Deep learning

DR.Fatemi Zadeh

electrical engineering department

Ahmadreza Majlesara 400101861

computer assignment repository

assignment 3

December 15, 2024

Deep learning

assignment 3

Ahmadreza Majlesara 400101861
computer assignment repository

Q1

part A

Consider a standard convolutional layer with a kernel of size $K \times K$, stride $= 1$, and Same Padding applied to a feature map of dimensions $H \times W \times M$ consisting of M channels. Let the number of filters in this layer be N. Derive the output size of the layer and compute the computational cost (number of multiply-accumulate operations). Express your results in terms of H, W, M, N, K.

solution

The output spatial dimensions remain the same as the input when same padding is used, so the output dimensions are:

$$\text{Output Height (H')} = H$$

$$\text{Output Width (W')} = W$$

Since the number of filters in the layer is N, the number of output channels is:

$$\text{Output Channels} = N$$

Thus, the final output size is:

$$H \times W \times N$$

The computational cost of a convolutional layer is given by the number of multiply-accumulate operations. The number of operations required to compute a single output element is equal to the number of weights in the kernel, which is $K \times K \times M$, so we can say that:

$$\text{Total Operations} = H \times W \times N \times K \times K \times M$$

part B

Now, consider a convolutional network where the input is a color image of size $128 \times 128 \times 3$. Assume the network consists of three consecutive convolutional layers with kernels of size 5×5, padding $= 2$, stride $= 2$, and ReLU activation functions. The layers have 64, 128, and 256 filters, respectively. Answer the following:

- Compute the output size and the computational cost of each convolutional layer.

- Examine the receptive field of the last convolutional layer. How does the number of neurons in the final layer depend on the input image resolution?

solution

The output size for a convolutional layer is given by:

$$H' = \left\lfloor \frac{H + 2P - K}{S} \right\rfloor + 1,$$

$$W' = \left\lfloor \frac{W + 2P - K}{S} \right\rfloor + 1,$$

The computational cost (number of multiply-accumulate operations) for a single convolutional layer is:

$$\text{Cost} = H' \cdot W' \cdot N \cdot K^2 \cdot M,$$

based on these formula we can say that:

- For the first convolutional layer:

$$H' = W' = \left\lfloor \frac{128 + 2 \cdot 2 - 5}{2} \right\rfloor + 1 = 64.$$

  Number of output channels: $N = 64$.

$$\text{Cost} = 64 \times 64 \times 64 \times 5^2 \times 3$$

- For the second convolutional layer:

$$H' = W' = \left\lfloor \frac{64 + 2 \cdot 2 - 5}{2} \right\rfloor + 1 = 32.$$

  Number of output channels: $N = 128$.

$$\text{Cost} = 32 \times 32 \times 128 \times 5^2 \times 64$$

- For the third convolutional layer:

$$H' = W' = \left\lfloor \frac{32 + 2 \cdot 2 - 5}{2} \right\rfloor + 1 = 16.$$

  Number of output channels: $N = 256$.

$$\text{Cost} = 16 \times 16 \times 256 \times 5^2 \times 128$$

For layer l, the receptive field size R_l can be computed recursively as:

$$R_l = R_{l-1} + (K - 1) \cdot S_{l-1},$$

where:

- R_{l-1} is the receptive field size of the previous layer.
- K is the kernel size.
- S_{l-1} is the stride of the previous layer.

so we can say that:

- For the first convolutional layer:

$$R_1 = 5.$$

- For the second convolutional layer:

$$R_2 = 5 + (5-1) \times 2 = 13.$$

- For the third convolutional layer:

$$R_3 = 13 + (5-1) \times 2 = 21.$$

As the input resolution increases, each convolutional layer produces larger feature maps. Because the stride reduces the spatial dimensions of the feature maps by a factor of 2 at each layer, the number of neurons in the final layer scales approximately linearly with the input resolution. For example, doubling the input resolution will roughly double the height and width of the feature maps at every layer, leading to a fourfold increase in the number of neurons in the final layer.

part C

This section deals with Depthwise Separable Convolutions used in architectures like MobileNet. Address the following:

- Derive the number of parameters and computational cost for a depthwise separable convolutional layer and compare it with a standard convolutional layer (from Part (a)).
- Revisit the network in Part (b). Replace the second and third convolutional layers with depthwise separable convolutions (similar to MobileNet). Compare the number of parameters and computational cost between the standard convolutional layers and the depthwise separable layers.

solution

For the depthwise convolution:

$$\text{Params (depthwise)} = M \cdot K^2$$

$$\text{Cost (depthwise)} = H \cdot W \cdot M \cdot K^2$$

For the pointwise convolution:

$$\text{Params (pointwise)} = M \cdot N$$

$$\text{Cost (pointwise)} = H \cdot W \cdot M \cdot N$$

For the entire depthwise separable convolution:

$$\text{Params (depthwise separable)} = M \cdot K^2 + M \cdot N$$

$$\text{Cost (depthwise separable)} = H \cdot W \cdot M \cdot K^2 + H \cdot W \cdot M \cdot N$$

The depthwise separable convolution reduces both the number of parameters and computational cost.

- For the first convolutional layer:

$$\text{Params (standard)} = 5^2 \cdot 3 \cdot 64$$

$$\text{Cost (standard)} = 64 \times 64 \times 5^2 \times 3$$

- For the second convolutional layer:

$$\text{Params (standard)} = 5^2 \cdot 64 + 64 \cdot 128$$

$$\text{Cost (standard)} = 32 \times 32 \times 64 \times 5^2 \times 3 + 32 \times 32 \times 128 \times 64$$

- For the third convolutional layer:

$$\text{Params (standard)} = 5^2 \cdot 128 + 128 \cdot 256$$

$$\text{Cost (standard)} = 16 \times 16 \times 128 \times 5^2 \times 64 + 16 \times 16 \times 256 \times 128$$

now by replacing the second and third convolutional layers with depthwise separable convolutions:
for second layer:

- Parameters:

$$\text{Params (depthwise separable)} = 64 \cdot 5^2 + 64 \cdot 128$$

- Computational cost:

$$\text{Cost (depthwise separable)} = 32 \cdot 32 \cdot (64 \cdot 5^2 + 64 \cdot 128)$$

for third layer:

- Parameters:

$$\text{Params (depthwise separable)} = 128 \cdot 5^2 + 128 \cdot 256$$

- Computational cost:

$$\text{Cost (depthwise separable)} = 16 \cdot 16 \cdot (128 \cdot 5^2 + 128 \cdot 256)$$

by comparing results of this part with the previous part we can see that we have about 86% reduction in the number of parameters and 91% reduction in computational cost in second layer and 92% reduction in the number of parameters and 94% reduction in computational cost in the third layer.

part D

Assume a classification task with **200 classes**. To perform this task, we add the following layers:

- **Flatten layer** that converts the output of the final convolutional layer into a 1D vector.
- **Fully Connected (FC) layer** with **200 output neurons**, followed by a **SoftMax activation**.

You are required to:

- **Compute the total number of parameters** introduced by these layers.
- **Compare the number of parameters** in the FC layer with the total number of parameters in the convolutional layers from Part (b) (standard convolution in layers 2 and 3) and Part (c) (depthwise convolution in layers 2 and 3).
- **Discuss how the contribution of the Fully Connected layer's parameters can be reduced**, and analyze the impact of such reductions on the network's performance.

Specifically, you need to compute and compare the FC parameters in the following two cases:

- After using **standard convolutions** in layers 2 and 3.
- After using **depthwise convolutions** in layers 2 and 3.

solution

The output of the third convolutional layer is:

$$16 \times 16 \times 256$$

After the Flatten layer, this becomes a vector of size:

$$16 \cdot 16 \cdot 256 = 65,536$$

The Fully Connected (FC) layer maps this vector to 200 output neurons. The total number of parameters introduced by the FC layer includes:

$$\text{Params (weights)} = 65,536 \cdot 200 = 13,107,200$$

$$\text{Params (biases)} = 200$$

so we can say that:

$$\text{Params (FC)} = 13,107,200 + 200 = 13,107,400$$

From Part (b), the number of parameters in the second and third convolutional layers with standard convolutions is:

$$\text{Params (standard)} = 204,800\,(\text{second layer}) + 819,200\,(\text{third layer}) = 1,024,000$$

From Part (c), the number of parameters in the second and third convolutional layers with depthwise separable convolutions is:

$$\text{Params (depthwise separable)} = 9,792\,(\text{second layer}) + 35,968\,(\text{third layer}) = 45,760$$

so the total number of parameters in the first approach is:

$$1,024,000 + 13,107,400 = 14,131,400$$

and in the second approach is:

$$45,760 + 13,107,400 = 13,153,160$$

the number of parameters in the FC layer is significantly higher than the number of parameters in the convolutional layers. to address this issue, we can use techniques like Global Average Pooling (GAP) and dimensionality reduction before the FC layer. GAP reduces the spatial dimensions of the feature maps to 1x1, which allows us to directly apply the FC layer without flattening the feature maps. Dimensionality reduction techniques like 1x1 convolutions can be used to reduce the number of channels before the FC layer, which can help reduce the number of parameters and computational cost.
these methods have some advantages like:

- Reducing the number of parameters in the FC layer can help reduce overfitting, as it reduces the model's capacity.
- Reducing the number of parameters can also help speed up training and inference, as there are fewer parameters to update and compute.
- Using dimensionality reduction techniques can help capture more abstract features in the feature maps before the FC layer, which can improve the model's performance.

also they have some disadvantages like:

- Reducing the number of parameters in the FC layer may reduce the model's capacity to learn complex patterns, which can lead to underfitting.
- Dimensionality reduction techniques like 1x1 convolutions may introduce additional non-linearity to the model, which can make it harder to train.

Q2

In this exercise, we aim to examine Densely Connected Convolutional Networks. To study this network, you can refer to the link below.

Densely Connected Convolutional Networks

part a

Explain the primary differences between ResNet's residual connections and DenseNet's dense connections. Briefly describe the advantages of each case.

solution

In ResNet, residual connections allow the input to be added to the output of a layer, forming a residual block. This addition helps gradients flow more easily during backpropagation, alleviating the vanishing gradient problem and allowing for deeper networks. The primary advantage of ResNet's residual connections is that they simplify optimization by ensuring that the identity function can be learned, improving training stability.
In contrast, DenseNet uses dense connections, where each layer receives input from all previous layers. This means that the output of every layer is concatenated with the outputs of earlier layers and passed through subsequent layers. DenseNet's advantage lies in improved feature reuse, as each layer can access all previous feature maps, leading to more efficient learning and better gradient flow. DenseNet also reduces the number of parameters compared to traditional networks by avoiding redundant feature learning.

part b

Explain how DenseNet reduces the vanishing gradient problem and what the computational benefits are.

solution

DenseNet reduces the vanishing gradient problem through its dense connections, where each layer receives input from all previous layers.
The computational benefits of DenseNet arise from its architecture of dense connections. Since each layer receives input from all previous layers, DenseNet promotes feature reuse, which allows the network to learn more compact and efficient representations.

This leads to fewer parameters compared to traditional networks, as DenseNet does not require each layer to learn entirely new feature maps but can reuse existing ones. As a result, DenseNet networks tend to be more parameter-efficient while achieving competitive performance. Additionally, the reduced parameter count decreases memory usage and accelerates training, making DenseNet computationally efficient despite its depth.

part c

Propose a practical problem where the use of DenseNet architecture is suitable. Provide a real-world example that supports this.

solution

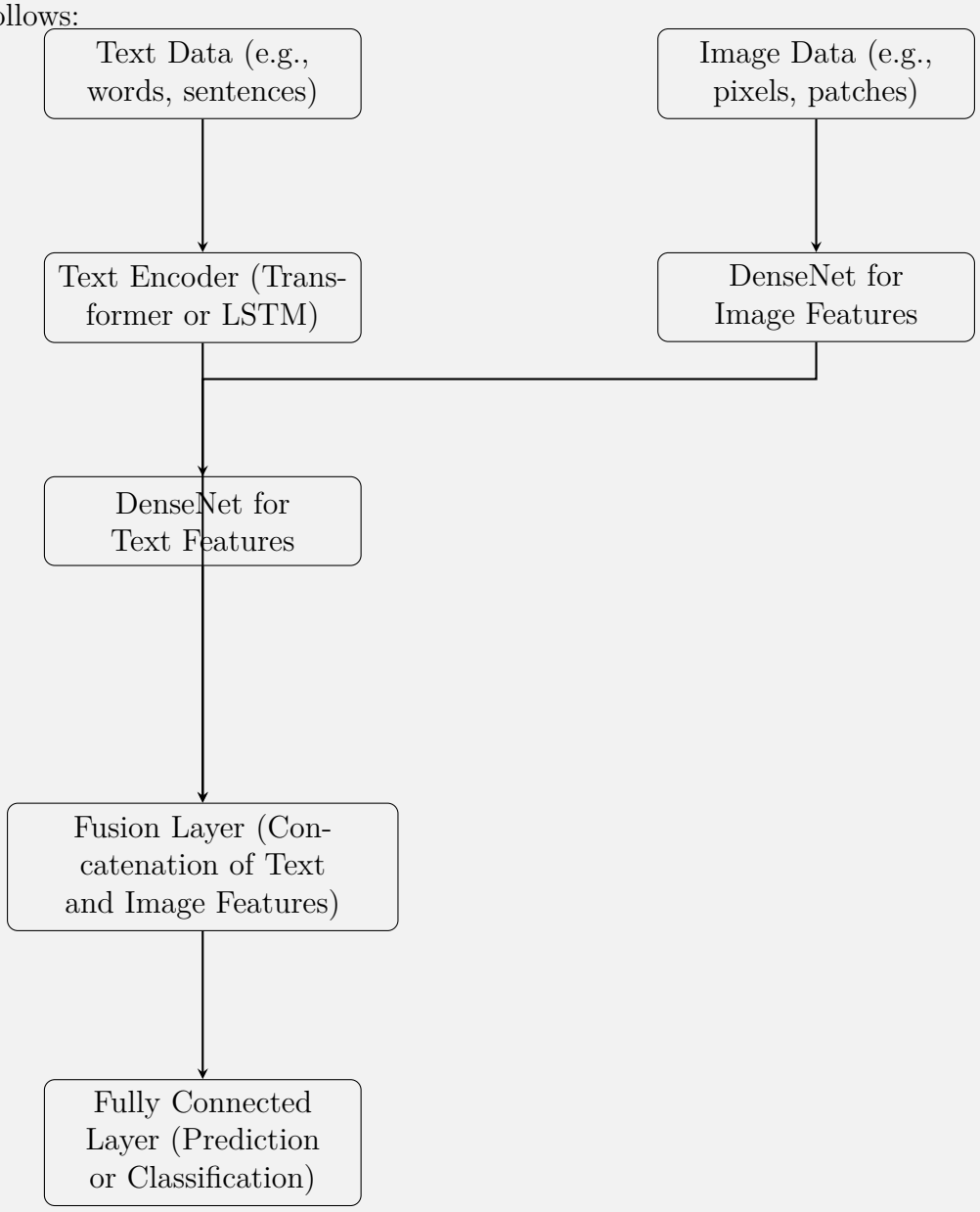
DenseNet is particularly suitable for tasks where feature reuse and efficient gradient flow are essential, such as medical image analysis. In medical imaging, the complexity and fine-grained details of the images require deep networks to capture subtle patterns across various scales. DenseNet's architecture, with its dense connections, allows for the efficient sharing and reuse of features, making it ideal for extracting complex features from high-resolution medical images like CT scans, MRI scans, or X-rays.
A real-world example where DenseNet is highly effective is in the task of detecting tumors in medical images. For instance, in a study for breast cancer detection, DenseNet was used to analyze mammogram images. The dense connections allowed the network to reuse feature maps from earlier layers, helping the model focus on key regions in the images without needing to learn redundant representations. This resulted in higher accuracy and better generalization, especially in challenging cases with small or ambiguous tumors.

part d

If the input data is multi-modal (e.g., text and image), how can DenseNet be adapted for processing such data? Draw and justify your proposed architecture.

solution

In the case of multi-modal data (e.g., text and image), DenseNet can be adapted as follows:



In this architecture, DenseNet is used to process the image data by leveraging its dense connections to promote feature reuse and effective gradient flow. The text data, on the other hand, is processed using a separate encoder like a Transformer or LSTM to capture the sequential or contextual relationships between words.
After extracting features from both modalities, the features are concatenated in the fusion layer. Finally, a fully connected layer is used to make the final predictions or classifications based on the combined feature representation.

Q3

In the lesson, we became familiar with the U-Net structure. In this question, we aim to review the main features and training of this network. At the end, we analyze the functionality of Transposed Convolution. Below, the overall architecture of the network is provided for visualization. To gain a deeper understanding, it is recommended to study the related article.

part a

In this network, the encoder and decoder are connected using Skip Connections. Explain the reason and impact of having these connections with respect to the article's content.

solution

In the U-Net architecture, skip connections link the encoder (contracting path) and decoder (expanding path) layers. The primary reason for these connections is to preserve high-resolution spatial features that are lost during the downsampling process. By transferring these fine-grained details to the decoder, the network ensures that both contextual information and precise localization are retained.
The skip connections have a significant impact on U-Net's performance. They improve segmentation accuracy by helping the network recover fine spatial details, which is crucial for tasks requiring precise boundaries, such as biomedical image segmentation. Additionally, skip connections facilitate better gradient flow during backpropagation, enabling effective training of deep networks. The symmetric U-shaped architecture created by these connections allows the network to combine deep contextual features with high-resolution information seamlessly, leading to sharper and more accurate segmentation results.

part b

For training the network, the Random Deformation technique is used to increase the amount of training data. Based on the article, explain how this technique is implemented and its effect on the model's performance.

solution

Random deformation is used in U-Net to augment training data by applying smooth elastic transformations. This is achieved by displacing grid control points with random values sampled from a Gaussian distribution and interpolating the deformation smoothly across the image using bicubic interpolation.
This technique effectively simulates realistic variations, such as tissue deformations, which are common in biomedical images. As a result, the model becomes more robust to such transformations and generalizes better, even with limited annotated training data.

part c

Consider the matrices below. Using the specified filter and input, apply the operation of Transposed Convolution.

$$\text{Input} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \text{Filter} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Note: For part (c), provide a step-by-step explanation. You can use Python libraries such as PyTorch to check your answer.

solution

The input size is 2×2, and the filter size is 2×2. The output size will be:

$$\text{Output size} = (H + K - 1) \times (W + K - 1) = (2 + 2 - 1) \times (2 + 2 - 1) = 3 \times 3.$$

We start with a 3×3 output matrix initialized to zeros:

$$\text{Output} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Multiply 1 (top-left element of input) by the filter:

$$1 \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.$$

Add this result to the top-left of the output matrix:

$$\text{Output} = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Multiply 2 (top-right element of input) by the filter:

$$2 \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}.$$

Add this result to the output matrix starting from the top-right position:

$$\text{Output} = \begin{bmatrix} 1 & 4 & 4 \\ 3 & 10 & 8 \\ 0 & 0 & 0 \end{bmatrix}.$$

Multiply 3 (bottom-left element of input) by the filter:

$$3 \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 9 & 12 \end{bmatrix}.$$

Add this result to the output matrix starting from the bottom-left position:

$$\text{Output} = \begin{bmatrix} 1 & 4 & 4 \\ 6 & 16 & 8 \\ 9 & 12 & 0 \end{bmatrix}.$$

Multiply 4 (bottom-right element of input) by the filter:

$$4 \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 12 & 16 \end{bmatrix}.$$

Add this result to the output matrix starting from the bottom-right position:

$$\text{Output} = \begin{bmatrix} 1 & 4 & 4 \\ 6 & 20 & 16 \\ 9 & 24 & 16 \end{bmatrix}.$$

we can use this code to verify it:

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

# Define input and filter tensors
input_tensor = torch.tensor([[1, 2], [3, 4]], dtype=torch.float32).
    unsqueeze(0).unsqueeze(0)
filter_tensor = torch.tensor([[1, 2], [3, 4]], dtype=torch.float32).
    unsqueeze(0).unsqueeze(0)

# Apply transposed convolution
output = F.conv_transpose2d(input_tensor, filter_tensor, stride=1,
    padding=0)
print("Output Matrix:")
print(output.squeeze(0).squeeze(0))

```

the output of this code is as follows:

```
        Output Matrix:
tensor([[ 1.,  4.,  4.],
        [ 6., 20., 16.],
        [ 9., 24., 16.]])

```

Q4

In the field of object detection, YOLO (You Only Look Once) algorithms are one-stage, real-time detection systems designed based on full image processing. YOLO versions are widely used in real-world projects due to their accuracy and real-time capabilities. To better understand the base versions of YOLO, you can refer to the following links:

- You Only Look Once: Unified, Real-Time Object Detection
- YOLO9000: Better, Faster, Stronger
- YOLOv3: An Incremental Improvement

part a

Suppose an object detection model includes 80 classes. Compare the output of YOLOv1 and YOLOv3 in terms of the number of channels (depth) for each cell and explain the reasons for the differences (consider the number of bounding boxes per cell as described in the original papers).

solution

For a model with $C = 80$ classes, YOLOv1 predicts $B = 2$ bounding boxes per cell. Each bounding box includes 4 coordinates (x, y, w, h) and 1 confidence score, while the grid cell as a whole predicts 80 class probabilities. This results in a total output depth of $B \times 5 + C = 2 \times 5 + 80 = 90$ channels per cell.
In YOLOv3, $B = 3$ bounding boxes are predicted per cell, with each bounding box outputting 4 coordinates, 1 objectness score, and 80 class probabilities. This gives $B \times (4 + 1 + C) = 3 \times 85 = 255$ channels per cell. Since YOLOv3 performs detection at three different scales, it generates three separate sets of predictions.
The differences in output depth arise from changes in design. YOLOv3 increases the number of bounding boxes per cell and introduces an explicit objectness score for each box. Unlike YOLOv1, it predicts class probabilities for each bounding box independently, enabling multi-label predictions. YOLOv3's multi-scale prediction also enhances its ability to detect objects of different sizes, contributing to its increased output complexity. These changes make YOLOv3 more accurate and flexible but increase computational cost.

part b

It is possible that some samples in the dataset may not belong to a specific class or belong to some classes simultanously. What solutions are provided in YOLOv3 to overcome this problem?

solution

YOLOv3 addresses class ambiguity by replacing the softmax layer with independent logistic classifiers for class predictions. This allows the model to output probabilities for each class independently, enabling multi-label predictions where objects belong to multiple categories simultaneously. Additionally, the objectness score suppresses predictions for samples that do not belong to any class, ensuring robustness to background or unlabeled data.

part c

In YOLO papers, how do they prevent duplicate detection and differentiate between distinct objects in the proposed algorithm?

solution

YOLO uses Non-Maximum Suppression (NMS) to eliminate duplicate detections by retaining only the highest-confidence bounding box when overlapping boxes have a high IoU. Distinct objects are differentiated by assigning each object to the grid cell containing its center and optimizing for IoU during training, ensuring accurate localization and separation.

part d

The main issue with anchor boxes in YOLO has been discussed in YOLOv2 paper. Explain the problem and the solutions YOLOv2 provided in detail.

solution

YOLOv2 identifies two main issues with anchor boxes: predefined dimensions do not match object distributions, and absolute location predictions cause training instability. To address these, YOLOv2 uses k-means clustering with an IoU-based distance metric to determine anchor box dimensions that better align with the dataset. Additionally, it predicts box centers relative to the grid cell with logistic constraints, stabilizing training and improving localization accuracy.

part e

What architectural differences exist between YOLOv3 and previous version?

solution

YOLOv3 introduces several key architectural improvements over previous versions. It replaces the backbone network with Darknet-53, which includes residual connections for better gradient flow and deeper feature extraction. Unlike YOLOv2, which uses a single-scale detection approach, YOLOv3 predicts bounding boxes at three different scales, improving detection for objects of varying sizes. It also replaces the softmax function for classification with independent logistic classifiers, enabling multi-label predictions for overlapping classes. Additionally, YOLOv3 increases the number of anchor boxes per grid cell to three, enhancing detection flexibility and accuracy.