Deep Learning

homework 3

Amirparsa Bahrami 401101332
github repository

Question 1

(a) Number of Parameters and Computational Cost

For a standard convolutional layer with kernel size $K \times K$, stride $S = 1$, and same padding applied to an input feature map of dimensions $H \times W$ with M channels, and producing N output channels:

- **Number of Parameters:** Each filter has dimensions $K \times K \times M$. Since there are N such filters, the total number of parameters (including biases) is:

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

- **Computational Cost (Number of Multiply-Accumulate Operations - MACs):** For each position in the output feature map, each filter performs $K \times K \times M$ multiplications and accumulations. The output feature map has dimensions $H \times W$ due to stride 1 and same padding. Therefore, the total computational cost is:

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

(b) Convolutional Network Calculations

1. Dimensions, Parameters, and Computational Cost

First Convolutional Layer:

- **Output Dimensions:**

$$O = \left\lfloor \frac{I + 2P - K}{S} \right\rfloor + 1 = \left\lfloor \frac{128 + 4 - 5}{2} \right\rfloor + 1 = 64$$

So, output size is $64 \times 64 \times 64$.

- **Number of Parameters:**

$$\text{Parameters} = N \times (K \times K \times M) + N = 64 \times (5 \times 5 \times 3) + 64 = 4800 + 64 = 4864$$

- **Computational Cost:**

$$\text{MACs} = H_{\text{out}} \times W_{\text{out}} \times N \times (K \times K \times M) = 64 \times 64 \times 64 \times 75 = 196,608,000$$

Second Convolutional Layer:

- **Output Dimensions:**

$$O = \left\lfloor \frac{64 + 4 - 5}{2} \right\rfloor + 1 = 32$$

  Output size is $32 \times 32 \times 128$.

- **Number of Parameters:**

$$\text{Parameters} = 128 \times (5 \times 5 \times 64) + 128 = 204,800 + 128 = 204,928$$

- **Computational Cost:**

$$\text{MACs} = 32 \times 32 \times 128 \times (5 \times 5 \times 64) = 32 \times 32 \times 128 \times 16,000 = 2,097,152,000$$

Third Convolutional Layer:

- **Output Dimensions:**

$$O = \left\lfloor \frac{32 + 4 - 5}{2} \right\rfloor + 1 = 16$$

  Output size is $16 \times 16 \times 256$.

- **Number of Parameters:**

$$\text{Parameters} = 256 \times (5 \times 5 \times 128) + 256 = 819,200 + 256 = 819,456$$

- **Computational Cost:**

$$\text{MACs} = 16 \times 16 \times 256 \times (5 \times 5 \times 128) = 16 \times 16 \times 256 \times 40,000 = 2,621,440,000$$

2. Receptive Field Analysis The receptive field (RF) of a neuron in the last convolutional layer represents the region of the input image that affects this neuron. We calculate it recursively:

- **Initial RF and Stride:**

$$\text{RF}_0 = 1, \quad \text{Stride}_0 = 1$$

- **First Layer:**

$$\text{RF}_1 = \text{RF}_0 + (K - 1) \times \text{Stride}_0 = 1 + (5 - 1) \times 1 = 5$$

$$\text{Stride}_1 = \text{Stride}_0 \times S = 1 \times 2 = 2$$

- **Second Layer:**

$$\text{RF}_2 = \text{RF}_1 + (K - 1) \times \text{Stride}_1 = 5 + 4 \times 2 = 13$$

$$\text{Stride}_2 = 2 \times 2 = 4$$

- **Third Layer:**

$$\text{RF}_3 = \text{RF}_2 + (K - 1) \times \text{Stride}_2 = 13 + 4 \times 4 = 29$$

$$\text{Stride}_3 = 4 \times 2 = 8$$

Conclusion: Each neuron in the last convolutional layer has a receptive field of 29×29 pixels in the input image.

(c) Depthwise Separable Convolutions

1. Parameters and Computational Cost Comparison

In depthwise separable convolutions:

[leftmargin=2cm]**Depthwise Convolution:**

- Applies a single filter per input channel.
- Parameters: $M \times K \times K$
- MACs: $H \times W \times M \times K \times K$

- **Pointwise Convolution:**
  - Uses 1×1 convolutions to combine features.
  - Parameters: $M \times N$
  - MACs: $H \times W \times M \times N$

Total for Depthwise Separable Convolution:

- Parameters: $M \times K \times K + M \times N$
- MACs: $H \times W \times M \times (K \times K + N)$

Comparison with Standard Convolution:

- Standard Convolution Parameters: $N \times M \times K \times K$
- Standard MACs: $H \times W \times N \times M \times K \times K$

Reduction Factor:

- Parameters Reduction: $\frac{N \times M \times K^2}{M \times K^2 + M \times N}$
- Computational Cost Reduction: Significant, especially when N and K are large.

2. Applying to the Convolutional Network

Second Layer with Depthwise Separable Convolution:

- **Parameters:**
  - Depthwise: $64 \times 5 \times 5 = 1,600$
  - Pointwise: $64 \times 128 = 8,192$
  - Total: $1,600 + 8,192 = 9,792$

- **Computational Cost:**
  - Depthwise MACs: $32 \times 32 \times 64 \times 25 = 1,638,400$
  - Pointwise MACs: $32 \times 32 \times 64 \times 128 = 8,388,608$
  - Total MACs: $10,027,008$

Third Layer with Depthwise Separable Convolution:

- **Parameters:**
  - Depthwise: $128 \times 5 \times 5 = 3,200$
  - Pointwise: $128 \times 256 = 32,768$
  - Total: $3,200 + 32,768 = 35,968$
- **Computational Cost:**
  - Depthwise MACs: $16 \times 16 \times 128 \times 25 = 819,200$
  - Pointwise MACs: $16 \times 16 \times 128 \times 256 = 8,388,608$
  - Total MACs: $9,207,808$

Comparison:

- **Parameters Reduced:**
  - Second Layer: From 204,928 to 9,792
  - Third Layer: From 819,456 to 35,968
- **Computational Cost Reduced:**
  - Significant reduction in MACs, making the network more efficient.

(d) Fully Connected Layer and Parameter Reduction

1. Total Number of Parameters

Using Standard Convolutions (Part b):

- Convolutional Layers Total Parameters:
  $4,864 + 204,928 + 819,456 = 1,029,248$
- Flatten Layer Output Size:
  $16 \times 16 \times 256 = 65,536$
- Fully Connected Layer Parameters:
  $65,536 \times 200 + 200 = 13,107,200 + 200 = 13,107,400$
- Total Parameters:
  $1,029,248 + 13,107,400 = 14,136,648$

Using Depthwise Separable Convolutions (Part c):

- Convolutional Layers Total Parameters:
  $4,864 + 9,792 + 35,968 = 50,624$
- Fully Connected Layer Parameters:
  Remains $13,107,400$
- Total Parameters:
  $50,624 + 13,107,400 = 13,158,024$

2. Contribution of the Fully Connected Layer

The Fully Connected (FC) layer dominates the total number of parameters:

- Over 13 million parameters in the FC layer versus around 1 million (standard) or 50,000 (depthwise) in convolutional layers.

3. Solutions to Reduce Parameters

Use Global Average Pooling (GAP):

- Replace the Flatten layer with GAP to reduce each feature map to a single value.
- Output after GAP: $1 \times 1 \times 256$
- New Fully Connected Layer Parameters:
  $256 \times 200 + 200 = 51,200 + 200 = 51,400$
- Total Parameters with GAP:
  - **Using Standard Convolutions:**
    $1,029,248 + 51,400 = 1,080,648$
  - **Using Depthwise Separable Convolutions:**
    $50,624 + 51,400 = 102,024$

Effect:

- **Parameter Reduction:** Dramatic decrease in the number of parameters.
- **Potential Impact:** May slightly affect accuracy but significantly improves efficiency and reduces overfitting.

Question 2

ResNet (Residual Network)

- **Residual Connections:** Add the input x to the output of a few layers $F(x)$ to form $y = F(x) + x$.
- **Purpose:** Address the vanishing gradient problem by allowing gradients to flow directly through the identity connections.
- **Operation:** Element-wise addition requires the input and output to have the same dimensions.

DenseNet (Densely Connected Network)

- **Dense Connections:** Concatenate the outputs from all preceding layers, so each layer receives the feature maps of all previous layers.
- **Purpose:** Encourage feature reuse and strengthen feature propagation.
- **Operation:** Concatenation increases the number of input channels for subsequent layers.

Main Differences

- **Connection Method:** ResNet uses addition; DenseNet uses concatenation.
- **Feature Reuse:** DenseNet explicitly reuses features from all previous layers.
- **Model Complexity:** DenseNet can be more parameter-efficient due to feature reuse.

How DenseNet Mitigates the Vanishing Gradient Problem and Its Computational Advantage

- **Vanishing Gradient Mitigation:**
  - **Direct Gradient Flow:** Dense connections provide short paths from the loss function to early layers, ensuring that gradients remain strong.
  - **Feature Reuse:** Early layers receive supervision from the loss function directly, enhancing learning.
- **Computational Advantage:**
  - **Parameter Efficiency:** By reusing features, DenseNet reduces the need to learn redundant mappings, leading to fewer parameters.
  - **Reduced Overfitting:** Fewer parameters can lead to better generalization, especially on smaller datasets.

When to Use DenseNet and Real-World Example

Recommended Use Cases:

- **Limited Data Scenarios:** When the dataset is small, and overfitting is a concern.
- **Resource-Constrained Environments:** Where model size and computational efficiency are critical.

Real-World Example: Medical Imaging Analysis

- **Application:** Classifying or segmenting medical images like X-rays or MRI scans.
- **Justification:** Medical datasets are often limited in size. DenseNet's parameter efficiency and feature reuse make it suitable for achieving high performance without overfitting.

Adapting DenseNet for Multi-Modal Input Data

Proposed Architecture:

1. **Separate Modalities Processing:**
   - **Image Modality:** Use a DenseNet to process image data.
   - **Text Modality:** Use an appropriate network (e.g., embeddings followed by LSTM or Transformer layers) to process text data.
2. **Feature Fusion:**
   - **Concatenate or Combine Features:** Merge the high-level features from both modalities. This can be done via concatenation or attention mechanisms.
3. **Joint Processing:**
   - **Combined Dense Layers or Further Dense Blocks:** Process the fused features to learn interactions between modalities.
4. **Output Layer:**
   - **Final Classification or Regression Layer:** Use a fully connected layer to map to the desired output.

Justification:

- **Modality-Specific Feature Extraction:** Allows each modality to be processed optimally.
- **Feature Interaction Learning:** The architecture enables learning of complex relationships between images and text.
- **Leveraging DenseNet Strengths:** Dense connections in the image pathway enhance feature propagation and reuse.

Visualization (Simplified)

```
[Image Input] ---> [DenseNet Blocks] ---
                                        \
                                         --> [Feature Fusion] --> [Joint Processing] -->
                                        /
[Text Input] ---> [Text Processing Network] ---
```

Conclusion: By carefully analyzing and optimizing the network architectures, we can significantly reduce computational costs and model size while maintaining performance, which is crucial for practical applications.

Question 3

(a) The Role and Impact of Skip Connections in U-Net

In the U-Net architecture, the network is divided into two main sections: the encoder (contracting path) and the decoder (expanding path). These two sections are connected through skip connections that link corresponding layers in the encoder and decoder.

The primary reasons for using skip connections in U-Net are:

1. **Preservation of Spatial Information:** As the encoder compresses the input image through successive convolutions and pooling operations, spatial resolution and fine-grained details are lost. The skip connections transfer high-resolution feature maps from the encoder directly to the decoder, allowing the network to retain important spatial information that is crucial for tasks like semantic segmentation.

2. **Facilitation of Gradient Flow:** Skip connections help mitigate the vanishing gradient problem by providing shortcut paths for gradients to flow back through the network during backpropagation. This leads to better training of deeper networks.

Impact on the Network:

- **Improved Localization:** By combining low-level features (from the encoder) with high-level features (from the decoder), the network can make more precise predictions, enhancing the localization of objects within the image.

- **Better Convergence:** Skip connections enable the network to converge faster and achieve higher accuracy, as they alleviate issues related to deep network training.

(b) Implementation and Effect of Random Deformation in Data Augmentation

Implementation of Random Deformation: Random deformation is a data augmentation technique that involves applying random geometric transformations to the training images and their corresponding labels (masks). Here's how it's typically implemented:

1. **Elastic Deformations:** Small random distortions are applied to the images by moving pixels around using a displacement field, which is often generated using random Gaussian filters. This simulates realistic variations such as tissue deformations in medical images.

2. **Affine Transformations:** Random rotations, translations, scaling, and shearing are applied to the images to create variations in orientation and size.

3. **Intensity Variations:** Adjusting the brightness, contrast, or adding noise to simulate different imaging conditions.

Effect on Model Performance:

- **Increased Data Diversity:** By augmenting the dataset with deformed images, the model is exposed to a wider variety of scenarios, which helps it generalize better to unseen data.

- **Reduced Overfitting:** Data augmentation helps prevent the model from learning spurious patterns specific to the training set, thus reducing overfitting.

- **Enhanced Robustness:** The model becomes more robust to variations and deformations in the input data, improving its performance in real-world applications.

(c) Performing Transposed Convolution on the Given Matrices

Given:

$$\text{Input Matrix (I): } I = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\text{Filter Matrix (F): } F = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Objective: Perform a transposed convolution of the input matrix I with the filter F and explain the result step by step.

Step-by-Step Explanation

1. **Determine Output Size:** For a transposed convolution with stride $s = 1$ and padding $p = 0$, the output size O can be calculated as:

   $$O = (\text{Input Size} - 1) \times s - 2p + \text{Filter Size}$$

   Substituting values:

   $$O = (2 - 1) \times 1 - 0 + 2 = 3$$

   So, the output will be a 3×3 matrix.

2. **Initialize the Output Matrix:**

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

3. **Process Each Element in the Input Matrix:** For each element $I(i, j)$ in the input matrix, we:

   - Multiply the filter F by $I(i, j)$.
   - Place the resulting matrix onto the output matrix, starting at position (i, j).
   - Sum the values if there are overlapping positions.

4. **Processing Elements:**

   - **Element at (0,0):** $I(0, 0) = 1$

     $$1 \times F = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

     Place it onto the output matrix starting at (0,0).

     $$\begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

   - **Element at (0,1):** $I(0, 1) = 2$

     $$2 \times F = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

     Place it onto the output matrix starting at (0,1).

- **Element at (1,0):** $I(1,0) = 3$

$$3 \times F = \begin{bmatrix} 3 & 6 \\ 9 & 12 \end{bmatrix}$$

- **Element at (1,1):** $I(1,1) = 4$

$$4 \times F = \begin{bmatrix} 4 & 8 \\ 12 & 16 \end{bmatrix}$$

5. **Final Output Matrix:**

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

Verification Using PyTorch

```python
import torch
import torch.nn as nn

# Define input and filter tensors
input_tensor = torch.tensor([[1., 2.],
                             [3., 4.]]).unsqueeze(0).unsqueeze(0)  # Shape: [N, C_in, H_in, W_in]

filter_tensor = torch.tensor([[1., 2.],
                              [3., 4.]]).unsqueeze(0).unsqueeze(0)  # Shape: [C_out, C_in, H_k, W_k]

# Create a ConvTranspose2d layer
conv_transpose = nn.ConvTranspose2d(in_channels=1, out_channels=1, kernel_size=2, stride=1, bias=False)
conv_transpose.weight = nn.Parameter(filter_tensor)

# Perform transposed convolution
output_tensor = conv_transpose(input_tensor)

print(output_tensor.squeeze())
```

Output:

$$\text{tensor}\left(\begin{bmatrix} 1 & 4 & 4 \\ 6 & 20 & 16 \\ 9 & 24 & 16 \end{bmatrix}\right)$$

Question 4

(a) Predictions in YOLOv1 and YOLOv3

In YOLOv1, each grid cell predicts bounding boxes and class probabilities as follows:

- **Bounding Boxes:** Each cell predicts 2 bounding boxes, with each box consisting of 5 values: x, y, w, h, and confidence score. This totals to $2 \times 5 = 10$ values per cell.
- **Class Probabilities:** Class probabilities are predicted once per grid cell for all classes, totaling 80 values for an 80-class dataset.

Therefore, the output depth per grid cell in YOLOv1 is:

$$10 + 80 = 90 \text{ channels.}$$

In YOLOv3, predictions are made using anchor boxes at three different scales, with each grid cell predicting:

- **Anchor Boxes:** Each cell predicts 3 anchor boxes per scale, and since predictions are made at three scales, there are $3 \times 3 = 9$ anchor boxes in total.
- **Per Anchor Box Predictions:** Each anchor box predicts $5 + 80 = 85$ values (5 for bounding box coordinates and objectness score, and 80 for class probabilities).

Therefore, the output depth per grid cell in YOLOv3 is:

$$3 \times 85 = 255 \text{ channels per scale.}$$

Reason for the Difference:

- YOLOv1 predicts class probabilities once per grid cell, sharing them across all bounding boxes within that cell.
- YOLOv3 predicts class probabilities for each anchor box individually, increasing the number of predictions and hence the output channels. This per-anchor box prediction allows for more precise detection but results in a higher output depth.

(b) Multi-Class Objects and Overlapping Labels in YOLOv3

YOLOv3 addresses the problem of overlapping labels and objects belonging to multiple classes by replacing the softmax activation function with independent logistic regression classifiers (sigmoid activation functions) for each class. This allows the model to predict a separate probability for each class independently, enabling multi-label classification where an object can belong to multiple classes simultaneously.

(c) Non-Maximum Suppression (NMS) in YOLO

YOLO uses the Non-Maximum Suppression (NMS) algorithm to prevent duplicate and multiple detections of the same object. NMS works by:

1. Selecting the bounding box with the highest confidence score for each object.

2. Suppressing other overlapping boxes whose Intersection over Union (IoU) with the selected box exceeds a predefined threshold.

This ensures that only the most relevant bounding box is retained for each detected object.

(d) Variable Image Sizes in YOLOv2 and YOLOv3

In YOLOv2 and YOLOv3, the networks are designed to be fully convolutional, eliminating fully connected layers that require a fixed input size. This architectural change allows the network to accept images of different sizes because convolutional layers can process inputs of varying dimensions.

Implementation of Variable Image Sizes:

- **Multi-Scale Training:** During training, the input image size is changed every few iterations. The network randomly selects a new image dimension (e.g., between 320 and 608 pixels in steps of 32) every certain number of batches.

Benefits:

- **Robustness to Scale:** Training on images of different sizes helps the network learn to detect objects at various scales, improving its generalization.
- **Speed-Accuracy Trade-off:** At inference time, the network can process images at different resolutions, allowing users to balance between detection speed and accuracy based on their needs.

(e) Problems and Solutions with Anchor Boxes in YOLOv2

The two main problems mentioned in the YOLOv2 paper regarding the use of anchor boxes are:

1. **Decrease in Recall:** Introducing anchor boxes initially led to a reduction in the model's ability to detect all relevant objects, resulting in fewer true positives.
2. **Training Instability:** The model became unstable during training when using anchor boxes, making it difficult to converge to an optimal solution.

Solutions Provided:

1. **Dimension Clusters for Anchor Boxes:**
   - **Problem Addressed:** Both the decrease in recall and training instability.
   - **Solution:** By performing k-means clustering on the bounding box dimensions in the training dataset, the model derives a set of anchor box priors that better match the objects' sizes. This leads to more appropriate anchor boxes, improving recall and stabilizing training.
2. **Direct Location Prediction:**
   - **Problem Addressed:** Training instability.
   - **Solution:** YOLOv2 predicts bounding box coordinates relative to the grid cell using logistic activation functions. This constrains the predictions to fall within the grid cell, which helps stabilize training by preventing extreme coordinate predictions.

Additionally, YOLOv2:

- **Increases Input Resolution:** Enhancing the grid size (e.g., from 7×7 to 13×13) improves the detection of smaller objects, addressing the recall issue.
- **Applies Batch Normalization:** This accelerates convergence and improves overall performance by stabilizing the learning process.

(f) Differences Between YOLOv3 and YOLOv2

The key differences between the YOLOv3 network architecture and its predecessor include:

1. **Backbone Network Upgrade:**
   - YOLOv2: Utilizes Darknet-19 as the backbone.
   - YOLOv3: Introduces Darknet-53, a deeper network with 53 convolutional layers and residual connections inspired by ResNet. This enhances feature extraction capabilities.
2. **Multi-Scale Predictions:** YOLOv3 makes predictions at three different scales by concatenating feature maps from different layers. This allows the detection of objects of various sizes more effectively.
3. **Residual Connections:** Incorporation of residual blocks helps in training deeper networks by mitigating the vanishing gradient problem, which was not present in YOLOv2.
4. **Class Prediction Method:**
   - YOLOv2: Uses softmax activation for class prediction, assuming mutually exclusive classes.
   - YOLOv3: Employs independent logistic classifiers with sigmoid activation functions for each class, enabling multi-label classification.
5. **Feature Fusion:** YOLOv3 combines low-level and high-level features through upsampling and concatenation, improving the detection performance across different object scales.
6. **Bounding Box Predictions:** While both versions use anchor boxes, YOLOv3 refines this by predicting bounding boxes across multiple scales and using logistic regression for more accurate coordinate predictions.

These enhancements allow YOLOv3 to achieve better accuracy and robustness, particularly in detecting small objects and handling scenarios where objects may belong to multiple classes.

End of homework 3