DLP - Week-10 - PA

1) Given an image dataset for object detection, how would you preprocess the data and create region proposals for Fast R-CNN? Consider the following code snippet: **1 point**

```
import cv2
import numpy as np
from skimage.feature import hog

image = cv2.imread('path_to_image.jpg')
resized_image = cv2.resize(image, (600, 600))
hog_features = hog(resized_image, orientations=9, pixels_per_cell=(8, 8),
   cells_per_block=(2, 2), visualize=False, multichannel=True)

def generate_proposals(image, window_size=(128, 128), step_size=32):
   proposals = []
   for y in range(0, image.shape[0] - window_size[1], step_size):
      for x in range(0, image.shape[1] - window_size[0], step_size):
         proposals.append((x, y, x + window_size[0], y + window_size[1]))
   return proposals

region_proposals = generate_proposals(resized_image)
```

→ resize

→ hog features → this is not present in faster RCNN

→ creating ROIs here

Which of the following steps is NOT a part of Fast R-CNN preprocessing?

- (a) Resize the input image to a fixed size.
- ✓ (b) Extract features using HOG.
- (c) Generate region proposals.
- (d) Classify each region proposal directly.

2) Which of the following statements correctly differentiates Fast R-CNN from YOLO? **1 point**

- (a) Fast R-CNN is a single-stage detector, while YOLO is a two-stage detector. → 2-stage → 1 stage ✗
- ✓ (b) Fast R-CNN relies on external region proposals, whereas YOLO divides the image into a grid for detection. → given by algo to find RoI → YOLO → SxS → Bounding Box (grid like structure)
- (c) YOLO uses Selective Search for region proposals, while Fast R-CNN performs end-to-end detection in one step.
- (d) Both Fast R-CNN and YOLO perform detection and classification in a single stage. ✗ ✗

3) Consider the following code snippet that outlines the steps of extracting regions of interest (RoI) for object detection: **1 point**

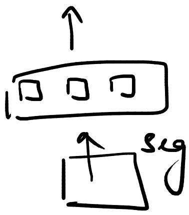
```
import cv2
import numpy as np
from sklearn.svm import SVC
from skimage.feature import hog

image = cv2.imread('dog_cat.jpg')
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
   ss.setBaseImage(image)
   ss.switchToSelectiveSearchFast()
region_proposals = ss.process()

## Extract features using convnet
# Step 3: Train a classifier (SVM) on the extracted features
classifier = SVC(kernel='linear')
classifier.fit(features, labels) # labels are pre-defined for training
```

Slow RCNN

SVM Linear



segmentation

Which architecture follows this approach of using region proposals, feature extraction, and a classifier for object detection?

- (a) Fast R-CNN
- ✓ (b) R-CNN (Slow R-CNN)
- (c) YOLO
- (d) Faster RCNN

4) The following code snippet demonstrates a simplified implementation of a Region Proposal Network (RPN) used in Faster R-CNN: **1 point**

```
import torch
import torch.nn as nn
import torch.nn.functional as F

# Input feature map from a backbone network (e.g., ResNet)
feature_map = torch.randn(1, 512, 50, 50) # (Batch, Channels, Height, Width)

# RPN layers
class RegionProposalNetwork(nn.Module):
   def __init__(self, in_channels, num_anchors):
      super(RegionProposalNetwork, self).__init__()
      self.conv = nn.Conv2d(in_channels, 512, kernel_size=3, stride=1, padding=1)
      self.cls_layer = nn.Conv2d(512, num_anchors * 2, kernel_size=1) # Classification
      self.reg_layer = nn.Conv2d(512, num_anchors * 4, kernel_size=1) # Regression
```

```
def forward(self, x):
    x = F.relu(self.conv(x))
    cls_logits = self.cls_layer(x) # Objectness scores
    reg_deltas = self.reg_layer(x) # Bounding box adjustments
    return cls_logits, reg_deltas

# Initialize the RPN with 9 anchors per location
rpn = RegionProposalNetwork(in_channels=512, num_anchors=9)

# Generate RPN outputs
cls_logits, reg_deltas = rpn(feature_map)
print("Classification logits shape:", cls_logits.shape) # (1, 18, 50, 50)
print("Regression deltas shape:", reg_deltas.shape) # (1, 36, 50, 50)
```

classify obj./obj not n → regression → coordinates $4n$ } faster RCNN

Which of the following architectures incorporates this type of Region Proposal Network (RPN) for object detection?

- (a) YOLO
- (b) Faster R-CNN
- (c) HOG detector
- (d) Fast R-CNN

5) Consider the following code snippet, which generates predictions and ground truth for an object detection task: ***1 point***

```
# Predicted bounding boxes and confidence scores
predictions = [
    {'bbox': [50, 50, 100, 100], 'score': 0.9},
    {'bbox': [30, 30, 70, 70], 'score': 0.75},
    {'bbox': [200, 200, 250, 250], 'score': 0.6}
]

# Ground truth bounding boxes
ground_truths = [
    {'bbox': [50, 50, 100, 100]},
    {'bbox': [30, 30, 70, 70]}
]

# Function to calculate IoU (Intersection over Union)
def calculate_iou(box1, box2):
    x1 = max(box1[0], box2[0])
    y1 = max(box1[1], box2[1])
    x2 = min(box1[2], box2[2])
    y2 = min(box1[3], box2[3])
    intersection = max(0, x2 - x1) * max(0, y2 - y1)
    area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
    area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])
    union = area1 + area2 - intersection
    return intersection / union if union > 0 else 0
threshold = 0.5
true_positives = 0
false_positives = 0
false_negatives = len(ground_truths)
for pred in predictions:
    matched = False
    for gt in ground_truths:
        iou = calculate_iou(pred['bbox'], gt['bbox'])
        if iou >= threshold:
            true_positives += 1
            false_negatives -= 1
            matched = True
            break
    if not matched:
        false_positives += 1

# Calculate precision and recall using the precision_recall function.
precision, recall = find_precission(true_positives, false_positives, true_negative, false_negative)
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

$TP = 2$

$FP = 1$

$FN = 0$

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{2}{2+1} = \frac{2}{3} = 0.67$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{2}{2+0} = \frac{\cancel{2}}{\cancel{2}} = 1$$

Given the predictions and ground truths, what are the precision and recall values?

- (a) Precision: 0.67, Recall: 1.00
- (b) Precision: 1.00, Recall: 1.00
- (c) Precision: 0.67, Recall: 0.67
- (d) Precision: 1.00, Recall: 0.67

6) The following code snippet demonstrates the calculation of Mean Average Precision (mAP) for an object detection task: *1 point*

```
import numpy as np

# Predicted boxes with confidence scores and ground truth boxes
predictions = [
  {'bbox': [50, 50, 100, 100], 'score': 0.9, 'class': 'cat'},
  {'bbox': [30, 30, 70, 70], 'score': 0.8, 'class': 'dog'},
  {'bbox': [200, 200, 250, 250], 'score': 0.7, 'class': 'cat'}
]

# Ground truth boxes
ground_truths = [
  {'bbox': [50, 50, 100, 100], 'class': 'cat'},
  {'bbox': [30, 30, 70, 70], 'class': 'dog'}
]

# Function to calculate IoU (Intersection over Union)
def calculate_iou(box1, box2):
  x1 = max(box1[0], box2[0])
  y1 = max(box1[1], box2[1])
  x2 = min(box1[2], box2[2])
  y2 = min(box1[3], box2[3])
  intersection = max(0, x2 - x1) * max(0, y2 - y1)
  area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
  area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])
  union = area1 + area2 - intersection
  return intersection / union if union > 0 else 0
  # Evaluate precision and recall
  tp, fp = 0, 0
  matched_gt = set()
  precision_recall_curve = []
  for pred in predictions:
    matched = False
    for i, gt in enumerate(ground_truths):
      if gt['class'] == pred['class'] and i not in matched_gt:
        iou = calculate_iou(pred['bbox'], gt['bbox'])
        if iou >= 0.5: # IoU threshold
          tp += 1
          matched_gt.add(i)
          matched = True
          break
    if not matched:
      fp += 1

  precision, recall = precision_recall(tp, fp, tn, fn)
  # Calculate mAP as the average precision
  mAP = np.mean([p for p, r in precision_recall_curve])
  print(f"Mean Average Precision (mAP): {mAP:.2f}")
  precision_recall_curve.append((precision, recall))
```

tp = 2
fp = 1
fn = 0
tn = 0

precision = 0.67
recall = 1

Given the predictions and ground truths, what is the Mean Average Precision (mAP)?

- (a) 0.33
- (b) 0.50
- ✓ (c) 0.67

7) Consider the following code snippet *1 point*

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision.ops import roi_pool

Step 1
feature_map = # Input feature map (e.g., from a CNN backbone like ResNet)
# Region proposals as (x1, y1, x2, y2)
region_proposals = torch.tensor([
  [0, 0, 7, 7],
  [2, 2, 10, 10],
  [4, 4, 14, 14]
], dtype=torch.float32)

Step 2
# ROI Pooling
pooled_features = roi_pool(
  feature_map,
  [region_proposals],
  output_size=(7, 7) # Fixed size for each region
)

Step 3
# Fully connected layer for classification and regression
class FastRCNNHead(nn.Module):
  def __init__(self, in_features, num_classes):
    super(FastRCNNHead, self).__init__()
    self.fc1 = nn.Linear(in_features, 1024)
    self.fc2 = nn.Linear(1024, num_classes + 4) # +4 for bbox regression
  def forward(self, x):
    x = F.relu(self.fc1(x))
    return self.fc2(x)
```

fast RCNN ⇒ bbox + linear SVM

Which architecture incorporates the above components for object detection?

- (a) R-CNN (Slow R-CNN)
- ✓ (b) Fast R-CNN
- (c) Faster R-CNN
- (d) YOLO

8) Consider the following code snippet, which calculates the F-score for an object detection task: *1 point*

```
# Predicted labels and ground truth labels
predictions = [1, 0, 1, 1, 0, 1, 0]
ground_truth = [1, 0, 1, 0, 0, 1, 1]

# Confusion matrix components
true_positive = sum([1 for p, g in zip(predictions, ground_truth) if p == 1 and g == 1])
false_positive = sum([1 for p, g in zip(predictions, ground_truth) if p == 1 and g == 0])
false_negative = sum([1 for p, g in zip(predictions, ground_truth) if p == 0 and g == 1])

# F-score calculation find_f1_Score()
f_score = find_f1_Score(true_positive, false_positive, false_negative)
print(f"F-Score: {f_score:.2f}")
```

3
1
1

$$\text{precision} = \frac{3}{4} = 0.75$$

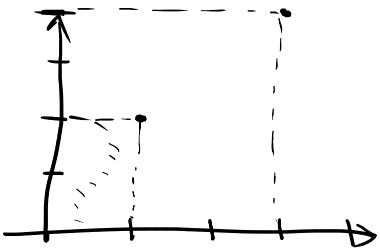
$$\text{recall} = \frac{3}{4} = 0.75$$

$$\frac{2 \times p \times r}{p + r} = \frac{\cancel{2 \times 0.75} \times 0.75}{\cancel{1.5}\ \cancel{0.75}}$$

Given the predictions and ground truth, what is the calculated F-score?

- (a) 0.67
- (b) 0.71
- (c) 0.75
- (d) 0.80

9) Consider the following code snippet, which calculates the Intersection over Union (IoU) between two bounding boxes: *1 point*



150 × 200 = 30000
50 × 100 = 5000

```
# Bounding box format: [x1, y1, x2, y2]
box1 = [50, 50, 150, 150]
box2 = [100, 100, 200, 200]

# Calculate the intersection coordinates
x1 = max(box1[0], box2[0])
y1 = max(box1[1], box2[1])
x2 = min(box1[2], box2[2])
y2 = min(box1[3], box2[3])

# Intersection area
intersection_area = max(0, x2 - x1) * max(0, y2 - y1)

# Areas of the two boxes
area_box1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
area_box2 = (box2[2] - box2[0]) * (box2[3] - box2[1])

# Union area
union_area = area_box1 + area_box2 - intersection_area

# IoU calculation
iou = intersection_area / union_area if union_area > 0 else 0
print(f"IoU: {iou:.2f}")
```

Given the bounding boxes [50,50,150,150] and [100,100,200,200], what is the calculated IoU?

- (a) 0.14

```
# Total loss
total_loss = coord_loss + object_loss + no_object_loss + iou_loss + class_loss
print(f"Total Loss: {total_loss:.4f}")
```

→ coordinate
→ object → classification

Which of the following loss components are included in the YOLO loss function illustrated in the code snippet?

- (a) Coordinate loss, Object loss, No-object loss, Classification loss.
- (b) Coordinate loss, Object loss, IoU loss, Classification loss.
- (c) Coordinate loss, IoU loss, Object loss, No-object loss, Classification loss.
- (d) IoU loss, Object loss, Classification loss, Regularization loss.

11) Which of the following networks can be used as feature extractors in Faster R-CNN? *1 point*

- (a) VGG16, ResNet50, MobileNet
- (b) YOLO, SSD, RetinaNet
- (c) AlexNet, LeNet, GAN
- (d) RPN, FPN, DenseNet

No, the answer is incorrect.
Score: 0

Accepted Answers:

(a) VGG16, ResNet50, MobileNet

12) Which of the following statements correctly differentiates Fast R-CNN from Slow R-CNN? *1 point*

- (a) Fast R-CNN uses external region proposals, while Slow R-CNN generates region proposals directly during training.
- (b) Slow R-CNN uses ROI Pooling for fixed-size feature extraction, while Fast R-CNN processes entire images in a single pass.
- (c) Fast R-CNN performs feature extraction on the entire image once, while Slow R-CNN extracts features sepa rately for each region proposal.
- (d) Slow R-CNN integrates region proposal generation into the network, while Fast R-CNN relies on Selective Search for proposals.

13) YOLO uses a single neural network to predict bounding boxes and class probabilities directly from the entire image in one pass. (True/False)

true

14) InFaster R-CNN,theRegion Proposal Network (RPN) doesn't share convolutional layers with the backbone network to generate region proposals, reducing computation time compared to external proposal methods. (True/False)

false