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

**Local Connectivity: CNNs use local connectivity through convolutional layers, which allows them to learn spatial hierarchies of features. Fully connected DNNs lack this property and treat each input feature independently.**

**Parameter Efficiency: CNNs share weights across spatial locations, reducing the number of parameters compared to fully connected DNNs.**

**Translation Invariance: CNNs are capable of learning features that are translationally invariant, making them suitable for tasks like image recognition.**

**Hierarchical Feature Extraction: CNNs use multiple layers with increasing receptive fields to learn complex features hierarchically.**

**Weight Sharing: Weight sharing in convolutional layers leads to feature reuse and generalization, making CNNs robust to variations in input.**

2. Consider a CNN composed of three convolutional layers, each with 3 × 3 kernels, a stride of

2, and &quot;same&quot; 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?

**Each convolutional layer has 3 × 3 × number\_of\_input\_channels × number\_of\_output\_feature\_maps parameters.**

**Layer 1: 3 × 3 × 3 × 100 = 2700 parameters**

**Layer 2: 3 × 3 × 100 × 200 = 180,000 parameters**

**Layer 3: 3 × 3 × 200 × 400 = 720,000 parameters**

**Total parameters in the CNN = 2700 + 180,000 + 720,000 = 902,700 parameters.**

**When using 32-bit floats, the network would require approximately 3.4 MB of RAM to make a prediction for a single instance. For a mini-batch of 50 images, it would require approximately 171 MB of RAM.**

3. If your GPU runs out of memory while training a CNN, what are five things you could try to

solve the problem?

**Reduce Batch Size: Decreasing the batch size reduces the memory requirement during training. However, this might also affect the convergence speed.**

**Decrease Model Complexity: Reduce the number of layers, neurons, or parameters in the model to make it more memory-efficient.**

**Use Mixed Precision Training: Employ mixed precision training, which uses 16-bit floating-point numbers for model weights and activations to reduce memory usage while maintaining training stability.**

**Data Augmentation: Apply data augmentation techniques to generate additional training samples on-the-fly, reducing the need to load all data at once.**

**Gradient Checkpointing: Implement gradient checkpointing techniques to trade computation time for memory usage by recomputing intermediate activations during backpropagation.**

4. Why would you want to add a max pooling layer rather than a convolutional layer with the

same stride?

**Max-pooling layers are used to downsample the spatial dimensions of feature maps while retaining the most important information. Adding a convolutional layer with the same stride as max-pooling would reduce spatial dimensions but without the downsampling effect.**

**Max-pooling introduces translation invariance, reduces the computational burden, and helps the network focus on the most salient features by selecting the maximum value within a region. Convolutional layers with the same stride do not offer these benefits and may lead to overfitting due to a large number of parameters.**

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

**Local Response Normalization (LRN) layers were previously used in some CNN architectures, like AlexNet, to provide local contrast normalization. They were intended to enhance the robustness and generalization of models.**

**LRN layers are less commonly used today because other normalization techniques like Batch Normalization are more effective. However, you might consider adding an LRN layer in cases where you specifically want to replicate older architectures for compatibility reasons or perform experiments to evaluate its impact.**

6. Can you name the main innovations in AlexNet, compared to LeNet-5? What about the main

innovations in GoogLeNet, ResNet, SENet, and Xception?

**AlexNet:**

**Introduced deep CNNs for image classification.**

**Used ReLU activation functions and dropout for regularization.**

**Utilized local response normalization.**

**GoogLeNet (Inception):**

**Introduced the concept of inception modules with multiple filter sizes.**

**Utilized 1x1 convolutions for dimensionality reduction.**

**ResNet (Residual Networks):**

**Introduced residual connections that enable the training of very deep networks.**

**Overcame the vanishing gradient problem by allowing gradients to flow directly through shortcuts.**

**SENet (Squeeze-and-Excitation Networks):**

**Introduced attention mechanisms to CNNs.**

**Adaptively recalibrated channel-wise feature responses.**

**Xception:**

**Utilized depthwise separable convolutions for efficient and effective feature extraction.**

**Achieved competitive performance with fewer parameters.**

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

convolutional layer?

**A Fully Convolutional Network (FCN) is a neural network architecture designed for dense prediction tasks, such as semantic segmentation. It replaces fully connected layers with convolutional layers to allow for variable-sized input images and produce spatially dense output predictions.**

**To convert a dense layer into a convolutional layer, you can set the dense layer's weights as the kernel weights of the convolutional layer, and the output size of the convolutional layer should match the desired output shape. This allows the convolutional layer to slide over the entire input spatially and produce dense predictions.**

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

**The main technical difficulty of semantic segmentation is the need for pixel-level classification of objects and regions within an image. This requires models to understand and distinguish between fine-grained object boundaries, handle occlusions, and produce dense predictions. It is a more complex and computationally demanding task compared to image classification or object detection.**

9. Build your own CNN from scratch and try to achieve the highest possible accuracy on MNIST.

**Building a CNN from scratch for MNIST classification involves defining a convolutional architecture, specifying activation functions, and training the model with appropriate hyperparameters. Achieving the highest accuracy would require experimentation and hyperparameter tuning.**

10. Use transfer learning for large image classification, going through these steps:

a. 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).

b. Split it into a training set, a validation set, and a test set.

c. Build the input pipeline, including the appropriate preprocessing operations, and

optionally add data augmentation.

d. Fine-tune a pretrained model on this dataset.

**a. Create a training set containing at least 100 images per class. Collect or curate a dataset containing images belonging to different classes (e.g., locations like beach, mountain, city).**

**b. Split it into a training set, a validation set, and a test set. Divide the dataset into three parts for training, model evaluation, and final testing.**

**c. Build the input pipeline, including the appropriate preprocessing operations, and optionally add data augmentation. Create a data input pipeline using TensorFlow's tf.data or other data loading techniques. Apply preprocessing and augmentation as needed.**

**d. Fine-tune a pretrained model on this dataset. Choose a suitable pretrained model (e.g., from TensorFlow Hub or a pre-trained TensorFlow model) and fine-tune it on your custom dataset. Adjust the final classification layer for the number of classes in your dataset.**