1. **After each stride-2 conv, why do we double the number of filters?**

Increasing the number of filters in a convolutional layer allows the model to learn more complex features from the input data. In a stride-2 convolution, the spatial dimensions of the input are reduced by half, which can lead to a reduction in the amount of information the model has available to learn from. Doubling the number of filters can help compensate for this loss of information and allow the model to continue learning effectively.

2. **Why do we use a larger kernel with MNIST (with simple cnn) in the first conv?**

The MNIST dataset consists of images of handwritten digits, which are relatively simple and highly structured. In order to capture the essential features of these images, a convolutional neural network (CNN) typically uses a larger kernel size in the first convolutional layer. A larger kernel allows the model to capture the overall structure of the input images and extract the important features, such as the shape and orientation of the digits. This initial set of learned features can then be used by the subsequent layers of the network to more accurately identify the digits in the images.

3. **What data is saved by ActivationStats for each layer?**

ActivationStats is a tool for collecting statistics about the activations (i.e. the output values) of each layer in a neural network. It typically collects the following data for each layer:

* The minimum and maximum activation values
* The mean and standard deviation of the activation values
* The number of zeros and the sparsity (proportion of zeros) in the activations

4. **How do we get a learner's callback after they've completed training?**

A learner's callback can be executed after the training process is complete by specifying the callback function as an argument to the fit() method of the learner object.

For example, if we have a learner object named my\_learner and a callback function named my\_callback, we can use the following code to execute the callback after training is complete:

**my\_learner.fit(X\_train, y\_train, callback=my\_callback)**

5. **What are the drawbacks of activations above zero?**

* One potential drawback of using activation functions that have non-zero values for some inputs is that they can cause the network to saturate. Saturation occurs when the activations of the neurons in a network become extremely large or small, which can prevent the network from learning effectively. This can happen because, as the activations become very large or small, the gradients of the activation functions also become very small, which makes it difficult for the network to update its weights and learn from the data.
* Another potential drawback of non-zero activations is that they can make it more difficult to interpret the output of the network. In general, the activations of a neural network can be thought of as the intermediate representations that the network learns from the data. When the activations are non-zero, it can be more difficult to understand how these intermediate representations are related to the input and output of the network.

6.**Draw up the benefits and drawbacks of practicing in larger batches?**

Some potential benefits of using larger batch sizes when training a neural network include:

Faster training: Using a larger batch size can allow the network to process more data at once, which can make the training process faster. This is because the model can make more updates to its weights in each iteration, which can help it converge to a good solution more quickly.

Better generalization: In some cases, using a larger batch size can also improve the generalization performance of the model. This is because training on a larger batch of data can provide the model with more diverse information, which can help it learn more robust and generalizable representations of the data.

More stable gradients: When using stochastic gradient descent (SGD) to train a neural network, using a larger batch size can make the gradients more stable. This is because the gradients are calculated using the average of the gradients of the individual examples in the batch, which can help reduce the variance of the gradients and make the training process more stable.

Some potential drawbacks of using larger batch sizes when training a neural network include:

Memory requirements: Using a larger batch size can increase the amount of memory required to train the model. This is because the model needs to store the activations and gradients for each example in the batch, which can quickly become impractical for large batch sizes.

Decreased learning rate: In some cases, using a larger batch size can require a lower learning rate in order to achieve good performance. This is because the model makes larger weight updates when using a larger batch size, which can cause it to overshoot the optimal solution if the learning rate is too high.

Loss of fine-grained information: Training on larger batches can cause the model to lose some of the fine-grained information in the data. This is because the model only sees a subset of the data in each iteration, and the larger the batch size, the less frequently it sees the individual examples in the dataset. This can make it more difficult for the model to learn subtle patterns in the data.

7. **Why should we avoid starting training with a high learning rate?**

It is generally recommended to avoid starting the training of a neural network with a high learning rate. This is because, if the learning rate is too high, the model may make large and unstable weight updates, which can cause it to diverge and fail to learn effectively.

When the learning rate is too high, the model may oscillate or diverge instead of converging to a good solution. This can happen because the model is making large weight updates that are not well-aligned with the true gradient of the loss function, which can cause it to overshoot the optimal solution and get stuck in a suboptimal state.

**8. What are the pros of studying with a high rate of learning?**

There are some potential benefits to using a high learning rate when training a neural network, including:

Faster training: Using a high learning rate can allow the model to make larger weight updates in each iteration, which can make the training process faster. This is because the model can move closer to the optimal solution more quickly, which can reduce the number of iterations required to reach a good solution.

Better generalization: In some cases, using a high learning rate can also improve the generalization performance of the model. This is because a high learning rate can allow the model to escape from suboptimal solutions and explore the solution space more extensively, which can help it find a more robust and generalizable solution.

Improved performance: In some cases, using a high learning rate can lead to better performance on the training set and on the validation or test sets. This is because the model is able to make more aggressive weight updates, which can allow it to fit the training data more closely and improve its accuracy on the validation or test sets.

However, it is important to note that using a high learning rate can also have some potential drawbacks, such as increased instability and a higher risk of divergence. It is therefore important to carefully tune the learning rate to find the best balance between performance and stability.

9. **Why do we want to end the training with a low learning rate?**

It is generally recommended to end the training of a neural network with a low learning rate. This is because, as the model approaches the optimal solution, the gradients of the loss function become smaller and the weight updates become less important. Using a low learning rate at this stage can help the model make small and precise weight updates that are well-aligned with the true gradient of the loss function, which can help it converge to the optimal solution more reliably.