Problem Set 3

Sharvita Paithankar

1)

(a) Image classification basically means that you are assigning a label to an entire image based on its content. A neural network for image classification looks at an image as an input and produces an output of a single class label or set of class probabilities. For example, a model could classify an image as "cat" or "dog." Object detection means the model is identifying and figuring out all the objects in an image given as an input. The neural network produces output bounding boxes and class labels for each object that was detected. For example, a model could detect cars and pedestrians in a street image. Instance segmentation is similar to object detection but each object is assigned a pixel-wise mask which helps with separating different objects of the same class.Semantic segmentation classifies each pixel in an image depending on its category without separating between different instances of the same class.

The similarity between them is that they all need to process images with neural networks and use CNNs for feature extraction. The Instance and semantic segmentations both output a pixel wise mask. The differences are that only image classification provides only a global label, while the others offer spatial information. Object detection provides bounding boxes but does not differentiate objects at the pixel level. Another difference is that the instance segmentation differentiates between individual objects, but semantic segmentation does not.

(b) Real-World Use Cases

1. Image Classification: Amazon and Flipkart use image classification to categorize clothing styles, detect fake products, and make visual search better
2. Object Detection: Tesla detecting vehicles and pedestrians in autonomous driving systems to make sure there is road safety.
3. Instance Segmentation: Snapchat and Instagram filters use instance segmentation to identify and overlay effects on specific facial features in real time.
4. Semantic Segmentation: Google maps using satellite image analysis for distinguishing roads, water bodies, mountains, etc.

---

2)

i. Parameters per layer: First layer (flattened input → hidden):

- Input size: 3 × 224 × 224 = 150528
- Hidden layer size: 150,528/2 = 75264
- Weights: 150528 × 75264 = 11329339392
- Biases: 75264

- Total: 11329414656

Second layer (hidden → output):

- Weights: 75264 × 100 = 7526400
- Biases: 100
- Total: 7526500

ii. Total parameters: 11329414656 + 7526500 = 11336941156

(b) Convolutional architecture:

i. Single 7×7 conv layer parameters:

- Weights: 7 × 7 × 3 × 3 = 441
- Biases: 3
- Total: 444 parameters

ii. Number of conv layers needed: Starting at 224×224, each layer reduces by 6 (7-1) 224 → 218 → 212 → ... → 8 Need 36 layers to reach 8×8

iii. Final FC layer parameters:

- Input: 3 × 8 × 8 = 192 (flattened)
- Output: 100
- Weights: 192 × 100 = 19200
- Biases: 100
- Total: 19300

iv. Total architecture parameters:

- Conv layers: 36 × 444 = 15984
- FC layer: 19300
- Total: 35284

3.

(a)

For regular convolution (1*1 stride, no padding):

Position (0,0):
Intermediate matrix:
[[2 0 -1] [1 -1 1]

[0 1 0] × [0   1 0]
[-2 0 1]   [1 -1 1]

Calculation:
(2×1 + 0×-1 + -1×1) +
(0×0 + 1×1 + 0×0) +
(-2×1 + 0×-1 + 1×1)
= (2 + 0 - 1) + (0 + 1 + 0) + (-2 + 0 + 1)
= 1

Position (0,1):

Intermediate matrix:
[[0 1 -1]   [1 -1 1]
 [0 0 0] × [0   1 0]
 [0 -1 1]   [1 -1 1]

Calculation:
(0×1 + 1×-1 + -1×1) +
(0×0 + 0×1 + 0×0) +
(0×1 + -1×-1 + 1×1)
= (0 - 1 - 1) + (0 + 0 + 0) + (0 + 1 + 1)
= 0

Position (1,0):

Intermediate matrix:
[[0 -1 0]   [1 -1 1]
 [0 0 0] × [0   1 0]
 [1 -1 0]   [1 -1 1]

Calculation:
(0×1 + -1×-1 + 0×1) +
(0×0 + 0×1 + 0×0) +
(1×1 + -1×-1 + 0×1)
= (0 + 1 + 0) + (0 + 0 + 0) + (1 + 1 + 0)
= -1

Position (1,1):

Intermediate matrix:

[[1 0 2]   [1 -1 1]
 [0 1 0] × [0   1 0]
 [1 0 -1]   [1 -1 1]

Calculation:
(1×1 + 0×-1 + 2×1) +
(0×0 + 1×1 + 0×0) +
(1×1 + 0×-1 + -1×1)
= (1 + 0 + 2) + (0 + 1 + 0) + (1 + 0 - 1)
= 4

Final Output Matrix:

[1 0]
[-1 4]

(b)

Intermediate result Before
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
Intermediate result After
[[2. -2. 2. 0. 0. 0.]
 [0. 2. 0. 0. 0. 0.]
 [2. -2. 2. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]

Input Matrix Values: 2 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 2. 0. 0. 0.]

```
 [ 0.  2.  0.  0.  0.  0.]
 [ 2. -2.  2.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]]


Input Matrix Values: 0 * [[ 1 -1  1]
 [ 0  1  0]
 [ 1 -1  1]]



Intermediate result After
[[ 2. -2.  1.  1. -1.  0.]
 [ 0.  2.  0. -1.  0.  0.]
 [ 2. -2.  1.  1. -1.  0.]
 [ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]]


Input Matrix Values: -1 * [[ 1 -1  1]
 [ 0  1  0]
 [ 1 -1  1]]



Intermediate result After
[[ 2. -2.  1.  0.  0. -1.]
 [ 0.  2.  0. -1. -1.  0.]
 [ 2. -2.  1.  0.  0. -1.]
 [ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]]


Input Matrix Values: -1 * [[ 1 -1  1]
 [ 0  1  0]
 [ 1 -1  1]]
```

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 2. 0. -1. -1. 0.]
 [2. -2. 1. 0. 0. -1.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]

Input Matrix Values: 0 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 0. -1. 0.]
 [2. -2. 2. 0. 0. -1.]
 [0. 1. -1. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]

Input Matrix Values: 1 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 0. -1. 0.]
 [2. -2. 2. 0. 0. -1.]
 [0. 1. -1. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]

Input Matrix Values: 0 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 2. -3. 2.]
 [2. -2. 2. 0. 2. -1.]
 [0. 1. -1. 3. -2. 2.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]

Input Matrix Values: 2 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 2. -3. 2.]
 [0. 0. 0. 0. 2. -1.]
 [0. -1. -1. 3. -2. 2.]
 [-2. 2. -2. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]

Input Matrix Values: -2 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 2. -3. 2.]
 [0. 0. 0. 0. 2. -1.]
 [0. -1. -1. 3. -2. 2.]
 [-2. 2. -2. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]

Input Matrix Values: 0 * [[1 -1 1]
 [0 1 0]

[1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 2. -3. 2.]
 [0. 0. 1. -1. 3. -1.]
 [0. -1. -1. 4. -2. 2.]
 [-2. 2. -1. -1. 1. 0.]
 [0. 0. 0. 0. 0. 0.]]

Input Matrix Values: 1 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 2. -3. 2.]
 [0. 0. 1. 0. 2. 0.]
 [0. -1. -1. 4. -1. 2.]
 [-2. 2. -1. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0.]]

Input Matrix Values: 1 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 2. -3. 2.]
 [0. 0. 1. 0. 2. 0.]
 [1. -2. 0. 4. -1. 2.]
 [-2. 3. -1. 0. 0. 1.]
 [1. -1. 1. 0. 0. 0.]]

Input Matrix Values: 1 * [[1 -1 1]
 [0 1 0]

[1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 2. -3. 2.]
 [0. 0. 1. 0. 2. 0.]
 [1. -1. -1. 5. -1. 2.]
 [-2. 3. 0. 0. 0. 1.]
 [1. 0. 0. 1. 0. 0.]]

Input Matrix Values: 1 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 2. -3. 2.]
 [0. 0. 1. 0. 2. 0.]
 [1. -1. -1. 5. -1. 2.]
 [-2. 3. 0. 0. 0. 1.]
 [1. 0. 0. 1. 0. 0.]]

Input Matrix Values: 0 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

Intermediate result After
[[2. -2. 1. 0. 0. -1.]
 [0. 3. -1. 2. -3. 2.]
 [0. 0. 1. 0. 2. 0.]
 [1. -1. -1. 4. 0. 1.]
 [-2. 3. 0. 0. -1. 1.]
 [1. 0. 0. 0. 1. -1.]]

Input Matrix Values: -1 * [[1 -1 1]
 [0 1 0]
 [1 -1 1]]

4.

1. Down-sampling Stage:

Channel progression: [3→16→32→64→128→256→256...]

Using 3×3 filters, calculating parameters for each layer type:

First five layers:

- 3→16: (3×3×3×16) + 16 = 448
- 16→32: (3×3×16×32) + 32 = 4640
- 32→64: (3×3×32×64) + 64 = 18496
- 64→128: (3×3×64×128) + 128 = 73856
- 128→256: (3×3×128×256) + 256 = 295168
- Remaining 256→256: (3×3×256×256) + 256 = 590080 each

Number of layers needed to reach 16×16: 224→222→220→218→...→16

Total 99 layers at 256 channels

Total down-sampling parameters: 448 + 4640 + 18496 + 73856 + 295168 + (590080 × 99) = 58810528

2. Up-sampling Stage:

Using 9×9 filters:

- 256→256: (9×9×256×256) + 256 = 5308672 per layer
- 256→128: (9×9×256×128) + 128 = 2654336
- 128→64: (9×9×128×64) + 64 = 663616
- 64→32: (9×9×64×32) + 32 = 165920
- 32→16: (9×9×32×16) + 16 = 41488
- 16→3: (9×9×16×3) + 3 = 3891

Need 21 layers of 256→256 to reach spatial size of 224×224

Total up-sampling parameters: (5308672 × 21) + 2654336 + 663616 + 165920 + 41488 + 3891 = 115011363

3. Total Model Parameters: 58810528 + 115011363 = 173821891

The up-sampling stage has significantly more parameters due to the larger 9×9 filters compared to the 3×3 filters in down-sampling.