



HW 1_Object Detection

Justin Thiadi 程煒財 | NTHU_112006234

1. Model Description

a. Brief Introduction

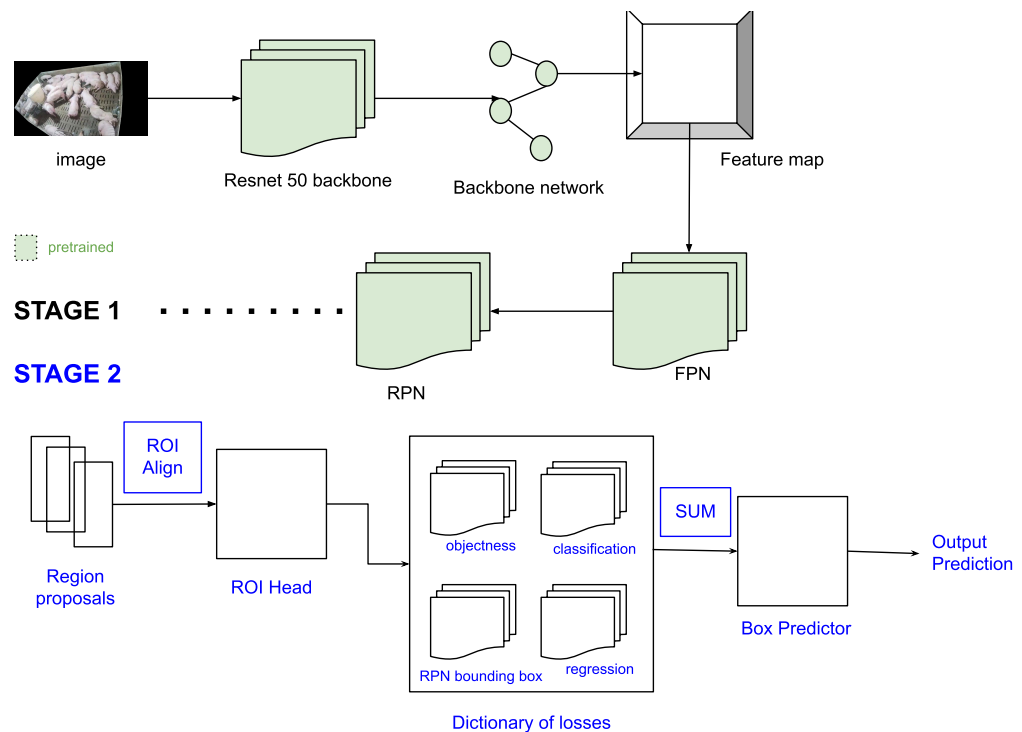
A Faster R-CNN object detection model is used with a ResNet-50 backbone and a Feature Pyramid Network (FPN).

The model detects pigs in images by combining a pretrained feature extractor with custom-trained prediction layers.

COCO-pretrained weights are used for the feature extractor parts, including the backbone, FPN, Region Proposal Network (RPN), and ROI feature extractor to take advantage of general visual features learned from the dataset.

The classification and bounding box regression layers are newly initialized and trained from scratch on the pig dataset.

b. Architecture Illustration



Faster R-CNN is a two-stage detector.

- Stage 1: Region Proposal Network (RPN) → identify where object is
 - Input image → ResNet-50 FPN
 - The backbone (ResNet + FPN) extracts feature maps from the input image
 - The FPN (Feature Pyramid Network) combines multi-scale features to detect small and large pigs.

- Feature maps → Anchors
 - The RPN slides small convolutional filters over the feature maps.
 - At each location, it predicts:
 - Objectness score (is there an object here?)
 - Box regression offsets (how to adjust anchor box coordinates).
- Generate proposals
 - The RPN refines anchor boxes into region proposals (top 1000–2000 per image).
 - Non-Maximum Suppression (NMS) removes overlapping boxes
- Outputs from Stage 1:
 - Region proposals (candidate bounding boxes)
 - Objectness scores
- Stage 2: ROI Head (Fast R-CNN Head) → determine what object is
 - ROI Align
 - Each proposal (from Stage 1) is mapped back onto the feature map.
 - ROIAlign extracts a fixed-size feature patch per proposal.
 - Fully Connected Layers (ROI Head)
 - These small features predict:
 - Classification scores (object class, “pig” vs background)
 - Bounding box refinements (fine-tuned box coordinates)
 - Box Predictor
 - Combines outputs of classification and regression heads.
 - Losses are computed:
 - objectness (binary classification)
 - bbox_regression (L1 or Smooth L1 loss)
 - Output Prediction
 - After post-processing (softmax + NMS), final boxes and labels are produced.
 - Outputs from Stage 2:
 - Final bounding boxes (boxes)
 - Confidence scores (scores)
 - Class labels (labels)

Inference Part → Predict bounding boxes on test images

- Load test images
 - Read each .jpg file and convert it to a tensor
 - Apply torchvision.transforms.ToTensor()
- Discard low-confidence detections (score < 0.2).
- Convert to [score, x, y, w, h, 0] format

c. Any Modifications

- i. The initial plan was to train the Faster R-CNN model for 30 epochs, but due to time constraints (the best model was obtained approximately one hour before the submission deadline), the training was divided into two separate stages in the notebook:

#1: 9 epochs of training followed by inference

#2: 10 additional epochs of training and inference

Both stages independently handle training and inference, meaning each can:

1. Save the best-performing model (best_model.pth) based on validation mAP@50
2. Generate predictions for the test set

2. Implementation Details

1. Data Preprocessing

- Each input frame was read from the dataset and converted to RGB using the Python PIL library
- Bounding box annotations were parsed from the provided gt.txt file and converted into the [xmin, ymin, xmax, ymax] format
- Degenerate bounding boxes (zero or negative width/height) were filtered out to prevent invalid samples
- Images were loaded as torch.Tensor objects with pixel values normalized to the range [0, 1] using torchvision.transforms.ToTensor()
- No resizing or cropping was applied to preserve the original aspect ratio of the dataset.

2. Data Augmentation

To improve generalization and reduce overfitting, several custom augmentations were applied during training via a composed transformation pipeline:

- ColorJitter: Randomly adjusted brightness, contrast, saturation, and hue (brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1).
- RandomHorizontalFlip: Applied with probability p=0.5 to mirror both image and bounding boxes horizontally.
- RandomGaussianBlur: Occasionally applied a Gaussian blur (radius=2, p=0.2) to simulate motion blur or camera noise
- ToTensor: Converted images to normalized PyTorch tensors

For validation and inference, only ToTensor() was applied to ensure consistent evaluation without random transformations.

3. Hyperparameters

Batch size	2	Weight decay	0.0005
Optimizer	SGD	LR Scheduler	CosineAnnealingWarmRestarts
Learning rate	0.01 (base)	Warmup	First 2 epochs
Momentum	0.9	Epochs	9 + 10

4. Loss Functions

- The Faster R-CNN model internally optimizes multiple losses jointly:
 - RPN Classification Loss: Binary cross-entropy for object vs. background proposals
 - RPN Regression Loss: Smooth L1 loss between predicted and ground-truth box offsets
 - ROI Classification Loss: Cross-entropy for final object class prediction.
 - ROI Regression Loss: Smooth L1 loss for refined bounding box coordinates.

The total loss is computed as the sum of all component losses

5. Training Strategy

- Training was conducted on GPU (cuda).
- Each iteration computed the total loss, performed backpropagation, and updated the model parameters using Stochastic Gradient Descent (SGD).
- The learning rate was controlled using a Cosine Annealing Warm Restarts scheduler, which gradually decreased the learning rate and periodically reset it to encourage better generalization and recovery from local minima.
- A linear warmup during the first two epochs helped stabilize early optimization and prevent gradient explosions.
- Hyperparameter Tuning
 - Several hyperparameters were tuned to achieve stable convergence and higher mAP scores:
 - Learning Rate (LR): Tested in the range [0.001, 0.05].
 - Weight Decay: Compared between 0.0001, 0.0005, and 0.001.
 - The final choice of 0.0005 minimized overfitting without underfitting.
 - Batch Size: Experimented with 1, 2, and 4.
 - A batch size of 2 was selected to fit GPU memory limits while maintaining stable gradients.
 - The model was evaluated every three epochs using COCO mAP@50, and the best-performing checkpoint was automatically saved as best_model.pth.
 - This incremental training approach enabled controlled hyperparameter refinement and efficient reuse of previously learned weights without restarting the full pipeline.

3. Result Analysis

a. Quantitative Improvements

The Faster R-CNN model was trained for a total of 19 epochs (9 + 10 in two stages). During training, the average loss steadily decreased from 0.793 (epoch 1) to 0.206 (epoch 19), showing consistent convergence and stable optimization.

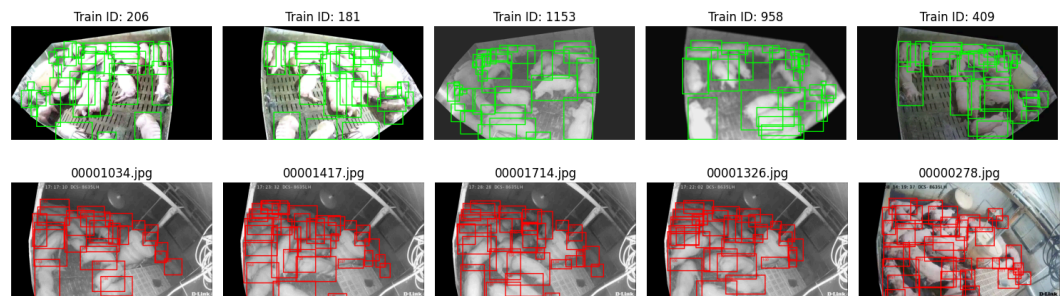
Performance was evaluated using COCO-style metrics.

	Epoch 3	Epoch 6	Epoch 9	Final (Epoch 19)
mAP@[0.50:0.95]	0.703	0.787	0.823	0.846
mAP@50	0.968	0.979	0.980	0.980
mAP@75	0.811	0.891	0.904	0.915
AR@[0.50:0.95]	0.740	0.815	0.845	0.865

These results demonstrate strong and consistent improvement across training, with high precision and recall maintained across different object size

The model achieved its best mAP@50 of 0.980 and best mAP@[0.50:0.95] of 0.846, indicating a relatively high accuracy

b. Visualizations



- Predictions closely matched ground-truth bounding boxes, even under varying poses and lighting (including colors)
- The model effectively localized pigs of different scales, with tight and accurate boxes
- False positives are rare and typically occurred in regions with shadows or overlapping animals
- Confidence scores for correct detections consistently exceeded 0.85, validating prediction reliability

4. Short Conclusion

A Faster R-CNN with ResNet-50 FPN backbone was implemented for pig detection using a custom dataset. The model was trained incrementally over 19 epochs with tuned hyperparameters and applied augmentations to enhance generalization.

Training achieved stable convergence, reducing the average loss from 0.793 to 0.206, and the final model reached a validation mAP@50 of 0.980 and mAP@[0.50:0.95] of 0.846, indicating strong localization and classification performance. Visual inspections further confirmed accurate bounding boxes and consistent confidence scores across both training and test images.

Overall, the results demonstrate that careful preprocessing, hyperparameter tuning, and staged fine-tuning significantly improved detection accuracy. We can try to explore larger batch sizes, mixed-precision training, and additional data augmentation to further enhance performance and efficiency.

5. References

- <https://discuss.pytorch.org/t/compute-validation-loss-for-faster-rcnn/62333/9>
- <https://medium.com/@RobuRishabh/understanding-and-implementing-faster-r-cnn-248f7b25ff96>
- <https://stackoverflow.com/questions/64238709/low-training-and-validation-loss-but-a-low-map-on-faster-r-cnn-on-mapillary-d>
- <https://medium.com/@freshtechyy/a-detailed-introduction-to-resnet-and-its-implementation-in-pytorch-744b13c8074a>