**Construction Safety**

**Key insights acquired from actual work.**

***Please note that all the insights listed below are:***

- Referring to the previous works of similar projects.

- Selected from our own research.

- Actual experiments: Trial and error process.

- Leverage Auto-training from Roboflow, on the same dataset but different augmentation and processing.

**ITERATIVE WORKFLOW**

- Identifies **misclassifications**, **false positives**, and false negatives, and makes necessary improvements.

- Allows for fine-tuning of the model's performance by **updating hyper-parameters used**.

- Involves in adjusting settings, **revisiting data augmentation methods**, or incorporating additional labeled data.

**DATA PREPARATION**

**01.** **Data collection insights:**

- **Diverse and representative data**: Train on both different classes and diverse images with all classes.

- **Balanced class distribution**: Imbalanced datasets can lead to biased training and affect the model's ability to detect objects from underrepresented classes.

- Check the **dimension insight** in data health check for resizing accordingly - median size.

- **Hard negative mining**: We include hard negative samples (background or non-object regions), marking them as null, in the dataset can help the model learn to discriminate between objects and backgrounds more effectively.

**02.** **Data annotation insights:**

- **Multi-scale and complex annotation:** polygon annotations instead of rectangular shapes: Annotating objects at multiple scales can improve the model's ability to detect objects of different sizes. Including annotations for small, medium, and large objects ensures that the model learns to detect objects at various scales and maintains good performance across the entire size spectrum.

- **Annotator training and guidelines**: only a few members are mapping annotations for consistency of data, strictly following guidelines.

-  **Smart annotation:** Leverage smart annotation from Roboflow.

**03.** **Data preprocessing insights:**

- No black and white.

- No horizontal flip.

- No random rotation.

- No perspective transformation.

- Edge detection is unnecessary.

**MODEL TRAINING**

When it comes to model training, various aspects we should take into account such as hyper-parameters, pre-trained models, optimizers, and so on. In this part, we only highlight insights from model tuning:

**YOLO v5, v7, v8:**

- Choose a small **learning rate**: 0.001. (Default value for YOLO models is 0.01).

- More **epochs.**

- Set **early stop** for training: **‘patience’** = 50, so that we can set more epochs without worrying about much training because if the model is not improved after 50 epochs, the algorithm will stop.

- Set **‘cache’** for faster training.

- Medium **batch size**. Small batch size does not improve performance because it offers low memory while training but also comes up with unstable training due to different behaviors of each data batch (so small that the model could not generalize well).

- **weight\_decay**: Weight decay is a regularization technique used in deep learning to prevent overfitting by adding a penalty term to the loss function. Default = 0.0005

- **freeze**: There are 12 layers for this architecture. The default number of freezing in YOLO is 1, the first layer. We can consider freezing 11 layers and unfreezing the last layer of “Linear activation function” for training.

- Consider using **Non-max suppression** to eliminate redundant or overlapping bounding box predictions generated by the model. Set **nms** = True.

**DE⫶TR (Detection Transformer) with Pytorch Lightning:**

- Set **early stop** for training: using callback.

- **weight\_decay**: to prevent overfitting.

- Set asmaller learning rate for **backbone layers** for more effective training and flexibility.

- I**ncrease the speed of training** by using 32-bit floating point precision (precision='32-true').

- accumulate\_grad\_batches=16: This parameter controls gradient accumulation. Instead of performing an optimizer step after every batch, gradients are accumulated over a certain number of batches (16 in this case) before performing the optimizer step. This can be useful when the **batch size is limited by memory constraints** but still allows for stable training updates.

- **Gradient clipping** helps mitigate the exploding gradient problem and ensures stable training.

**Mask R-CNN model (Using Keras, Pytorch and Detectron 2):**

- Using Mask R-CNN instead of Faster R-CNN due to complex annotations (non-rectangular shapes)

- Conducted model using various frameworks Keras, Detectron 2 and Pytorch but it seems that there are many errors to recall the dependencies.

**Efficient Det with Tensorflow framework:**

- Has trouble with data loaders since this model requires each image to have its own annotation file.

**MODEL COMPARISON**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Model** | **mAP**  **(%)** | **Speed /**  **Training time (Hour)** | **Model size**  **(MB)** | **Framework Compatibility** |
| 1 | YOLOv5 | 77.3 | 0.591 | 14.5 | Darknet, OpenCV, Pytorch, Tensorflow, Keras, … |
| 2 | YOLOv7 | 79.7 | 6.224 | 74.8 | Darknet, OpenCV, Pytorch, Tensorflow, Keras, … |
| 3 | YOLOv8 | 79.1 | 0.779 | 22.5 | Darknet, OpenCV, Pytorch, Tensorflow, Keras, … |
| 4 | DETR | 48.2 | 6.61 | 166.6 | Pytorch |
| 5 | AutoML - Roboflow | 80.3 | 0.681 | NA | Variety |