Object Detection & RNNs

Zhuoyi Huang

Partial slides credit to JunYoung Gwak
1. Object Detection
Motivation

- **Image classification**: often assume there is a single object in the image.
Motivation

- **Image classification**: often assume there is a single object in the image
- Real-world images can include multiple instances of objects with the same/different classes
Motivation

- **Image classification**: often assume there is a single object in the image
- Real-world images can include multiple instances of objects with the same/different classes
- **Object Detection**: produce bounding boxes that surround each instance
Problem Definition: Object Detection

Object Detection
- Input: Image
- Output: multiple instances of
  - object location (bounding box)
  - object class

Instance
- Distinguishes individual objects, in contrast to considering them as a single semantic class
Problem Definition: Object Detection

Object Detection

- Input: Image
- Output: multiple instances of
  - object location (bounding box)
  - object class

Bounding box

- Rigid box that confines the instance
- Multiple possible parametrizations
  - (width, height, center x, center y)
  - (x1, y1, x2, y2)
  - (x1, y1, x2, y2, rotation)
Problem Definition: Object Detection

Object Detection

- Input: Image
- Output: multiple instances of
  - object location (bounding box)
  - object class

Object class

- Semantic class of the instance
  - Similar to image classification
  - Predict a vector of scores
Modern Object Detection Architecture

- R-CNN
- Fast R-CNN
- Faster R-CNN
- Mask R-CNN
- SSD
- YOLO (v1, v2, v3, v4)
- FPN
- DETR
Object Detection: how can we detect multiple instances?

- Boxes can be centered at any location in the image
- Varying width/height
- Sliding window: infeasible
Object Detection: Anchor Boxes!

- Neural network prefers discrete prediction over continuous regression
- Preselect templates of bounding boxes to alleviate the regression problem
- For each anchor box, NN decides
  - Does it contain an object? (objectness classification)
  - Small refinement to the box (object localization)
Object Detection: Single-Stage and Two-Stage Architectures

Stage 1

- For every output pixels
  - For every anchor boxes
    - Predict bounding box offsets
    - Predict anchor confidence (objectness/class)
  - Output
    - Bounding boxes if single-stage
    - Region proposals (region-of-interest, RoI) if two-stage

Stage 2

- For RoI
  - Perform pooling using the RoI (RoI pooling)
  - Predict bounding box offsets
  - Predict object class

Object Detection: Single-Stage and Two-Stage Architectures

Stage 1

- For every output pixels
  - For every anchor boxes
    - Predict bounding box offsets
    - Predict anchor confidence (objectness/class)
  - Output
    - Bounding boxes if single-stage
    - Region proposals (region-of-interest, RoI) if two-stage

Stage 2

- For RoI
  - Perform pooling using the RoI (RoI pooling)
  - Predict bounding box offsets

Object Detection: Single-Stage vs Two-Stage Architectures

- **Single-Stage**
  - + Faster
  - - Can perform worse on small objects due to the low resolution of feature maps

- **Two-Stage**
  - + Performance is often higher
  - + Easily extendable to various instance-based tasks
  - - Slow
Details for Two-Stage Object Detectors

Stage 1

● **For every output pixels**
  ○ For every anchor boxes
    ■ Predict bounding box offsets
    ■ Predict anchor confidence (objectness/class)

● Output
  ○ Region proposals (region-of-interest, RoI)

Stage 2

● **For RoI**
  ○ Perform pooling using the RoI (RoI pooling)
  ○ Predict bounding box offsets
  ○ Predict object class

Feature extractor

- Every pixel makes prediction
- Image classification: single output
Details for Two-Stage Object Detectors

Stage 1

- For every output pixels
  - For every anchor boxes
    - Predict bounding box offsets
    - Predict anchor confidence (objectness/class)
  - Output
    - Region proposals (region-of-interest, RoI)

Stage 2

- For RoI
  - Perform pooling using the RoI (RoI pooling)
  - Predict bounding box offsets
  - Predict object class

Extract Anchor Boxes

- For each output pixel
  - "Objectness" classification
  - Regression
- Often thousands of anchors for an image
- Pass anchors that correspond to ground-truth locations to the next stage, plus negative anchors

Bounding Box Regression

Given
- Anchor box size \((p_w, p_h)\)
- Output pixel center location \((p_x, p_y)\)

Predict bounding box refinement toward \(b\)
- Log-scaled scale relative ratio
  \[ d_w = \log\left(\frac{b_w}{p_w}\right), \quad d_h = \log\left(\frac{b_h}{p_h}\right) \]
- Relative center offset
  \[ d_x = \frac{(b_x - p_x)}{p_w}, \quad d_y = \frac{(b_y - p_y)}{p_h} \]
Details for Two-Stage Object Detectors

Stage 1

- For every output pixels
  - For every anchor boxes
    - Predict bounding box offsets
    - Predict anchor confidence (objectness/class)
  - Output
    - Region proposals (region-of-interest, RoI)

Stage 2

- For each RoI
  - Perform pooling using the RoI (RoI pooling)
  - Predict bounding box offsets
  - Predict object class

RoI Pooling

- Given region-of-interests (RoIs), we want to pool from the backbone features.

Details for Two-Stage Object Detectors

Stage 1

- For every output pixels
  - For every anchor boxes
    - Predict bounding box offsets
    - Predict anchor confidence (objectness/class)
  - Output
    - Region proposals (region-of-interest, RoI)

Stage 2

- For each RoI
  - Perform pooling using the RoI (RoI pooling)
  - Predict bounding box offsets
  - Predict object class (semantic class / background)

Details for Two-Stage Object Detectors
Are We Done?

- Prediction might contain multiple boxes of the same instance
Post-Processing: Non-Maximum Suppression

- For boxes overlapping with each other above a threshold: keep the one with the maximum confidence score

- Implementation
  - Sort by confidence
  - For each box (conf high to low)
    - If overlaps with confirmed predictions above a threshold
      - Discard
    - Else
      - Add to predictions
Feature Pyramid Network as the feature extractor

- **Traditional backbone**
  - Small feature maps have larger receptive field and contain better-extracted overall semantic information
  - Want this semantic information in larger feature maps for prediction

- **Feature Pyramid Network**
  - Richer representation
  - Enables multi-scale predictions
How should we evaluate our results?

- Start with the most simple case
- Given
  - a single ground-truth box
  - a single predicted box
How should we evaluate our results?

- Start with the most simple case
- Given
  - a single ground-truth box
  - a single predicted box
- Use Intersection-over-Union (IoU)

\[ \text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \]
What if there are multiple boxes?

- Multiple ground-truth boxes
- Multiple predictions
- Might include
  - True positive (prediction matched with GT)
  - False positive (prediction not matched with any GT)
  - False negative (GT not matched with any prediction)
Bounding Box Matching

- Use IoU threshold
- Matched if
  - IoU above certain threshold
  - Class prediction is correct
  - GT not matched with other boxes (1-to-1)
Evaluation Metrics: Precision and Recall

- True Positive (TP)
- False Negative (FN)
- False Positive (FP)

Precision = $\frac{TP}{TP + FP}$
Recall = $\frac{TP}{TP + FN}$
Evaluation Metrics: Average Precision

- Go through every prediction in descending order of the prediction confidence
- Plot Precision-Recall Curve
- Area below the curve is Average Precision (AP)
Evaluation Metrics: Average Precision

- To make AP more stable to score ordering, we sometimes take max precision to the right of the PR curve
- Use different IoU threshold for matching
  - AP50, AP75, AP95: match IoU threshold of 0.5, 0.75, 0.95
  - AP: average of AP with match IoU threshold of [0.5, 0.55, 0.6, ..., 0.95]
Two-Stage Detectors can do more!

- In addition to detecting boxes, at the final stage using RoI features, we can predict:
  - 3D bounding boxes
  - Instance segmentation
  - Keypoints (human pose)
  - Meshes
  - Scene graphs
  - ...

- A family of R-CNNs!

3D Object Detection

- **Input**
  - 2D image and/or 3D point cloud

- **Output**
  - 3D bounding box
    - Center location: x, y, z
    - Bounding box size: w, h, l
    - Rotation
3D Object Detection

Stage 1
- For every output pixel (from backbone)
  - For every anchor boxes
    - Predict bounding box offsets
    - Predict anchor confidence

(Optional, if two-stage networks) Stage 2
- For every region proposals
  - Predict bounding box offsets
  - Predict its semantic class

For example, Point R-CNN
Mask R-CNN

- Final stage parallel to box prediction
  - Predict instance mask using a convolution head
- RoI Align especially helpful for segmentation by aggregating fine-grained features
Mesh R-CNN

- Final stage parallel to box prediction
  - Predict voxels
  - Align and refine meshes with graph convolution
Graph R-CNN

- Object detection + relationship detection
- Additional Relation Proposal Network
- Use Graph Convolution Network (GCN) for scene graph refinement

DETR: End-to-End Object Detection with Transformers

- Using Transformer to directly produce boxes
- Predict objects (much larger than number of boxes) using learned fixed number of object queries

Conclusion

Stage 1

● For every output pixels
  ○ For every anchor boxes
    ■ Predict bounding box offsets
    ■ Predict anchor confidence (objectness/class)
  ● Output
    ○ Region proposals (region-of-interest, RoI)

Stage 2

● For each RoI
  ○ Perform pooling using the RoI (RoI pooling)
  ○ Predict bounding box offsets
  ○ Predict object class (semantic class / background)
  ○ Predict other stuff! (segmentation, pose, mesh, etc.)
● Non-maximum Suppression

Implementing a Detector: Detectron2

- Open-source software for object detection and more
- Developed by Facebook with PyTorch
- Easily extendable with extensive documentations
2. RNNs
Recurrent Neural Networks

Traditional Neural Networks
- Can’t use its reasoning about previous events to inform later ones.

Recurrent Neural Networks
- Networks with loops allow information to persist.
- Chain-like, multiple copies of the same network
LSTM Networks

Recurrent Neural Networks

- Difficult to handle long-term dependencies.

Long Short Term Memory Networks (LSTMs)

- A special kind of RNN.
- The repeating module has a different structure.
Core Idea Behind LSTM Networks

Key to LSTMs

- **Cell state:**
  - Only some minor linear interactions, information flow unchanged.

- **“Gates”**
  - Remove or add information to the cell state:
  - Composed of:
    - Sigmoid neural net layer: output 0 - 1
    - Pointwise multiplication operation
Step-by-Step LSTM walk through

1. Throw away information from the cell state
   - Forget gate layer
     - For each number in the Cell State $C_{t-1}$: Input $h_{t-1}, x_t$, output a number between 0 and 1.

2. Store new information in the cell state
   - Input gate layer
     - Sigmoid layer, output between 0 and 1, decides which values we’ll update.
     - A tanh layer
       - Creates a vector of new candidate values $\tilde{C}_t$

\[
f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)
\]

\[
i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)
\]

\[
\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)
\]

\[
C_t = f_t \ast C_{t-1} + i_t \ast \tilde{C}_t
\]
Step-by-Step LSTM walk through

3. Output – A filtered cell state version

- **Sigmoid gate layer**
  - Decides what parts of the cell state we’re going to output.
- **Cell State tanh layer**
  - Input: Cell state
  - Output: Push the values to be between −1 and 1
- **Multiply the above two, and get the final output** $h_t$.

\[
o_t = \sigma (W_o [h_{t-1}, x_t] + b_o) \\
h_t = o_t \times \text{tanh} (C_t)
\]
Variants on LSTM

1. Adding “peephole connections”
   - Let the gate layers look at the cell state

2. Use coupled forget and input gates
   - Instead of separately deciding what to forget and what we should add new information to, we make those decisions together.

\[
\begin{align*}
f_t &= \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \\
o_t &= \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)
\end{align*}
\]

\[
C_t = f_t \cdot C_{t-1} + (1 - f_t) \cdot \tilde{C}_t
\]
Variants on LSTM

1. Gated Recurrent Unit (GRU)
   - Combines the forget and input gates into a single “update gate”.
   - Merges the cell state and hidden state.
   - Simpler than standard LSTM models, growing increasingly popular.

\[
\begin{align*}
  z_t &= \sigma (W_z \cdot [h_{t-1}, x_t]) \\
  r_t &= \sigma (W_r \cdot [h_{t-1}, x_t]) \\
  \tilde{h}_t &= \tanh (W \cdot [r_t \ast h_{t-1}, x_t]) \\
  h_t &= (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t
\end{align*}
\]
RNNs take advantage of sequences

● Sequences are not only text or music, they can also be videos (sets of images).

● E.g. Understand actions in videos: Using RNNs to focus on tracking the convolutional features.

● Using RNNs and CNNs together is possible, and in fact, it could be the most advanced use of Computer Vision we have.

● Action classification, movie generation …
Suggested Readings

- Rich feature hierarchies for accurate object detection and semantic segmentation
- Fast R-CNN
- Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
- Mask R-CNN
- Fast Point R-CNN
- Mesh R-CNN
- Graph R-CNN for Scene Graph Generation
- You Only Look Once: Unified, Real-Time Object Detection
- SSD: Single Shot MultiBox Detector
- End-to-End Object Detection with Transformers
- Detectron2
- Recurrent Models of Visual Attention