Consider a CNN composed of three convolutional layers, each with 3 × 3 kernels, a stride of 2, and SAME padding. The lowest layer outputs 100 feature maps, the middle one outputs 200, and the top one outputs 400. The input images are RGB images of 200 × 300 pixels. Then address the following questions in a text file.

1. What is the total number of parameters in the CNN?
2. If you are using 64-bit floats, at least how much RAM will this network require when making a prediction for a single instance?
3. What about when training on a mini-batch of 50 images?
4. Is it okay to initialise all the weights to the same value? And, is it okay to initialise the bias terms to 0?
5. Why would you want to add a max pooling layer rather than a convolutional layer with the same stride?

Answers

1. **Total Number of Parameters**: To calculate the total number of parameters in the CNN, we need to consider the parameters in each convolutional layer. Each convolutional layer has 3 × 3 kernels, and the number of parameters in each kernel is equal to the product of the kernel size (3 × 3), the number of input channels (3 for RGB images), and the number of output channels. There is also a bias term.

For the lowest layer:

* + Number of parameters per kernel = (3 × 3 × 3 + 1) = 28 (weights + bias)
  + Total number of parameters = (3 × 3 × 28) × 100 = 25200

For the middle layer:

* + Number of parameters per kernel = (3 × 3 × 100 + 1) = 901 (weights + bias)
  + Total number of parameters = (3 × 3 × 901) × 200 = 1621800

For the top layer:

* + Number of parameters per kernel = (3 × 3 × 200 + 1) = 1801 (weights + bias)
  + Total number of parameters = (3 × 3 × 1801) × 400 = 19447600

Total number of parameters in the CNN = 25200 + 1621800 + 19447600 = 21066400 parameters.

1. **RAM Required for Prediction with Single Instance**:

To calculate the RAM required for prediction with a single instance, we need to consider the size of the input image, the size of the output feature maps, and the size of each parameter (assuming 64-bit floats).

1 - Input image size = 200 × 300 pixels × 3 channels (RGB)

- Total size of the input image = 200 × 300 × 3 × 8 bytes (64-bit floats) = 1440000 bytes

2. For each convolutional layer, we need to calculate the size of the output feature maps.

- Lowest layer output size = 100 × 150 × 100 feature maps

- Middle layer output size = 50 × 75 × 200 feature maps

- Top layer output size = 25 × 38 × 400 feature maps

3. We already calculated the total number of parameters for each layer:

- Lowest layer: 25200 parameters

- Middle layer: 1621800 parameters

- Top layer: 19447600 parameters

Size of parameters for each layer:

- Lowest layer: 25200 × 8 bytes (64-bit floats) = 201600 bytes

- Middle layer: 1621800 × 8 bytes = 12974400 bytes

- Top layer: 19447600 × 8 bytes = 155580800 bytes

4. Total RAM required for prediction with a single instance:

- For input image: 1440000 bytes

- For output feature maps:

- Lowest layer: 100 × 150 × 100 × 8 bytes = 12000000 bytes

- Middle layer: 50 × 75 × 200 × 8 bytes = 6000000 bytes

- Top layer: 25 × 38 × 400 × 8 bytes = 7600000 bytes

- For parameters:

- Lowest layer: 201600 bytes

- Middle layer: 12974400 bytes

- Top layer: 155580800 bytes

Total RAM required = Input Image Size + Output Feature Maps Size + Parameters Size

Total RAM required = 1440000 + (12000000 + 6000000 + 7600000) + (201600 + 12974400 + 155580800) bytes

≈ 200 MB (approximately)

So, at least 200 megabytes of RAM will be required when making a prediction for a single instance using this network.

1. **RAM Required for Training on Mini-Batch of 50 Images**: To calculate the RAM required for training on a mini-batch of 50 images, we need to consider the RAM required for each image as well as the RAM required for storing gradients during backpropagation.
2. **Initializing Weights and Biases**: Initializing all weights to the same value might hinder the network's ability to learn complex patterns as it restricts the initial diversity of the weights. However, initializing biases to 0 is a common practice and generally does not cause issues.
3. **Max Pooling vs. Convolutional Layer**: Max pooling layers are used to downsample feature maps, reducing the computational load and controlling overfitting by providing a form of translation invariance. Using a convolutional layer with the same stride would reduce the spatial resolution without capturing the most salient features within each receptive field, making it less effective for downsampling. Therefore, max pooling layers are preferred for downsampling in CNN architectures.