# CNN and Image Processing Scenario-Based Questions

May 22, 2025

## 1 Multiple Choice Questions (MCQs)

1. **ReLU Activation Function**
   *Scenario*: A developer trains a CNN on CIFAR-10 to classify images (e.g., airplanes, dogs). The CNN uses ReLU after convolutional layers.

```
import torch
import torch.nn as nn
conv_layer = nn.Conv2d(3, 32, kernel_size=3,
    padding=1)
relu = nn.ReLU()
# Example input and output
x = torch.tensor([[-1.0, 2.0], [0.5, -0.3]])
y = relu(x)
print(f"ReLU output: {y}")
```

   *Output*: `ReLU output: [[0.0, 2.0], [0.5, 0.0]]`
   *Question*: What is the purpose of ReLU in this CNN?

   A) To normalize pixel values

   B) To introduce non-linearity

   C) To reduce image dimensions

   D) To compute loss

   **Answer**: **B**
   *Explanation*: ReLU, defined as $f(x) = \max(0, x)$, introduces non-linearity by setting negative values to zero, enabling the CNN to learn

complex, non-linear patterns in images, crucial for tasks like CIFAR-10 classification. Normalization is handled by preprocessing, dimension reduction by pooling, and loss by functions like CrossEntropyLoss.

2. **Normalization in CIFAR-10**
*Scenario*: A data scientist preprocesses CIFAR-10 images (32×32, RGB) for a CNN, normalizing pixel values from [0, 255] to [-1, 1].

```
from torchvision import transforms
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5,
       0.5, 0.5))
])
# Example pixel value transformation
pixel = 128
normalized = (pixel / 255 - 0.5) / 0.5
print(f"Normalized pixel: {normalized:.2f}")
```

*Output*: `Normalized pixel: 0.00`
*Question*: Why is normalization applied?

A) To enhance image contrast

B) To stabilize CNN training

C) To increase image resolution

D) To remove color channels

**Answer**: **B**
*Explanation*: Normalization, using Normalized $= \frac{\text{pixel}/255-\mu}{\sigma}$ with $\mu = 0.5$, $\sigma = 0.5$, scales pixel values to [-1, 1], stabilizing gradient updates during CNN training and improving convergence. It doesn't affect contrast, resolution, or channels.

3. **Convolutional Layer Output Size**
*Scenario*: A CNN processes a 32×32 RGB image with a Conv2d layer (3 input channels, 64 output channels, 3×3 kernel, padding=1, stride=1).

```
import torch.nn as nn
conv = nn.Conv2d(3, 64, kernel_size=3, padding=1,
   stride=1)
```

```
# Input: (batch, 3, 32, 32)
# Output shape calculation
output_shape = (32 - 3 + 2*1) // 1 + 1
print(f"Output height/width: {output_shape}")
```

*Question*: What is the output spatial size?

A) 30×30

B) 32×32

C) 16×16

D) 64×64

**Answer**: **B**
*Explanation*: The output size is calculated as Output $= \lfloor \frac{W-K+2P}{S} \rfloor + 1$, where $W = 32$, $K = 3$, $P = 1$, $S = 1$. Thus, $\lfloor \frac{32-3+2}{1} \rfloor + 1 = 32$, preserving the 32×32 size due to padding.

4. **Max Pooling Effect**
*Scenario*: A CNN applies max pooling (2×2 kernel, stride=2) to a 32×32 feature map.

```
import torch.nn as nn
pool = nn.MaxPool2d(kernel_size=2, stride=2)
# Input: (batch, channels, 32, 32)
# Output shape calculation
output_shape = 32 // 2
print(f"Output height/width: {output_shape}")
```

*Question*: What is the output spatial size after pooling?

A) 32×32

B) 16×16

C) 8×8

D) 64×64

**Answer**: **B**
*Explanation*: Max pooling with a 2×2 kernel and stride=2 halves the spatial dimensions: Output $= \lfloor \frac{W}{S} \rfloor = \lfloor \frac{32}{2} \rfloor = 16$. This reduces computational load while retaining key features.

5. **CrossEntropyLoss Usage**
*Scenario*: A CNN for CIFAR-10 uses CrossEntropyLoss for training.

```
import torch.nn as nn
criterion = nn.CrossEntropyLoss()
# Example: Predicted logits and true label
logits = torch.tensor([[2.0, 1.0, 0.1], [0.5, 2.5,
    0.3]])
labels = torch.tensor([0, 1])
loss = criterion(logits, labels)
print(f"Loss: {loss.item():.2f}")
```

*Question*: What does CrossEntropyLoss measure?

A) Pixel intensity differences

B) Mean squared error

C) Classification error

D) Feature map variance

**Answer**: **C**
*Explanation*: CrossEntropyLoss, $-\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{\text{true},c}^{(i)} \log(y_{\text{pred},c}^{(i)})$, measures the error between predicted class probabilities (after softmax) and true labels, guiding CNN optimization for classification tasks like CIFAR-10.

6. **Edge Detection Kernel**
*Scenario*: A medical imaging CNN uses a vertical edge detection kernel.

```
import torch
conv_layer = nn.Conv2d(1, 1, kernel_size=3,
    bias=False)
vertical_kernel = torch.tensor([[[[1., 0., -1.],
                                  [1., 0., -1.],
                                  [1., 0., -1.]]]])
conv_layer.weight.data = vertical_kernel
```

*Question*: What does this kernel detect?

A) Horizontal edges

B) Vertical edges

C) Diagonal edges

D) Color gradients

**Answer**: **B**

*Explanation*: The kernel $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ detects vertical edges by highlighting bright-to-dark transitions (left-to-right) with positive values and dark-to-bright with negative values, as used in medical imaging for boundary detection.

7. **Region of Interest (ROI)**
*Scenario*: A CNN processes MRI scans, focusing on a tumor region as the ROI.

```
import cv2
image = cv2.imread('mri_scan.jpg', 0)
roi = image[50:150, 50:150]  # Extract 100x100
    tumor region
print(f"ROI shape: {roi.shape}")
```

*Question*: Why focus on the ROI?

A) To increase image resolution

B) To reduce computational cost

C) To normalize pixel values

D) To apply pooling

**Answer**: **B**
*Explanation*: Focusing on the ROI (e.g., tumor region) reduces the image area processed, lowering computational cost and noise, improving efficiency and accuracy in tasks like tumor detection, as noted in medical imaging applications.

8. **Sigmoid Activation**
*Scenario*: A CNN uses sigmoid activation for binary classification.

```python
import torch
sigmoid = torch.nn.Sigmoid()
x = torch.tensor([0.0, 1.0, -1.0])
y = sigmoid(x)
print(f"Sigmoid output: {y}")
```

*Output*: `Sigmoid output:  [0.5000, 0.7311, 0.2689]`
*Question*: What is the output range of sigmoid?

A) $[-1, 1]$
B) $[0, 1]$
C) $[0, \infty)$
D) $(-\infty, \infty)$

**Answer**: **B**
*Explanation*: Sigmoid, $f(x) = \frac{1}{1+e^{-x}}$, maps inputs to $[0, 1]$, suitable for probability outputs in binary classification—unlike tanh, which maps to $[-1, 1]$, or ReLU, which maps to $[0, \infty)$.

# 2 Numeric Type Questions (NTQs)

1. **Convolution Output Calculation**
*Scenario*: A 5×5 grayscale image is convolved with a 3×3 kernel (stride=1, no padding). The kernel is applied at the top-left corner.

```python
import numpy as np
image = np.array([[1, 1, 1, 0, 0],
                  [0, 1, 1, 1, 0],
                  [0, 0, 1, 1, 1],
                  [0, 0, 1, 1, 0],
                  [0, 1, 1, 0, 0]])
kernel = np.array([[1, 0, 1],
                   [0, 1, 0],
                   [1, 0, 1]])
output = np.sum(image[0:3, 0:3] * kernel)
print(f"Output value: {output}")
```

*Question*: What is the output value at the top-left position?
**Answer**: **4**

*Explanation*: Convolution computes $Y[i,j] = \sum_{m=0}^{2}\sum_{n=0}^{2} X[i+m, j+n] \cdot K[m,n]$. For the top-left 3×3 region, element-wise multiplication and summation yield: $(1\cdot 1)+(1\cdot 0)+(1\cdot 1)+(0\cdot 0)+(1\cdot 1)+(1\cdot 0)+(0\cdot 1)+(0\cdot 0)+(1\cdot 1) = 4$.

2. **Normalized Pixel Value**
   *Scenario*: A CIFAR-10 image pixel (RGB=128) is normalized using mean=0.5, std=0.5.

```
pixel = 128
normalized = (pixel / 255 - 0.5) / 0.5
print(f"Normalized value: {normalized:.2f}")
```

   *Output*: `Normalized value:  0.00`
   *Question*: What is the normalized pixel value, rounded to two decimal places?
   **Answer**: **0.00**
   *Explanation*: Normalization is Normalized $= \frac{\text{pixel}/255-0.5}{0.5}$. For pixel=128, $\frac{128}{255} \approx 0.502$, then $\frac{0.502-0.5}{0.5} = 0.004 \approx 0.00$.

3. **Output Size with Stride**
   *Scenario*: A 64×64 feature map is convolved with a 5×5 kernel, stride=2, no padding.

```
W, K, S, P = 64, 5, 2, 0
output_size = (W - K + 2*P) // S + 1
print(f"Output size: {output_size}")
```

   *Question*: What is the output spatial size?
   **Answer**: **30**
   *Explanation*: Output size is $\lfloor \frac{W-K+2P}{S} \rfloor + 1$. With $W = 64$, $K = 5$, $P = 0$, $S = 2$, $\lfloor \frac{64-5}{2} \rfloor + 1 = \lfloor 29.5 \rfloor + 1 = 30$.

4. **ReLU Output Value**
   *Scenario*: A CNN feature map value of -0.7 is passed through ReLU.

```
import torch
x = torch.tensor(-0.7)
y = torch.nn.ReLU()(x)
print(f"ReLU output: {y.item()}")
```

*Question*: What is the ReLU output?
**Answer**: **0.0**
*Explanation*: ReLU, $f(x) = \max(0, x)$, outputs 0 for negative inputs like -0.7, introducing sparsity and non-linearity.

# 3 Multiple Select Questions (MSQs)

1. **CNN Preprocessing Steps**
*Scenario*: A developer prepares CIFAR-10 data for a CNN.
*Question*: Which steps are part of the preprocessing pipeline?

   A) Resizing to 224×224

   B) Normalization to [-1, 1]

   C) Random horizontal flip

   D) Fully connected layer

   **Answers**: **B, C**
   *Explanation*:
   - A) CIFAR-10 images are 32×32, not resized to 224×224, false.
   - B) Normalization to [-1, 1] stabilizes training, true.
   - C) Random horizontal flip augments data, true.
   - D) Fully connected layers are part of the CNN, not preprocessing, false.

2. **Limitations of 2D Convolution**
*Scenario*: A CNN processes high-resolution images for object detection.
*Question*: Which are limitations of 2D convolution?

   A) Fixed receptive field

   B) Loss of contextual information

   C) Inability to model temporal changes

   D) Automatic feature extraction

   **Answers**: **A, B, C**
   *Explanation*:
   - A) Fixed kernel size limits capturing large patterns, true.
   - B) Local focus misses distant relationships, true.

- C) 2D convolution cannot handle temporal data, true.
- D) Automatic feature extraction is a strength, not a limitation, false.

3. **Activation Function Properties**
   *Scenario*: A CNN uses various activation functions for CIFAR-10.
   *Question*: Which statements are true about activation functions?

   A) ReLU outputs $[0, \infty)$

   B) Sigmoid outputs $[0, 1]$

   C) Tanh outputs $[-1, 1]$

   D) Leaky ReLU outputs $(-\infty, \infty)$

   *Explanation*:

   A) ReLU, $\max(0, x)$, outputs $[0, \infty)$.

   B) Sigmoid, $\frac{1}{1+e^{-x}}$, outputs $[0, 1]$.

   C) Tanh, $\tanh(x)$, outputs $[-1, 1]$.

   D) Leaky ReLU, $\max(\alpha x, x)$, outputs $(-\infty, \infty)$.