

Training accuracy plot for one epoch. For the dense layer, the accuracy drops rapidly after four layers. For the residual layer, the accuracy is stable at around 84 percent as the number of layers increases.

Training accuracy plot for ten epochs. For the dense layer, the accuracy drops after four layers (with some fluctuating between two and four layers). For the residual layer, the accuracy is more stable (only fluctuating a bit) at around 90 percent even as the number of layers increases.

Increased depth can improve the performance of a neural network, but it can also

make the training process more time-consuming.

Deeper neural networks might be able to learn more complex patterns and

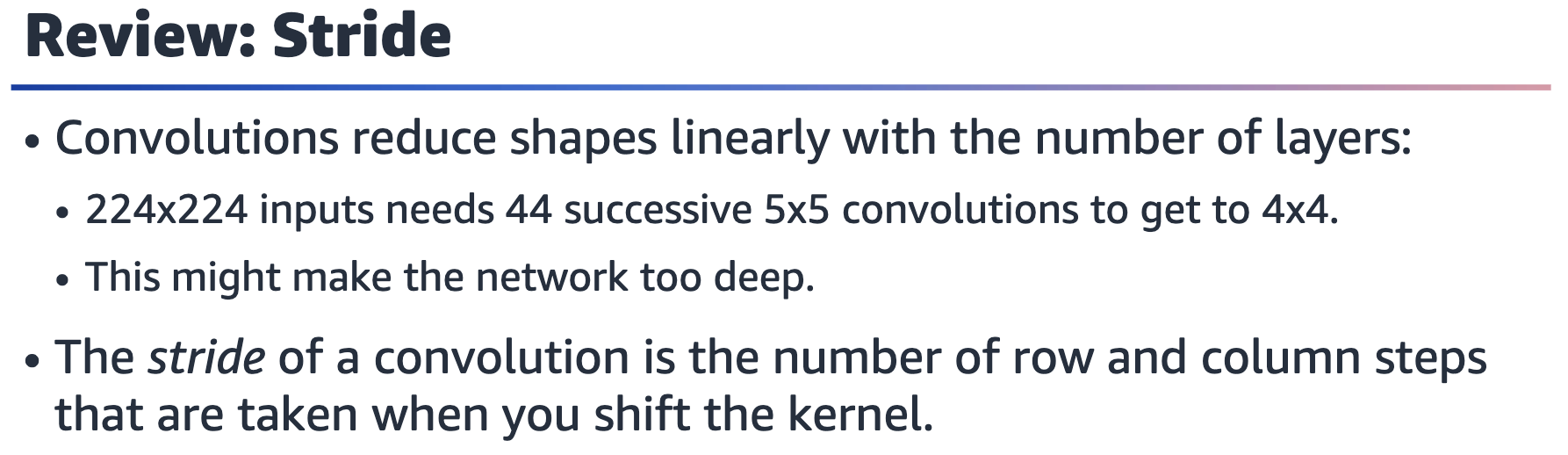
representations in data, but they might also be more susceptible to overfitting.

Increasing the depth of a neural network can improve its ability to generalize to new

data, but it can also make the network more sensitive to small changes in the input data.

One issue with deep models could be the vanishing gradient problem, which can

occur when you train deep neural networks with many layers.



In image processing, stride refers to the step size that is used when applying a convolutional filter to an input image. Specifically, stride is the number of pixels that the filter is moved horizontally and vertically across the image when performing the convolution.

A larger stride value means that the filter skips more pixels when moving across the image, which results in a smaller output size. Conversely, a smaller stride value means that the filter moves more slowly across the image, which results in a larger output size.

Stride is an important parameter to consider when building deep learning models for

image processing. Increasing the stride can reduce the computational complexity of

the network and speed up the training process, but it can lead to a loss of

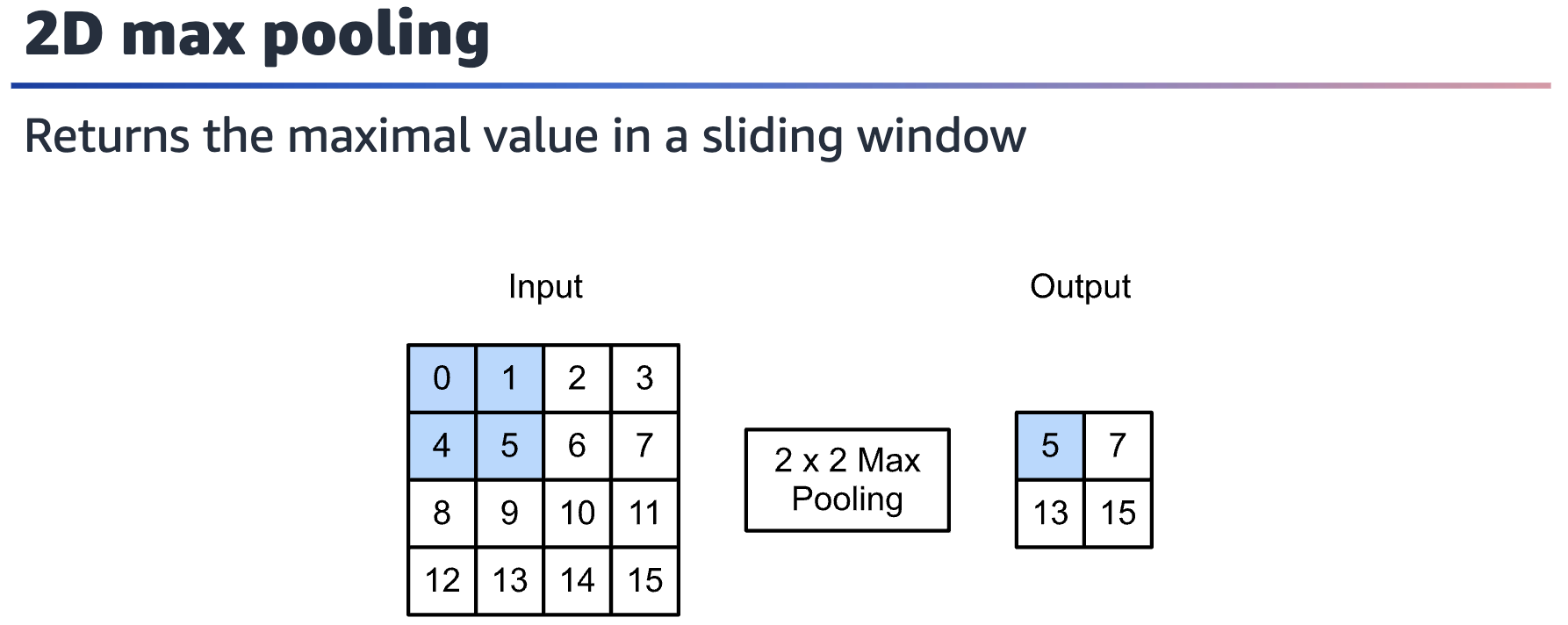
information if important features are skipped. Decreasing the stride can improve the

accuracy of the model by allowing it to capture more fine-grained details, but it also

increases the computational cost and might lead to overfitting.

Overall, choose the stride value based on the specific requirements of the problem

& the available computational resources. In general, a smaller stride value is better for tasks that require high accuracy and fine-grained feature detection, while a larger stride value is better for tasks that require fast computation and coarse feature detection.



The main purpose of pooling is to reduce the size of feature maps or inputs without

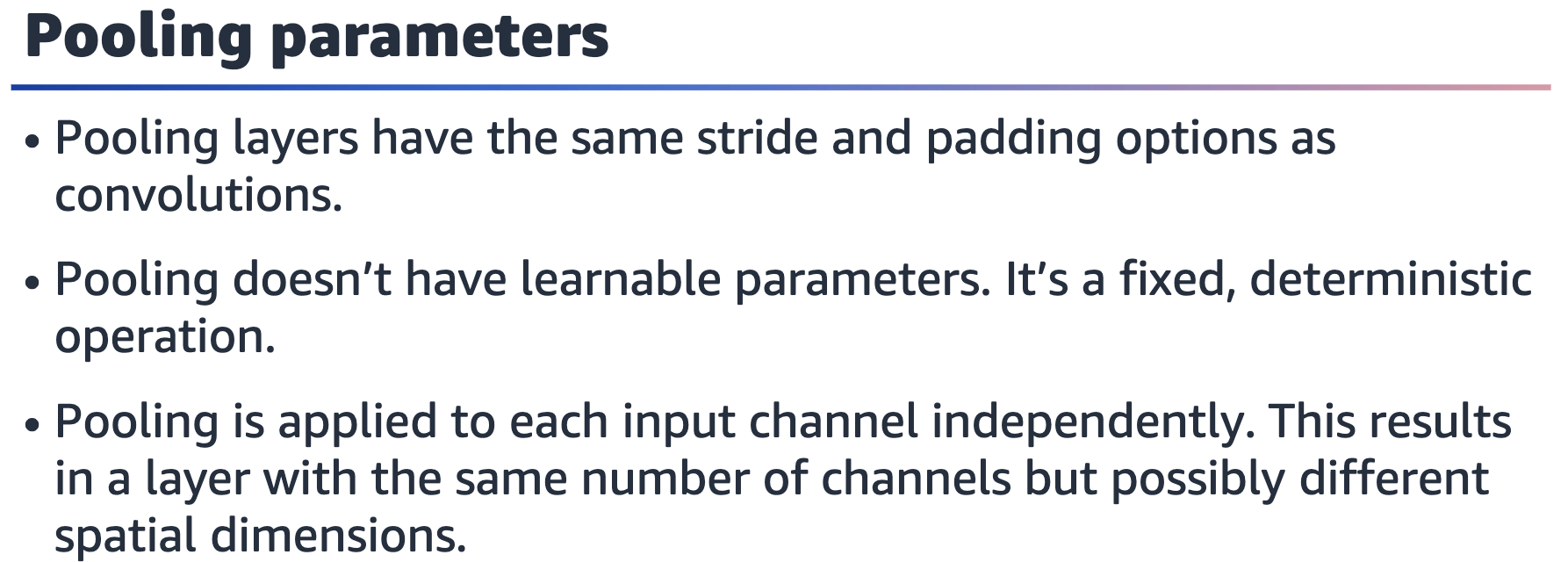
losing a lot of information. This makes computation faster because the number of

training parameters is reduced. Pooling reduces the spatial dimensionality of the

feature maps while preserving the most important information. This is achieved by

applying a fixed function, such as max pooling or average pooling, to small local

regions of the input feature maps.



Pooling doesn’t have learnable parameters because it’s a fixed and deterministic

operation that is applied to the input data.

Pooling is applied to each input channel independently in a convolutional neural

network (CNN) because different channels can contain different features or patterns

that are important for the overall classification task.

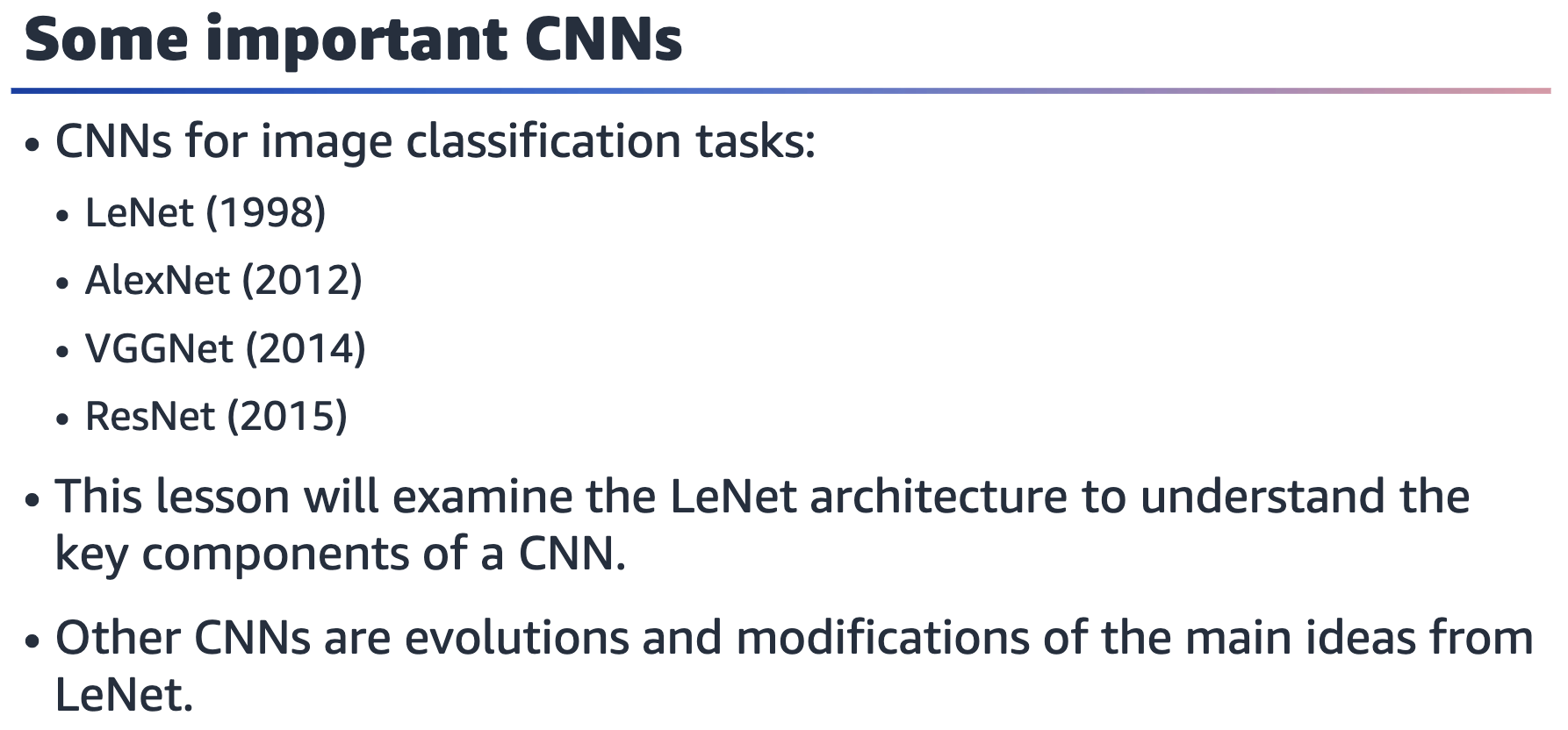
In a CNN, each input channel corresponds to a different feature map or channel from

the previous convolutional layer. The purpose of pooling is to reduce the dimensionality of the feature maps while preserving the most important information. Applying pooling to each input channel independently ensures that each channel is reduced in dimensionality by the same factor, which helps to maintain the relative

importance of each channel's information.

For example, if the pooling operation was applied to all input channels at once, the

information in one channel might be overemphasized, while the information in another channel might be lost or downplayed. By applying pooling independently to each channel, the relative importance of the information in each channel is preserved, which can improve the overall performance of the CNN.



Over the years, several important CNN architectures were widely adopted by the

research community. The first famous CNN was an architecture called LeNet.

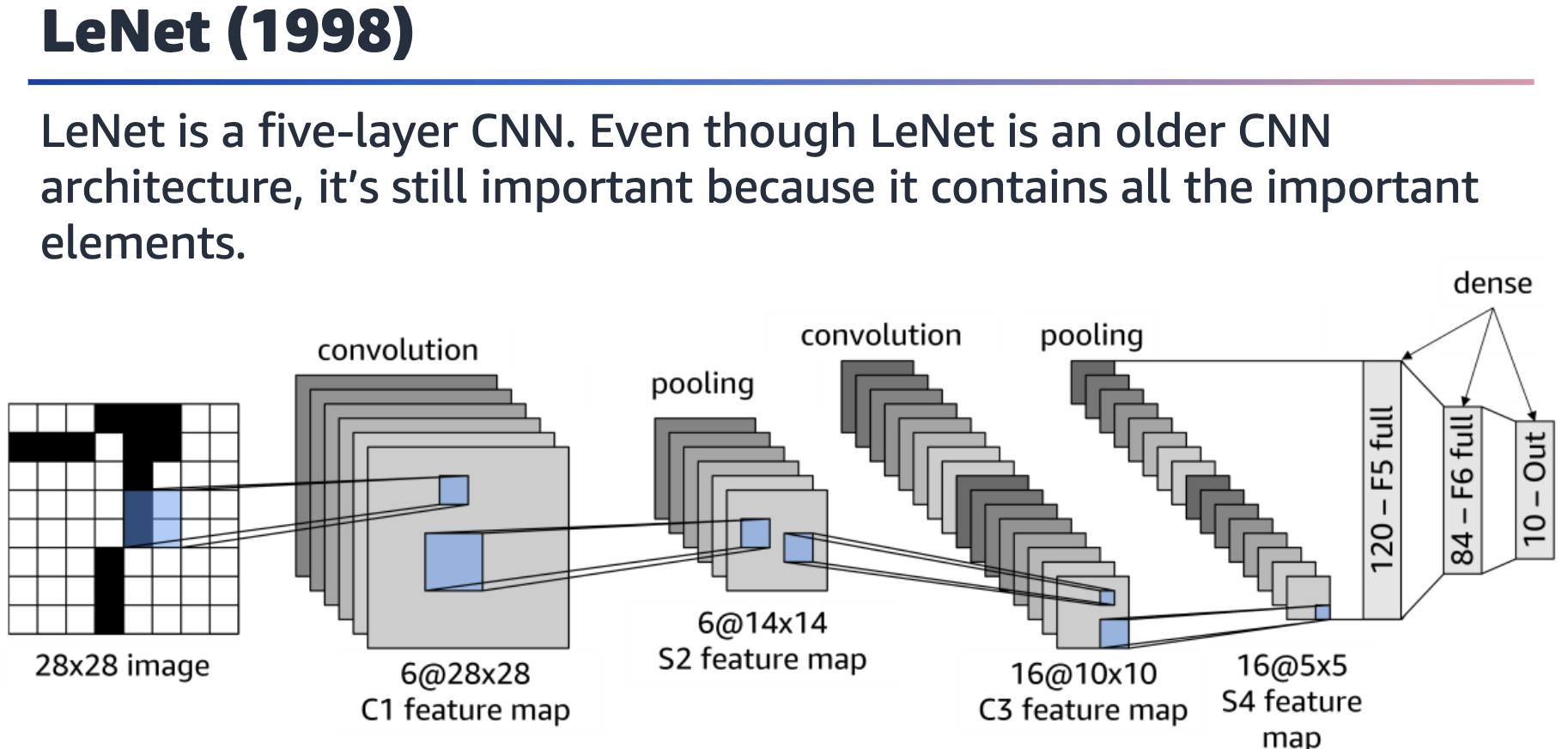
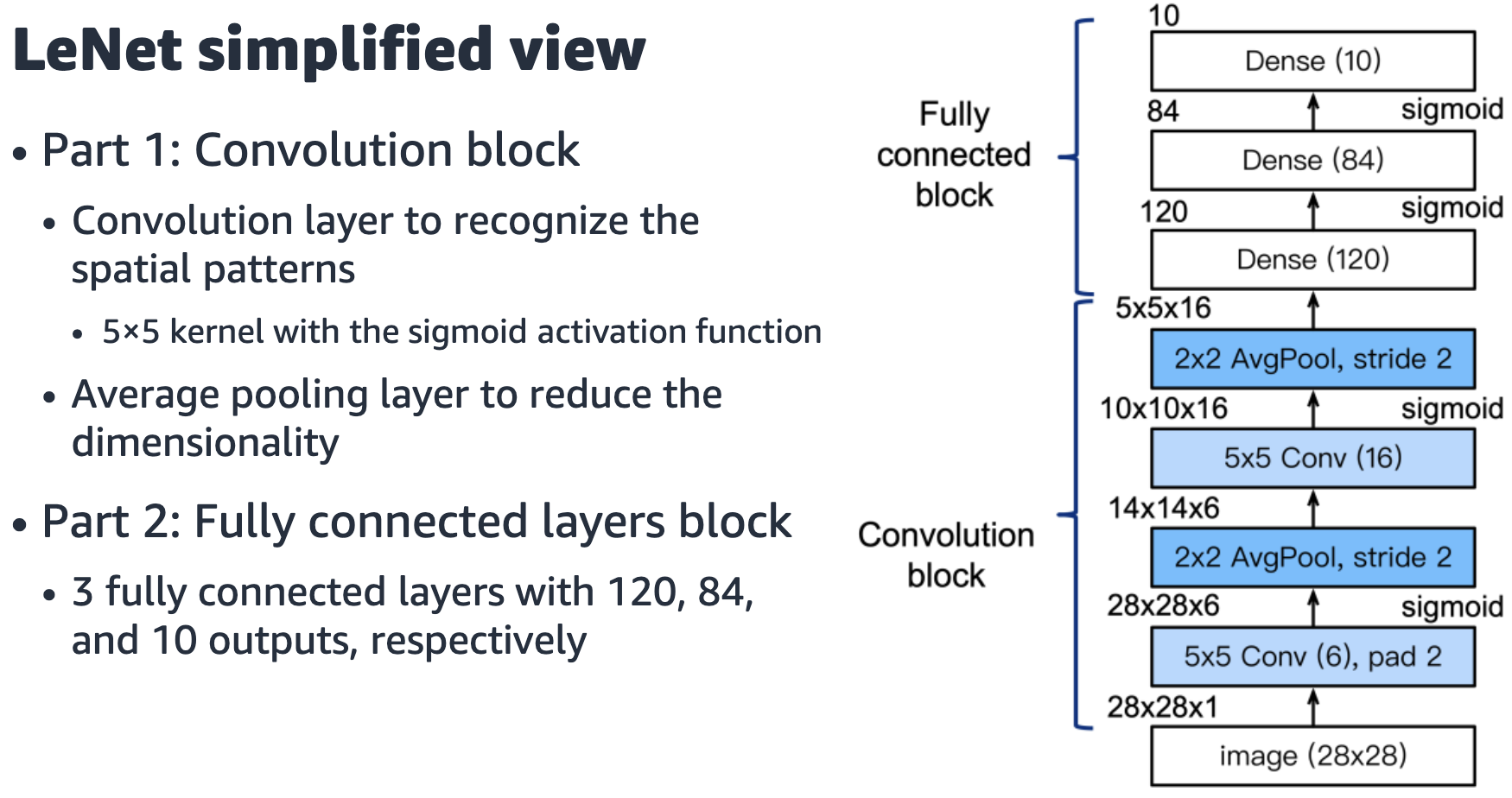


Diagram of LeNet’s five-layer CNN. Shows the input image, a convolutional layer, a pooling layer, another convolutional layer, another pooling layer, and a final sequence of three dense layers.

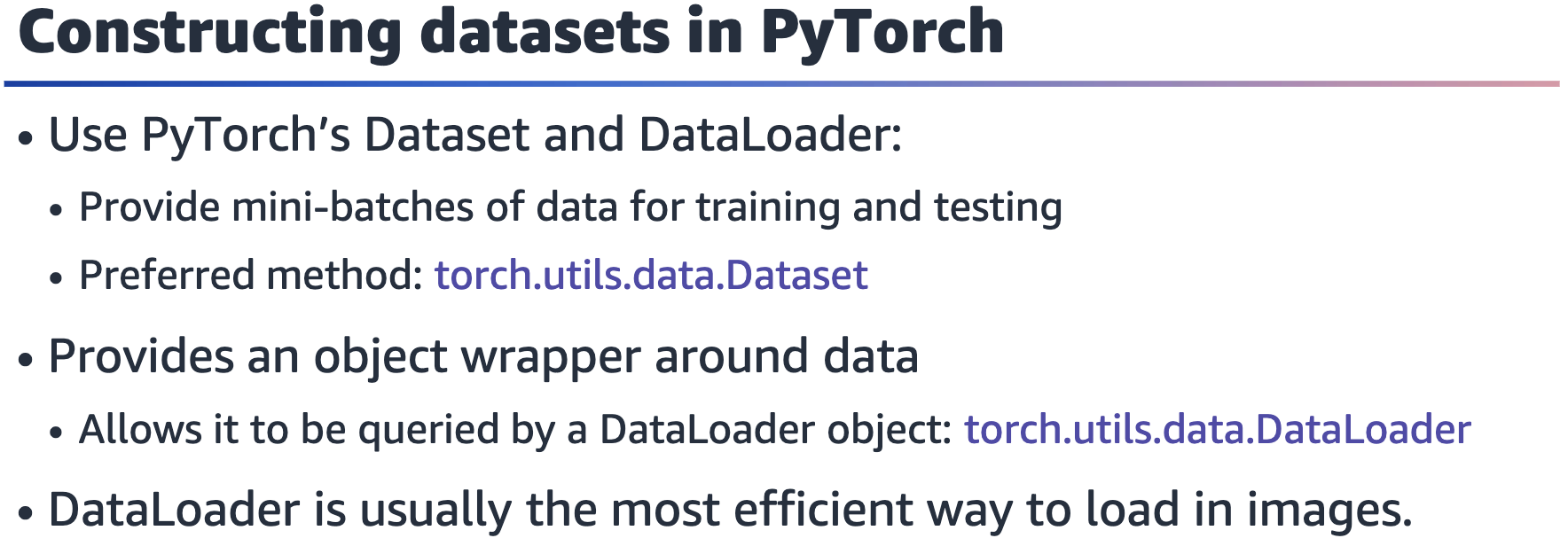
LeNet is a five-layer CNN. Convolution layers are followed by pooling layers. At the

end, dense layers take in the pooling layer’s results. Feature maps get smaller in height and width, and the number of channels increases as you go deeper in the network.



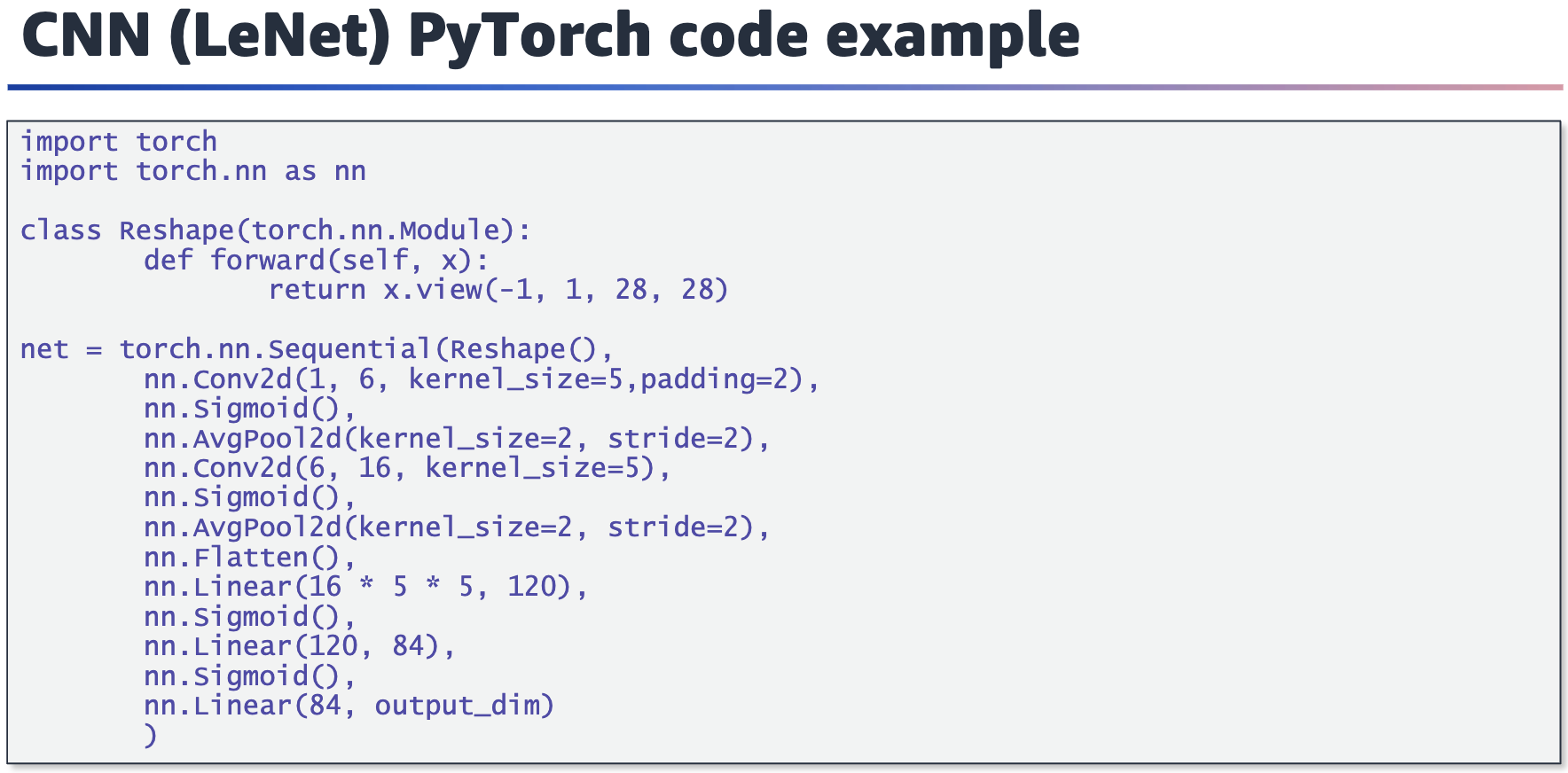
Simplified diagram of LeNet's 5 layer CNN that shows input image, the convolutional layer, a pooling layer, followed by another convolutional & pooling layer and a final sequence of 3 dense layers.

It can be helpful to look at a simplified view of a CNN that only indicates the network’s main characteristics, rather than actually visualizing the number of layers. For LeNet, you can also distinguish between a set of layers that performs convolution and the fully connected block of dense layers.



This slide provides a quick introduction to DataLoader in PyTorch. This construct is

commonly found when building CNNs to load in image data.

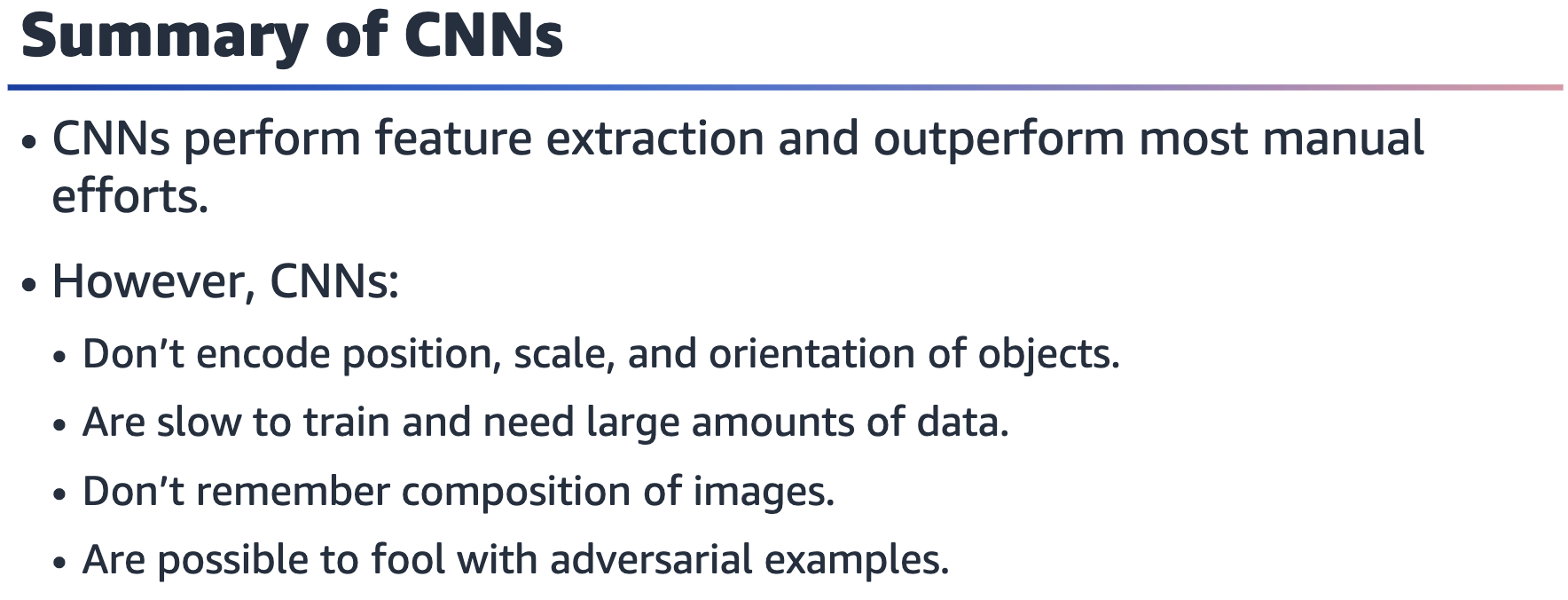


First, you need to reshape input images to 28x28 pixels. Then, you construct the

network exactly as discussed in the previous section with convolutional layers,

pooling layers, and dense (linear) layers. Between the layers, you can see sigmoid

activation functions



CNNs will struggle to classify images correctly if the orientation of the object or

position is different.

CNNs are also slow to train and don’t preserve the composition of different elements

that make up an object. This is part of why CNNs can be fooled.

