Assignment 3

Ehsan Merrikhi 400101967

github repository

Deep Learning

Dr. Fatemizadeh

December 13, 2024

Deep Learning

Assignment 3

Ehsan Merrikhi 400101967
github repository

Contents

Question 1 (25 Points)

In this question, we examine computational topics related to convolutional networks.

Part 1.

A standard convolutional layer with a kernel $K \times K$ is applied, assuming **Same Padding** and **Stride = 1**, to a **Feature Map** with dimensions $H \times W \times M$, where M is the number of channels. The number of channels in the output of this layer is equal to N. Determine the number of parameters and the computational cost (number of multiply operations) of this convolutional layer in terms of the variables H, W, M, N, K.

Soloution

1. Number of parameters

Each kernel is of size $K \times K$, and there are M input channels for each output channel. Therefore, the total number of weights is:

$$\text{Number of Weights} = K \times K \times M \times N$$

Additionally, each output channel has a bias term. Therefore, the total number of parameters is:

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

2. Computational Cost

For a convolution operation:

- At each spatial location, the kernel performs $K \times K \times M$ multiplications and accumulates the results (additions).
- There are $H \times W$ spatial locations on the output feature map, and N output channels.

Thus, the total number of multiply-add operations is:

$$\text{Multiply-Add Operations} = H \times W \times K \times K \times M \times N$$

Final Answer:

1. **Number of Parameters:**
$$K^2 \times M \times N + N$$
2. **Computational Cost:**
$$H \times W \times K^2 \times M \times N$$

Part 2.

Now, consider a convolutional network where the input is a color image with dimensions 128×128. Assume the network has three convolutional layers with:

- Kernel size: 5×5, **Padding = 2**, **Stride = 2**, ReLU activation.

The layers have 64, 128, and 256 filters, respectively. Answer the following:

1. Calculate the number of parameters, computational cost, and output dimensions for each convolutional layer.
2. Compute the **receptive field** for a neuron in the last convolutional layer. In another saying, explain how the number of pixels in the input image affects a neuron in the last layer of the network.

Soloution

2. Receptive Field Calculation

For the given network with three convolutional layers, each having a kernel size of 5×5, stride of 2, and padding of 2, the receptive field can be calculated as follows:

$$\begin{aligned}
&\text{Layer 1:}\\
&\qquad \text{Receptive Field} \quad R_1 = K = 5\\
&\qquad \text{Jump} \quad J_1 = S = 2\\
\\
&\text{Layer 2:}\\
&\qquad R_2 = R_1 + (K-1) \times J_1 = 5 + (5-1) \times 2 = 13\\
&\qquad J_2 = J_1 \times S = 2 \times 2 = 4\\
\\
&\text{Layer 3:}\\
&\qquad R_3 = R_2 + (K-1) \times J_2 = 13 + (5-1) \times 4 = 29\\
&\qquad J_3 = J_2 \times S = 4 \times 2 = 8
\end{aligned}$$

Thus, the receptive field of a neuron in the last convolutional layer is:

$$\boxed{R_3 = 29 \times 29 \text{ pixels}}$$

Soloution

1. First Convolutional Layer

- Input Dimensions: $128 \times 128 \times 3$, Kernel Size: 5×5, Number of Filters: 64, Stride: 2, Padding: 2

Number of Parameters:

$$\text{Number of Parameters} = (5 \times 5 \times 3 \times 64) + 64 = 4800 + 64 = 4864$$

Computational Cost:

$$\text{Output Height} = \frac{128 - 5 + 2 \times 2}{2} + 1 = 64, \quad \text{Output Width} = 64$$

$$\text{Multiply Operations} = 64 \times 64 \times \overbrace{5 \times 5 \times 3}^{\text{Operations per pixel}} \times 64 = 19,660,800$$

Output Dimensions: $64 \times 64 \times 64$

Second Convolutional Layer

- Input Dimensions: $64 \times 64 \times 64$, Kernel Size: 5×5, Number of Filters: 128, Stride: 2, Padding: 2,

Number of Parameters:

$$\text{Number of Parameters} = (5 \times 5 \times 64 \times 128) + 128 = 204800 + 128 = 204,928$$

Computational Cost:

$$\text{Output Height} = \frac{64 - 3 + 2 \times 1}{2} + 1 = 32, \quad \text{Output Width} = 32$$

$$\text{Multiply Operations} = 32 \times 32 \times \overbrace{5 \times 5 \times 64}^{\text{Operations per pixel}} \times 128 = 209,715,200$$

Output Dimensions: $32 \times 32 \times 128$

Third Convolutional Layer

- Input Dimensions: $32 \times 32 \times 128$, Kernel Size: 5×5, Number of Filters: 256, Stride: 2, Padding: 2

Number of Parameters:

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

Computational Cost:

$$\text{Output Height} = \frac{32 - 2 + 2 \times 0}{2} + 1 = 16, \quad \text{Output Width} = 16$$

$$\text{Multiply Operations} = 16 \times 16 \times \overbrace{5 \times 5 \times 128}^{\text{Operations per pixel}} \times 256 = 209,715,200$$

Output Dimensions: $16 \times 16 \times 256$

Part 3.

In this section, we discuss **Depthwise Separable Convolutions**, which are used in the MobileNet architecture. Answer the following questoins:

- Calculate the number of parameters and computational cost for a **Depthwise Separable Convolution layer**, and compare it with a regular convolution (Section A).
- Reconsider the convolutional network from **Section B**, but now assume that the second and third convolutional layers are replaced with **Depthwise Separable Convolutions** like in MobileNet. Compare the number of parameters and computational cost of the convolutional layers with Section B.

Soloution

1. Depthwise Separable Convolution Properties

Parameters:

- **Depthwise Convolution:**

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

- **Pointwise Convolution:**

$$\text{Parameters} = (1 \times 1 \times M \times N) + N$$

$$\textbf{Total Parameters} = K^2M + MN + M + N$$

Computational Cost:

- **Depthwise Convolution:**

$$\text{Computations} = H_{\text{out}} \times W_{\text{out}} \times M \times K^2$$

- **Pointwise Convolution:**

$$\text{Computations} = H_{\text{out}} \times W_{\text{out}} \times N \times M$$

$$\textbf{Total Computations} = H_{\text{out}} \times W_{\text{out}} \times M \times (K^2 + N)$$

Regular Convolution Properties:

$$\text{Parameters} = (K \times K \times M \times N) + N \quad \text{Computations} = H_{\text{out}} \times W_{\text{out}} \times N \times K^2 \times M$$

Comparative Analysis

- **Parameters:** Depthwise separable convolutions reduce the number of parameters.
- **Computational Cost:** Computational cost is slightly increased.

Overall: Depthwise is favored due to significant reduction in number of parameters.

Soloution

2. Modified Convolutional Network

A. Original Network Specifications (Section B)

- **Layer 1:** Regular Convolution
  - Parameters: 4,864 & Computational Cost: 19,660,800 operations
- **Layer 2:** Regular Convolution
  - Parameters: 204,928 & Computational Cost: 209,715,200 operations
- **Layer 3:** Regular Convolution
  - Parameters: 819,456 & Computational Cost: 209,715,200 operations

B. Modified Network with Depthwise Separable Convolutions

Modified Layer 2: Depthwise Separable Convolution

- **Parameters:**

$$\left.\begin{array}{l}\text{Depthwise:} \quad 5 \times 5 \times 64 + 64 = 1{,}664 \\ \text{Pointwise:} \quad 1 \times 1 \times 64 \times 128 + 128 = 8{,}320\end{array}\right\} \quad \text{Total Parameters:} \quad = 9{,}984$$

- **Computational Cost:**

$$\left.\begin{array}{l}\text{Depthwise:} \quad 32 \times 32 \times 64 \times 25 = 2,560,000 \\ \text{Pointwise:} \quad 32 \times 32 \times 128 \times 64 = 8,388,608\end{array}\right\} \quad \text{Total Parameters:} \quad = 10,948,608$$

Modified Layer 3: Depthwise Separable Convolution

- **Parameters:**

$$\left.\begin{array}{l}\text{Depthwise:} \quad 5 \times 5 \times 128 + 128 = 3,328 \\ \text{Pointwise:} \quad 1 \times 1 \times 128 \times 256 + 256 = 33,024\end{array}\right\} \quad \text{Total Parameters:} \quad = 36,352$$

- **Computational Cost:**

$$\left.\begin{array}{l}\text{Depthwise:} \quad 16 \times 16 \times 128 \times 25 = 819,200 \\ \text{Pointwise:} \quad 16 \times 16 \times 256 \times 128 = 8,388,608\end{array}\right\} \quad \text{Total Parameters:} \quad = 9,207,808$$

Comparative Analysis

- **Parameters:** Depthwise separable convolutions reduce the number of parameters.
- **Computational Cost:** Computational cost is slightly increased.

Overall: Depthwise is favored due to significant reduction in number of parameters.

Part 4.

Suppose we examine a classification problem with 200 classes. To this end, we add a **Flatten** layer along with a **Fully Connected** layer with **SoftMax** activation functions to the convolutional layers. Calculate the total number of parameters of this network for the convolutional architectures in sections **b** and **c**, and compare them with each other. What is the proportion of the Fully Connected layer in the number of parameters? Propose solutions to reduce the number of parameters and examine their impact.

Soloution

A. Calculating the Number of Parameters

Assuming sections **b** and **c** refer to two different convolutional architectures, we'll calculate the total number of parameters for each, including the added Flatten and Fully Connected layers.

a) Convolutional Layers Parameters: For a convolutional layer:

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

where: $K \times K = \text{Kernel size}$, $M = \#$ of input channels, $N = \#$ of output channels.

b) Fully Connected Layer Parameters: $(D \times 200) + 200$
where: D = Number of features from Flatten layer, 200 = Number of classes

B. Proportion of Fully Connected Layer Parameters

- for architecture b we have: $\text{Proportion}_{\text{FC}} = \frac{\text{Parameters}_{\text{FC}}}{\text{Total Parameters}} \times 100\%$
- for architecture c we have: $\text{Proportion}_{\text{FC}} = \frac{\text{Parameters}_{\text{FC}}}{\text{Total Parameters}} \times 100\%$

C. Suggestions to Reduce the Number of Parameters

- **Use Global Average Pooling:** Replace the Fully Connected layer with a Global Average Pooling layer to drastically reduce parameters.
- **Implement Depthwise Separable Convolutions:** Utilize Depthwise Separable Convolutions to minimize parameters in convolutional layers.
- **Apply Parameter Sharing:** Share weights across different layers to decrease the total number of unique parameters.
- **Use Sparse Connections:** Use sparse connections in the last layers instead of fully connected layer.

D. Impact of Reducing Parameters

Reducing the number of parameters leads to:

- **Reduced Model Size:** Makes the model more suitable for deployment on devices with limited memory.
- **Faster Computations:** Enhances inference speed due to fewer computations.

Question 2 (25 Points)

Examine DenseNet by addressing the following questions:

1. Compare ResNet's residual connections with DenseNet's dense connections.
2. Explain how DenseNet mitigates the vanishing gradient problem and its computational advantages.
3. Identify scenarios where DenseNet is recommended for a problem, providing a real-world example.
4. How could DenseNet be Adapted for multi-modal inputs (e.g., image and text), propose your suggested structure.

1. DenseNet vs ResNet

Soloution

ResNet's Residual Connections:

- Introduces shortcut connections that bypass one or more layers.
- Each block learns a residual mapping: $y = \mathcal{F}(\mathbf{x}, \mathbf{W}) + \mathbf{x}$, where $\mathcal{F}(\mathbf{x})$ is the desired mapping.
- Helps in training deeper networks by addressing the vanishing gradient problem.

DenseNet's Dense Connections:

- Connects each layer to every other layer in a feed-forward manner.
- Each layer receives feature maps from all preceding layers and passes its own feature maps to all subsequent layers.
- Enhances feature reuse, improves gradient flow, and reduces the number of parameters.

2. Mitigating Vanishing Gradient and Computational Advantages

Soloution

Reducing Vanishing Gradients:

- Dense connections provide a direct pathway for gradients from the loss function to each layer, preventing them from diminishing.
- These shorter gradient paths enable the effective training of deeper networks.

Computational Advantages:

- **Parameter Efficiency:** DenseNet uses fewer parameters than traditional CNNs by reusing features across layers.
- **Reduced Overfitting:** Promotes diverse feature representations, which helps in minimizing overfitting.

3. Recommended Use Cases and Real-World Example

Soloution

When to Use DenseNet:

- Situations requiring deep feature reuse and efficient parameter utilization.
- Tasks benefiting from improved gradient flow, such as image classification, segmentation, and detection.

Real-World Example:

- **Medical Image Analysis:** DenseNet is effective in tasks like tumor detection and organ segmentation due to its ability to capture intricate patterns with fewer parameters, ensuring efficient training on limited medical datasets.

4. Adapting DenseNet for Multi-Modal Inputs (Image and Text)

Soloution

Proposed Structure:

1. **Separate Feature Extractors:**
   - **Image Branch:** Utilize DenseNet to extract visual features from images.
   - **Text Branch:** Employ a DenseNet-like architecture or another suitable CNN to process textual data.
2. **Feature Fusion:**
   - Concatenate the feature maps from both branches.
   - Pass the combined features through additional dense layers for classification.

Justification:

- Dense connections in both branches facilitate efficient feature reuse and gradient flow.
- Feature fusion leverages complementary information from different modalities, enhancing model performance.

Diagram Explanation:

- The image and text inputs are processed through their respective DenseNet branches.
- Feature maps are concatenated and fed into fully connected layers for the final classification.

Question 3 (25 Points)

U-Net Architecture and Transposed Convolution
In the lesson, we became familiar with the U-Net architecture. In this question, we aim to first examine the main features of this architecture and its training, and finally, address the Transposed Convolution operation. In the image below, you can see an overview of the network architecture. For further familiarity, it is recommended to read the related article.

Part 1.

In this network, the encoder and decoder sections are connected using Skip Connections. Explain the reason and the impact of these connections based on the concepts discussed in the article.

Soloution

In an encoder-decoder neural network architecture, skip connections play a critical role in improving the model's performance. Below is an explanation of the reason for using skip connections and their impact:

Reason for Using Skip Connections

1. **Mitigation of Information Loss:** During the encoding process, features are compressed, leading to potential information loss. Skip connections help reintroduce the lost information to the decoding process, ensuring better detail retention.
2. **Facilitating Gradient Flow:** By providing shorter paths for gradient flow during backpropagation, skip connections help mitigate the vanishing gradient problem, leading to more stable and efficient training.
3. **Improved Learning Dynamics:** Skip connections make it easier for the network to learn identity mappings, ensuring that critical features are preserved without unnecessary distortion.

Impact of Skip Connections

1. **Enhanced Accuracy:** Skip connections improve the quality of the output by effectively combining high-level (semantic) and low-level (spatial) features.
2. **Better Convergence:** Networks with skip connections converge faster during training due to improved gradient flow and reduced risk of vanishing gradients.
3. **Reduction in Artifacts:** In tasks such as image reconstruction, skip connections minimize artifacts and ensure realistic and accurate outputs.
4. **Improved Structural Consistency:** By reintroducing low-level features from the encoder, the decoder can produce outputs that maintain structural integrity, such as preserving edges and boundaries in image segmentation tasks.

Conclusion

Skip connections are a critical component in encoder-decoder architectures. They address challenges such as information loss and vanishing gradients while improving accuracy, convergence, and structural consistency of the output. Their inclusion is essential for achieving superior model performance in tasks that require detailed and precise outputs.

Part 2.

For training the network, the **Random Deformation** technique has been utilized to increase the size of the training dataset. Based on the concepts discussed in the article, explain the implementation of this technique and its impact on the model's performance:

Soloution

How the Random Deformation Technique is Performed

1. **Data Augmentation:** Random deformation applies transformations such as scaling, rotation, translation, and elastic deformation to the existing training samples. This generates new variations of the data, simulating different real-world scenarios.
2. **Randomness:** The deformations are applied randomly to ensure diversity in the generated data, which prevents the model from overfitting to specific patterns in the original dataset.
3. **Consistency in Labels:** While augmenting the data, care is taken to ensure that the transformed data still corresponds to the correct labels, preserving the semantic integrity of the dataset.

Impact on Model Performance

1. **Improved Generalization:** By training the model on a more diverse dataset, it becomes more robust to variations in the input and generalizes better to unseen data.
2. **Reduced Overfitting:** Random deformation increases the effective size of the training dataset, reducing the risk of overfitting, especially when the original dataset is small.
3. **Enhanced Performance:** The model learns to handle real-world variations more effectively, leading to improved accuracy and reliability in practical applications.
4. **Training Efficiency:** By providing diverse data during training, the convergence of the model improves, as it encounters a wider range of scenarios within the training phase.

Conclusion

The **Random Deformation** technique is an effective method for data augmentation. It enhances the diversity of the training dataset, improving the model's ability to generalize, reducing overfitting, and ultimately resulting in better performance in real-world applications.

Part 3.

Consider the two matrices below. Using the given filter and input, perform the **Transposed Convolution** operation.

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

Note: In part (c), you must fully describe the steps to verify the final answer. You can also use PyTorch to check the result.

Soloution

Step 1: Flip the Filter

The filter is flipped horizontally and vertically:

$$\text{Flipped Filter} = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}$$

Step 2: Expand the Input Matrix for Transposed Convolution

The input matrix:

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

To perform transposed convolution, we expand the input by inserting zeros between rows and columns, resulting in:

$$\text{Expanded Input} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & 0 \\ 3 & 0 & 4 \end{bmatrix}$$

Step 3: Perform the Convolution Operation

We slide the flipped filter over the expanded input and compute the sum of element-wise products at each position.

Position-by-position Calculation:

1. Center the filter at the top-left corner of the expanded input. Multiply element-wise and sum:

$$1 \times 4 = 4 \quad \text{(Top-left of Output)}$$

2. Repeat for all positions, filling in the output matrix. The complete computations result in:

$$\text{Output} = \begin{bmatrix} 1 & 4 & 4 \\ 5 & 16 & 10 \\ 6 & 11 & 4 \end{bmatrix}$$

Step 4: Verification

The results can be verified programmatically using Python or other computational tools.

Question 4 (25 Points)

In the field of object detection, efficient and integrated algorithms such as YOLO (You Only Look Once) are designed based on single-pass processing of the entire image. Due to their ability for real-time and accurate object detection, they are widely used in real-world applications. In this exercise, we will review the basic versions of YOLO from the first to the third. To study the articles, 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 1.

Assume we are using a dataset for object detection, which includes 80 classes. To analyze the differences in the number of channels (depth) of the outputs in YOLOv1, YOLOv2, and YOLOv3, let us consider the following:

Soloution

- **YOLOv1:** Divides the image into an $S \times S$ grid, predicting B bounding boxes per cell, with each output including class probabilities, objectness score, and coordinates. The total depth is:

$$(B \times 5) + C = (2 \times 5) + 80 = 90$$

- **YOLOv2:** Introduces anchor boxes, predicting B bounding boxes per cell. Each output includes box attributes, object confidence, and class probabilities. The total depth is:

$$B \times (5 + C) = 5 \times (5 + 80) = 425$$

- **YOLOv3:** Improves on YOLOv2 by using multi-scale predictions at three resolutions. Each scale predicts B bounding boxes per cell, resulting in:

$$3 \times (B \times (5 + C)) = 3 \times (3 \times (5 + 80)) = 765$$

The increasing channel depth reflects YOLOv2's use of anchor boxes and YOLOv3's multi-scale predictions, enhancing their ability to detect objects at varying sizes and resolutions.

Part 2.

It is possible that some samples in the dataset are labeled as "background" and do not belong to any class. What solution has been proposed in YOLOv3 to address this issue?

Soloution

In YOLOv3, a new scoring system was introduced to handle overlapping labels in the dataset. Instead of forcing each object into a single class, it uses a multi-label classification approach, allowing objects to belong to multiple classes with confidence scores for each. This makes the model more flexible and better at dealing with overlapping labels.

▬ *Part 3.*

In the YOLO articles, what algorithm is used to avoid repetitive and multiple detections of objects?

Soloution

In the YOLO articles, the algorithm used to avoid repetitive and multiple detections of objects is **Non-Maximum Suppression (NMS)**.

1. **Thresholding:** Select all bounding boxes with confidence scores above a predefined threshold.
2. **Selecting the Primary Box:** Choose the bounding box with the highest confidence score as the primary detection.
3. **Suppressing Overlaps:** Remove all other bounding boxes that have a high overlap (measured by Intersection over Union, IoU) with the selected box.
4. **Repeating:** Continue the process for the remaining boxes until no high-confidence boxes are left.

This algorithm is essential for YOLO, as it eliminates redundant detections and ensures only one bounding box is retained for each object, improving both accuracy and efficiency.

▬ *Part 4.*

Explain why, unlike YOLOv1, YOLOv2 and YOLOv3 allow training on images of different sizes? How has this idea been implemented, and why is it beneficial?

Soloution

Unlike YOLOv1, YOLOv2 and YOLOv3 can train on images of different sizes thanks to their fully convolutional network (FCN) architecture. This makes the model resolution-independent, so it can handle varying image dimensions without needing fixed-size inputs.
Implementation:

- The network uses convolutional layers and global average pooling instead of fully connected layers, allowing it to adapt to different image sizes.
- During training, image sizes are randomly changed at regular intervals (e.g., every 10 iterations) using predefined dimensions like 320x320 or 416x416.

Benefits:

- It helps the model detect objects of various sizes by learning across different resolutions.
- This flexibility makes the model better suited for real-world applications where input images may vary in size.
- Training on multiple resolutions improves accuracy for both small and large objects.

Part 5.

In the YOLOv2 article, the two main problems related to using the anchor box have been mentioned. Present these issues and the solutions that have been proposed in the YOLOv2 article.

Soloution

Problem 1: Poorly Chosen Anchor Box Dimensions

- Anchor box dimensions that do not match the distribution of object sizes in the dataset can lead to inefficient training and inaccurate predictions.

Solution:

- Use k-means clustering on the dataset's bounding boxes to determine optimal anchor box dimensions. This ensures that the anchor boxes are better aligned with the actual object sizes, improving both convergence and detection accuracy.

Problem 2: Handling of Large and Small Objects

- A fixed set of anchor boxes may not perform well for detecting both very small and very large objects, leading to suboptimal predictions.

Solution:

- Introduce multiple anchor boxes of varying sizes to better capture the scale diversity in the dataset. This allows the model to detect objects across a wide range of sizes more effectively.

Part 6.

What are the key architectural differences in YOLOv3 compared to its predecessor?

Soloution

- **Multi-scale Predictions:** YOLOv3 detects objects at three different scales, making it better at identifying objects of various sizes.
- **Enhanced Feature Extraction:** It uses a deeper backbone network, Darknet-53, and a feature pyramid structure to improve detection, especially for smaller objects.
- **Refined Bounding Boxes:** YOLOv3 improves bounding box predictions with logistic regression and better objectness metrics.
- **Multi-label Classification:** Instead of softmax, YOLOv3 uses independent logistic regression, allowing it to handle objects belonging to multiple classes.