1. What are the advantages of a CNN over a fully connected DNN for image classification?

Advantages of a CNN over a fully connected DNN for image classification:

* Local receptive fields: CNNs use convolutional layers that focus on local regions, capturing spatial hierarchies in the image, which is important for image features.
* Parameter sharing: CNNs share weights among neurons in a layer, reducing the number of parameters and enabling better generalization.
* Translation invariance: CNNs are capable of recognizing features irrespective of their position in the image, making them well-suited for detecting patterns in images.

1. 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.

What is the total number of parameters in the CNN? If we are using 32-bit floats, at least how much RAM will this network require when making a prediction for a single instance? What about when training on a mini-batch of 50 images?

To calculate the total number of parameters in the CNN:

* The first convolutional layer has 3 \* 3 \* 3 (kernel size \* input channels) parameters per filter and 100 filters.
* The second convolutional layer has 3 \* 3 \* 100 parameters per filter and 200 filters.
* The third convolutional layer has 3 \* 3 \* 200 parameters per filter and 400 filters.
* Additionally, there are biases associated with each filter.

Total parameters = (3 \* 3 \* 3 \* 100) + (3 \* 3 \* 100 \* 200) + (3 \* 3 \* 200 \* 400) + (100 + 200 + 400)

To estimate RAM usage for a single instance:

* 32-bit floats require 4 bytes per parameter.
* Multiply the total parameters by 4 bytes to estimate RAM usage.

For a mini-batch of 50 images, multiply the RAM usage for a single instance by 50.

1. If your GPU runs out of memory while training a CNN, what are five things you could try to solve the problem?

Solutions to GPU running out of memory while training a CNN:

* Reduce batch size: Use smaller mini-batches during training.
* Reduce model size: Decrease the number of layers or neurons in each layer.
* Use mixed-precision training: Utilize lower precision (e.g., float16) for model parameters.
* Gradient accumulation: Accumulate gradients over mini-batches to reduce memory consumption.
* Memory-efficient architectures: Choose architectures designed for memory efficiency, like MobileNet.

1. Why would you want to add a max pooling layer rather than a convolutional layer with the same stride?

Max pooling layers downsample the spatial dimensions of feature maps while retaining important features, reducing computational load. Convolutional layers with the same stride do not reduce spatial dimensions.

1. When would you want to add a local response normalization layer?

Local response normalization layers were used in some older architectures to introduce a form of lateral inhibition, which could enhance contrast and help in the detection of edges and patterns. However, they are less commonly used in modern CNN architectures.

1. Can you name the main innovations in AlexNet, compared to LeNet-5? What about the main innovations in GoogLeNet, ResNet, SENet, and Xception?

Innovations in mentioned architectures:

* AlexNet: Introduced deeper networks, ReLU activations, dropout, and GPU acceleration.
* GoogLeNet: Introduced inception modules for network depth and efficiency.
* ResNet: Introduced residual connections to ease training of very deep networks.
* SENet: Introduced squeeze-and-excitation blocks to adaptively recalibrate feature maps.
* Xception: Introduced depthwise separable convolutions for efficiency.

1. What is a fully convolutional network? How can you convert a dense layer into a convolutional layer?

A fully convolutional network (FCN) is designed for tasks like semantic segmentation. To convert a dense layer into a convolutional layer, set the dense layer's weights as the kernel of the convolutional layer, and reshape the layer to have 1x1 spatial dimensions.

1. What is the main technical difficulty of semantic segmentation?

The main technical difficulty of semantic segmentation is achieving pixel-level classification while preserving spatial information. This requires dealing with class imbalance, handling large receptive fields, and ensuring precise localization of object boundaries.

1. Build your own CNN from scratch and try to achieve the highest possible accuracy on MNIST.
2. Use transfer learning for large image classification, going through these steps:
   1. Create a training set containing at least 100 images per class. For example, you could classify your own pictures based on the location (beach, mountain, city, etc.), or alternatively you can use an existing dataset (e.g., from TensorFlow Datasets).
   2. Split it into a training set, a validation set, and a test set.
   3. Build the input pipeline, including the appropriate preprocessing operations, and optionally add data augmentation.
   4. Fine-tune a pretrained model on this dataset.