EBU4203 Introduction to AI – Week 3 Tutorial 2023

Q1: Explain static graph and dynamic graph used in deep learning frameworks.

Slides: P20 Programming model in the slides notes (listen to the class carefully)

The computation process can be view as a graph.

A static graph is one that defines the entire computation graph before performing the computation. When the data is obtained, it is calculated according to the defined calculation graph.

Dynamic graphs, on the other hand, generate computational graphs as they are computed, and the complete graph is known only when the computation is completed.

Q2: Can you explain the fundamental steps involved in processing an image in Computer Vision?

Slides: P98 steps to build a computer vision model

The fundamental steps in image processing in Computer Vision include

1. data collection (capturing an image)
2. data cleaning (noise reduction)
3. data preparation (resizing)
4. build and train the model (identifying relevant characteristics, choosing important features)
5. classification or recognition (making sense of the visual information).

Q3. Can you explain the concept of object recognition, and how is it used in Computer Vision applications?

Slides: P144 What is Objects Recognition

Object recognition is the process of identifying specific objects or instances within an image or video stream.

It is used in various applications, such as autonomous navigation (recognizing obstacles), augmented reality (overlaying digital information on real-world objects), and robotics (identifying objects for manipulation).

Q4. In this question, you will be asked to calculate the dimensions of the outputs after an operation in a Convolution Neural Network (CNN).

Slides: P76-78

a) Given an input image with dimensions 128x128 pixels, a 3x3 kernel, and a stride of 1, calculate the size of the output feature map.

- Width: (128 - 3 + 1) = 126 pixels
- Height: (128 - 3 + 1) = 126 pixels
- So, the output feature map is 126x126 pixels.

b) If you have a 16x16 feature map and apply max-pooling with a 2x2 window and a stride of 2, what will be the dimensions of the resulting pooled feature map?
the resulting pooled feature map will have dimensions:
   - Width: 16 / 2 = 8 pixels
   - Height: 16 / 2 = 8 pixels
   - So, the pooled feature map is 8x8 pixels.

Q5. Consider a grayscale input image with dimensions of 6x6 pixels and a 3x3 convolutional kernel. The values of the input image are as follows:

Input Image:
[[2, 1, 2, 0, 3, 1],
 [1, 3, 2, 1, 2, 0],
 [3, 0, 1, 1, 2, 2],
 [0, 2, 3, 1, 1, 3],
 [2, 2, 0, 2, 1, 2],
 [1, 1, 3, 0, 2, 3]]

a) **2D Convolution:** Calculate the result of applying the 3x3 convolutional kernel with the following weights:

Kernel Weights:
[[1, 0, -1],
 [0, 1, 0],
 [-1, 0, 1]]

Use valid padding and a stride of 1. Show the resulting feature map.

Slides: P76-78

To calculate the resulting feature map after applying the convolutional kernel with the given weights to the input image, we follow these steps:

1.Slide the 3x3 kernel over the input image with a stride of 1 and calculate the element-wise multiplication between the kernel and the corresponding region of the input

2.Sum the results of the element-wise multiplications to obtain the value for each location in the feature map.

3.Repeat this process for each position where the kernel can be applied without exceeding the dimensions of the input image

Here's the resulting feature map:

```
[[ 1,  4,  1,  2],

 [ 2,  2,  -1,  5],

 [ 2,  2,  1,  0],

 [ 1,  0,  3,  2]]
```

Steps:

1. For the top-left position of the kernel:

   Input Portion:

   ```
   [[ 2, 1, 2],
    [ 1, 3, 2],
    [ 3, 0, 1]]
   ```

   Element-wise multiplication with the kernel:

   ```
   [[2*1, 1*0, 2*(-1)],
    [1*0, 3*1, 2*0],
    [3*(-1), 0*0, 1*1]]
   ```

   Sum of the results:

   ```
   markdown
   (2*1 + 1*0 + 2*(-1) + 1*0 + 3*1 + 2*0 + 3*(-1) + 0*0 + 1*1) = 1
   ```

2. Move the 3x3 kernel one step to the right and repeat the process for the next position:

   Input Portion:

   ```
   [[ 1, 2, 0],
    [ 3, 2, 1],
    [ 0, 1, 1]]
   ```

   Element-wise multiplication with the kernel:

   ```
   [[1*1, 2*0, 0*(-1)],
    [3*0, 2*1, 1*0],
    [0*(-1), 1*0, 1*1]]
   ```

   Sum of the results:

   ```
   markdown
   (1*1 + 2*0 + 0*(-1) + 3*0 + 2*1 + 1*0 + 0*(-1) + 1*0 + 1*1) = 4
   ```

   The final output matrix will look something like this:

```
[[ 1,  4,  1,  2],
 [ 2,  2, -1,  5],
 [ 2,  2,  1,  0],
 [ 1,  0,  3,  2]]
```

This is the result of applying the 3x3 convolutional kernel with the given weights to the input image.

b) **Average Pooling:** For the feature map obtained in Part A, perform average pooling with a 2x2 pooling window and a stride of 2. Calculate the resulting pooled feature map and show its values.

To calculate average pooling, we take the average value within each 2x2 pooling window:
Average Pooled Feature Map:

```
[[ 2.25, 1.75],
 [ 1.25, 1.5]]
```

Steps:

Start : Input Portion:

```
[[ 1, 4],
 [ 2, 2]]

Average(1+4+2+2)/4=2.25
```

Move the 2x2 kernel 2 steps to the right again.

```
[[ 1, 2],
 [ -1, 5]]

Average(1+2-1+5)/4=1.75
```

Repeat, the result :

```
[[ 2.25, 1.75],
 [ 1.25, 1.5]]
```

c) **Pooling/Stride Effect:** How does adjusting the stride in convolution or pooling operations impact the spatial dimensions of the output feature map?
Increasing the stride in convolution or pooling reduces the spatial dimensions of the output. For example, with a stride of 2, we would move the convolutional kernel or pooling window by 2 pixels at a time, resulting in a smaller output feature map

d) **Padding Effect:** What is the purpose of adding zero-padding to an input image before convolution, and how does it affect the size of the output feature map?
Padding is used to control the size of the output feature map. Adding zero-padding to the input image before convolution preserves its spatial dimensions. Without padding, the output feature map is smaller than the input.

Q6. Consider a 3D volume with dimensions of 3x3x3 and a 2x2x2 3D convolutional kernel. The values of the 3D volume are as follows:

3D Volume:

```
[
  [
    [2, 1, 2],
    [1, 3, 2],
    [3, 0, 1]
  ],
  [
    [0, 2, 3],
    [1, 1, 3],
    [2, 2, 0]
  ],
  [
    [1, 1, 2],
    [2, 0, 2],
    [1, 3, 0]
  ]
]
```

Tips: This one is a little different. Refer to Slides: P80

a) **3D Convolution**: Calculate the result of applying the 2x2x2 3D convolutional kernel with the following weights:

3D Kernel Weights:

```
[  [    [1, 0],
        [0, 1]
  ],
  [    [-1, 0],
       [0, -1]
  ]
]
```

Use valid padding and a stride of 1 in all three dimensions (x, y, z). Show the resulting 3D feature volume.

To calculate the resulting 3D feature volume after applying the 3D convolutional kernel with the given weights to the 3D volume, we follow these steps:

Slide the 2x2x2 3D kernel over the 3D volume with a stride of 1 in all dimensions (x, y, z) and calculate the element-wise multiplication between the kernel and the corresponding region of the 3D volume.

Sum the results of the element-wise multiplications to obtain the value for each location in the 3D feature volume.

Repeat this process for each position where the 3D kernel can be applied without exceeding the dimensions of the 3D volume.

Here's the resulting 3D feature volume:

```
[  [     [4, 2],
         [-2, 3]
  ],
  [     [0, 2],
        [-2, 1]
  ]
]
```

Now, let's calculate the result step by step:

Place the 2x2x2 kernel at the top-left corner of the input 3D volume.

Input Portion:

```
[
  [
    [2, 1],
    [1, 3]
  ],
  [
    [0, 2],
    [1, 1]
  ]
],
[
  [
    [0, 2],
    [1, 1]
  ],
  [
    [1, 1],
    [2, 0]
  ]
]
```

Element-wise multiplication with the kernel:

```
[
  [
    [2*1, 1*0],
    [0*0, 3*1]
  ],
  [
    [0*(-1), 2*0],
    [1*0, 1*(-1)]
  ]
],
[
  [
    [0*1, 2*0],
    [1*0, 1*1]
  ],
  [
    [1*(-1), 1*0],
```

```
    [2*0, 0*(-1)]
  ]
]
```

Result:

```
[
  [
    [2*1+1*0+0*0+3*1=5]+[0*(-1)+2*0+1*0+1*(-1)=-1]=4
  ]
  [
    [0*1+2*0+1*0+1*1=1]+[1*(-1)+1*0+2*0+0*(-1)=-1]=0
  ]
]
```

- Place the 2x2x2 kernel at the top-right corner of the input 3D volume.

Input Portion:

```
[
  [
    [1, 2],
    [3, 2]
  ],
  [
    [2, 3],
    [1, 3]
  ]
],
[
  [
    [2, 3],
    [1, 3]
  ],
  [
    [1, 2],
    [0, 2]
  ]
]
```

Element-wise multiplication with the kernel:

```
[
  [
    [1*1, 2*0],
    [3*0, 2*1]
  ],
  [
    [2*(-1), 3*0],
    [1*0, 3*(-1)]
  ]
],
[
  [
    [2*1, 3*0],
```

```
    [1*0, 3*1]
  ],
  [
    [1*(-1), 2*0],
    [0*0, 2*(-1)]
  ]
]
```

Result:

```
[
  [
    [1*1+2*0+3*0+2*1=3]+[2*(-1)+3*0+1*0+3*(-1)=-5]=2
  ]
  [
    [2*1+3*0+1*0+3*1=5]+[1*(-1)+2*0+0*0+2*(-1)=-3]=2
  ]
]
```

Place this to down-left

Input Portion:

```
[
  [
    [1, 3],
    [3, 0]
  ],
  [
    [1, 1],
    [2, 2]
  ]
],
[
  [
    [1, 1],
    [2, 2]
  ],
  [
    [2, 0],
    [1, 3]
  ]
]
```

Element-wise multiplication with the kernel:

```
[
  [
    [1*1, 3*0],
    [3*0, 0*1]
  ],
  [
    [1*(-1), 1*0],
    [2*0, 2*(-1)]
  ]
],
[
```

```
  [
    [1*1, 1*0],
    [2*0, 2*1]
  ],
  [
    [2*(-1), 0*0],
    [1*0, 3*(-1)]
  ]
]
```

Result:

```
[
  [
    [1*1+3*0+3*0+0*1=1]+[1*(-1)+1*0+2*0+2*(-1)=-3]= -2
  ]
  [
    [1*1+1*0+2*0+2*1=3]+[2*(-1)+0*0+1*0+3*(-1)=-5]= -2
  ]
]
```

Place this to down-right

Input Portion:

```
[
  [
    [3, 2],
    [0, 1]
  ],
  [
    [1, 3],
    [2, 0]
  ]
],
[
  [
    [1, 3],
    [2, 0]
  ],
  [
    [0, 2],
    [3, 0]
  ]
]
```

Element-wise multiplication with the kernel:

```
[
  [
    [3*1, 2*0],
    [0*0, 1*1]
  ],
  [
    [1*(-1), 3*0],
    [2*0, 0*(-1)]
  ]
```

```
],
[
  [
    [1*1, 3*0],
    [2*0, 0*1]
  ],
  [
    [0*(-1), 2*0],
    [3*0, 0*(-1)]
  ]
]
```

Result:

```
[
  [
    [3*1+2*0+0*0+1*1=4]+[1*(-1)+3*0+2*0+0*(-1)=-1]=3
  ]
  [
    [1*1+3*0+2*0+0*1=1]+[0*(-1)+2*0+3*0+0*(-1)=0]=1
  ]
]
```

Here's the resulting 3D feature volume:

```
[  [     [4, 2],
         [-2, 3]
  ],
  [     [0, 2],
        [-2, 1]
  ]
]
```

b) **3D Max Pooling**: For the 3D feature volume obtained in Part A, perform max pooling with a 2x2x2 pooling window and a stride of 2 in all three dimensions (x, y, z). Calculate the resulting 3D pooled feature volume and show its values.
For the 3D feature volume obtained in Part A, we perform max pooling with a 2x2x2 pooling window and a stride of 2 in all dimensions (x, y, z). The values in the resulting 3D pooled feature volume are as follows:

3D Pooled Feature Volume:
[[[4]]]