”Integrating YOLOv8 and MediaPipe for Accurate Pose Detection and Replication in Unity 3D Environments”

Raja Ali Akhtar
Department of Computer Engineering
Information Technology University
Lahore,Pakistan.
Email : bsce21035@itu.edu.pk

***Abstract*—This research addresses a key challenge in virtual character animation: enabling seamless motion replication by simply recording human movements. The primary focus is on fine-tuning the YOLOv8 architecture for real-time pose detection. A custom dataset, curated by merging samples from the COCO 2017 dataset and the OChuman dataset, is utilized to train YOLOv8, optimizing its performance for detecting human poses accurately.**

In parallel, a pretrained MediaPipe model in OpenCV (cv2) extracts pose keypoints from video frames, offering an alternative approach for pose detection. Comparative analysis evaluates the efficacy of both the fine-tuned YOLOv8 model and the pretrained MediaPipe model, emphasizing accuracy, speed, and practical applicability.

The key innovation lies in the seamless integration of the fine-tuned YOLOv8 model with Unity 3D environments, enabling automatic motion replication. Experimental results validate the effectiveness of the proposed solution, showcasing its potential for various applications, including virtual reality experiences, gaming, and motion capture. The research contributes to advancing the field of computer vision and virtual character animation, offering a user-friendly approach for animating 3D characters with real-world movements.

***Index Terms*—YOLOv8, fine-tuning, pose detection, Unity 3D, motion replication, COCO 2017,OChuman**

I. Introduction

Recent advancements in computer vision and virtual reality have increased interest in automating 3D character animation. Traditional approaches, such as manual keyframing or elaborate motion capture setups, are time consuming and resource costly. This paper solves this issue by introducing a revolutionary deep learning-based mechanism for real-time posture identification and replication. The goal is to fine-tune the YOLOv8 architecture and compare it to a pretrained MediaPipe model for pose estimation. The final objective is to seamlessly incorporate these models into Unity 3D settings, automating 3D character animation. This method streamlines the animation process while also increasing realism by obtaining exact posture keypoints from video footage. The relevance of this study stems from its potential to transform virtual character animation across several fields.

Identify applicable funding agency here. If none, delete this.

Fig. 1. (a) character in 3d environment (b) person moving in real environment

II. Related Work

Related work in the field of video-to-3D character motion transfer:

1) **Markerless Motion Capture:** Computer vision techniques to extract human motion from video without markers or specialized equipment such as sensors. [1]
2) **Pose Estimation and Tracking:** Algorithms estimate 2D or 3D human pose and track motion across frames. [2]
3) **Motion Retargeting and Synthesis**: Techniques transfer motion between characters or generate new motion sequences. [3]
4) **Deep Learning for Motion Generation:** Neural networks learn motion patterns from data and generate realistic motion. [4]
5) **Applications in Animation and VR:** Motion capture technology enhances animation, virtual reality, and gaming experiences. [5]
6) **Evaluation Metrics and Benchmarks**: Metrics assess pose accuracy, temporal coherence, and realism of motion sequences. [6]

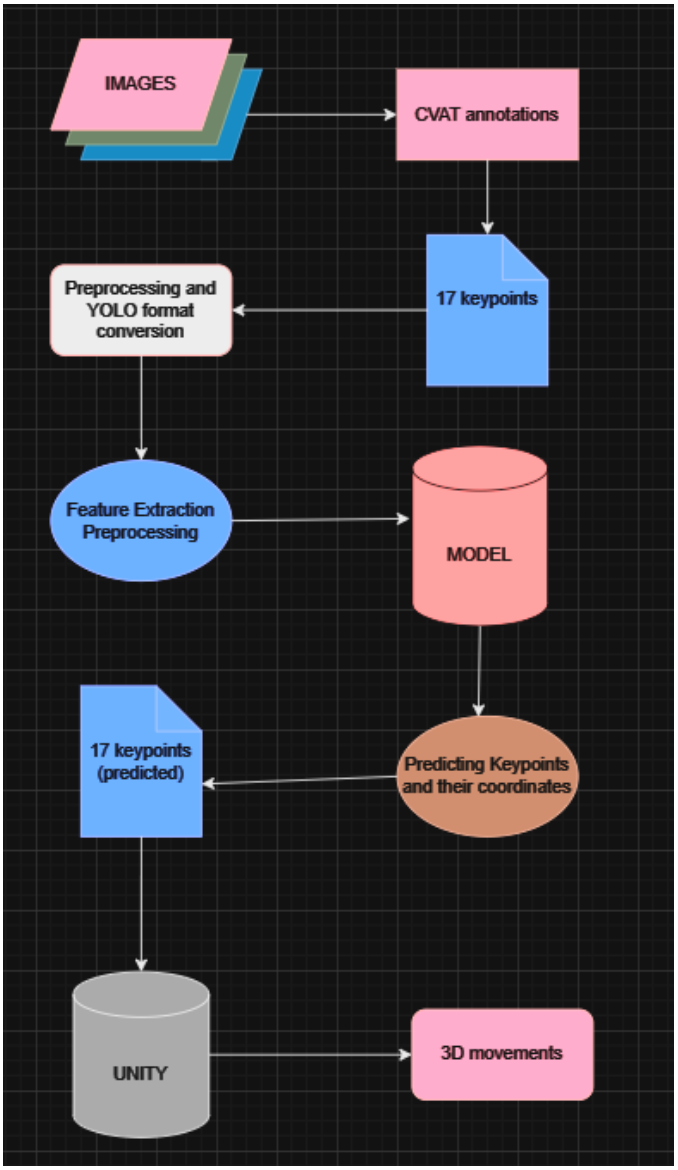

Fig. 2. Overall working of system

III. Methodology

The overall Methodology is shown in Fig 2. Images of 640 x 640 enters the system.it can also be the frames of a video. The output of the system is 17 keypoints with every keypoint having x, y and z coordinate values. Other stages are defined below :

A. Implementation setup

1) **Hardware Setup Details:**

-Processor : AMD Ryzen 3600

- Frequency: 3.6 GHz / 4.2 GHz (Turbo Core)
- 6 cores / 12 threads.
- 35 MB GameCache (32 MB L3 + 3 MB L2)
- Socket AMD AM4.
- Memory controller : Dual Channel DDR4.
- Engraving : 7nm FinFET.

Fig. 3. Enter Caption

- Maximum thermal envelope (TDP): 65W.

-Graphics Card: Nvidia GTX 1660 Super

a) Hardware Specifications:

- 6GB VRAM GDDR6
- 192-bit memory interface
- 1,408 CUDA cores

2) **Software Setup Details:**

- Environments

a) Development Tools:

- Environment for programming Python-related functionalities was **PyCharm**.
- **Unity 3D** 2024 for 3D environments.
- **Microsoft Visual Studio Community** for C# scripts.

- Programming Languages

a) Software Tools:

- **Python 3.12** and Python 3.8 (for cv2)
- **C#** for Unity environment scripting

- Annotations / Preprocessing

a) Tools Used:

- Computer vision annotation Tool is used for annotations [7].
- Robo flow for data preprocessing and splitting [8].

- Libraries

a) Tools Used:

- Ultralytics
- Ultralytics YOLO-v8
- CvZone
- CvZone 2
- PyTorch
- TensorFlow
- CUDA Toolkit

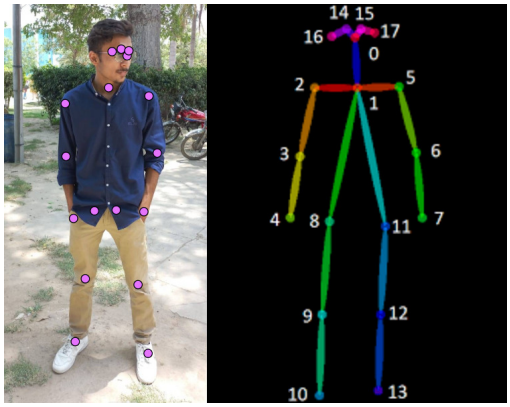

Fig. 4. Data Annotations

B. Data Acquisition

Dataset Description: This dataset contains images from Both COCO dataset 2017 and OChuman datset. since both these dataset contains images in thousands. My dataset contains almost 3000 images where single subject is in different postures as shown in Fig 3.

- ***Dataset Annotations:*** Using Computer Vision Annotation Tool images are mannually annotated and all data is stored in the form of YOLO txt format which is acceptable by YOLOv8 model.One annotation file for one image.
- ***Preprocessing :*** In preprocessing I used roboflow for removing pictures with multiple persons in it.
  - Auto-Orient (orientation correction)
  - Resize (All input images should be 640 x 640p)
  - Auto-Adjust Contrast (color correct the images)
- ***Augmentation :*** In augmentation 10 percent of the training dataset was augmented with grayscale and fliped images. As you can see in Fig 3.
- ***Dataset Spliting:*** To facilitate robust model evaluation and generalization, the dataset underwent stratified random splitting into training, validation, and test sets. Out of the total 3400 images, approximately 15 percent (500 images) were reserved for testing, and an additional 15 percent (500 images) were allocated for validation. The remaining 70 percent of the dataset (approximately 2400 images) was utilized for training the model. This partitioning strategy ensures that the model is trained on a diverse range of data while allowing for unbiased performance assessment on unseen data. Each set was carefully curated to maintain a representative distribution of poses and activities, thereby facilitating robust evaluation and validation of the trained model's performance.

C. Model : YOLO-v8

The architecture of YOLOv8, a convolutional neural network (CNN) used for object detection. YOLOv8 builds upon prior YOLO versions and incorporates various architectural and developer experience improvements . Here's a breakdown of the key architectural components:

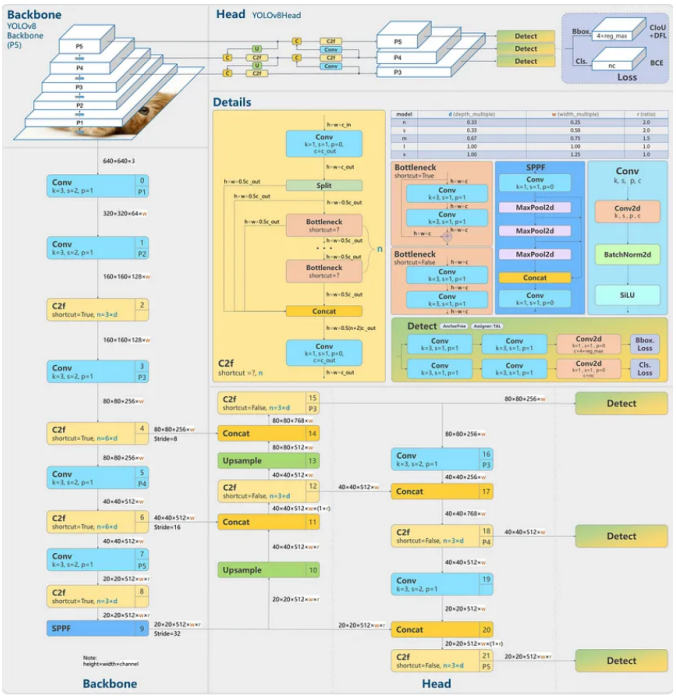

Fig. 5. YOLO-v8 architecture

- **Backbone:** This is the initial layer that extracts features from the input image. YOLOv8 utilizes a modified version of the CSPDarknet53 architecture, which consists of 53 convolutional layers and employs cross-stage partial connections to enhance information flow across various layers .
- **Neck :** While not explicitly labelled in the image, the neck section merges feature maps extracted from the backbone. In YOLOv5, the neck utilizes SPP (Spatial Pyramid Pooling) and PAN (Path Aggregation Network) structures .
- **Head:** The head is responsible for generating the final predictions. It consists of multiple convolutional layers followed by fully-connected layers. The image shows several detections (Detect) throughout the network, indicating the model can predict objects at various scales within the image .

Here are some additional details:

- The image shows two pathways through the network: P5 and P4. These likely represent feature maps of different resolutions used for object detection at various scales .
- Conv layers (convolutional layers) are used throughout the network to extract features from the image.
- Bottleneck refers to a residual block that allows for deeper networks while mitigating the vanishing gradient problem .
- Skip connections (represented by arrows) bypass certain layers and directly connect to subsequent layers, facilitating the flow of information throughout the network .

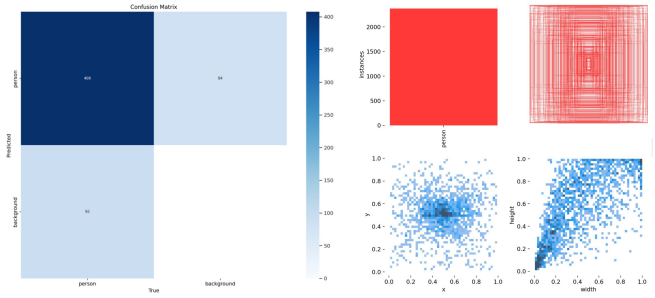

Fig. 6. Confusion Matrix

Overall, the YOLOv8 architecture leverages a backbone for feature extraction, a neck for combining features, and a head for generating detections, achieving efficient object detection across various image resolutions.

D. Model Training

The Yolo V8 accepts the data only in the form of YOLO txt format. So I converted my COCO format to YOLO format. With the following configurations,, the model was trained. `EPOCH 84/84 Batchsize=16 imgsz=640` Other parameters are `weight decay = 0.005 , momentum=0.937 , learning rate=0.01.` Other model properties are `250 layers , 329540 parameters , 3295470 gradients , 9.3 Gflops , 397 pretrained weights`.The dataset split looked like this :`Train=2368 , Val=500 , Test =500`. The optimiser used in model training is `optimizer = ADAM Weights`.The model took 10.29 hours to train.

E. Performance of the Model

- Figure 8 shows graphs of training and validation loss for both bounding box and pose estimation tasks. Across about 84 epochs, we see a decrease in loss for both tasks. However, this model was trained for 100 epochs and the results were clearly showing validation and training loss moving apart from each other so we used early stopping in order to avoid over fitting.
- Training this model requires a lot of computational resources, especially in terms of time and processing power. This highlights the demanding nature of the training process so that is why all combinations of tune-able parameters are not checked.
- The correlogram in Fig 7 analysis reveals that labels within the YOLO architecture are normalized. This means that the data is adjusted to a standard mean position, which is important for stabilizing and optimizing the training process. This normalization is achieved through batch normalization layers within the YOLO architecture.
- The precision Graphs shows the accuracy of the model.The plots of mAP50 and 50to 95 are shown here which are saturating after 0.8.mAP 50 is used for general accuracies while mAP 95 is used for specific application

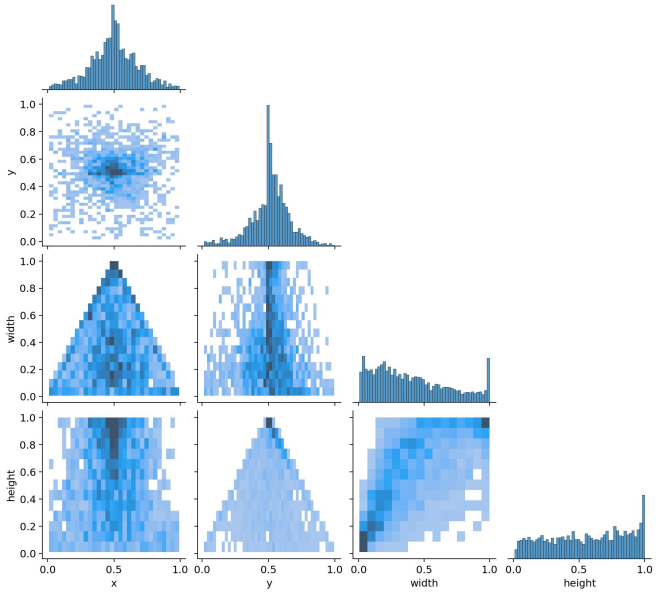

Fig. 7. correlogram

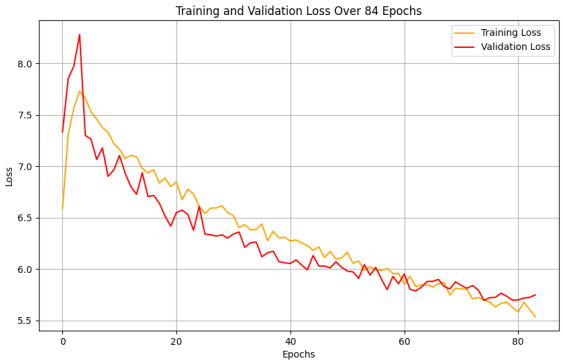

Fig. 8. Loss functions with respect to Epoch

errors.precision and accuracy graphs are also suggesting that the model is performing well enough.

- The figure 10 shows my my friend being tracked by my model doing Karate.The model is accurately tracking all pose landmarks even in complex state.
- The confusion matrix Figure 6 showing how well the classification of person or empty background is taking place. 18 percent of the person image are not recognized while other are recognized correctly.

F. Integration with Unity

- The model outputs keypoints along with their corresponding x, y, and z coordinates. These keypoints represent specific points on the detected object, such as joints in a pose detection task. Once the model generates these

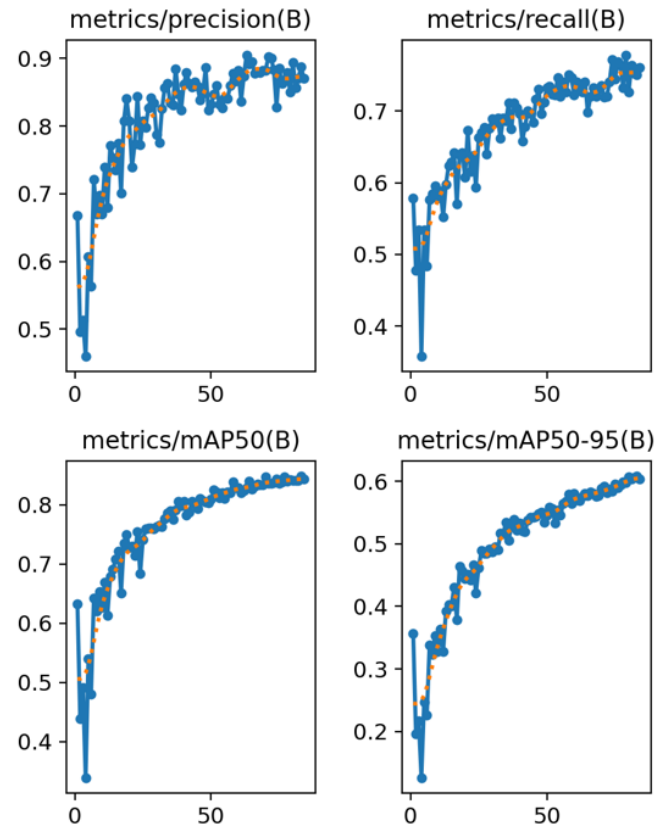

Fig. 9. Precision-Recall Graphs

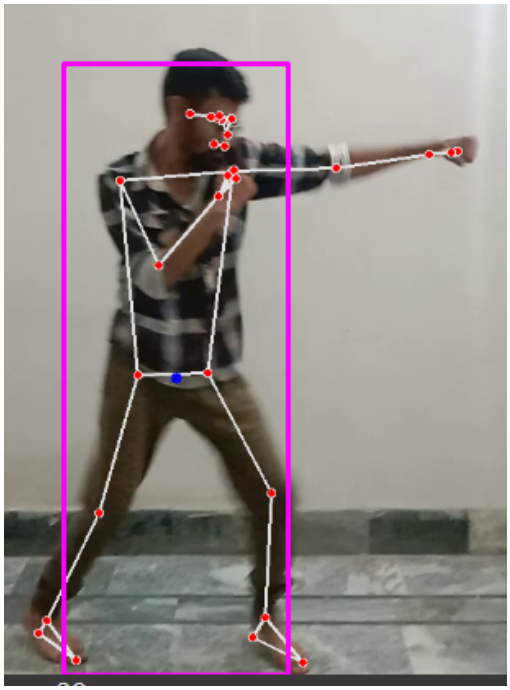

Fig. 10. Model tracking landmarks

Metric	YOLO V8	Mediapipe
Accuracy	60 to 70%	75 to 80 %
Speed	15-45 FPS	30-60 FPS
Size	50-200 MB	5-15 MB

TABLE I
COMPARISON OF YOLOV8 AND MEDIAPIPE PERFORMANCE IN POSE ESTIMATION.

keypoints, they are passed to a Unity C sharp script for further processing.

In the Unity environment, the C sharp script reads the keypoints and their coordinates received from the model. This script then uses this information to animate a 3D character or object within the Unity scene. By mapping the keypoints to specific parts of the character's body or object, the script can accurately replicate the movements detected by the model in real-time. This allows for seamless integration of the model's output into the Unity environment, enabling the creation of interactive applications or experiences that respond to real-world movements captured by the model.

G. Model Comparison YOLO-V8 vs Mediapipe

1) YOLOv8 Performance: YOLO; Optimal Speed and Accuracy of Object Detection - This paper provides detailed benchmarks for YOLO models, including mAP and FPS metrics. While it's focused on YOLO, it gives insights into the performance trajectory of the YOLO family. *Source: YOLOv4 Paper*

GitHub Repositories and Benchmarks:

- Several GitHub repositories provide implementations and performance benchmarks of YOLOv8 for pose estimation. These repositories often include details on model size, accuracy, and inference speed.
- Example: YOLOv8 Pose Estimation GitHub

2) MediaPipe Performance:

1) Official MediaPipe Documentation and Papers:
   - Google provides detailed documentation and research papers on MediaPipe, including benchmarks on various devices.
   - Source: MediaPipe Documentation
2) Research Paper on MediaPipe Hands:
   - Provides detailed benchmarks on the accuracy and speed of MediaPipe for hand pose estimation, which is closely related to body pose estimation.
   - Source: MediaPipe Hands Paper
3) User Case Studies and Performance Evaluations:
   - Blog posts and articles where users share their experiences with MediaPipe for pose estimation, including performance metrics on different devices.
   - Example: Real-time Pose Estimation using MediaPipe

CONCLUSION

n conclusion, this research presents an innovative approach to virtual character animation by integrating YOLOv8 and

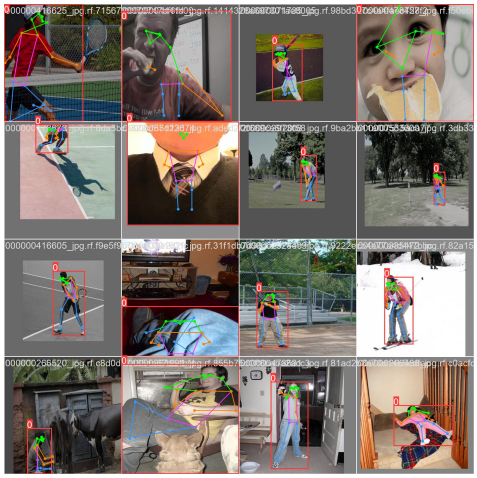

Fig. 11. Some outputs

MediaPipe for accurate pose detection and replication within Unity 3D environments. The fine-tuned YOLOv8 model, trained on a custom dataset derived from COCO 2017 and OChuman datasets, demonstrates promising results in real-time pose detection. Comparative analysis with a pretrained MediaPipe model showcases the efficacy of the proposed solution in terms of accuracy, speed, and practical applicability.

The seamless integration of the YOLOv8 model with Unity 3D environments enables automatic motion replication, allowing for the creation of interactive applications that respond to real-world movements captured by the model. By leveraging deep learning techniques, this research streamlines the animation process and enhances realism in virtual environments.

Overall, this study contributes to advancing the fields of computer vision and virtual character animation, offering a user-friendly approach for animating 3D characters with real-world movements. The proposed solution holds significant potential for various applications, including virtual reality experiences, gaming, and motion capture, paving the way for future developments in interactive storytelling and immersive content creation.

Fig. 12. Some outputs

REFERENCES

[1] Davies, T., Taylor, J., & Morten, T. (2017). "Markerless motion capture using multiple color cameras." *IEEE Transactions on Circuits and Systems for Video Technology, 27*(4), 783-797.

[2] Cao, Z., Simon, T., Wei, S. E., & Sheikh, Y. (2017). "Realtime multi-person 2D pose estimation using part affinity fields." *CVPR*.

[3] Lee, Y., Liu, M. Y., & Grauman, K. (2019). "MoCoGAN: Decomposing motion and content for video generation." *CVPR*.

[4] Holden, D., Saito, J., & Komura, T. (2017). "Deep learning of human character motion." *ACM Transactions on Graphics (TOG), 36*(4), 1-16.

[5] Magnenat-Thalmann, N., & Thalmann, D. (2004). "The Making of Virtual Humans." *Springer Science & Business Media*.

[6] Loper, M., Mahmood, N., & Black, M. J. (2014). "MoSh: Motion and shape capture from sparse markers." *ACM Transactions on Graphics (TOG), 33*(6), 1-14.

[7] Computer Vision Annotation Tool .

[8] Roboflow .