1. What are the advantages of a CNN for image classification over a completely linked DNN?

**- Local Connectivity: CNNs use convolutional layers that are locally connected, meaning each neuron is only connected to a small region of the input image. This is more biologically inspired and helps capture local patterns and features in images efficiently, while fully connected DNNs connect every neuron to every other neuron, making them less suitable for image data due to the high dimensionality.**

**- Parameter Sharing: CNNs share parameters across the entire image, reducing the number of parameters and enabling the model to learn translation-invariant features. Fully connected DNNs have a much larger number of parameters, making them prone to overfitting.**

**- Hierarchical Features: CNNs typically consist of multiple layers with increasing abstraction levels, allowing them to learn hierarchical features from simple edges and textures to more complex objects and structures. Fully connected DNNs may struggle with this hierarchical feature learning.**

**- Spatial Invariance: CNNs are capable of recognizing patterns regardless of their spatial position in the image, thanks to pooling layers and convolution operations. Fully connected DNNs do not inherently possess this property.**

2. Consider a CNN with three convolutional layers, each of which has three kernels, a stride of two,

and SAME padding. The bottom layer generates 100 function maps, the middle layer 200, and the

top layer 400. RGB images with a size of 200 x 300 pixels are used as input. How many criteria does

the CNN have in total? How much RAM would this network need when making a single instance

prediction if we&#39;re using 32-bit floats? What if you were to practice on a batch of 50 images? -

**Each convolutional layer generates one feature map for each kernel.**

**- The bottom layer generates 100 feature maps, the middle layer 200, and the top layer 400.**

**- To calculate the total number of parameters, you need to consider the weights for each kernel, which depends on the size of the kernels and the number of input channels (3 for RGB images).**

**- Assuming each kernel in the convolutional layers is 3x3 in size, the calculation for each layer would be as follows:**

**- Bottom layer: (3 \* 3 \* 3 + 1) \* 100 = 2800 parameters (adding 1 for bias)**

**- Middle layer: (3 \* 3 \* 100 + 1) \* 200 = 180200 parameters**

**- Top layer: (3 \* 3 \* 200 + 1) \* 400 = 720400 parameters**

**- The total number of parameters in the CNN is the sum of parameters in all layers: 2800 + 180200 + 720400 = 903400 parameters.**

**- To calculate the RAM needed for a single inference with 32-bit floats, you would multiply the number of parameters by 4 (32 bits = 4 bytes): 903400 \* 4 = 3,613,600 bytes or approximately 3.45 MB.**

**- For a batch of 50 images, you would need to multiply the RAM requirement by the batch size: 3.45 MB \* 50 = 172.5 MB.**

3. What are five things you might do to fix the problem if your GPU runs out of memory while

training a CNN?

**- Reduce Batch Size: Decreasing the batch size during training can lower GPU memory usage, but it might lead to slower convergence.**

**- Use Smaller Models: If possible, use a smaller architecture with fewer layers and parameters to reduce memory consumption.**

**- Data Augmentation: Apply data augmentation techniques like random cropping, rotation, and flipping to generate additional training data without increasing the model's size.**

**- Gradient Checkpointing: Implement gradient checkpointing, a technique that trades off computation time for memory, allowing you to train larger models with limited GPU memory.**

**- Mixed Precision Training: Use mixed precision training, which combines 16-bit and 32-bit precision for training, reducing memory usage without significant loss of accuracy.**

4. Why would you use a max pooling layer instead with a convolutional layer of the same stride?

**- Max pooling layers are used in CNNs to downsample feature maps spatially and reduce dimensionality. They are typically employed after convolutional layers to reduce computational complexity.**

**- A max pooling layer with the same stride as a convolutional layer helps capture the most important information in the feature maps while discarding less relevant details. This reduces the risk of overfitting and improves the model's ability to generalize to new data.**

**- Max pooling introduces translation invariance and reduces the sensitivity of the network to small variations in object position within the receptive field.**

5. When would a local response normalization layer be useful?

**- LRN layers were popularized in early CNN architectures like AlexNet. They are useful for enhancing the model's ability to discriminate between features by normalizing the responses of neurons within a local neighborhood.**

**- LRN layers help suppress responses that are uniformly large within a neighborhood, emphasizing relative activations. This can be beneficial when you want to highlight specific features or suppress noise.**

**- LRN layers are less commonly used in modern CNN architectures, as other normalization techniques like batch normalization have become more popular.**

6. In comparison to LeNet-5, what are the main innovations in AlexNet? What about GoogLeNet and

ResNet&#39;s core innovations?

**- AlexNet: Innovations in AlexNet included the use of multiple GPUs for training, ReLU activation functions, dropout for regularization, and local response normalization. It won the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and significantly advanced deep learning for computer vision.**

**- GoogLeNet (Inception): GoogLeNet introduced the concept of inception modules with multiple filter sizes to capture features at different scales. It also used global average pooling instead of fully connected layers at the end, reducing the number of parameters.**

**- ResNet: ResNet introduced residual connections, allowing for very deep networks. These skip connections mitigate the vanishing gradient problem, enabling training of extremely deep networks with hundreds of layers.**

7. On MNIST, build your own CNN and strive to achieve the best possible accuracy.

**Building a CNN for MNIST:**

**- To build your own CNN for MNIST, you can use libraries like TensorFlow or PyTorch. The architecture typically consists of convolutional layers, pooling layers, fully connected layers, and softmax output for classification.**

**- You can experiment with different architectures, activation functions, dropout rates, and optimization techniques to achieve the best accuracy on the MNIST dataset.**

8. Using Inception v3 to classify broad images. a.

Images of different animals can be downloaded. Load them in Python using the

matplotlib.image.mpimg.imread() or scipy.misc.imread() functions, for example. Resize and/or crop

them to 299 x 299 pixels, and make sure they only have three channels (RGB) and no transparency.

The photos used to train the Inception model were preprocessed to have values ranging from -1.0 to

1.0, so make sure yours do as well.

**- To use Inception v3 for image classification, you would typically follow these steps:**

**a. Load and preprocess your images to match the format expected by Inception v3 (299x299 pixels, RGB, values in the range [-1.0, 1.0]).**

**b. Load the pre-trained Inception v3 model and freeze all layers up to the bottleneck layer.**

**c. Replace the output layer with a new one that has the appropriate number of neurons for your classification task.**

**d. Train the modified model on your dataset, fine-tuning the last layers while keeping the pre-trained weights fixed.**

**e. Evaluate the model's performance on a test set.**

9. Large-scale image recognition using transfer learning.

a. Make a training set of at least 100 images for each class. You might, for example, identify your

own photos based on their position (beach, mountain, area, etc.) or use an existing dataset, such as

the flowers dataset or MIT&#39;s places dataset (requires registration, and it is huge).

b. Create a preprocessing phase that resizes and crops the image to 299 x 299 pixels while also

adding some randomness for data augmentation.

c. Using the previously trained Inception v3 model, freeze all layers up to the bottleneck layer (the

last layer before output layer) and replace output layer with appropriate number of outputs for

your new classification task (e.g., the flowers dataset has five mutually exclusive classes so the

output layer must have five neurons and use softmax activation function).

d. Separate the data into two sets: a training and a test set. The training set is used to train the

model, and the test set is used to evaluate it.

**Large-Scale Image Recognition Using Transfer Learning:**

**a. Collect or create a training set with at least 100 images for each class you want to classify. Ensure your images are labeled.**

**b. Create a preprocessing pipeline that resizes and crops images to 299x299 pixels while applying random transformations for data augmentation.**

**c. Load the pre-trained Inception v3 model, freeze all layers up to the bottleneck layer (before the output layer), and replace the output layer with the appropriate number of neurons for your classification task.**

**d. Split your dataset into a training set and a test set.**

**e. Train the modified Inception v3 model on the training set, fine-tuning the last layers.**

**f. Evaluate the model's performance on the test set using appropriate metrics.**

**g. We can further fine-tune hyperparameters and experiment with different augmentation techniques to improve performance.**