INFO8010: Project report - Image segmentation by UNet

Federico Ferraro,[1] **Thibaut Lefèvre**,[2] and **Duy Vu Dinh**[3]

[1]*federico.ferraro@student.uliege.be (s203840)*
[2]*thibaut.lefevre@student.uliege.be (s164094)*
[3]*DuyVu.Dinh@student.uliege.be (s2401627)*

I. INTRODUCTION

Semantic segmentation is a fundamental problem in computer vision, where the goal is to classify each pixel of an image into a predefined category. In recent years, deep learning has significantly advanced the accuracy and robustness of segmentation models, particularly through the use of convolutional neural networks (CNNs). These models are widely applied in domains such as autonomous driving, medical imaging, and sports analytics.

In this project, we focus on the task of football player segmentation in match images. This task is particularly relevant in sports analytics, where accurate localization of players can support tactical analysis, automatic highlight generation, and augmented broadcast visuals. The problem is inherently challenging due to complex backgrounds, frequent occlusions, and varying lighting conditions on the field.

To address this task, we implement and evaluate multiple deep neural network architectures for image segmentation, starting from the original U-Net architecture. U-Net is known for its symmetric encoder-decoder structure with skip connections that preserve spatial information. We extend our analysis to modern variants such as U-Net++, Attention U-Net, and ResUNet, which introduce architectural enhancements to improve feature representation and segmentation accuracy. After that, we implement custom made models and analyse their results.

Our objective is to compare these architectures on the *Football Player Segmentation* dataset using both quantitative metrics, including IoU, Dice score, and qualitative visualizations. All models are implemented from scratch in PyTorch, and trained under similar conditions to ensure fair comparisons. This work aims to provide insights into the strengths and limitations of each model variant, and to contribute a reproducible benchmark for sports-related segmentation tasks.

II. RELATED WORK

Regarding this dataset, a significant number of contributors have implemented the U-Net architecture for semantic segmentation of football players. For instance, Doyeun [1] and Chetan [2] used PyTorch-based pipelines to train U-Net models with pixel-wise supervision, focusing on binary segmentation masks. WeigelK [3] followed a similar approach, analyzing player positions through COCO-formatted annotations. These works typically involve data preprocessing, model definition, training loops, and basic evaluation metrics.

Beyond binary segmentation, some researchers explored more complex setups. Asma Masssad [4] implemented a multiclass segmentation model using U-Net to distinguish player roles or teams. Drmwnnrafi [5] integrated both U-Net and U2Net models in TensorFlow to compare performance across architectures, aiming to enhance accuracy and generalization.

The Segment Anything Model (SAM), proposed by Meta AI, has also been applied to this dataset. Squidbob [6] demonstrated the use of SAM for zero-shot segmentation of football players, showing promising results without task-specific fine-tuning. This highlights the potential of foundation models in domain-specific segmentation.

Ihelon [7], the creator of the dataset, provided baseline modeling techniques to guide segmentation tasks, serving as a reference point for comparative studies. Additionally, Xinkai Liao [8] adopted an object detection approach rather than segmentation, offering a complementary method for identifying players in diverse field conditions.

III. METHODS

A. Dataset

The dataset used in this project is the *Football Player Segmentation* dataset [9]. It contains a total of 527 RGB image frames extracted from a video recording of a football match, with corresponding player segmentation masks provided in COCO format. Each annotation includes a pixel-perfect polygon-based segmentation for distinguishing player regions on the field. Players and referees are considered as mask and all other elements are considered as background.

This dataset is made available under the license Attribution 4.0 International (CC BY 4.0) which allows for sharing and manipulating the data under the condition that it is properly credited and that the changes applied to the data are indicated. No personal data is used, and there are no ethical concerns to raise.

Due to the sequential nature of the video frames, consecutive images are highly correlated, resulting in redundancy that can negatively affect model generalization and increase training time. To address this, we sampled every 5th frame, reducing the total number of images used for training and evaluation to 106. Each image is a full-color RGB frame, and the corresponding mask is a binary image in which pixels corresponding to players are labeled as 1 (foreground) and all others as 0 (background).

The dataset was specifically designed for player segmentation, but it has some limitations. Since all images originate from a single match, the dataset may lack diversity in terms of player uniforms, stadium lighting, camera angles, and backgrounds. These limitations could potentially impact the model's generalization to other matches or broader football scenarios.

B. Exploratory data analysis

There are no missing or corrupted images in the dataset. It can be observed that the number of masks is not equal to the number of players on the field. This is not dependent on the resolution, it is thus inherent to the dataset and the way it is annotated. Possible causes for this are the overlap of multiple players or objects covering part of the players, as can be seen for example in Image idx 351, where the goalkeeper is behind one of the goal poles, thus their mask is being split in multiple elements.

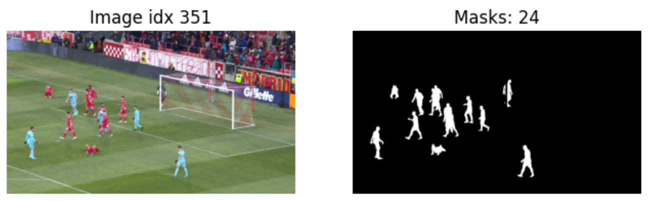


FIG. 1: Image idx351, the goalkeeper is standing behing the goal pole.

It is also possible to notice that the referee on the field is annotated in the masks, as well as the linesmen. This could potentially lead to issues as linesmen have a similar appearance as the referee but occupy a different position on the field, often overlapping with background elements like billboards, the border of the field or even other elements. This difference with the other people on the field may later introduce errors in the predictions : potential false positives for identifying the linesman when it was background or false negatives for excluding them unrightfully. We will however proceed with set as it is the best one easily available to us, and it is of high quality and usability.

The EDA is further detailed in the notebook DLEDA.ipynb which can be found attached to this project.

C. Preprocessing and augmentation

To enhance generalization and prevent overfitting due to the limited and highly correlated dataset, we applied a set of data preprocessing and augmentation techniques tailored to the training and evaluation phases.

For the training set, we implemented a custom `JointTrainTransform` class that applies transformations jointly to both the input image and its corresponding mask. The preprocessing pipeline includes:

- Resizing all images and masks to a fixed resolution of 256×256 pixels.
- Random horizontal flipping.
- Random vertical flipping.
- Color jittering (brightness, contrast, saturation, hue) applied only to the image, to simulate lighting variations and enhance robustness to visual changes.
- Normalization of image pixel values using the standard ImageNet mean and standard deviation.
- Tensor conversion for both image and mask to enable compatibility with PyTorch models.

These augmentations and manipulations were applied exclusively to the training data to maintain fair and consistent evaluation on unseen data.

For the validation and test sets, we applied a minimal transformation pipeline via a `JointEvalTransform` class. It includes resizing to 256×256 pixels, tensor conversion and normalization using the same statistics as the training set.

D. Model architectures

In this project, four convolutional neural network architectures from the literature were implemented and compared. These are designed for semantic segmentation tasks. All models follow an encoder-decoder structure and are implemented from scratch using PyTorch. After that, the group attempted to combine the strengths of these models in a novel, custom-built architecture that was finally compared with the previous models to assess the change in performance.

1. *UNet*

The original UNet [10] is a widely adopted architecture for biomedical image segmentation and has been used across multiple domains. It features a symmetric encoder-decoder design with skip connections that bridge corresponding layers in the downsampling and upsampling paths. These skip connections help preserve spatial information and improve segmentation accuracy. Each block comprises two convolutional layers followed by ReLU activations and batch normalization.

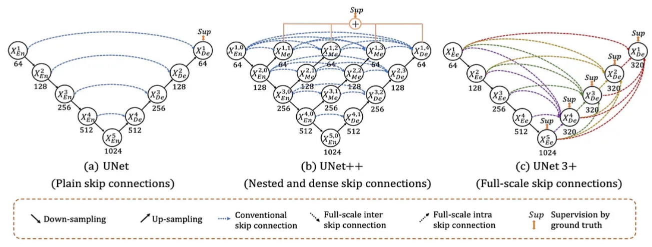

(a) UNet
(Plain skip connections)
(b) UNet++
(Nested and dense skip connections)
(c) UNet 3+
(Full-scale skip connections)
Down-sampling
Up-sampling
Conventional skip connection
Full-scale inter skip connection
Full-scale intra skip connection
Sup Supervision by ground truth

FIG. 2: UNet, UNet++ and UNet3+ architectures

2. *TinyUNet*

TinyUNet [11] is a lightweight version of the original U-Net architecture, designed with fewer layers and smaller feature maps to reduce training time and memory usage. It comprises three downsampling blocks, a bottleneck, and three corresponding upsampling blocks with skip connections. Despite its compact size, TinyUNet effectively captures essential features and serves as a baseline for evaluating segmentation performance under limited resources.

3. *UNet++*

U-Net++ [12] enhances the original U-Net by introducing nested and dense skip connections to better capture multi-scale features. It replaces simple skip connections with a series of convolutional blocks that progressively refine the segmentation maps. This design increases representational capacity and helps reduce the semantic gap between encoder and decoder feature maps.

4. *ResUNet*

ResUNet [13] enhances the original U-Net architecture by incorporating residual connections, which help mitigate the vanishing gradient problem and enable the training of deeper networks. It integrates residual blocks within both the encoder and decoder paths, allowing for more effective feature propagation and improved gradient flow. This design not only enhances the network's ability to capture intricate details but also boosts segmentation performance by preserving both low- and high-level features throughout the network.

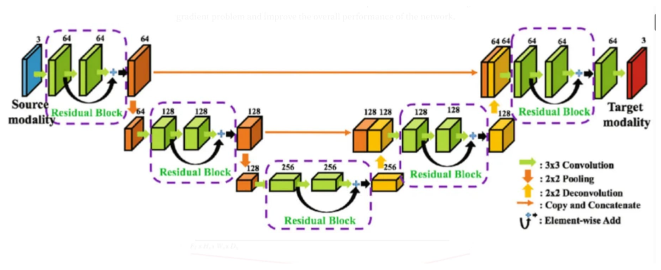


FIG. 3: Residual UNet architecture

5. *Custom model: deeper ResUNet*

During this project, different iterations of the ResUNet architecture were trained and tested, varying in the number of residual layers in the model, such as the architecture in figure 4.

6. *Custom model: Residual UNet++*

Based on the observations that are described in section IV C, it was decided to reuse the idea of UNet++ with its dense skip connections but to enhance it with residual connections from ResUNet. The idea behind this is to combine the building blocks of these two architectures to assimilate part of their strengths. This will be discussed in further details later in the report.

E. Training setup

To train and evaluate our segmentation models, we split the dataset into three subsets using a 70-15-15 ratio. Specifically, 70% of the images were used for training, 15% for validation during model development and tuning, and the remaining 15% for final testing. The splitting was randomized but stratified to maintain consistent distribution of foreground (player) instances.

PyTorch's DataLoader is used with batch sizes of 2, 4, or 8, depending on our GPU memory. Data augmentation was applied to the training set using horizontal and vertical flipping, random color jitter, and resizing to a fixed resolution of 256×256 pixels. Validation and test sets were only resized and normalized to ensure consistent evaluation.

All models were trained using the Adam optimizer with learning rates ranging from 10^{-5} to $10-3$. The number of epochs varied from 20 to 40 based on the hyperparameter grid. The main loss function used

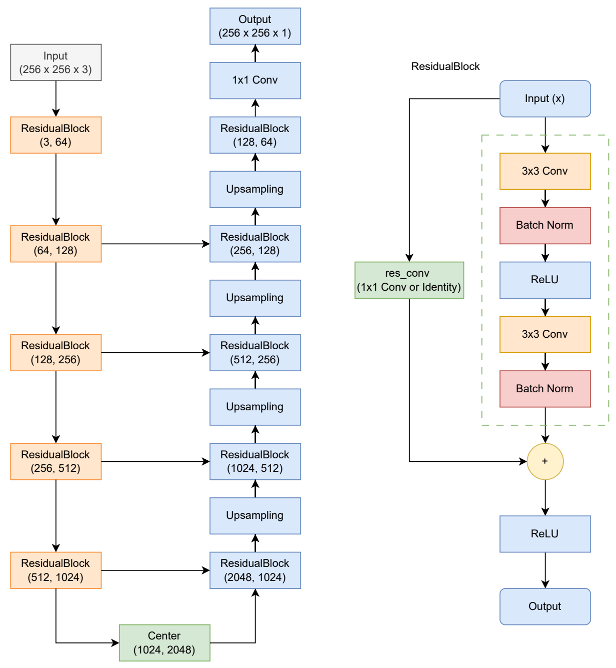


FIG. 4: Custom Residual Unet with additional layers.

was BCEWithLogitsLoss but other loss functions were tested, e.g. the dice loss function. Final performance was evaluated on the test set using multiple segmentation metrics.

F. Post-processing

Following the prediction of segmentation masks by the trained models, we applied a simple yet essential post-processing step to convert the raw network output into binary masks suitable for evaluation and visualization. Since the models output logits (unnormalized values), we first transformed the data by applying a sigmoid function to map predictions to probabilities between 0 and 1. These probabilities were then thresholded using a fixed cutoff value of 0.5 to generate binary segmentation masks.

No additional morphological operations such as dilation, erosion, or contour extraction were applied in this study, as our goal was to compare different model architectures under minimal post-processing assumptions. This approach ensures that performance differences are mainly attributable to the model's learning capacity rather than auxiliary processing steps.

This minimal post-processing pipeline provides a direct and transparent way to evaluate model outputs while preserving the integrity of pixel-level predictions.

IV. RESULTS

The primary metric for model evaluation is Intersection over Union (IoU), which is more suited to the task and more informative than accuracy scores that typically range from 98-99% for all models(due to the class imbalance). IoU penalizes both the inclusion of pixels that should not be part of the mask and the exclusion of pixels that should be included. This metric effectively balances precision and recall into a single measure while maintaining a stricter criterion for precise boundaries compared to the Dice coefficient. Given that the annotations are ”pixel-perfect” rather than simple bounding boxes, it was decided to more heavily penalize the model for imperfect boundaries.

A. Image manipulation

The first test that was conducted is the verification that the image manipulations conducted during the training of the model rendered it more robust. To assess whether this assumption was true, the simple UNet architecture has been evaluated on manipulated and non-manipulated data, and then validated and tested on non-manipulated validation and test sets. Hyperparameters were set as follows: learning_rate $= 10^{-4}$, batch_size $= 4$, num_epochs $= 40$. loss function was BCEWithLogitsLoss from the module torch.nn. The results can be found in I and show that there is a slight improvement of the IoU coefficient. The image treatment was thus validated and it was applied to all subsequent experiments.

Image manipulation	IoU	Dice	Pixel accuracy
no	0.7144	0.8333	0.9940
yes	0.7409	0.8511	0.9947

TABLE I: Comparison of the results with and without image manipulation applied to the training set.

As noted previously, Pixel Accuracy is not a very insightful metric, as such it will be excluded from the subsequent analyses. After that, a comparison of the architectures from the literature was made.

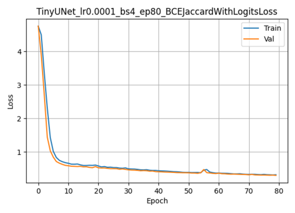
B. Loss Function

For this section, the model considered is TinyUNet, the hyperparameters were chosen as follows: learning_rate $= 10^{-4}$, batch_size $= 4$, num_epochs $= 80$. The compared loss functions will be BCEWithLogitsLoss, BCEJacquardWithLogitsLoss and BCEDiceWithLogitsLoss.

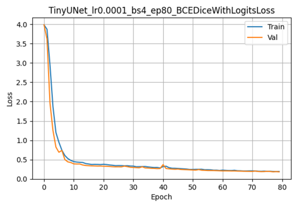
Loss function	IoU	Dice	Precision	Recall
BCEWithLogitsLoss	0.7834	0.8785	0.8745	0.8830
BCEJaccardWithLogitsLoss	0.7740	0.8725	0.8628	0.8830
BCEDiceWithLogitsLoss	0.7608	0.8641	0.8759	0.8529
FocalWithLogitsLoss	0.7092	0.8298	0.8599	0.8020
MixedLoss	0.7621	0.8649	0.8845	0.8466
FocalLoss	0.7767	0.8742	0.8795	0.8693

TABLE II: Comparison of the models found in the literature.

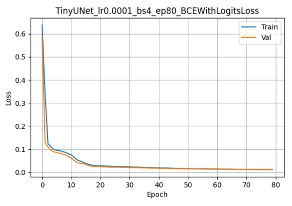
In the table II, the most interesting loss functions appear to be BCEWithLogitsLoss, BCEJaccardWithLogitsLoss and FocalLoss. Further investigation was conducted by comparing the convergence of these losses: for this purpose, the evolution of accuracy from epoch to epoch was tracked and plotted. As shown in Figure 5.



(a) BCEJaccardWithLogitsLoss



(b) BCEDiceWithLogitsLoss



(c) BCEWithLogitsLoss

FIG. 5: Evolution of the value of the loss function score for different loss functions.

The chosen loss function for this work was BCEWithLogitsLoss to standardize comparisons across different models and configurations. This decision was made considering that the differences in performance between BCEWithLogitsLoss and the other evaluated loss functions (such as BCEDiceWithLogitsLoss and BCEJaccardWithLogitsLoss) were marginal. By using a single, consistent loss function, we ensure that any observed differences in model performance are attributable to the model architectures or hyperparameters rather than variations in the loss function. Additionally, BCEWithLogitsLoss is computationally efficient and provides a stable training process, which further supports its selection for this study.

C. Literature model comparison

For this section, the hyperparameters were chosen as follows: learning_rate $= 10^{-4}$, batch_size $= 4$, num_epochs $= 40$. loss function was BCEWithLogitsLoss from the module torch.nn.

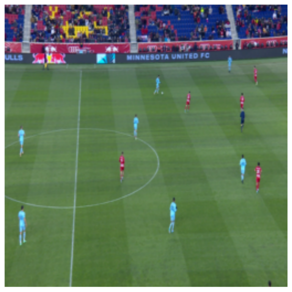
Model	IoU	Dice	Precision	Recall
TinyUNet	0.6908	0.8171	0.8688	0.7714
UNet	0.7409	0.8504	0.8504	0.8526
UNet++	0.8163	0.8988	0.9269	0.8728
ResUNet	0.8984	0.9464	0.9443	0.9489

TABLE III: Comparison of the models found in the literature.

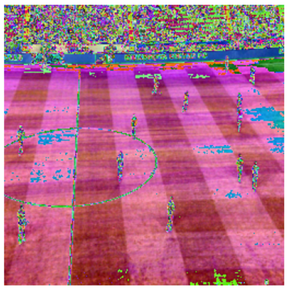
From the table III, we can see that the most interesting models are UNet++ and ResUNet

D. Mask generation

To qualitatively assess the segmentation performance, we visualize one representative instance from the validation set, showing the input image, the corresponding ground truth mask, and predicted masks from each model.



(a) Resized image



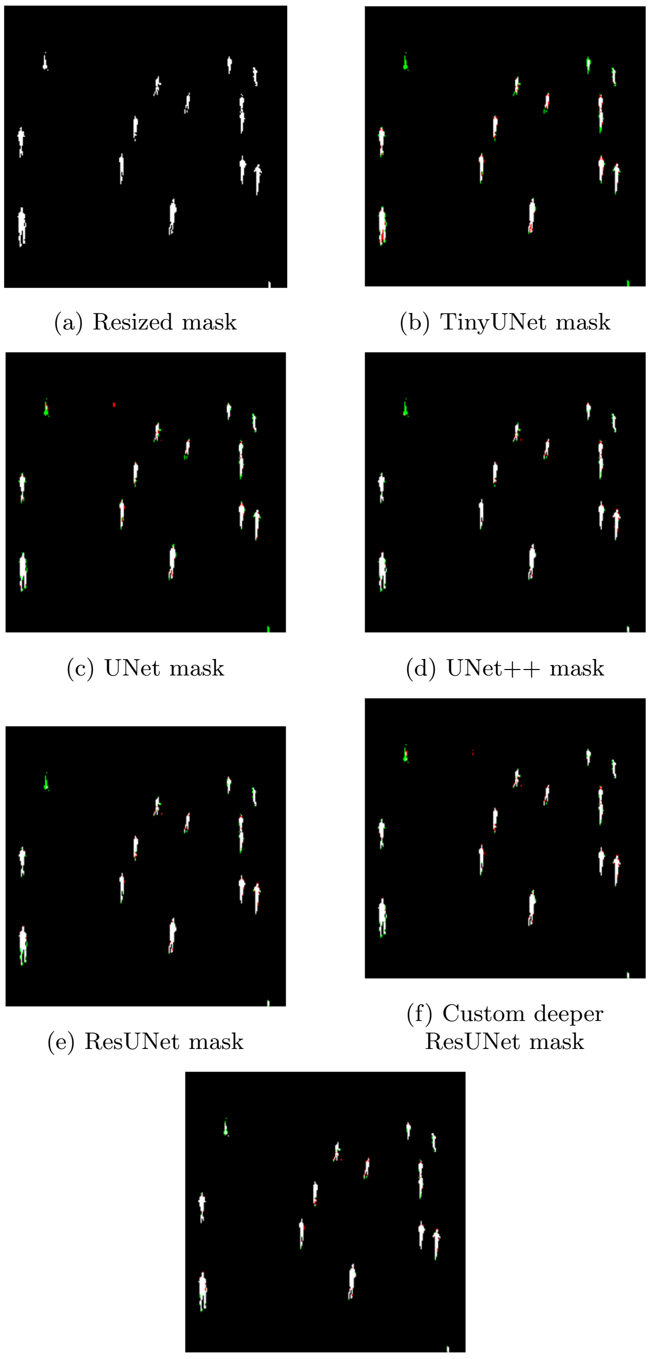
(b) Transformed image

FIG. 6: Image after resizing and transforming.

Figure 6 illustrates the image after preprocessing, including resizing to 256 × 256 pixels and transformation via normalization. Despite the reduced resolution, the overall scene layout and player silhouettes remain visually distinguishable.

Figure 7 shows the corresponding ground truth mask alongside predicted masks generated by five models: TinyUNet, UNet, UNet++, ResUNet, and a deeper custom UNet. In these visualizations, white pixels indicate correct predictions (true positives), green pixels highlight false positives (background incorrectly classified as player), red pixels indicate false negatives (players missed by the model).

Across models, several patterns emerge:



(a) Resized mask

(b) TinyUNet mask

(c) UNet mask

(d) UNet++ mask

(e) ResUNet mask

(f) Custom deeper ResUNet mask

(g) Custom ResUNet++ mask

FIG. 7: Truth mask and predicted masks.

- TinyUNet, while lightweight and fast, often misses finer object boundaries and tends to merge nearby players. It struggles with thin or partially visible silhouettes, especially for players farther from the camera.

- UNet improves segmentation quality, producing fuller player shapes and slightly reducing edge-related errors. However, predictions still degrade in regions with clutter or variable lighting.

- UNet++, thanks to its nested skip connections, better preserves object boundaries and handles occlusions more effectively. It yields smoother contours and reduces scattered prediction noise.

- ResUNet demonstrates increased robustness through residual connections, delivering more accurate predictions, especially for players with partial occlusions or under motion blur.

- Deeper ResUNet on the other hand showed poor results, especially with low learning rate and few epochs where the performances were bad. With higher learning rate and number of epochs it showed results closer but still a bit lower than other models (best deeper ResUNet model, with a learning rate of 0.00100 trained for 40 epochs had a IoU index of 0.849725 and a dice index of 0.918749.

- Custom UNet++ with residual blocks performs best among all models. It consistently detects complete player silhouettes and is visually closest to the ground truth with fewer false positives and negatives. Looking at the different predicted masks a difference with other models arises with the referees and linemen that are more clearly identified.

Despite these strengths, a consistent failure across all models is the segmentation of players located near the advertising boards or grandstand. In these areas, color and texture similarities, such as red jerseys blending with red advertisements or audience patterns resembling human figures, introduce ambiguity. Models often either miss these players or mistakenly classify background structures as foreground.

These visual results suggest that deeper architectures improve segmentation quality, but complex backgrounds remain challenging. Future improvements could involve attention mechanisms or context-aware modules to better separate players from distractor elements.

E. Model performance comparison

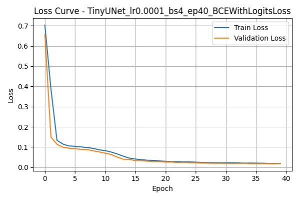
The table IV shows the final results of the models compared previously, with the addition of the custom made model. For this section, the hyperparameters were chosen as follows: learning_rate = 10^{-4}, batch_size = 4, num_epochs = 40. loss function was BCEWithLogitsLoss from the module torch.nn.

The results show that deeper and more sophisticated architectures generally outperform smaller models in segmentation accuracy. Among all tested models, the custom ResUNet exhibited the best performance, demonstrating strong capability in segmenting both foreground players and object boundaries. UNet++ also delivered robust results, benefiting from its dense skip connections

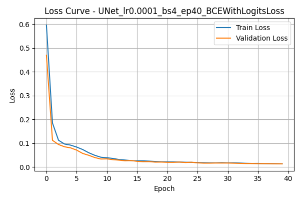
Model	IoU	Dice	Precision	Recall
TinyUNet	0.6908	0.8171	0.8688	0.7714
UNet	0.7409	0.8504	0.8504	0.8526
UNet++	0.8163	0.8988	0.9269	0.8728
ResUNet	0.8984	0.9464	0.9443	0.9489
Custom ResUNet++	0.8529	0.9206	0.9322	0.9094
Deeper ResUNet	0.8346	0.9098	0.9250	0.8954

TABLE IV: Comparison of the models found in the literature with the custom made models.

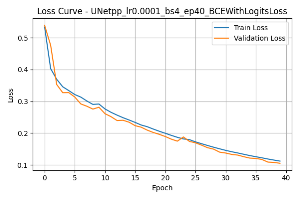
that enable multi-scale feature aggregation. Standard UNet performed adequately, while TinyUNet showed clear limitations in detecting fine details and managing background clutter. However, the deeper did not always mean much better: It was the case with the residual UNet model whose depth was increased, adding more layers of residual blocks. Its training took longer but the results were not significantly better, attaining an apparent limitation from the technology.



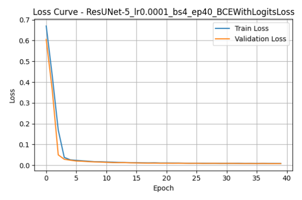
(a) TinyUNet



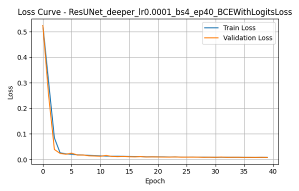
(b) UNet



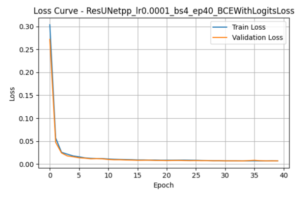
(c) UNet++



(d) ResUNet



(e) Custom deeper ResUNet



(f) Custom ResUNet++

FIG. 8: Loss curves across architectures.

Based on the loss curves, it can be seen that multiple models follow a similar trend of a sharp initial decline in both the training loss and validation loss. Most of the models reach a stationary point around zero.Despite this, the very low and stable validation loss suggests that the learning is effective and good generalisation can be made to unseen data. This will be subject to discussion in the section V C.

F. Retrained model

Based on the results of the hyperparameter combinations, the custom ResUNet++ model achieved a good performance on the validation set in terms of Dice coefficient and Intersection over Union (IoU). Therefore, the custom ResUNet++ is selected as the retrained model for deployment and evaluation. To make optimal use of the labeled data, the selected model was retrained from scratch using the combined training and validation sets. Once trained, the model was evaluated on the held-out test set to estimate its generalization performance. The final model achieved a Dice score of 0.93 and an IoU of 0.86, on the test set, confirming its robust segmentation capabilities. A qualitative analysis of the test predictions also shows that the model successfully captures most player, although some challenging edge cases near the background structures still persist. The loss curves are shown in Figure 9.

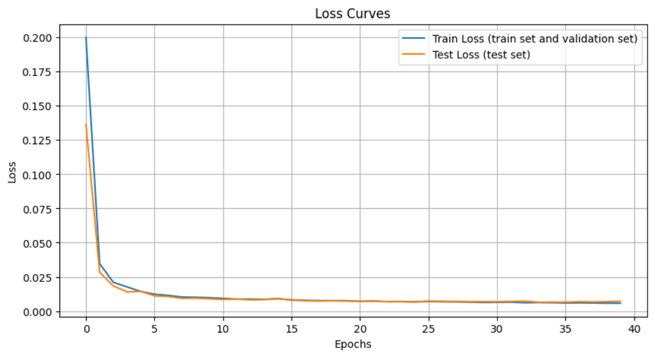


FIG. 9: Loss curves for Custom ResUNet++ model.

V. DISCUSSION

While conducting the analysis of the performances, it was noted that the dataset considered in our work only includes data from a single football match, so there will inevitably be a generalisation error if the model is tested on a different dataset. See Section V C.

A. Working methods

While comparing the results, it was noticed that there were discrepancies in the results yielded by the models. In order to reduce these effects, all the final results are computed on the same machine, in the same environment. Additionally, it was noted that in order to present consistent results, there is a need to train and evaluate the same model multiple times and retain the average of the results. This allows for a robustification of the results with respect to random effects.

B. Trade-offs between performance and efficiency

The performance improvements of deeper models came at the cost of increased computational complexity and memory usage. For instance, ResUNet and UNet++ required longer training times and higher GPU memory compared to TinyUNet. In contrast, TinyUNet trained faster and with minimal resource usage, making it more suitable for deployment on edge devices or under hardware constraints-albeit at a significant drop in segmentation quality. The storage of these models can also have an impact on the choice of the model, the more complex models being heavier than the other ones. Therefore, selecting a model for real-world deployment requires balancing performance and resource availability.

C. Limitations

This project faced several limitations:

- **Dataset limitations:** The dataset comprised 527 annotated images from a single football match. Despite frame sub-sampling, many frames were visually similar, limiting the diversity of training examples and introducing temporal redundancy. The viewing angle of the images of the dataset is also very similar and unlike color properties that are modified and augmented with *pytorch* through transformations on images, it is not the case of the angle in which the images were taken. Another limit which in this case was noticed too late is the way the data was split, which is randomly. In the case of our data, images are spaced apart by a few frames only, they might therefore be very similar. With random split, it is therefore possible that nearly identical images can be found in two different sets (training, validation and test set), resulting in possibly too optimistic results. If the split was made taking this information into account (for example based on the ordering of the images), results on the test and validation set might get a bit lower but be more reliable.
- **Memory constraints:** Due to limited GPU resources, we used a fixed resolution of 256×256 and relatively small batch sizes. This constrained our ability to scale model complexity or image fidelity.
- **Generalisation:** Currently, the model can only recognize players from an elevated and central perspective on the field. If used for other perspectives, the model would likely perform poorly because the backgrounds would differ significantly. For example, a player might be standing in front of an advertisement, the football stands, goals, or other elements that differ enormously from the green grass that appears behind most of the players in this dataset.

D. Future work

To address current limitations and further improve performance, several directions could be explored:

- **Efficient architectures:** Models such as DeepLabV3+, Swin-Unet, or Mobile-UNet offer improved performance with reduced computational cost and could be explored as scalable alternatives.
- **Video-based learning:** Incorporating temporal information from video sequences using 3D CNNs or recurrent modules can enhance the consistency and robustness of predictions. However, this approach requires high-quality annotated video feeds, which can be challenging to obtain. The idea is to gather additional information from the correlation between successive images to improve prediction accuracy.
  Alternatively, researching and training unsupervised models like U2Seg could alleviate the need for annotations, though it would increase the computational complexity of the models. [14]
- **Dataset enhancement:** Collecting data from multiple matches and applying synthetic data generation or advanced augmentations would increase generalizability.
- **Incorporation of more diversified data:** Including high-quality datasets with diversified camera perspectives will improve the generalization potential. This will potentially enable the model to recognize portions of players and focus more on person-specific traits such as hairstyle, skin color, or shoes, details that are difficult to discern with the current dataset.

VI. CONCLUSION

This project explored the application of various deep learning architectures for the task of football player segmentation in match images. By implementing and evaluating multiple models—including U-Net, TinyUNet, U-Net++, and ResUNet—the study aimed to identify the most effective approaches for accurate and robust segmentation in sports analytics.

The findings indicate that deeper and more sophisticated architectures, such as ResUNet and U-Net++, generally outperform simpler models like TinyUNet, up to a certain limitation of the technique. These advanced models demonstrated superior performance in segmenting players and preserving object boundaries, thanks to their enhanced feature representation capabilities. However, these improvements come at the cost of increased computational complexity and memory usage, highlighting the trade-offs between performance and efficiency.

Despite the promising results, the study faced several limitations, including dataset constraints, and memory limitations. The dataset's lack of diversity posed challenges to model generalization and performance. Additionally, the current model's ability to recognize players is limited to an elevated and central perspective, which may not generalize well to other viewpoints or backgrounds. It is possible to cite the example of a goalkeeper standing behind a pole, or the linesman standing in front of billboards which posed difficulties for the precision of the segmentation.

This work contributes to the field of semantic segmentation in deep learning by providing a comprehensive comparison of different architectures and their performance on a specific sports-related task. The findings offer insights into the strengths and limitations of each model, guiding future research and development in sports analytics and beyond.

VII. ACKNOWLEDGEMENTS

This project has been conducted with the help of AI: Open AI's Chat GPT and Mistral AI's Le Chat were used in order to gather informations and to aid the writing of the report.
Github's Copilot was used for code completion and debugging.

The images used for the description of the models were either taken from Sik-Ho Tsang's blog [15] or made by group members in the case of our custom model.

[1] Doyeun. Football players segmentation with u-net (pytorch). `https://www.kaggle.com/code/doyeun/football-players-segmentation-with-u-net-pytorch`, 2023.

[2] Chetan KRJNNCE. Football players segmentation with u-net (pytorch). `https://www.kaggle.com/code/chetankrjnnce/football-players-segmentation-with-u-net-pytorch`, 2023.

[3] WeigelK. Football player segmentation with u-net. `https://www.kaggle.com/code/weigelk/football-player-segmentation-with-u-net`, 2023.

[4] Asma Masssad. U-net multiclass segmentation v3. `https://www.kaggle.com/code/asmamasssad/unet-multiclass-segmentation-v3`, 2023.

[5] Drmwnnrafi. Football players segmentation with tf u-net + u2net. `https://www.kaggle.com/code/drmwnnrafi/football-players-segmentation-with-tf-unet-u2net`, 2023.

[6] Squidbob. Segment anything model - how to. `https://www.kaggle.com/code/squidbob/segment-anything-model-how-to`, 2023.

[7] Yaroslav Isaienkov. Football player segmentation modeling. `https://www.kaggle.com/code/ihelon/football-player-segmentation-modeling`, 2023.

[8] Xinkai Liao. Football player detection. `https://www.kaggle.com/code/xinkailiao/football-player-detection`, 2023.

[9] Sadhliroomyprime. Football semantic segmentation. `https://www.kaggle.com/datasets/sadhliroomyprime/football-semantic-segmentation`, 2022. Accessed: 2025-04-29.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[11] Junren Chen, Rui Chen, Wei Wang, Junlong Cheng, Lei Zhang, and Liangyin Chen. TinyU-Net: Lighter yet Better U-Net with Cascaded Multi-Receptive Fields . In *proceedings of Medical Image Computing and Computer Assisted Intervention – MICCAI 2024*, volume LNCS 15009. Springer Nature Switzerland, October 2024.

[12] Nima Tajbakhsh Jianming Liang Zongwei Zhou, Md Mahfuzur Rahman Siddiquee. Football semantic segmentation. `https://arxiv.org/abs/1807.10165`, 2018.

[13] Peter Caccetta Chen Wu Foivos I. Diakogiannis, François Waldner. Resunet-a: a deep learning framework for semantic segmentation of remotely sensed data. `https://arxiv.org/abs/1904.00592`, 2019.

[14] Xinyang Han Long Lian Roei Herzig Trevor Darrell Dantong Niu, Xudong Wang. Unsupervised universal image segmentation. `https://arxiv.org/abs/2312.17243`, 2023.

[15] Sik-Ho Tsang. Unet 3+ — a full-scale connected unet (medical image segmentation). `https://sh-tsang.medium.com/reading-unet-3-a-full-scale-connected-unet-medical-image-s` 2020.