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

A CNN has several advantages over a fully connected deep neural network (DNN) for image classification tasks.

One of the main advantages of a CNN is that it uses a special type of layer called a convolutional layer, which is able to automatically learn spatial hierarchies of features from the input data. This means that a CNN can learn to recognize complex patterns in images by breaking them down into smaller and simpler patterns, such as edges, corners, and textures. This is in contrast to a fully connected DNN, which does not have this ability and must be trained on all possible combinations of the input features.

Another advantage of a CNN is that it is able to reduce the number of parameters that need to be trained in the network, compared to a fully connected DNN. This is because a CNN uses a process called parameter sharing, where the same set of weights is used for multiple patches of the input image. This allows a CNN to learn more efficiently from the data, and makes it more resistant to overfitting.

Additionally, CNNs are able to take advantage of the spatial structure of the input data, such as the two-dimensional nature of an image. This allows them to learn spatial hierarchies of features, and to make more accurate predictions about the objects in an image. This is not possible with a fully connected DNN, which treats all input features as independent of each other.

Overall, the use of convolutional layers, parameter sharing, and the ability to learn spatial hierarchies of features make CNNs well-suited for image classification tasks, and give them several advantages over a fully connected DNN.

1. **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're using 32-bit floats? What if you were to practice on a batch of 50 images?**

* In general, a CNN with three convolutional layers and three kernels per layer would have a relatively small number of parameters. Each convolutional layer applies its filters to the output of the previous layer, and produces a set of feature maps as its output. The number of feature maps generated by each layer depends on the number of filters used in that layer.

In the case of the CNN described in the question, the bottom layer generates 100 feature maps, the middle layer generates 200, and the top layer generates 400. However, without knowing the size of the convolutional filters or the size of the input data, it is not possible to determine the total number of parameters in the network.

As for the amount of memory needed by the CNN when making a single instance prediction, this would depend on the size of the input data and the size of the intermediate tensors generated by the network. If the input data is 200x300 RGB images, and the network uses 32-bit floating point values, then each image would require 200x300x3x4 = 36,000,000 bytes of memory to store. If the network generates intermediate tensors of similar size, then it would require a similar amount of memory to make a prediction.

* If the network is making predictions on a batch of 50 images, then it would require 50 times as much memory as making a prediction on a single image. This means that the network would require 36,000,000 x 50 = 1,800,000,000 bytes, or approximately 1.8 GB of memory to make predictions on a batch of 50 images. However, this is a rough estimate and the actual memory usage may vary depending on the specific architecture of the network.

1. **What are five things you might do to fix the problem if your GPU runs out of memory while training a CNN?**

If a GPU runs out of memory while training a convolutional neural network (CNN), there are several steps that can be taken to fix the problem. Here are five potential solutions:

**Reduce the batch size:** One way to reduce the amount of memory needed by a CNN is to decrease the batch size, which is the number of samples processed by the network at each iteration of training. This will decrease the amount of memory required to store intermediate tensors in the network, and can help avoid running out of memory.

**Use a smaller network architecture:** Another way to reduce memory usage is to use a smaller network architecture. This could involve using fewer convolutional filters in each layer, or using smaller filter sizes. This will decrease the number of parameters in the network, which will in turn reduce the amount of memory needed to store the network weights and intermediate tensors.

**Use a lower precision data type:** By default, most deep learning frameworks use 32-bit floating point values to store data in a CNN. However, using a lower precision data type, such as 16-bit floating point values or 8-bit integer values, can significantly reduce the amount of memory needed to store the data. This can be a useful strategy for reducing memory usage when training a CNN.

**Use a different optimizer:** The choice of optimizer can also affect the amount of memory needed to train a CNN. Some optimizers, such as Adam, can require more memory than others, such as SGD. If memory is an issue, switching to a more memory-efficient optimizer can help avoid running out of memory.

**Increase the amount of available GPU memory:** If none of the above strategies are sufficient to fix the problem, the only remaining option is to increase the amount of available GPU memory. This can be done by using a GPU with more memory, or by using multiple GPUs with a distributed training approach. This will increase the amount of memory available for training the CNN, and can help avoid running out of memory.

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

One reason to use a max pooling layer instead of a convolutional layer with the same stride is that max pooling can help to reduce overfitting in the network. This is because max pooling effectively performs feature selection, by only retaining the most important features in the data. This can help to prevent the network from learning irrelevant details in the input data, which can lead to overfitting.

Another reason to use max pooling is that it can reduce the computational complexity of the network. This is because max pooling reduces the size of the output from a convolutional layer, which means that the following layers in the network will also have smaller inputs. This can make the network faster to train and more efficient to run.

Overall, max pooling can be used in combination with a convolutional layer to improve the performance and efficiency of a CNN. It can help to reduce overfitting, improve feature selection, and reduce the computational complexity of the network.

1. **When would a local response normalization layer be useful?**

A local response normalization (LRN) layer is a type of normalization layer that is commonly used in convolutional neural networks (CNNs). An LRN layer performs normalization on the input data by normalizing the values of each output feature map across the spatial dimensions of the input. This has the effect of making the output of the LRN layer more robust to large changes in the input data, which can improve the performance of the CNN.

1. **In comparison to LeNet-5, what are the main innovations in AlexNet? What about GoogLeNet and ResNet's core innovations?**

AlexNet is a convolutional neural network (CNN) designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. It was the first CNN to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual competition in visual recognition that is considered one of the benchmarks for the performance of image recognition algorithms. In comparison to LeNet-5, one of the main innovations in AlexNet was the use of rectified linear units (ReLUs) as the activation function, instead of the traditional sigmoid function, which allowed for faster training. AlexNet also introduced the concept of dropout regularization, which helped prevent overfitting by randomly setting some of the activations to zero during training.

GoogLeNet, also known as Inception-v1, is a CNN developed by Google in 2014. It was the first CNN to win the ILSVRC, achieving a top-5 error rate of 6.67%. One of the key innovations in GoogLeNet was the use of inception modules, which allowed the network to learn both spatial and temporal information from the input data. Inception modules are made up of multiple parallel convolutional filters of different sizes, which can learn a wide range of spatial and temporal features from the input data.

ResNet, short for residual network, is a CNN developed by Microsoft Research in 2015. It was the first CNN to win the ILSVRC with a top-5 error rate of 3.57%, significantly lower than the previous state-of-the-art results. The core innovation of ResNet was the introduction of the concept of residual learning, which allows the network to learn multiple layers of representation by adding together the input from the previous layer and the output from the current layer. This allows the network to train much deeper networks, with up to 152 layers, without suffering from the vanishing gradient problem, which is a common issue when training very deep neural networks.

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

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.

9. **Large-scale image recognition using transfer learning.**

In the context of large-scale image recognition, transfer learning is often used to pre-train a model on a large dataset, such as ImageNet, and then fine-tune the model on a smaller dataset that is specific to the task at hand. This allows the model to learn from the large, general-purpose dataset, and then adapt to the specific characteristics of the smaller dataset.

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'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.