**1. Introduction**

This report outlines my experiences working on a custom object detection project using YOLOv8n, focusing on bee frame cell classification. My objectives were to achieve reliable classification of seven cell types: CappedHoneyCell, CappedWorkerBroodCell, EmptyCombCell, PollenCell, UncappedNectarCell, UncappedWorkerLarvaCell, and BeeHiveFrame. In this report, I describe the project setup, the adjustments made to the learning rate and other hyperparameters, training results, evaluation metrics, and the challenges encountered.

**2. Installation and Setup**

**2.1 Installation Process**

I installed YOLOv8 using pip install ultralytics. My setup involved a Windows 11 system with Python 3.11 and Torch 2.4.1 configured for CPU training, as no GPU was available. The installation process was straightforward, but the training speed was somewhat limited on the CPU.

**2.2 Dataset Preparation**

The dataset consisted of 52 annotated bee frame images provided in .png format, along with corresponding .txt files containing labels and bounding boxes. I split the dataset into 80% for training (42 images) and 20% for validation (10 images). A data.yaml file was created to define the dataset paths and class names.

**3. Training Process**

**3.1 Base Training Configuration**

The initial model used was yolov8n.pt, a pre-trained YOLOv8 nano model, chosen for its smaller size, which is advantageous on CPU. I set the training to run for 100 epochs, using an image size (imgsz) of 640x640 pixels.

**3.2 Hyperparameter Tuning: Learning Rate**

To fine-tune the model’s performance, I experimented with the lr0 (initial learning rate) parameter. This controls how fast the model learns at the beginning of training. The default setting was 0.01, which I adjusted to explore different convergence rates and accuracy levels:

* **First Trial**: Set lr0 = 0.01 (default). The training curve showed oscillations, suggesting a potentially unstable learning process.
* **Second Trial**: Set lr0 = 0.001. This lower learning rate produced smoother convergence but required more epochs to achieve significant improvement.
* **Third Trial**: Set lr0 = 0.005. This midpoint setting struck a balance between stability and learning speed, improving F1 scores and mean average precision (mAP) significantly over the baseline.

**3.3 Additional Parameters Adjusted**

* **Epochs**: The number of epochs was set at 100 for each trial. Although lower learning rates required additional epochs for better accuracy, 100 epochs provided a sufficient range for observing trends.
* **Batch Size**: Set to 16 due to memory constraints on the CPU.
* **Other Hyperparameters**: Momentum and weight decay were left at defaults but could be considered for further tuning in GPU environments.

**4. Evaluation and Results**

**4.1 Key Evaluation Metrics**

* **F1 Score**: A critical metric balancing precision and recall, used to evaluate classification accuracy. F1 scores were calculated after each trial to gauge improvements in model performance.
* **Mean Average Precision (mAP)**: Assesses model accuracy across different confidence thresholds, useful for comparing results from each learning rate trial.

**4.2 Results Summary**

The best results were achieved with lr0 = 0.005, which provided balanced learning progress and stability. The F1 score improved by 8 percentage points, and mAP increased to 0.76 at the 0.5 threshold.

**4.3 F1 and Precision-Recall Curves**

Analyzing the F1 curve for each trial provided insights into model learning patterns. The optimal learning rate trial displayed a steady increase in F1 scores, without drastic fluctuations. Precision-recall curves also showed that the best lr0 achieved higher recall values, indicating fewer missed detections of object categories.

**4.4 Challenges**

The CPU training environment limited processing power and required extended training time. Hyperparameter tuning was also slower on CPU, and several trials required extended periods to complete. The limited dataset size added variability in results, as minor changes impacted the small validation set more significantly.

**5. Conclusion and Future Directions**

The YOLOv8n model provided a practical base for training a custom bee frame cell classifier. My primary findings indicated that reducing the initial learning rate from the default 0.01 to 0.005 yielded improvements in F1 and mAP scores. Given more computational resources, further experimentation with batch size, momentum, and additional epochs could enhance model performance.

Future directions could include:

1. **Utilizing a GPU** to speed up training and allow for more extensive hyperparameter exploration.
2. **Expanding the dataset** to improve model robustness and generalization.
3. **Testing different YOLO architectures** (e.g., yolov8s.pt or yolov8m.pt) to compare performance on CPU.

Overall, this project provided valuable insights into custom object detection using YOLOv8n and the impact of hyperparameters like learning rate on model performance.