Pipeline:

Data Preparation & Sanitization

- Initially reviewed the raw dataset and COCO-formatted JSON files containing bounding box coordinates and architectural labels.
- Developed a script to parse coordinates and crop individual objects for the ResNet classification stage
- Implemented a sanitization script to remove characters incompatible with Windows environments (e.g., colons, square brackets, spaces) from filenames and JSON paths
- Built a transformation engine to convert standard pixel bounding boxes (x, y, w, h) into YOLO-normalized center-point coordinates (cx, cy, w, h)

Preprocessing & Handling Imbalance

- Instead of standard resizing, which skews architectural shapes, I utilized **letterbox padding** to standardize image sizes while preserving original proportions.
- Used a Stratified Split (70/15/15) to ensure all five cabinet classes, including rare ones, were proportionally represented across Train, Val, and Test sets
- **For ResNet**: Applied inverse-frequency weights to the Cross-Entropy Loss function
- **For YOLO**: Implemented **Image Oversampling**, duplicating images with rare cabinet types in the training set to force the model to prioritize minority features

Architecture

- Evaluated **ResNet-18** as a baseline and **ResNet-50** for potentially better accuracy with complex features. **YOLOv8n** was selected for detection due to its small size and efficient architecture for training and deployment
- Selected **Adam** for its efficiency with sparse gradients and adaptive learning rate capabilities

What didn't work?

- Initial tests with random splits caused "blind spots" where rare classes were missing from training or evaluation.
- Simple cropping and resizing distorted the furniture geometry, making identification unreliable

Challenges

- ResNet classifiers occasionally struggled with Base vs. Wall cabinet distinction because tight crops removed the floor-level height context
- Main issues - saving weights in Kaggle, leading to potential overfitting on the final ResNet-18 inference weights.

Future work

- Using OCR to read text labels (e.g., "BC-24") alongside images to reach near-perfect accuracy

- Implementing a sliding window to process small blueprint sections at full resolution, keeping tiny grid lines sharp
- Experimenting with larger YOLO models and more data to mitigate overfitting

Results

Yolo

```
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 2/2 1.
4s/it 2.8s5.1s
                   all         25        257      0.732      0.684      0.678      0.549
              lc_bcabo         15        154       0.72       0.87       0.73      0.562
              lc_wcabo          9         48       0.81      0.833      0.828      0.713
         lc_muscabinso          7         45      0.749      0.933      0.933      0.867
            lc_wcabcub          2         10      0.649        0.1      0.219     0.0541
Speed: 7.8ms preprocess, 12.4ms inference, 0.0ms loss, 20.9ms postprocess per image
Results saved to /kaggle/working/runs/detect/val
--- FINAL TEST METRICS ---
mAP50: 0.6775
mAP50-95: 0.5492
Precision: 0.7323
Recall: 0.6842
```

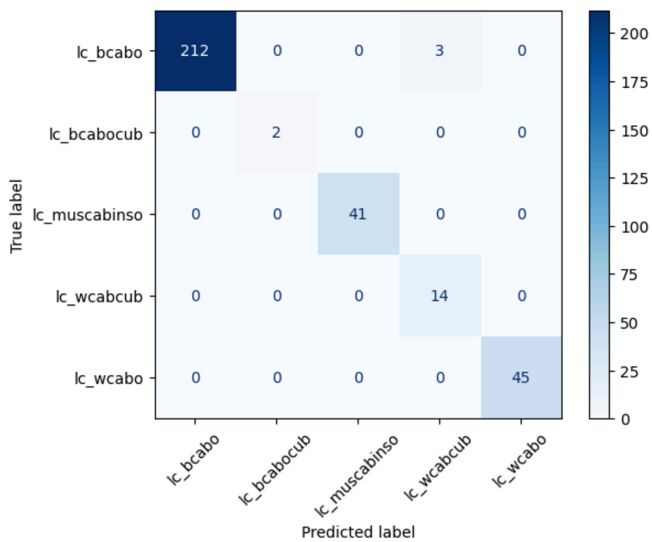
We can see that here the model, because of their fine-grained internal grid details and thin line weights, the detector often failed to distinguish them from standard wall cabinets at the current resolution. In general i think this is good results.

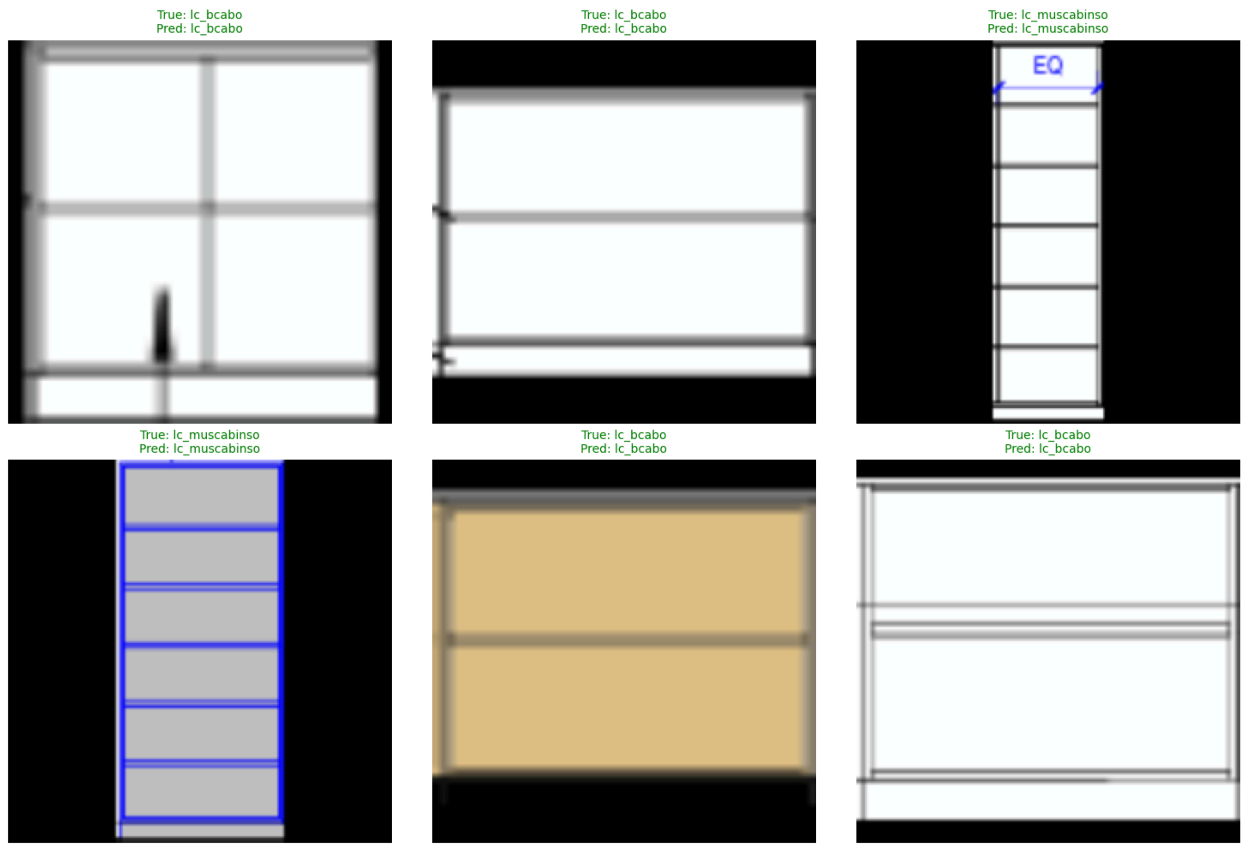
ResNet50

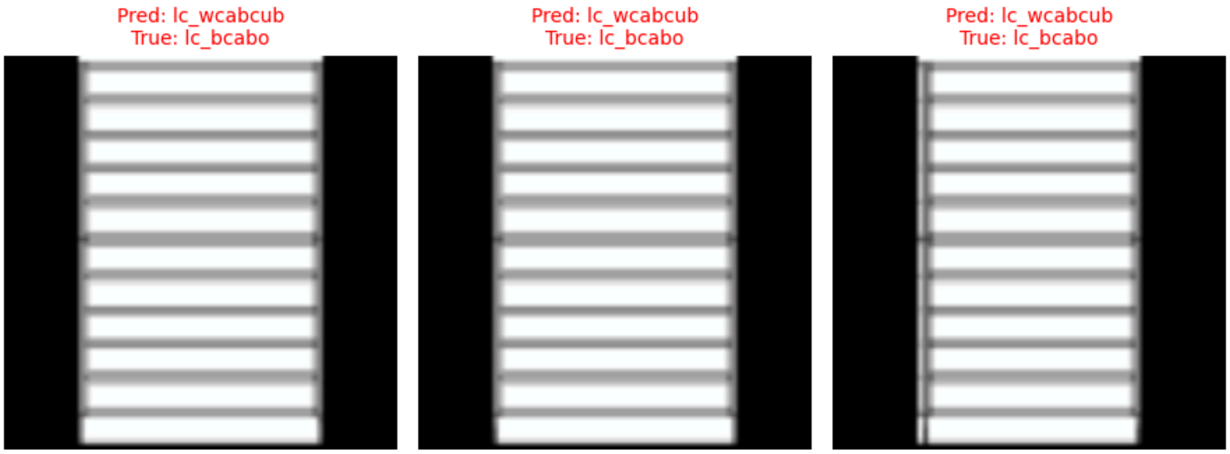
```
Evaluating on Unseen Test Data...
               precision    recall  f1-score   support

     lc_bcabo       1.00      0.99      0.99       215
   lc_bcabocub       1.00      1.00      1.00         2
lc_muscabinso       1.00      1.00      1.00        41
   lc_wcabcub       0.82      1.00      0.90        14
     lc_wcabo       1.00      1.00      1.00        45

     accuracy                           0.99       317
    macro avg       0.96      1.00      0.98       317
 weighted avg       0.99      0.99      0.99       317
```


True label
lc_bcabo
lc_bcabocub
lc_muscabinso
lc_wcabcub
lc_wcabo
212
0
0
3
0
0
2
0
0
0
0
0
41
0
0
0
0
0
14
0
0
0
0
0
45
lc_bcabo
lc_bcabocub
lc_muscabinso
lc_wcabcub
lc_wcabo
Predicted label
200
175
150
125
100
75
50
25
0


True: lc_bcabo
Pred: lc_bcabo
True: lc_bcabo
Pred: lc_bcabo
True: lc_muscabinso
Pred: lc_muscabinso
EQ
True: lc_muscabinso
Pred: lc_muscabinso
True: lc_bcabo
Pred: lc_bcabo
True: lc_bcabo
Pred: lc_bcabo



Here we can see that the results are very good - only 3 mistakes. Also here the imbalance problem solved perfectly. However here could be problem of overfitting