Assignment 3

Mahdi Tabatabaei 400101515
Github Repository

Deep Learning

Dr. Fatemizadeh

December 10, 2024

Deep Learning

Assignment 3

Mahdi Tabatabaei 400101515
Github Repository

Question 1 (25 Points)

In this question, we analyze computational aspects related to convolutional networks.

(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.

Soloution

1. Output Dimensions

Since the stride is 1 and "Same Padding" is used, the output spatial dimensions remain the same as the input dimensions:

$$H_{out} = H, \quad W_{out} = W$$

The output depth is equal to the number of filters, N.
Thus, the output feature map size is:

$$H_{out} \times W_{out} \times N = H \times W \times N$$

2. Computational Cost

- Each filter has $K \times K$ weights per input channel (M channels).
- For each spatial position, the filter performs $K^2 \times M$ MACs.
- The number of spatial positions on the feature map is $H \times W$.
- The number of filters is N.

Thus, the total number of MACs is:

$$\text{Total MACs} = H \times W \times N \times K^2 \times M$$

(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?

Soloution

Layer 1 (First Convolutional Layer)

- Input Dimensions: $128 \times 128 \times 3$
- Output Dimensions: $64 \times 64 \times 64$
- Number of Parameters:

$$\text{Parameters per filter} = 5 \times 5 \times 3 + 1 = 76$$

$$\text{Total Parameters} = 76 \times 64 = 4,864$$

- Computational Cost:

$$\text{MACs} = 64 \times 64 \times 64 \times 5^2 \times 3 = 3,145,728$$

Layer 2 (Second Convolutional Layer)

- Input Dimensions: $64 \times 64 \times 64$
- Output Dimensions: $32 \times 32 \times 128$
- Number of Parameters:

$$\text{Parameters per filter} = 5 \times 5 \times 64 + 1 = 1,601$$

$$\text{Total Parameters} = 1,601 \times 128 = 204,928$$

- Computational Cost:

$$\text{MACs} = 32 \times 32 \times 128 \times 5^2 \times 64 = 838,860,800$$

Layer 3 (Third Convolutional Layer)

- Input Dimensions: $32 \times 32 \times 128$
- Output Dimensions: $16 \times 16 \times 256$
- Number of Parameters:

$$\text{Parameters per filter} = 5 \times 5 \times 128 + 1 = 3,201$$

$$\text{Total Parameters} = 3,201 \times 256 = 819,456$$

- Computational Cost:

$$\text{MACs} = 16 \times 16 \times 256 \times 5^2 \times 128 = 524,288,000$$

Soloution

Receptive Field of the Last Convolutional Layer
The receptive field of a neuron in the convolutional network refers to the region of the input image that influences that neuron's output. As we move deeper into the network, the receptive field grows larger.
To calculate the receptive field of each convolutional layer, we use the formula:

$$R_i = K_i + (S_i - 1) \times (R_{i-1} - 1)$$

where:
- R_i is the receptive field at layer i,
- K_i is the kernel size at layer i,
- S_i is the stride at layer i,
- R_{i-1} is the receptive field from the previous layer.

Receptive Field Calculation
For each layer:

- Layer 1: $K_1 = 5$, $S_1 = 2$, $R_1 = 5$
- Layer 2: $K_2 = 5$, $S_2 = 2$, $R_2 = 9$
- Layer 3: $K_3 = 5$, $S_3 = 2$, $R_3 = 17$

Thus, the receptive field of the last convolutional layer is 17×17.

Dependence of Neurons in the Final Layer on Input Image Resolution
The number of neurons in the final layer depends on the input image resolution. As the resolution of the input image decreases, the number of neurons in the final convolutional layer also decreases.
For example, with:

- Input size 128×128: The output of the final layer is $16 \times 16 \times 256$.
- Input size 64×64: The output of the final layer would be $8 \times 8 \times 256$.
- Input size 32×32: The output of the final layer would be $4 \times 4 \times 256$.

Thus, the number of neurons in the final layer decreases as the input resolution decreases, because each convolution layer reduces the spatial dimensions by half due to the combination of stride and kernel size.

(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.

Soloution

Depthwise Separable Convolutions
A depthwise separable convolution divides a standard convolution operation into two steps:

(a) **Depthwise Convolution:** A single $K \times K$ filter is applied independently to each input channel.

(b) **Pointwise Convolution:** A 1×1 convolution is applied across all channels to combine the outputs of the depthwise convolution.

This decomposition reduces both the number of parameters and the computational cost compared to standard convolutions.

Depthwise Separable Convolution

1. **Depthwise Convolution:**

- **Number of Parameters:**

$$\text{Parameters (Depthwise)} = K^2 \times M$$

- **Computational Cost:**

$$\text{MACs (Depthwise)} = H \times W \times K^2 \times M$$

2. **Pointwise Convolution:**

- **Number of Parameters:**

$$\text{Parameters (Pointwise)} = M \times N$$

- **Computational Cost:**

$$\text{MACs (Pointwise)} = H \times W \times M \times N$$

3. **Total Parameters and Computational Cost:**

$$\text{Parameters (Total)} = K^2 \times M + M \times N$$

$$\text{MACs (Total)} = H \times W \times (K^2 \times M + M \times N)$$

Replacing 2nd and 3rd layer with Depthwise

Second Layer

- **Input Dimensions:** $64 \times 64 \times 64$

- **Kernel Size:** 5×5
- **Number of Filters:** 128

Standard Convolution:

- Parameters:
$$\text{Parameters} = 5^2 \times 64 \times 128 = 204,800$$
- Computational Cost:
$$\text{MACs} = 64 \times 64 \times 5^2 \times 64 \times 128 = 838,860,800$$

Depthwise Separable Convolution:

- Depthwise Convolution:
$$\text{Parameters (Depthwise)} = 5^2 \times 64 = 1,600$$
$$\text{MACs (Depthwise)} = 64 \times 64 \times 5^2 \times 64 = 6,553,600$$
- Pointwise Convolution:
$$\text{Parameters (Pointwise)} = 64 \times 128 = 8,192$$
$$\text{MACs (Pointwise)} = 64 \times 64 \times 64 \times 128 = 33,554,432$$
- Total Parameters:
$$\text{Total Parameters} = 1,600 + 8,192 = 9,792$$
- Total MACs:
$$\text{Total MACs} = 6,553,600 + 33,554,432 = 40,108,032$$

Third Layer

- **Input Dimensions:** $32 \times 32 \times 128$
- **Kernel Size:** 5×5
- **Number of Filters:** 256

Standard Convolution:

- Parameters:
$$\text{Parameters} = 5^2 \times 128 \times 256 = 819,200$$
- Computational Cost:
$$\text{MACs} = 32 \times 32 \times 5^2 \times 128 \times 256 = 524,288,000$$

Depthwise Separable Convolution:

- Depthwise Convolution:

$$\text{Parameters (Depthwise)} = 5^2 \times 128 = 3,200$$

$$\text{MACs (Depthwise)} = 32 \times 32 \times 5^2 \times 128 = 3,276,800$$

- Pointwise Convolution:

$$\text{Parameters (Pointwise)} = 128 \times 256 = 32,768$$

$$\text{MACs (Pointwise)} = 32 \times 32 \times 128 \times 256 = 33,554,432$$

- Total Parameters:

$$\text{Total Parameters} = 3,200 + 32,768 = 35,968$$

- Total MACs:

$$\text{Total MACs} = 3,276,800 + 33,554,432 = 36,831,232$$

Comparison

- **Standard Convolutions:**
  - Total Parameters: $4,864 + 204,800 + 819,200 = 1,028,864$
  - Total MACs: $3,145,728 + 838,860,800 + 524,288,000 = 1,366,294,528$
- **Depthwise Separable Convolutions:**
  - Total Parameters: $4,864 + 9,792 + 35,968 = 50,624$
  - Total MACs: $3,145,728 + 40,108,032 + 36,831,232 = 80,084,992$

Reductions:

- **Parameters:** $\frac{50,624}{1,028,864} \approx 4.92\%$ (95.08% reduction)
- **MACs:** $\frac{80,084,992}{1,366,294,528} \approx 5.86\%$ (94.14% reduction)

(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.

Soloution

FC Parameters after Standard Convolutions (Part b)
After the standard convolutions in Part b, the Flatten layer produces a vector of size:

$$\text{Flattened vector size} = 16 \times 16 \times 256 = 65,536 \text{ features.}$$

The FC layer has 200 output neurons, so the total number of parameters in the FC layer is:

$$\text{FC Parameters} = 65,536 \times 200 + 200 = 13,107,200 + 200 = 13,107,400.$$

The total parameters in the convolutional layers are:

$$\text{Convolutional Parameters (Part b)} = 4,864 + 204,928 + 819,456 = 1,029,248.$$

Thus, the total number of parameters in the network is:

$$\text{Total Parameters (Standard Convolutions)} = 1,029,248 + 13,107,400 = 14,136,648.$$

The proportion of FC parameters relative to all parameters is:

$$\frac{\text{FC Parameters}}{\text{Total Parameters}} = \frac{13,107,400}{14,136,648} \approx 0.926 \quad (\text{or } 92.6\%).$$

FC Parameters after Depthwise Convolutions (Part c)
After applying depthwise convolutions in Part c, the parameters in the convolutional layers are:

$$\text{Convolutional Parameters (Depthwise)} = 4,864 + 51,200 + 204,800 = 260,864.$$

The FC layer parameters remain the same:

$$\text{FC Parameters} = 13,107,400.$$

Thus, the total number of parameters in the network is:

$$\text{Total Parameters (Depthwise Convolutions)} = 260,864 + 13,107,400 = 13,368,264.$$

The proportion of FC parameters relative to all parameters is:

$$\frac{\text{FC Parameters}}{\text{Total Parameters}} = \frac{13,107,400}{13,368,264} \approx 0.9804 \quad \text{(or 98.04\%)}.$$

Methods of parameter reduction and their impact come in the following table.

Method	**Reduction in FC Parameters**	**Impact on Performance**
Global Average Pooling (GAP)	Drastic reduction (99.6%)	Reduces overfitting, but may lose fine spatial information due to averaging over the feature maps.
Bottleneck Layer	Significant reduction (up to 75%)	Maintains more spatial information while reducing the number of feature maps before feeding into the FC layer.
Dropout	No direct reduction in FC parameters	Helps to reduce overfitting by preventing the network from relying on specific features, thus improving generalization.
1x1 Convolutions	Significant reduction	Retains spatial information but reduces the number of FC parameters by applying a 1×1 convolution instead of flattening.
Reduce FC Size (Pruning)	Direct reduction by cutting neurons	Faster, but may limit the model's capacity to learn complex patterns. This is useful when output classes are fewer or simpler.
Depthwise Separable Convolutions	Indirect reduction (smaller feature map)	Efficient method that reduces both parameters and computation cost. However, it may slightly impact the network's ability to capture high-level features.

Table 1: Methods to Reduce FC Parameters and Their Effects

Question 2 (25 Points)

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

- Explain the primary differences between ResNet's residual connections and DenseNet's dense connections. Briefly describe the advantages of each case.
- Explain how DenseNet reduces the vanishing gradient problem and what the computational benefits are.

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

Soloution

Differences Between ResNet and DenseNet Connections

- **ResNet's Residual Connections:**
  - Residual connections add shortcut paths to skip one or more layers, enabling the network to learn residual mappings instead of direct mappings.
  - Advantage: This mitigates the vanishing gradient problem by allowing gradients to flow directly through the shortcut paths, making it easier to train.
- **DenseNet's Dense Connections:**
  - Dense connections connect each layer to all subsequent layers, ensuring maximum feature reuse by concatenating feature maps from all preceding layers.
  - Advantage: DenseNet reduces redundancy, encourages feature propagation, and substantially reduces number of parameters and improving performance.

The Vanishing Gradient Problem and Computational Cost

- **Reducing the Vanishing Gradient Problem:**
  - DenseNet directly connects each layer to all preceding layers. This ensures that gradients flow more effectively through the network, as every layer has direct access to the loss function and the original input.
  - This dense connectivity facilitates better gradient propagation, minimizing the risk of gradients diminishing as they backpropagate through the network.
- **Computational Benefits:**
  - DenseNet improves feature reuse by concatenating feature maps, reducing the need to relearn redundant information across layers.
  - This architecture significantly reduces the number of parameters compared to traditional convolutional networks, making it more computationally efficient while maintaining or improving performance.

Practical Application of DenseNet Architecture

- **Practical Problem: Medical Image Classification**
  - **Description:** Accurate classification of medical images, such as identifying malignant tumors in MRI or CT scans.
  - **Why DenseNet is Suitable:** DenseNet's ability to reuse features and its parameter efficiency make it ideal for handling high-dimensional medical images where preserving intricate details is crucial.

- **Real-World Example: Melanoma Detection**
  - **Application:** Utilizing DenseNet to classify skin lesion images to distinguish between benign moles and malignant melanoma.
  - **Supporting Evidence:** Studies have demonstrated that DenseNet models achieve high accuracy in melanoma detection with fewer parameters compared to traditional CNNs, facilitating faster training and deployment in clinical settings.

Adapting DenseNet for Multi-Modal Data

- **Proposed Architecture:**
  - For multi-modal data, such as text and images, separate DenseNet branches can be used to process each modality independently.
  - Each branch extracts features specific to its input type (e.g., a DenseNet for image data and a DenseNet or transformer-based module for text data).
  - The outputs of these branches are concatenated or combined using a fusion mechanism, such as fully connected layers, attention, or weighted summation.
  - The fused features are passed through additional DenseNet layers or classification layers for the final task.
- **Justification:**
  - The dense connectivity of DenseNet ensures efficient feature propagation and reuse, enabling effective learning of both image and text representations.
  - Fusion of the modalities at an intermediate stage leverages the complementary nature of text and image data, improving the performance on tasks like visual question answering or image captioning.

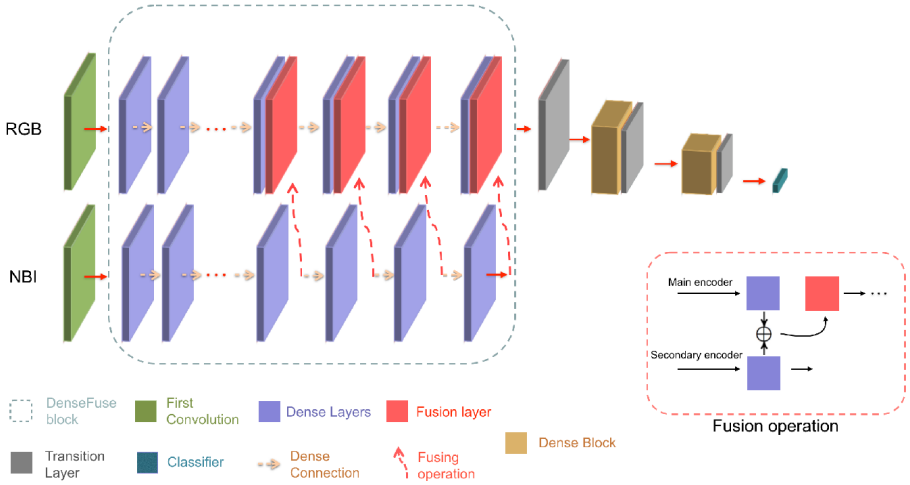

Figure 1: Diagram of Multimodal Densenet

Question 3 (25 Points)

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.

1. 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.

Soloution

In the U-Net architecture, the encoder and decoder are connected using **skip connections**. These connections help the network by passing important information directly from the encoder to the decoder, which improves segmentation accuracy.

1. Preserving Details (Localization):
The encoder downscales the image, losing some details. Skip connections bring these details from the encoder to the decoder, helping the network make more precise predictions at each pixel.

2. Combining Context and Precision:
The encoder captures large structures (context), and the decoder focuses on smaller details (precision). Skip connections combine both, allowing the network to handle both large and small structures at the same time.

3. Better Training:
Skip connections help the network learn more efficiently by making it easier for the gradients (which guide the learning) to flow through the network during training.

4. Improved Generalization with Limited Data:
Skip connections allow the network to use both low-level and high-level information, which helps it learn better even with limited data.

2. 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.

Soloution

The **Random Deformation** technique is used to artificially increase the amount of training data by applying random elastic deformations to the input images. This technique helps the network become more robust and generalize better, especially when only a small amount of annotated data is available.

How it's implemented:

- **Elastic Deformations:** Random displacements are applied to the pixels in the image. These displacements are smooth and local, simulating realistic deformations (such as stretching or bending).

- **Displacement Grid:** The displacements are applied on a **3x3 grid**. Each grid point is moved based on a random displacement vector sampled from a **Gaussian distribution**, which controls how much the image is deformed.
- **Bicubic Interpolation:** Once the displacement is applied, the image is resampled using **bicubic interpolation**. This smooths out the changes and ensures the deformed image looks realistic, avoiding jagged edges or unnatural distortions.

Effect on the model's performance:

- The technique makes the network more **robust** by teaching it to handle slight shifts, distortions, and variations in the data, which is common in biological tissues.
- It helps the network to learn from fewer labeled images by **simulating variations** in the training data, improving both **accuracy** and **generalization**.
- This approach reduces overfitting, especially when there is limited annotated data, and improves the model's ability to perform segmentation tasks on unseen data.

3. 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.

Soloution

Transposed convolution involves:

- Creating matrices by multiplying each input element with the filter
- Positioning these matrices at different offsets
- Summing the resulting matrices
- Transposed convolution "transposes" the typical convolution operation
- Each input element is multiplied by the entire filter
- Resulting matrices are positioned at different offsets
- The final output is obtained by summing these matrices

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

Stride $= 1$

$$\text{Transposed Convolution} = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 4 \\ 0 & 6 & 8 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 3 & 6 & 0 \\ 9 & 12 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 8 \\ 0 & 12 & 16 \end{bmatrix}$$

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

Stride $= 2$

$$\text{Transposed Convolution} = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 2 & 4 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 6 & 0 & 0 \\ 9 & 12 & 0 & 0 \end{bmatrix} + \dots$$

$$\dots + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 8 \\ 0 & 0 & 12 & 16 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 & 4 \\ 3 & 4 & 6 & 8 \\ 3 & 6 & 4 & 8 \\ 9 & 12 & 12 & 16 \end{bmatrix}$$

Question 4 (25 Points)

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

1. 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).

Soloution

The depth of the output for each cell in YOLOv1 and YOLOv3 is determined by the number of bounding boxes per cell and the predictions made for each bounding box.

YOLOv1

- **Output per cell:** $S \times S \times (B \times 5 + C)$
- **Details:**
  - S: Grid size (e.g., 7×7 for VOC dataset).
  - $B = 2$: Number of bounding boxes per cell.
  - 5 values per bounding box: x, y, w, h, and confidence score.
  - $C = 80$: Number of classes.
- **Depth per cell:** $2 \times 5 + 80 = 90$.

YOLOv3

- **Output per cell:** $N \times (B \times (4 + 1 + C))$
- **Details:**
  - $N = 3$: Predictions at three different scales.
  - $B = 3$: Number of bounding boxes per cell at each scale.
  - 4 values for box coordinates, 1 for objectness, and $C = 80$ for classes.
- **Depth per cell (per scale):** $3 \times (4 + 1 + 80) = 255$.

Comparison and Reasons

- **Bounding Boxes per Cell:** YOLOv3 predicts more bounding boxes ($B = 3$ vs. $B = 2$) to better handle varied object sizes and improve recall.
- **Multi-Scale Predictions:** YOLOv3 uses three scales, significantly increasing the depth of predictions.
- **Class Prediction Mechanism:** YOLOv1 uses a softmax function for classification, while YOLOv3 uses independent logistic classifiers for each class, enabling better multi-label handling.

2. 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?

Soloution

In YOLOv3, the following solutions address the issue where samples in the dataset may not belong to a specific class or may belong to multiple classes simultaneously:

1. Multi-label Classification Using Independent Logistic Regression

- Instead of using a softmax activation function for class predictions (as in YOLOv1), YOLOv3 uses independent logistic regression for each class.
- This approach allows the model to predict multiple classes for a single bounding box. If a sample belongs to multiple classes, the independent logistic outputs ensure that the probabilities for each class are independent.
- For example, an object can simultaneously be labeled as both *Person* and *Athlete* without conflict, as the logistic function calculates the probability of each class independently.

2. Binary Cross-Entropy Loss for Class Predictions

- YOLOv3 employs binary cross-entropy loss for class predictions instead of categorical cross-entropy. This choice accommodates the possibility of multiple classes per sample.
- Binary cross-entropy evaluates each class prediction independently, making it suitable for multi-label classification scenarios.

3. Flexible Bounding Box Assignment

- YOLOv3 uses objectness scores to determine whether a bounding box contains an object or not, regardless of its class.
- If a bounding box does not overlap sufficiently with any ground-truth object (e.g., it falls into the background), its objectness score is predicted as 0, effectively ignoring class predictions for that box.

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

Soloution

1. Non-Maximum Suppression (NMS)

- After predicting multiple bounding boxes, YOLO applies Non-Maximum Suppression (NMS) to remove duplicates.
- NMS retains the bounding box with the highest confidence score while discarding other overlapping boxes with an Intersection over Union (IoU) above a predefined threshold.
- This ensures that only one bounding box is kept for each detected object.

2. Spatial Constraints

- YOLO divides the input image into a grid and assigns each grid cell responsibility for detecting objects whose center falls within that cell.
- This spatial assignment reduces duplicate predictions for the same object, as only one cell is responsible for it.

3. Confidence Scores and Objectness

- YOLO predicts a confidence score for each bounding box, representing how likely it is to contain an object and how accurate the localization is.
- Bounding boxes with low confidence scores are ignored, further reducing false positives and duplicates.

4. Explain why YOLOv2 and YOLOv3, unlike YOLOv1, adopt an approach where the network is trained to work on images of varying sizes? How is this idea implemented, and why is it useful?

Soloution

YOLOv2 and YOLOv3 adopt training on images of varying sizes to make the model more robust and adaptable to different input resolutions. This is implemented using **multi-scale training**, where the input size is randomly changed every few iterations during training (e.g., between 320×320 to 608×608).

Implementation

- The network resizes the input image to a randomly chosen dimension (multiple of 32) before each batch.
- This ensures the model learns to predict effectively across different resolutions.

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

Soloution

Problem with Anchor Boxes in YOLOv2

(a) **Manually Defined Priors:** The dimensions of anchor boxes are hand-picked, which may not match the dataset's object distribution well.

(b) **Model Instability:** Predicting bounding box coordinates as offsets from anchor boxes caused instability during early training.

Solutions in YOLOv2

(a) **Dimension Clustering:**

- K-means clustering was applied to the training set's bounding boxes to generate anchor box dimensions.
- A distance metric based on Intersection over Union (IoU) was used instead of Euclidean distance to optimize for detection performance.

(b) **Direct Location Prediction:**

- YOLOv2 predicted bounding box center coordinates relative to the grid cell using a sigmoid activation to constrain the output between 0 and 1.
- This bounded prediction improved stability during training.

6. What architectural differences exist between YOLOv3 and previous version?

Soloution

1. Backbone Network

- **YOLOv2:** Uses *Darknet-19* as the backbone, consisting of 19 convolutional layers and 5 max-pooling layers.
- **YOLOv3:** Introduces *Darknet-53*, a deeper network with 53 convolutional layers and residual connections for better feature extraction and gradient flow.

2. Multi-Scale Predictions

- YOLOv3 predicts bounding boxes at 3 different scales, enabling the detection of objects of varying sizes.
- Previous versions predict bounding boxes at a single scale, making it less effective for detecting small objects.

3. Bounding Box Predictions

- YOLOv3 predicts 3 bounding boxes per cell for each scale, leading to a total of 9 predictions per grid cell.
- YOLOv2 and YOLOv1 use fewer bounding boxes per cell, limiting flexibility in detecting objects with diverse shapes and sizes.

4. Class Predictions

- **YOLOv3:** Uses independent logistic regression for class predictions, allowing multi-label classification.
- **YOLOv2/YOLOv1:** Use softmax for class predictions, restricting each bounding box to a single class.

5. Feature Extraction

- YOLOv3 adopts a feature pyramid network-like approach, combining high-level and low-level features for improved accuracy.
- YOLOv2 and YOLOv1 rely on a single feature map for predictions, which limits their ability to capture fine-grained details.

6. Activation Functions

- YOLOv3 uses Leaky ReLU for hidden layers and sigmoid activation for bounding box predictions.
- This is consistent with YOLOv2 but a refined implementation compared to YOLOv1.