Melika Mohammadi Fakhar - 99522086

1.

1.1

CNN with Sliding Window:

High Redundancy: Multiple CNN evaluations for each window, increasing computational load.
Inefficient: High computational burden due to separate processing for each window position, especially in large images.

CNN without Sliding Window (using methods like region proposals or fully convolutional networks):

Reduced Redundancy: Fewer evaluations needed as the network either focuses on likely object areas or processes the entire image in a single pass.
More Efficient: Lower computational demand and faster processing, benefiting from optimization techniques.

Overall, using a CNN without the sliding window method is generally more computationally efficient and cost-effective.

1.2

Multiple Classes: YOLO handles multiple classes by predicting class-specific probabilities in each grid cell, scaled by the confidence that an object is present and correctly bounded.
Bounding Boxes: Each cell predicts adjustments to multiple anchor boxes to fit the shapes and sizes of detected objects, making the model robust to variability in object dimensions.
Anchor Boxes: These are crucial for enabling the detection of multiple objects of different shapes and sizes within the same grid cell, improving both precision and recall of the model.

MORE DETAILED EXPLANATION FOR THIS QUESTION:

Grid Division and Prediction

Grid Division: YOLO divides the input image into an $S \times S$ grid. Each grid cell is responsible for detecting objects whose center falls within the cell.
Predictions per Grid Cell: Each cell predicts multiple bounding boxes and associated confidence scores. The confidence score reflects the accuracy of the bounding box and the probability that the bounding box contains a specific object.
Class Probabilities: Each grid cell also predicts the conditional class probabilities $P(\text{Class}_i \mid \text{Object})$ for C classes. These probabilities are conditioned on the grid cell containing an object.
Overall Prediction: The final prediction for each bounding box is the product of the conditional class probability and the individual box confidence:

$$\Pr(\text{Class}_i \mid \text{Object}) \times \Pr(\text{Object}) \times \text{IOU}_{\text{pred}}^{\text{truth}}$$

This gives the probability that the box contains an object of class i, the object is present in the box, and how well the predicted box matches the actual box.

Anchor Boxes
YOLO uses predefined anchor boxes, which are preset bounding box shapes and sizes. These anchor boxes help the model to detect objects with different shapes and sizes more effectively.
How Anchor Boxes Work:
Each grid cell uses these anchor boxes as references for predicting actual bounding boxes around objects.
The network learns to adjust the dimensions of these anchor boxes to fit the specific objects in the image during training.
Multiple anchor boxes per grid cell allow the model to detect multiple objects in each cell, each of a different shape or size.
Bounding Box Predictions: For each anchor box, the model outputs adjustments to the anchor box dimensions and position (i.e., center coordinates, width, and height). This allows the anchor boxes to better enclose the detected objects.
Advantages of Anchor Boxes
Handling Variability: They allow the model to specialize in detecting objects of certain sizes and aspect ratios, improving detection accuracy across a range of object sizes.
Efficiency: By using anchor boxes, YOLO can efficiently predict multiple bounding boxes in each grid cell, each adjusted for different object characteristics, without significantly increasing computational complexity.

2.
2.1

- **YOLO**

Architecture:
YOLO frames object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities.
It divides the image into a grid and predicts bounding boxes and probabilities for each grid cell.
The model applies a single neural network to the full image, making predictions at a global scale.

Speed:
Very fast, as it processes the entire image during inference in a single pass, hence "You Only Look Once."
Typically faster than SSD because it reduces the number of bounding box predictions per image.

Accuracy:
Generally offers strong performance in terms of accuracy, but can struggle with small objects compared to SSD, as it predicts fewer bounding boxes per grid cell.
Can miss objects that appear in groups or are close together because each grid cell only predicts a limited number of boxes.

Optimal Use Scenarios:
Best used in scenarios where real-time detection is crucial and there is a moderate variation in object sizes.
Ideal for surveillance, where objects are generally well-spaced and not clustered.

• **SSD**

Architecture:
SSD predicts bounding boxes using multiple feature maps at different scales, which makes it more effective at detecting objects of various sizes.
Like YOLO, SSD processes the whole image in one pass but uses multiple differently sized feature maps to handle objects of varying sizes.

Speed:
Fast, but typically a bit slower than YOLO due to the multiple feature maps and the larger number of predictions it makes at various scales.
Speed can be optimized with different configurations of feature maps and box aspect ratios.

Accuracy:
Tends to perform better at detecting small objects, thanks to its use of multiple feature maps and aspect ratios.
Generally, it achieves better overall accuracy across different object sizes and is less prone to missing closely spaced objects.

Optimal Use Scenarios:
Excellent for applications that require the detection of variously sized objects, such as autonomous driving or when objects may be at different distances from the camera.
Useful in crowded scenes where objects of different scales are present.

• **Scenarios Where One Might Outperform the Other:**

YOLO: More effective in scenarios where the objects are generally large or medium-sized and not closely packed. Examples include monitoring large animals in wildlife research or vehicles in relatively clear traffic conditions.
SSD: Better suited for complex scenarios involving small or multiple overlapping objects, such as detecting pedestrians in a busy street scene or small parts in industrial quality control, where objects come in various sizes and are densely packed.

2.2

• **How Focal Loss Works:**

In traditional cross-entropy loss used in classification tasks, each instance contributes equally to the loss, regardless of how easy or difficult it is to classify. This can cause an issue in scenarios like object detection, where easy negatives (background) vastly outnumber the positive examples (objects). The model can become biased towards predicting the majority class, often leading to poor detection performance.
Focal Loss addresses this by modifying the standard cross-entropy loss such that it down-weights the loss assigned to well-classified examples. This allows the model to focus more on hard, misclassified examples. The formula for Focal Loss is:

$$\text{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t)$$

p_t is the model's estimated probability for each class being the true class.

α_t is a balancing factor for the class, which adjusts the importance given to each class.

γ is a focusing parameter that smoothly adjusts the rate at which easy examples are down weighted. When $\gamma = 0$ Focal Loss is equivalent to Cross Entropy Loss, and as γ increases, the effect of the loss scaling by hard examples becomes more pronounced.

• **Addressing Class Imbalance**

Focal Loss effectively tackles class imbalance in object recognition by focusing training on difficult negative examples and by giving less weight to easy negatives:
Focus on Difficult Examples: By scaling down the loss for well-classified examples (those for which the model has high confidence), Focal Loss forces the model to focus training on examples that are hard to classify, which are often the minority positive examples. This is crucial in object detection, where rare but important objects might be easily overlooked by the overwhelming number of easy negatives.
Adjustable Focus: The γ parameter allows fine-tuning of how much the model focuses on the hardest examples versus continuing to improve classification performance across more moderate examples. Higher values of γ increase the model's focus on hard examples, making it very adaptive to the specific needs of the training data's imbalance.

3.
• **Necessity of NMS**
Reduces Redundancy: It eliminates multiple bounding boxes for the same object, ensuring each detected object is represented by the best possible bounding box.
Improves Accuracy: By filtering out less probable bounding boxes, NMS clarifies the detection output and directly improves precision and recall, crucial metrics for evaluating detection models.
• **Steps in Applying NMS**
1. Sort the Boxes: Organize detected bounding boxes by descending confidence scores.
2. Select and Suppress:
Choose the highest scoring box and retain it.
Compute the Intersection over Union (IoU) with other boxes.
Suppress boxes with an IoU above a certain threshold, indicating significant overlap.
3. Iterate: Repeat the process for the next highest scoring box until all boxes are processed.
• **Impact of IoU Threshold on NMS**
Higher IoU Threshold: Allows more overlap, potentially keeping more true positives but may result in multiple detections per object.
Lower IoU Threshold: Results in more aggressive suppression, reducing duplicates but might miss distinct objects in dense scenes.

4.
• Sort Boxes by Confidence:
1. (0.9, (50, 50, 100, 100))
2. (0.8, (55, 60, 105, 110))
3. (0.7, (100, 100, 150, 150))
4. (0.6, (45, 50, 95, 100))

• Select the Box with Highest Confidence:
Selected: (0.9, (50, 50, 100, 100))

• Calculate IoU with Remaining Boxes:
Need to calculate IoU between (50, 50, 100, 100) and each of the other boxes.
Suppress Boxes with IoU Greater Than 0.5:
The box (100, 100, 150, 150) is clearly non-overlapping so we can retain it.
For the boxes that potentially overlap, we calculate IoU using the formula:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Let's calculate the IoU for these boxes:
IoU for (55, 60, 105, 110):
Intersection coordinates: (55, 60, 100, 100)
$\max(0,\min(100,105) - \max(50,55)) \times \max(0,\min(100,110) - \max(50,60)) = 45 \times 40 = 1800$
Area of Box1 + Area of Box2 − Area of Intersection $= 2500 + 2500 - 1800 = 3200$

$$\frac{1800}{3200} \approx 0.56$$

IoU for (45, 50, 95, 100):
Intersection coordinates: (50, 50, 95, 100)
$\max(0,\min(100,95) - \max(50,45)) \times \max(0,\min(100,100) - \max(50,50)) = 45 \times 50 = 2250$
Area of Box1 + Area of Box2 − Area of Intersection $= 2500 + 2500 - 2250 = 2750$

$$\frac{2250}{2750} \approx 0.82$$

Given that both IoUs are greater than the threshold of 0.5, we suppress both (55, 60, 105, 110) and (45, 50, 95, 100).
Remaining Bounding Boxes:
(0.9, (50, 50, 100, 100))
(0.7, (100, 100, 150, 150))
These are the bounding boxes that remain after applying NMS with an IoU threshold of 0.5.

5.
YOLO Bounding Box Coordinates:
b_x, b_y: These represent the center of the bounding box as a fraction of the total width and height.
b_w, b_h: These represent the width and height of the bounding box as a fraction of the total width and height.

Convert to Pixel Coordinates for a 416 x 416 Image:
• Center Position:
Center X (cx): b_x times the width of the image. For $b_x = 0.5$ it is $0.5 \times 416 = 208$ pixels.
Center Y (cy): b_y times the height of the image. For $b_y = 0.6$ it is $0.6 \times 416 = 249.6$ rounding to about 250 pixels.

• Width and Height:
Width (w): b_w times the width of the image. For $b_w = 0.3$, it is $0.3 \times 416 = 124.8$, rounding to about 125 pixels.
Height (h): b_h times the height of the image. For $b_h = 0.4$, it is $0.4 \times 416 = 166.4$, rounding to about 166 pixels.

Top-left Corner (Derived from the Center):

- Top-left X (x1): Subtract half the width from the center X: 208 - 62.5 = 145.5, rounding to about 146 pixels.
- Top-left Y (y1): Subtract half the height from the center Y: 250 - 83 = 167 pixels.

The bounding box coordinates for the YOLO model with an image size of 416x416 pixels would therefore be:
Top-left corner at (146, 167).
Width of 125 pixels.
Height of 166 pixels.