

On the Reproducibility of the EAST-SPL Pipeline: Event-Aware Statistical Tiling for Soccer Player Localization

Abolfazl Chaman Motlagh¹[0000–0002–6179–724X] and Mikael Nilsson¹[0000–0003–1712–8345]

Centre for Mathematical Sciences,
Division of Computer Vision and Machine Learning
Lund University, Lund, Sweden

Abstract. This companion paper focuses on reproducibility aspects of the ICPR-accepted paper, "EAST-SPL: Event-Aware Statistical Tiling for Decomposable Soccer Player Localization with an auxiliary rejection network" [2]. It documents implementation and practical details that are necessary for reproduction, validation, and, most importantly, integration of the EAST-SPL pipeline.

The code is available at <https://github.com/AbolfazlChM95/EAST-SPL>.

Keywords: Object detection · Early rejection network · Decomposable methods · Efficient ML · Genetic Algorithm

1 Introduction

Reproducibility is essential for validating scientific claims, verifying the reliability of research findings, and enabling the scientific community to build upon published outputs. In machine learning, reproducibility is a key requirement for conducting, communicating, and evaluating research in a reliable and reusable manner [4].

Reproducibility of the EAST-SPL pipeline [2] is important for two reasons: first, EAST-SPL is a modular extension to the predecessor work DTSP-LEV [3] that can be used to reduce expected empirical FLOPs. We also hypothesize, as future work, that simultaneous training of the rejection head could work as a form of deep supervision. Second, the core contribution in EAST-SPL can be adapted to other spatially decomposable vision tasks under suitable conditions. These conditions include, for example, the availability of scene-level statistics and the possibility of decomposing the input image into tiles. Therefore, reproducibility of EAST-SPL is necessary to enable future work to explore this idea in different application domains.

EAST-SPL [2] was introduced for decomposable soccer player localization, where high-resolution inputs make full-frame inference expensive. The main paper builds on DTSP-LEV [3] and replaces single-frame tiling objectives with expected total FLOPs computed from projected player-location statistics. It also

introduces an auxiliary rejection head to bypass empty tiles and a genetic algorithm to optimize the non-differentiable tiling objective.

The reproducibility aspects described in this paper correspond to the following components of EAST-SPL [2]:

Section 2 : Section 3.1 of EAST-SPL, implementation of the rejection head.

Section 3 : Eq. (2), implementation of the Expected Total FLOPs objective.

Section 4 : Eq. (2), projected statistics and computation of $P(\text{player} \subset t)$.

Section 5 : Section 4.2, dataset generation for isolated rejection-head training.

Sections 6 - 7 : Discussion of applicability and limitations.

2 Integration of the Rejection Network

This section is related to Section 3.1 in EAST-SPL [2]. An important characteristic of the auxiliary rejection head attached to a mid-layer feature map is input-size adaptability. This is provided by global pooling of feature maps. EAST-SPL uses both per-channel average and max pooling layers and concatenates them to construct the input vector for the rejection head. See the method *Channel_pooling* in Code 1 and Fig. 1 in EAST-SPL [2].

The rejection head can be used inside the forward pass of the primary network to bypass empty tiles, which are detected by thresholding the rejection head logit. As discussed in Section 4.3 of EAST-SPL [2], the choice of rejection threshold θ (in Code 1, *self.rej_thresh*) directly controls the rejection rate, and thus the FLOPs, while also affecting the end-to-end accuracy of the main model. Therefore, θ should be tuned to obtain computational savings while keeping the reduction in recall as small as possible.

```

1  from torch.nn import AdaptiveAvgPool2d, AdaptiveMaxPool2d
2  # Initialize Average and Max pooling functions
3  AvgPooling = torch.nn.AdaptiveAvgPool2d((1,1))
4  MaxPooling = torch.nn.AdaptiveMaxPool2d((1,1))
5
6
7  ## Inside the torch model class for the primary network.
8  def Channel_pooling(self, features):
9      # pooling tiled feature array to constant length vector
10     avg_pool_f = AvgPooling(features)
11     max_pool_f = MaxPooling(features)
12     combined_features = torch.cat((avg_pool_f, max_pool_f), dim=1)
13     return combined_features
14
15  ## forward pass for our primary nn class with the rejection head
16  def forward(self, x):
17     f = self.Shared_Block(x)
18     f_pooled = self.Channel_pooling(f)
19     rej_prob = self.rejection_head(f_pooled)
20     y = self.Non_Shared_Block(f) if rej_prob>self.rej_thresh else torch.nan
21     return y

```

Code 1: Simplified implementation snippet: the forward pass for rejection head and bypass non-shared computation for rejected tiles

3 Calculating Expected Total FLOPs

This section describes the implementation of the Expected Total FLOPs ($\mathcal{ETF}$) objective defined in Eq. (2) of EAST-SPL [2]. The objective for the primary network M , with auxiliary rejection head N sharing neural block O over the tile set T , is given by

$$\mathcal{ETF}^{(M;N)}(\mathbb{T}) := \sum_{t \in \mathbb{T}} (\mathcal{F}^{(O+N)}(t) + \underbrace{\mathbb{P}[N(\text{Frame}[t])=1]}_{\doteq \mathbb{P}[\text{player} \subset t]} \cdot (\mathcal{F}^{(M)}(t) - \mathcal{F}^{(O)}(t))) \quad (1)$$

where $\mathcal{F}^{(A)}(t)$ indicates FLOPs of the network A on an input with the size of tile t .

During the genetic algorithm (GA), proposed in Section 3.3 of EAST-SPL [2], the $\mathcal{ETF}$ has to be calculated for each tile set $\mathbb{T}$ corresponding to a candidate solution. Note that each solution in GA is represented by 2D collocated coordinate points $\mathcal{P}$; then, using the mask $\mathcal{M}$ over the ROI (here, the soccer field in the image), the tile set $\mathbb{T}$ is constructed.

Code 2 contains two functions as methods of an object to be called during the GA optimization for calculating the FLOPs. The method *StatFLOPsTiles* calculates the $\mathcal{ETF}$ using a tabular FLOPs array of three different blocks and also projects statistics into the image plane, *self.Projected_Stats*.

The FLOPs of different blocks are calculated a priori and saved as tabular data for different input sizes. In terms of a straightforward architecture, the FLOPs could also be a direct term as function of the width and height of tiles. The method *Tiles2FLOPs* in Code 2 will calculate the FLOPs of running a network on a given tile set. The FLOPs are given as a tabular array. In our architecture (Fig. 1 in EAST-SPL [2]), the rejection head is a fixed-size MLP, and the total FLOPs of running it on tile set $\mathbb{T}$ is the number of tiles ($|\mathbb{T}|$) times constant FLOPs, see the first column of Table 2 in EAST-SPL [2].

4 Calculating probabilistic weight for every tile

An important building block of the EAST-SPL [2] pipeline is the calculation of the term $\mathbb{P}[N(\text{Frame}[t]) = 1]$ in Eq. (1) (Eq. (2) in the main paper). The replacement of the term $\mathbb{P}[N(\text{Frame}[t]) = 1]$ with $\mathbb{P}[\text{player} \subset t]$ as the statistical weight for calculating the non-shared primary network FLOPs that appears in Eq. (1), relies on an assumption about the performance of the classifier (here the rejection head) N . To calculate this probability in practice, we can approximate it by the empirical probability from the dataset, i.e., $\mathbb{P}[\text{player} \subset t] \approx \frac{\#(\text{player} \subset t)}{\# \text{dataset}}$, as briefly mentioned in the original paper. The latter approximation is justified by the law of large numbers, while its finite-sample error depends on how well the dataset represents the underlying distribution of real events.

The dataset used in EAST-SPL [2] to estimate soccer-game statistics is Spiideo SoccerNet-SynLoc [1]. It contains data from 15 arenas, with two camera

```

1 def StatFLOPsTiles(self, Tiles):
2     # Calculating Expected Total FLOPs  $\mathcal{ETF}$  Eq.1
3     # Term  $\mathcal{F}^{(O+N)}(t)$  : Flops of rejection head and shared block
4     # This could also be calculated using Tile2FLOPs()
5     l1 = self.Rejectionhead(Tiles)
6     # Term  $\mathcal{F}^{(M)}(t)$  : Flops of non-shared part of primary network
7     l2 = Tiles2FLOPs(Tiles, self.MainNet_FLOPs, self.MainNet_Shift, self.margin)
8     # Term  $\mathcal{F}^{(O)}(t)$  : Flops of shared part of primary network
9     l3 = Tiles2FLOPs(Tiles, self.Shared_FLOPs, self.Shared_Shift, self.margin)
10    # Term  $\mathbb{P}(\text{player} \subset t)$  : see section 4 and Code 4
11    prob_weights = Tiles2Prob(Tiles, self.Projected_Stats)
12    # Sum over tiles to construct  $\mathcal{ETF}$ 
13    ETF = np.sum(l1 + l3 + prob_weights * (l2-l3))
14    return ETF
15
16    ## Outside of Object class
17    def Tiles2FLOPs(Tiles, FLOPs_array, shift, margin=0):
18        # Tiles array to FLOPs vector correspondingly
19        Tiles = np.atleast_2d(Tiles) # Fix only one Tile case
20        w, h = Tiles[:, 2], Tiles[:, 3]
21        i, j = w - shift + 2*margin, h - shift + 2 * margin
22        if np.any(i<0) or np.any(j<0):
23            raise IndexError("width or height are small! Look for tabular shift and margin")
24        FLOPs = FLOPs_array[i, j]
25        return FLOPs
26
27    def RejNetFLOPs(Tiles):
28        # Calculating the FLOPs of rejection head for our application, given a Tiles set
29        return self.rej_const_FLOPs * np.ones(len(Tiles))

```

Code 2: Methods in *FLOPsCalculator* object for calculating Eq.1 from given tabular FLOPs of different blocks of the architecture (See Fig.1 in [2]).

setups per arena, resulting in 30 camera configurations. Since field dimensions are not fixed across arenas, directly aggregating player positions in pitch coordinates would introduce scale inconsistencies. To address this heterogeneity, we first group the images by arena. This grouping can be derived from the camera parameters provided in the dataset annotations: because the dataset is recorded using fixed cameras, images from the same arena and the same viewing direction share the same camera parameters. Player coordinates on the pitch are then normalized to a common coordinate system, for example $[0, 1]^2$, before being aggregated across arenas.

The calculated normalized 2D coordinates of soccer players over the entire dataset could be used to approximate the geometric statistics of the soccer game. Note that the SynLoc dataset does not provide ground-truth matching left- and right-looking cameras of the same scene; hence, because they share a view of the scene, some players are counted twice. This will create a cone-shaped artifact on the soccer field that can be seen in Fig. 3 (Left) of the EAST-SPL paper.

When optimizing tile decomposition for a sequential view (camera setting), the stacked statistics $(\oplus p_i)$ will first be unnormalized to the dimension of the corresponding arena ($\text{Unnorm}(\oplus p_d)$). Then the unnormalized 2D coordinates will be projected into the image plane using the known camera matrix K and distortion coefficients d . One way to do this is to calculate this once per pixel for the full resolution of the query scene. With this we then have an image $\mathcal{P} = \Pi_{K,d}(\text{Unnorm}(\oplus p_d))$. The image $\mathcal{P}$ has the same resolution as the query image I (and mask $\mathcal{M}$), where the value of each pixel is the number of observations

that a player was inside that pixel over the entire dataset divided by the dataset size.

```

1 # using tools from [1] (https://github.com/Spiideo/sskit)
2 from sskit import unnormalize, world_to_image
3 # Implementation of  $\Pi_{K,d}(x)$ : Projecting 3D coordinate to 2D
4 Projected_2D_Coordinate = unnormalize(world_to_image(camera_matrix, dist_poly, 3D_coord),
    ↪ [3, img.size[1], img.size[0]])

```

Code 3: The script for projecting 3D coordinate (for calculating the $\Pi_{K,d}(\text{Unnorm}(\oplus p_d))$) into the camera using sskit¹toolbox.

To **approximate** the $\mathbb{P}[player \subset t]$, we can simply sum over the tile t in the image $\mathcal{P}$. This approximation will hold under the assumption that $\mathcal{P}_{i,j}$ and $\mathcal{P}_{k,l}$ are independent and under a Taylor expansion. Since this calculation leads to a saturated probability for large tiles, we clip the final result to keep it under 1, see Code 4. The exact computation of the term could be provided by projecting 2D coordinate points directly to the image plane and calculating the empirical probability for each frame, at each $\mathcal{ETF}$ calculation.

```

1 def Tiles2Prob(Tiles, Projected_Statistics):
2     Probs = []
3     for tile in Tiles:
4         x, y, w, h = tile.astype(int)
5         Probs.append(np.sum(Projected_Statistics[x:x+w, y:y+h]))
6     return np.clip(np.array(Probs), a_min=0, a_max=1)

```

Code 4: Method to approximate $\mathbb{P}(player \subset t)$ for each t in the input vector $Tiles$ based on a priori calculated $Projected_Statistics$ as $\mathcal{P}$ for the camera (k, d) .

5 Training of rejection network

This section expands the isolated rejection-head training described in Section 4.2 of EAST-SPL [2] by detailing the construction of the secondary training dataset from the Spiideo SoccerNet-SynLoc dataset [1]. During rejection-head training, the shared block of the primary network is frozen. Training is accelerated by performing a single forward pass of the SynLoc training set through the shared block and storing the resulting feature maps. Since the selected U-Net block preserves the input spatial resolution, tiling can be applied either before or after this forward pass (subject to the receptive field). As the global pooling layers (*Channel_pooling* in Code 1) have no trainable parameters, the pooled feature vectors for each tile can be precomputed and used to train the rejection MLP.

Since the training of the rejection network is also isolated from the sought optimal tile configuration, we sampled 500 randomly generated tiles for each sample in the SynLoc dataset. The tiles have been generated with a rectangular shape (see section 7), with width and height selected randomly and independently between 100 and 700 pixels. The ground truth for each sample has been collected from the image ground truth as a binary variable indicating the presence of at least one player inside the generated tile, see Fig. 1. To balance the

¹ <https://github.com/Spiideo/sskit>

dataset, we enforced at least 40% of generated tiles with the positive label. The samples from the SynLoc training dataset have been used to train the rejection network. The reported sensitivity and rejection rate in Table 2 and Fig. 4 of EAST-SPL [2] have been calculated on the generated tiles from the test dataset.

Generated tiles from one sample for isolated training of rejection network.

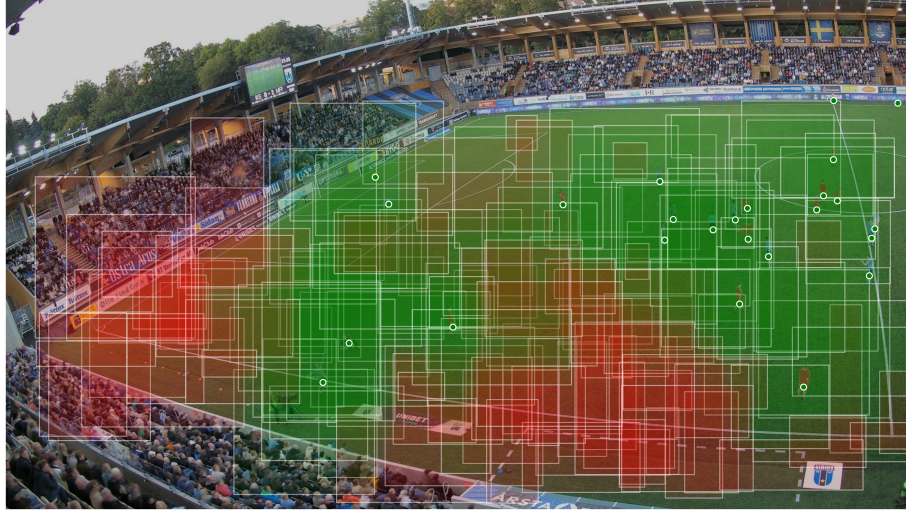


Fig. 1: An example of generated tiles for isolated training of rejection network. Each tile has a binary ground truth (here positive: green, negative: red) based on presence of at least one player in the tile (players shown as white dots)

6 Applications Beyond Soccer Player Localization

The general methodology in EAST-SPL [2] does not have any inherent limitation to soccer player localization. The work extends DTSP-LEV [3] as an improvement with additional use of input-image decomposition into smaller tiles. The general concept could be applied to a general neural network that accepts a 2D or 3D array, where two of the input dimensions have spatial correlations. However, the network’s architecture should be capable of being run in parallel without any practical hindrance on a patch of any size, which we call a tile. One example of such networks is fully convolutional neural networks [3].

Additionally, the downstream task is not limited to object detection or localization. However, a downstream task where the network has to access global properties of the image will not benefit from decomposing the image to reduce the total computation. Intuitively, the pipeline is suitable for a downstream task where the task depends on localized regions of interest. Bypassing the tiles that do not include the object of study can reduce computation.

7 Limitation of Axis-Aligned Rectangular Tiling

The tile notation used in EAST-SPL is based on the patch representation that has also been used in DTSPL-BEV [3], which is the baseline for non-statistical tiling and also grid-search optimization for comparison. The tile representation $t = [x, y, w, h]$ is based on four values: the top-left corner coordinate (x, y) , width w , and height h .

For the boundary of the region of interest, for example, in the soccer-field case, a non-rectangular tile might be a better option if we aim to cover the mask $\mathcal{M}$ with fewer tiles. The limitation of using rectangular tiles comes from the rectangular shape of the input array for our neural network. Additionally, if we also allow rotated rectangles, we might be able to cover the mask $\mathcal{M}$ with fewer tiles, and we can rotate the rectangle back to an axis-aligned rectangle, suitable as a network input array. This, however, requires stability of performance of the model under rotation, which remains for future work.

8 Conclusion

In this companion paper, we documented practical implementation details of the EAST-SPL [2] paper that are necessary to reproduce the experiments presented in the main paper. We additionally discussed the possible generalization of the proposed core idea to applications beyond soccer player localization. Lastly, we clarified an argument about non-axis-aligned rectangular or non-rectangular tilings that emerged during discussions and presentation of the work.

References

1. Ardö, H., Nilsson, M., Cioppa, A., Magera, F., Giancola, S., Liu, H., Ghanem, B., Droogenbroeck, M.V.: Spiideo soccernet synloc - single frame world coordinate athlete detection and localization with synthetic data. In: In Proceedings of the 20th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 2: VISAPP. pp. 278–285. INSTICC, SciTePress (2025)
2. Chaman Motlagh, A., Nilsson, M.: EAST-SPL: Event-aware statistical tiling for decomposable soccer player localization with an auxiliary rejection network (2026), accepted at International Conference on Pattern Recognition (ICPR) 2026
3. Persson, I., Motlagh, A.C., Nilsson, M.: DTSPL-BEV: Decomposable tiled soccer player localization. In: Proceedings of the 15th International Conference on Pattern Recognition Applications and Methods - ICPRAM. pp. 483–490. INSTICC, SciTePress (2026). <https://doi.org/10.5220/0014468500004067>
4. Pineau, J., Vincent-Lamarre, P., Sinha, K., Larivière, V., Beygelzimer, A., d’Alché Buc, F., Fox, E., Larochelle, H.: Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *Journal of machine learning research* **22**(164), 1–20 (2021)