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

Convolutional neural networks (CNNs) have several advantages over fully connected deep neural networks (DNNs) for image classification. One of the main advantages is that CNNs are able to automatically learn spatial hierarchies of features, which makes them well-suited for image classification tasks. Fully connected DNNs, on the other hand, require the input to be flattened into a single vector, which means that they cannot capture spatial hierarchies of features.

Another advantage of CNNs is that they are able to reduce the number of parameters that need to be learned, compared to fully connected DNNs. This is because CNNs use convolutional layers, which apply the same filter to different parts of the input, rather than having a separate set of weights for each input pixel like in a fully connected DNN. This means that CNNs are able to achieve better performance with fewer parameters, which makes them faster to train and less prone to overfitting.

Additionally, CNNs are able to take advantage of the spatial structure of images, which allows them to learn local patterns in the data. This is especially useful for image classification tasks, where local features such as edges and corners are often important for making accurate predictions. Fully connected DNNs, on the other hand, treat the input as a flat vector of values, which means that they are unable to capture local spatial patterns in the data.

Overall, CNNs are well-suited for image classification tasks because they can automatically learn spatial hierarchies of features, reduce the number of parameters that need to be learned, and take advantage of the spatial structure of 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?

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

If your GPU runs out of memory while training a CNN, there are several things you could try to solve the problem:

Use a smaller batch size. When training a CNN, the GPU needs to hold the data for the current batch in memory. By using a smaller batch size, you can reduce the amount of memory needed to train the model.

Reduce the size of the input images. If your input images are too large, they may be using up too much memory on the GPU. You can try resizing the images to a smaller size to reduce the amount of memory needed to process them.

Use data augmentation. Data augmentation is a technique that involves artificially increasing the size of the training dataset by applying transformations to the existing data. This can help to reduce overfitting and improve the performance of the model, and it can also reduce the amount of memory needed to train the model because you can generate additional data on the fly rather than having to store it all in memory.

Use a network with fewer parameters. If your network has too many parameters, it may be using up too much memory on the GPU. You can try using a network with fewer parameters, or you can try reducing the complexity of the network by using fewer layers or smaller filters.

Use a lighter-weight network architecture. Some network architectures, such as VGG and ResNet, can be very deep and complex, which can make them difficult to train on a GPU with limited memory. In this case, you can try using a lighter-weight network architecture, such as MobileNet, that is specifically designed to be efficient and run well on limited hardware.

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

There are a few reasons why you might want to use a max pooling layer rather than a convolutional layer with the same stride.

First, max pooling is generally faster and more computationally efficient than convolutional layers. This is because max pooling reduces the dimensionality of the input by selecting the maximum value from each window, whereas convolutional layers require more computation to perform the convolution operation.

Second, max pooling can help to reduce overfitting in a model. By downsampling the input, max pooling reduces the number of parameters that the model has to learn, which can prevent the model from overfitting to the training data.

Third, max pooling can also improve the performance of a model by making it more robust to small translations in the input. Because the max pooling operation takes the maximum value from each window, it is less sensitive to small translations of the input than a convolutional layer.

Overall, max pooling is a useful technique that can help to improve the performance, computational efficiency, and generalization of a model.

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

A local response normalization layer is typically used in conjunction with convolutional layers in a neural network. It is designed to normalize the activations of a convolutional layer, which can help to improve the performance of the network.

There are a few reasons why you might want to use a local response normalization layer in your model. First, normalization can help to improve the convergence of the model by making the distribution of the activations more stable. This can make it easier for the model to learn and can lead to faster training times.

Second, normalization can also help to improve the generalization of the model. By normalizing the activations, the model is less sensitive to the specific distribution of the training data, which can help it to perform better on unseen data.

Overall, local response normalization can be a useful technique to improve the performance of a convolutional neural network, and is often used in state-of-the-art models for tasks such as image classification.

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

AlexNet was a pioneering convolutional neural network (CNN) developed by Alex Krizhevsky and colleagues at the University of Toronto. It was the first large-scale CNN to achieve significant performance on the ImageNet dataset, and it is considered one of the most important innovations in the history of deep learning.

Compared to LeNet-5, the main innovations in AlexNet were:

* The use of much larger convolutional filters, which allowed the network to learn more complex features from the input.
* The use of overlapping pooling, which reduced the dimensionality of the input without losing too much spatial information.
* The use of dropout regularization, which prevented the network from overfitting to the training data.
* The use of multiple GPUs for training, which allowed the network to be trained much faster than previous models.

GoogLeNet, ResNet, SENet, and Xception are all CNNs that were developed after AlexNet. Each of these models introduced new innovations and improvements over AlexNet and other previous CNNs.

GoogLeNet was an early attempt at developing a very deep CNN, with up to 22 layers. It introduced the concept of "inception modules", which used multiple parallel convolutional filters at each layer to learn different types of features from the input.

ResNet introduced the idea of "residual connections", which allowed the network to train much deeper than previous models without suffering from the vanishing gradient problem. This made it possible to train very deep CNNs with hundreds of layers, which improved the performance of the network.

SENet introduced the idea of "squeeze-and-excitation" blocks, which modulated the channels of the activations in a way that allowed the network to better model relationships between different channels of the input.

Xception was a variant of GoogLeNet that used depthwise separable convolutions, which allowed the network to be trained more efficiently and to achieve better performance. It was also the first CNN to use a depthwise separable convolution in the first layer, which improved the ability of the network to learn from the input.

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

* A fully convolutional network (FCN) is a type of neural network that is designed to process inputs of arbitrary size, making it well-suited for tasks such as image segmentation. In an FCN, all of the layers are convolutional, which means that they are designed to operate on spatial data such as images. This makes FCNs efficient at processing spatial data, and allows them to be applied to inputs of varying size.
* To convert a dense layer into a convolutional layer, you can simply replace the dense layer with a convolutional layer that has the same number of filters and the same kernel size as the original dense layer. For example, if you had a dense layer with 64 units and a kernel size of 3x3, you could replace it with a convolutional layer that also has 64 filters and a kernel size of 3x3.

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

One of the main technical difficulties of semantic segmentation is the fact that it requires the model to make fine-grained predictions at a pixel level. This means that the model must be able to accurately identify and classify very small and potentially visually similar objects in an image, which can be challenging. Additionally, the amount of data required for training a semantic segmentation model can be very large, which can make it difficult to train the model effectively. Finally, the performance of a semantic segmentation model can be sensitive to changes in the camera angle, lighting conditions, and other factors, which can make it difficult to generalize the model to new situations.

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.