a 2D-Convolutional Layer to translate the input channels in the three channels returned by the model.

The model takes as input a grayscale patch and uses the corresponding RGB version of the patch as ground truth. After the training, the weights in the Feature Extractor part of the model (until the Backbone), will be used to improve the Ball Object Detection of base model.

Loss Functions:

- **Mean Square Error (MSE)** measures the average squared difference between the predicted colorized image and the actual target image. The MSE loss is defined as:

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

- **Mean Absolute Error (MAE)** measures the average absolute difference between the predicted colorized image and the actual target image. The MAE loss is defined as:

$$\text{MAE}(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

In the work by Zhang et al. [3], the authors did not identify the Mean Squared Error (MSE) loss function as the optimal solution for the colorization task. Instead, they formulated the problem as a classification task by discretizing the color space into 313 classes of *ab* values for each pixel and employing a SoftMax Cross-Entropy loss. This approach accounts for the inherent non-convexity of the real-world color distribution, where a single object can exhibit multiple plausible colorations.

However, for our study, we opted to use the simpler MSE loss function. This decision was motivated by the restrictive nature

of our domain, which involves images of robots playing football. The limited color palette in this setting, predominantly consisting of white and green patches, reduces the complexity of color variations. In the end, we have trained the model with both *MAE* and *MSE*, confronting the results.

D. *Edge model*

The architecture of the Edge Detection model is very similar to that of the Colorization model. In fact, after the *Feature Extractor* components, the Edge Detection model replaces the *Segmenter* and *Classifier* parts with a decoder:

- **Decoder**: Similar to the previous model, in this case as well, the decoder is formed by 3 deconvolution (transposed convolutional) layers that reduce the dimensions of the input, using ReLU activations each time to introduce non-linearity to the output. Additionally, we use a 2D-Convolutional Layer at the end, but differently from the previous model, it is used to translate the input channel into the single channel returned by the model.

The model takes as input a grayscale patch and uses the corresponding grayscale patch filtered by the Sobel operator as ground truth.

Loss Functions:

- **Mean Square Error (MSE)** measures the average squared difference between the predicted image and the target image obtained by applying the Sobel filter to the original patches. The MSE loss is defined as:

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

Sobel Filter: The Sobel filter is a discrete differentiation operator used to compute the gradient magnitude of an image, emphasizing its edges. It works by convolving the input image I with two 3x3 kernels, G_x and G_y, to approximate the derivatives in the x- and y-directions, respectively:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

The filtered image S is obtained by computing the gradient magnitude:

$$S = \sqrt{(I * G_x)^2 + (I * G_y)^2},$$

where $*$ represents the convolution operator. The result highlights the edges in the grayscale patch, which serves as the ground truth for training the model.

VI. RESULTS

The performance of the proposed models was evaluated using several metrics: **Loss**, **Accuracy**, **F1 Score**, and **Intersection over Union (IoU)**. The pretext task models were initially trained on a dataset of approximately 10,000 images extracted from the MakerFaire 2024 event. Subsequently, the base model, initialized with the self-supervised weights derived from these pretext tasks, was trained on a completely new and larger dataset comprising 150,000 images from the SPQR team database.

TABLE I
PERFORMANCE METRICS OF THE BASE MODEL TRAINED WITH PRETEXT TASK WEIGHTS AND RANDOM INITIALIZATION.

Pretext Task Weights	Loss	Accuracy (%)	F1 Score (%)	IoU (%)
Colorization (MAE)	1.69899	96.419	91.051	80.388
Edge Detection	1.75486	96.142	90.298	80.340
Triplet Loss	1.76519	96.129	90.257	79.018
Colorization (MSE)	1.77139	95.869	89.667	80.776
Base Model	1.77378	95.675	89.001	79.621

Table I summarizes the evaluation metrics for the base model trained with weights extracted from each pretext task. For comparison, a baseline version of the base model trained with randomly initialized weights is also included. Each row in the table represents a distinct experiment where:

- The **pretext task weights** were used to initialize the base model, followed by training on the SPQR dataset, or
- The base model was initialized with **random weights**.

The results shown are averaged over five runs, with each run representing the best performance achieved within five epochs. Each training session took approximately 13 hours using an RTX 3050 GPU.

The results indicate that initializing the base model with pretext task weights generally outperforms random initialization, demonstrating the effectiveness of self-supervised learning. Among the pretext tasks, weights derived from the **Colorization (MAE)** task yielded the best performance across all metrics, followed closely by **Edge Detection**. This suggests that pretext tasks enhancing spatial and color-related features provide a stronger foundation for the base model. In contrast, the baseline model trained with random initialization exhibited the lowest performance across all metrics.

A. *Conclusion and Future Work*

We presented several models to enhance the BallPerceptor system for RoboCup, including triplet loss, colorization, and edge detection tasks. These approaches improved feature extraction for ball detection, making the model more adaptable to challenging environments. Through this work, we made a small but meaningful contribution to the RoboCup league and to the development of better ball perception for NAO robots.

Future work will focus on:

- **Triplet Loss Optimization:** Exploring better triplet mining strategies.
- **Augmented Data:** Expanding the training dataset for greater diversity.
- **Real-Time Deployment:** Optimizing the model for real-time robot use.
- **MaML [4]:** Investigating Model-Agnostic Meta-Learning (MaML) by combining all three tasks to improve generalization.

These advancements will further enhance the model's performance and adaptability, pushing the boundaries of robot vision in RoboCup competitions.

REFERENCES

[1] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In Defense of the Triplet Loss for Person Re-Identification. *CoRR*, abs/1703.07737, 2017. http://arxiv.org/abs/1703.07737.

[2] SPQR Team, "SPQR Nao 2024 Repository," GitHub. [Online]. Available: https://github.com/SPQRTeam/spqrnao2024

[3] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.* In Proceedings of the 34th International Conference on Machine Learning (ICML), 2017.