1. Explain why ReLU is typically preferred over Sigmoid as the activation function in the convolutional block?

   Avoiding Vanishing Gradients: The output range of Sigmoid is between 0 and 1, which makes it prone to the vanishing gradient problem during backpropagation, especially when the network is deep. The maximum gradient of Sigmoid is 0.25, which means the gradient can become very small as it propagates through multiple layers, making training very difficult. On the other hand, ReLU has a derivative of either 1 or 0, which helps avoid the vanishing gradient problem effectively.

   More Efficient Computation: The computation of the ReLU function is very simple, as it only involves computing max(0, x). In contrast, Sigmoid involves calculating an exponential function, which is much more computationally complex. The simpler computation of ReLU allows for faster training, especially in deep networks.

   Sparse Activation: ReLU outputs 0 for any negative input, meaning that it can deactivate certain neurons at certain layers. This sparse activation can increase the efficiency of the model and improve generalization, as it is closer to how biological neurons work in the human brain. Sigmoid, on the other hand, outputs values between 0 and 1, which means neurons are almost always activated to some degree.

2. Describe how you design the CNN architecture and your findings in choosing parameters such as filter_size and pool_size for each layer?

   Layer 1: Convolution (filter_size=3, output_channel=16) with ReLU, followed by max pooling (pool_size=2).

   Layer 2: Convolution (filter_size=3, output_channel=32) with ReLU, followed by max pooling (pool_size=2).

   Layer 3: Convolution (filter_size=3, output_channel=64) with ReLU, followed by max pooling (pool_size=2).

   Flatten Layer: Converts feature maps into a 1D vector for the fully connected layers.

   Fully Connected Layer: Dense (input=1024, output=64) with ReLU. Reduces dimensionality while preserving learned features.

   Output Layer: Dense (input=64, output=1) with Sigmoid for binary classification.

   Findings:

   Filter Size: Chose 3x3 filters as they are efficient for feature extraction without excessive computational cost.

Pool Size: Used 2x2 pooling to effectively reduce feature map dimensions while retaining important information.

Model Depth: Used this approach with gradually increasing filters, allowing for effective feature extraction by increasing the representation capacity of each layer without overfitting.

3. Calculate and compare the number of learnable parameters between the CNN model and the NN model you designed for binary classification in Lab4. For simplicity, omit the bias parameters and calculate only the weights.

   For CNN:

   Convolutional Layer 1:

      Input Channels: 1, Output Channels: 16, Filter Size: $3 \times 3 = 9$

      Number of Parameters: $3 \times 3 \times 1 \times 16 = 144$

   Convolutional Layer 2:

      Input Channels: 16, Output Channels: 32, Filter Size: $3 \times 3 = 9$

      Number of Parameters: $3 \times 3 \times 16 \times 32 = 4608$

   Convolutional Layer 3:

      Input Channels: 32, Output Channels: 64, Filter Size: $3 \times 3 = 9$

      Number of Parameters: $3 \times 3 \times 32 \times 64 = 18432$

   Fully Connected Layer 1:

      Input Units: $4 \times 4 \times 64 = 1024$ (flattened output from convolutional layers)

      Output Units: 64

      Number of Parameters: $1024 \times 64 = 65536$

   Fully Connected Output Layer:

      Input Units: 64, Output Units: 1

      Number of Parameters: $64 \times 1 = 64$

   Total learnable parameters in CNN = 144 + 4608 + 18432 + 65536 + 64 = 88784

   For NN in Lab4:

   # of neurons on layer = [784, 128, 128, 64, 64, 32, 4]

   Total learnable parameters in NN = $784 \times 128 + 128 \times 128 + 128 \times 64 + 64 \times 64 + 64 \times 32 + 32 \times 4 = 100352 + 16384 + 8192 + 4096 + 2048 + 128 = 131200$

   Comparison:

   The CNN model has significantly fewer parameters (88784) compared to the fully connected NN model from Lab4 (131200). This difference is due to the use of convolutional layers, which share parameters across spatial locations. This parameter sharing makes CNNs more efficient for image tasks, reducing the overall number of learnable parameters while still retaining powerful feature extraction capabilities.