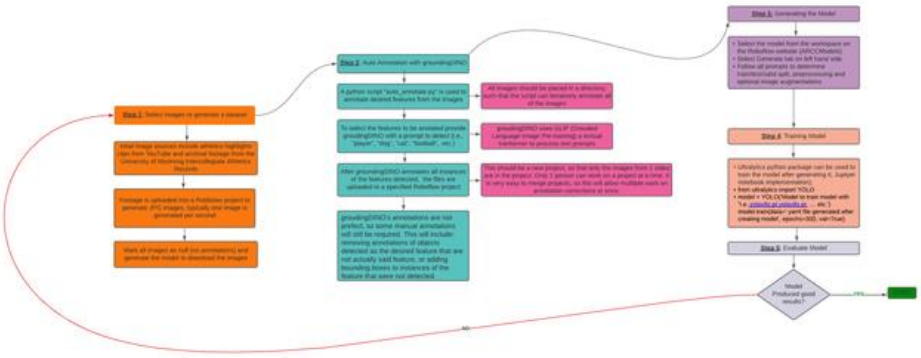
Author: Derek Walton

Process to Track Players

Process Summary:

The aim of this project is to identify and track athletes from the University of Wyoming. To achieve this, the detection of specific players was divided into three stages. Firstly, the players are detected using computer vision models generated by the groundingDINO (Zero-Shot Open-Set object detector) and YOLOv8 (Real time image detection and object segmentation model) techniques. These models can automatically annotate images and track players in real-time. Next, the location of the uniform numbers on the detected players is identified using the VGG16 convolutional neural network. Finally, a CNN is used to classify the integer value of the uniform number, which is then replaced with the corresponding player roster name and the positions of the players are tracked.

Step 1: Player Detection



1. Select Images to Train model
    1. Images were sourced from various YouTube clips and archival footage from University of Wyoming Intercollegiate Athletics Records.
    2. Once footage is deemed suitable Roboflow is used to generate images from the video
        1. This is done by selecting the number of frames per second to convert into JPG images
    3. After the images are generated a model is created to download the images
2. Auto-annotation with groundingDINO

    1. Images generated in previous step are placed into a ‘data’ file for ‘auto_annotate.py’ to use
        1. auto_annotate.py is a python script that takes images from ‘data’ and uses groundingDINO to detect objects in the images provided from a text prompt and annotate all instances of the object in the images.

            1. groundingDINO uses GLIP (Grounded Language-Image Pre-training) to process the text prompt
                1. GLIP uses a multi-task learning framework to pre-train a network to perform three tasks: image classification, text classification, and image-text matching. The network is trained to predict the labels of images and texts, and lean to match images to their associated texts.
        2. Once all detected instances of the desired object are detected, the auto_annotate.py uploads the image to a selected Roboflow project
    2. The annotate images uploaded to the project need to be processed further to ensure that the annotation was performed correctly
        1. This will include removing an unnecessary annotations that were added or adding bounding boxes to targets that were not detected.
3. Generating the Player model

    1. Once a suitable number of images are within a project a model can be generated on Roboflow
        1. Select the project from the workspace being used, (ARCCModels)
        2. Select the ‘Generate’ tab on the left hand side
        3. Follow all the prompts to determine the training/test/validation split, any necessary preprocessing, and image augmentations desired.
            1. training/test/validation split
                1. Used to prevent model from over-fitting the training data
                2. Train data set
                    1. used for learning (by model), to fit the parameters to the machine learning model
                3. Valid data set
                    1. used to provide unbiased evaluation of a model fitted on the training dataset while tuning model hyperparameters
                    2. also for other forms of model preparation: feature selection, threshold cut-off selection
                4. Test data set
                    1. used to provide an unbiased evaluation of a final model fitted on the training data set
            2. Preprocessing
                1. Resizing: This is used to resize the images to a standard size, which is often required for the model to work properly. Roboflow provides options to resize the images while maintaining aspect ratios.
                2. Normalization: This is used to scale the pixel values of the image to a common range, typically between 0 and 1 or -1 and 1. This helps in improving the performance of the model and making it more robust to lighting conditions.
                3. Standardization: This is used to transform the pixel values of the image so that they have a mean of zero and standard

deviation of one. This helps in reducing the effects of lighting conditions and making the model more robust.
4. Data balancing: This is used to balance the number of images in each class, which is important when dealing with imbalanced datasets.
5. Data augmentation: This is used to generate new training data by applying image augmentations, as described in the previous answer.

3. Image augmentations
    1. Flipping: This augmentation mirrors the image horizontally or vertically to create new training data.
    2. Rotation: The image is rotated by a specified degree to simulate variations in camera angles.
    3. Zooming: The image is zoomed in or out to provide variations in scale.
    4. Cropping: A part of the image is cropped out to create new training data.
    5. Color Jittering: The colors in the image are adjusted to simulate variations in lighting conditions.
    6. Blurring: The image is blurred to simulate variations in focus and to remove noise.
    7. Noise Addition: Noise is added to the image to simulate variations in sensor noise and other image artifacts.

4. Training model

    1. After generating the model use the Ultralytics python package to train it
        1. from ultralytics import YOLO

           model = YOLO('Model to train model with “i.e., yolov8s.pt, yolov8s.pt, … etc.')
           model.train(data='.yaml file generated after creating model', epochs=300, val=True)

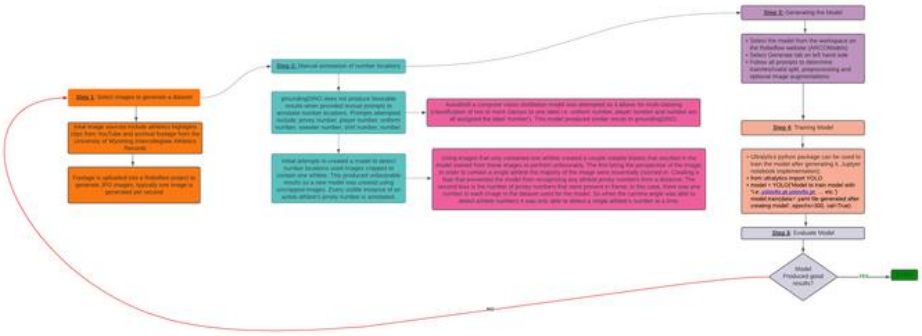
            1. Edit the valid/test/train paths in the .yaml file to the location that your generated model was saved
                1. train: Path to model
                   val: Path to model
                   test: Path to model

                   nc: 1
                   names: ['Name of class(es) annotated in model']

                   roboflow:
                   workspace: Yeceives player box from Step 1our project’s API key
                   project: Your project’s name

version: Version of your model
license: CC BY 4.0
url: Your project's URL

Step 2: Jersey Number Location Detection



1. Select images to train model
    1. As done in step one, images were sourced from athletic highlight clips on YouTube and archival footage from the University of Wyoming Intercollegiate Athletics Records.
    2. Also, like step one images are uploaded to a project on Roboflow, however, in this case auto-annotation of athlete jersey number locations provided unfavorable results.
        1. auto_annotate.py was used in the attempt to automatically annotate jersey number locations, however, correct annotation was seen every 10-15 images. This being the case it was decided that manual annotation of number locations would be more efficient than correcting any incorrect annotation that were produced by groundingDINO. Prompts used to attempt auto-annotation of jersey number locations include: jersey number, player number, shirt number, sweater number, uniform number, and number.
            1. It should also be noted that AutoDistill a computer vision distillation model developed by Roboflow was also used in the attempt to automate the annotation of jersey number locations. This was tested, because unlike groundingDINO, AutoDistill allows for multi-classing.
                1. Multi-classing is when two or more classes i.e., apples, oranges, and bananas are assigned to a single label in the case of this example ‘fruit’
                    1. This is possible with AutoDistill as it used an “Ontology” for textual prompts. An ontology in the case of AutoDistill is a dictionary that associates more general prompts i.e., ball with a more descriptive class i.e., basketball. This allows for the base model used by AutoDistill to detect more of

the desired features while providing an accurate label for the dataset.
    2. AutoDistill describes their base model as “large foundation models that know a lot about a lot”, at the time of testing the only base model available was GroundedSAM. This is a combination of IDEA Research’s groundingDINO (object detection) and Facebook Research’s Segment-Anything-Model (SAM) (object segmentation). However, the developers of AutoDistill plan to introduce OpenAI’s GPT-4 model.
    3. The Ontology also allows for multi-classing i.e., all prompts attempted using auto_annotate.py were assigned to the label ‘number’.
  2. Even with the addition of multi-classing that AutoDistill provides auto-annotation of jersey number locations produced comparable results to groundingDINO used in auto_annotate.py
2. Manual Annotation of Jersey Number Locations

    1. Initial attempts to create the number location detection model used images cropped to contain a single athlete. The images used for this dataset were created by taking images from the player detection dataset.
        1. crop.py uses a player detection model trained in step one to detect players in the images it is provided. Every time a player is detected a cropped image is created using the bounding box coordinates to crop and save the image.
        2. However, many of the images created by this Python script were unusable, this is because the player would be angled in such a way that their jersey number was not visible. This required manual filtering and deletion of all images that did not contain visible player jersey numbers.
        3. After a suitable number of cropped images were created (1500) manual annotation was performed. Once annotated a model was generated on Roboflow following all prompts mentioned in step 1 and it was trained using the same process in step 1.
    2. The model that was created using cropped images performed very poorly due to two notable biases that were created by using cropped images
        1. The first is the perspective of the images in the dataset. Since the cropped images were produced using a player detection model, they were zoomed in. This led to the model being unable to detect any player jersey number from a distance.
        2. The second bias is the number of instances of a player’s jersey number in each frame. In the case of the cropped images that were used to train the model there was a single instance in each image. So even when the camera angle allowed for detection in videos used to test the model. The model could only detect a single number location at a time.

3. The next attempt at generating a number location detection model entailed using the uncropped images from the player detection model.
    1. Every visible instance of a player's jersey number is manually annotated
    2. Using the uncropped images prevents the biases seen in the initial attempts and produced far better results than the initial model. Currently we are considering combing the model with cropped and uncropped images to see if any further improvement can be seen in detecting player jersey number locations.

Contributor: Kyle Lofthus

Step 3: Number Classification

1. Receives location from Step 2
    1. With the location of the number on the jersey from step 2, the CNN processes the target location to detect and then classify the number.
2. Classifying the number

    1. Testing will be conducted to determine whether the CNN will detect a multi-digit number as a group or individually.
        1. This will be determined by how the CNN classifies numbers such as 24. If it is outputting the result as 2, we know that we will train it on 0-9 and have it classify the number 2 and then the number 4 and combine them to output the correct result of 24.
        2. If the CNN is able to classify both numbers together without increasing the time to do so by a significant manner then the CNN will be trained on 0-99 and be able to detect and classify 24, as 24 immediately and output the correct result.
3. Outputting the Result

    1. After the CNN has classified the image that number will be output to YOLOv8.
    2. From here YOLOv8 will track the number on the jersey to correctly identify each athlete throughout a game.

(Alternate Option: image-to-text)

References

https://builtin.com/machine-learning/vgg16

https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

https://towardsdatascience.com/how-to-split-data-into-three-sets-train-validation-and-test-and-why-e50d22d3e54c

https://github.com/IDEA-Research/GroundingDINO

https://github.com/microsoft/GLIP

https://colab.research.google.com/github/roboflow-ai/notebooks/blob/main/notebooks/automated-dataset-annotation-and-evaluation-with-grounding-dino.ipynb#scrollTo=zJNOIAhGQGlk

https://docs.roboflow.com/

https://arxiv.org/pdf/2203.03605.pdf

https://arxiv.org/pdf/2112.03857.pdf

https://arxiv.org/abs/2203.03605

https://github.com/autodistill/autodistill

https://builtin.com/machine-learning/multiclass-classification