

DL – Unit 3 (Convolution Neural Network) – END-SEM PYQ Answers

MAY-JUNE 2023

Q1a Explain Pooling Layer with its need and different types.

[6 Marks]

Pooling Layer — Overview

A Pooling Layer (also called a sub-sampling or down-sampling layer) is placed after a Convolution Layer to reduce the spatial dimensions (width × height) of the feature maps while retaining the most important information. This reduces the number of parameters, mitigates overfitting, and makes the network more computationally tractable.

Need for Pooling

- Reduces spatial size of feature maps, lowering computation cost.
- Provides spatial invariance — small translations of the input do not change the pooled output.
- Reduces the number of learnable parameters, helping prevent overfitting.
- Makes features more abstract and robust to noise.

Types of Pooling

- **Max Pooling:** Selects the maximum value from each pooling window. Retains the most prominent feature in a region. Most commonly used.
- **Average Pooling:** Computes the average of values in each pooling window. Gives a smoother representation; used in some modern architectures.
- **Global Average Pooling (GAP):** Reduces each feature map to a single number (its mean). Often replaces fully-connected layers at the end of CNNs (e.g., GoogLeNet).
- **Min Pooling:** Selects the minimum value — rarely used in practice.
- **Stochastic Pooling:** Samples a value from the pooling region according to a multinomial distribution proportional to activations; used for regularisation.

Max Pooling — Numerical Example

Input feature map (4×4): After 2×2 Max Pool (stride 2):

1	3	2	4	→	3	4
5	6	1	2		6	3
7	1	3	0		2	4

Note: Pooling has NO learnable parameters. The window size (kernel) and stride are hyperparameters set by the designer. Typical choice: 2×2 kernel, stride 2 — halves each spatial dimension.

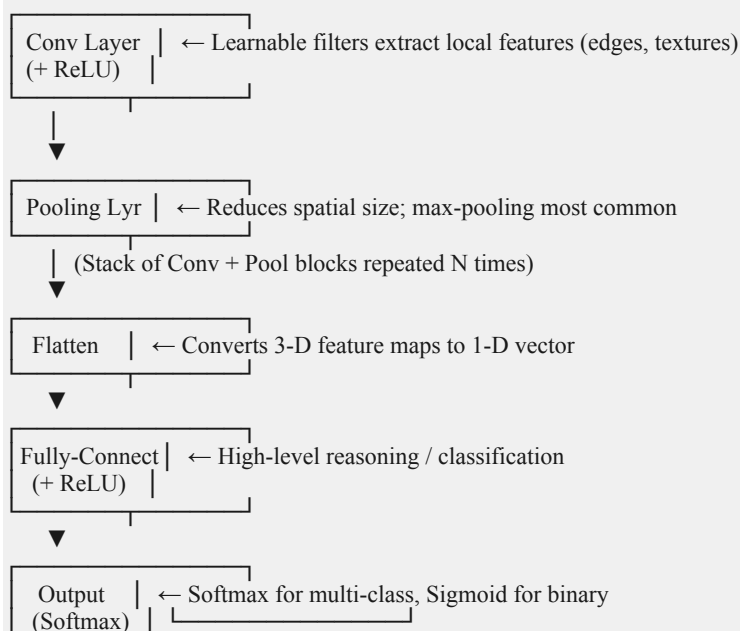
Q1b Draw and explain CNN (Convolution Neural Network) architecture in detail. [6 Marks]

CNN Architecture

A Convolutional Neural Network is a specialised deep neural network designed for grid-structured data such as images. Its architecture chains layers that progressively extract, refine, and classify features.

Architecture Flowchart

Input Image
↓
▼



Layer-by-Layer Explanation

- **Input Layer:** Raw pixel values of the image (e.g., $224 \times 224 \times 3$ for RGB).
- **Convolutional Layer:** Applies a set of learnable filters (kernels) that slide over the input to produce feature maps. Each filter detects a specific local pattern. Operation: $\text{Output} = \text{ReLU}(\text{Input} * \text{Filter} + \text{bias})$.
- **Activation (ReLU):** Introduces non-linearity. Replaces negative values with 0: $f(x) = \max(0, x)$.
- **Pooling Layer:** Down-samples the feature maps (see Q1a). Reduces dimensions while preserving dominant features.
- **Flatten Layer:** Reshapes the multi-dimensional feature maps into a 1-D vector to feed into dense layers.
- **Fully Connected (Dense) Layers:** Perform classification or regression based on the extracted features.
- **Output Layer:** Uses Softmax for multi-class classification to produce a probability distribution over classes.

Applications of CNNs

- Image Classification (ImageNet, CIFAR-10)
- Object Detection (YOLO, Faster-RCNN)
- Face Recognition
- Medical Image Analysis (tumour detection, X-ray diagnosis)
- Self-driving car perception
- Natural Language Processing (1-D convolutions over text)

Note: Landmark CNN architectures: LeNet-5 (1998), AlexNet (2012), VGGNet, GoogLeNet (Inception), ResNet, EfficientNet. ResNet introduced residual/skip connections to allow training of very deep networks (100+ layers).

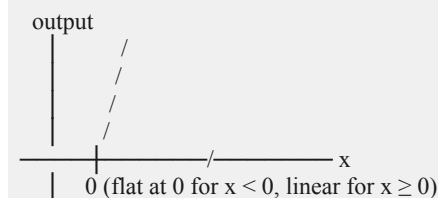
Q1c Explain ReLU Layer in detail. What are the advantages of ReLU over Sigmoid? [6 Marks]

ReLU (Rectified Linear Unit) — Activation Layer

ReLU is the most widely used activation function in deep learning. It is applied element-wise after a convolution (or any linear operation) to introduce non-linearity, enabling the network to learn complex decision boundaries.

Mathematical Definition

$$f(x) = \max(0, x)$$



Advantages of ReLU over Sigmoid

Property	ReLU vs Sigmoid
Vanishing Gradient	ReLU does NOT saturate for $x > 0$, so gradients flow freely. Sigmoid saturates near 0/1 causing near-zero gradients.
Computation Speed	ReLU is simply $\max(0, x)$ — extremely fast. Sigmoid requires exponentiation.
Sparsity	ReLU outputs exactly 0 for negative inputs, producing sparse activations — more efficient and interpretable.
Convergence Speed	Networks with ReLU converge $\sim 6\times$ faster than with tanh or sigmoid.
No Output Scaling Issue	ReLU output range is $[0, \infty)$. Sigmoid squashes to $(0, 1)$ causing vanishing gradients in deep nets.

Disadvantages of ReLU (and Variants)

- **Dying ReLU:** If a neuron's input is always negative, it outputs 0 permanently and never updates — the neuron 'dies'. Solutions: Leaky ReLU, ELU, Parametric ReLU.
- **Leaky ReLU:** $f(x) = \max(0.01x, x)$ — allows small negative slope to keep neurons alive.
- **ELU (Exponential Linear Unit):** Smooth for negative values, reduces bias shift.

Note: In practice, ReLU or its variants (Leaky ReLU, ELU, SELU) are the default choice for hidden layers. Sigmoid is still used at output layers for binary classification; softmax for multi-class.

Q2a Explain all the features of pooling layer.

[6 Marks]

[REPEATED] Pooling Layer is covered in Q1a above. Key additional features to emphasize:

- Pooling is a form of non-linear down-sampling.
- It has no learnable parameters — purely a fixed mathematical operation.
- Controls spatial hierarchy of features.
- Provides translation invariance within the pooling window.
- Reduces memory footprint and computational load for subsequent layers.
- Helps make representations approximately invariant to small translations, rotations, and scale

changes.

Q2b Explain Dropout Layer in Convolutional Neural Network.

[6 Marks]

Dropout Layer

Dropout is a regularisation technique introduced by Srivastava et al. (2014) to prevent overfitting in neural networks. During training, each neuron (along with its connections) is randomly 'dropped out' (set to zero) with a probability p (typically 0.5 for hidden layers, 0.2 for input layers).

How Dropout Works

Training Phase:

Full Layer: [N1][N2][N3][N4][N5]

After Drop: [N1][0][N3][0][N5] ← N2, N4 dropped ($p=0.4$)

Inference Phase:

All neurons active, but weights scaled by $(1-p)$ (or equivalently, training weights are scaled by $1/(1-p)$)

Why Dropout Works

- **Prevents co-adaptation:** Neurons can no longer rely on specific other neurons, forcing them to learn more robust independent features.
- **Ensemble effect:** With N neurons, 2^N possible subnetworks are trained, and at test time their predictions are implicitly averaged.
- **Reduces overfitting:** Acts as a strong regulariser, comparable in effect to L2 weight decay.

Dropout in CNNs vs Dense Layers

- In CNN feature layers, Spatial Dropout (dropping entire feature maps) is more effective than per-neuron dropout, preserving spatial structure.
- Standard per-neuron dropout is most effective in fully-connected layers.

Note: Dropout is only active during training. At inference, all neurons are active but outputs are scaled accordingly. Modern alternatives include DropBlock (drops contiguous regions in feature maps) and Stochastic Depth (drops entire residual blocks).

Q2c Explain working of Convolution Layer with its features.

[6 Marks]

Convolution Layer

The Convolution Layer is the core building block of a CNN. It learns spatial hierarchies of features by applying learnable filters (kernels) to the input feature map. The mathematical operation is a discrete cross-correlation (though conventionally called convolution in the ML community).

Working of Convolution

Input (5×5) Filter (3×3) Feature Map (3×3)

1	2	3	0	1	$\rightarrow \Sigma(\text{element-wise product}) + \text{bias}$
4	5	6	1	2	
7	8	9	2	3	
0	1	2	3	4	
5	6	7	8	9	

Filter slides left→right, top→bottom with given stride

Key Features of a Convolution Layer

- **Local Connectivity:** Each neuron in the output connects only to a small local region (receptive field) of the input — unlike fully connected layers.
- **Parameter Sharing:** The same filter is applied across the entire input, drastically reducing parameters. A 3×3 filter has only 9 weights regardless of input size.
- **Multiple Filters:** Multiple filters are applied in parallel, each learning a different feature. If there are K filters, the output has K channels (depth).
- **Stride:** Controls how many pixels the filter moves at each step. Stride 1 → dense sampling; Stride 2 → halves dimensions.
- **Padding:** Zero-padding preserves spatial dimensions after convolution.
- **Output Size Formula:** $\text{Output_size} = \lfloor (\text{Input_size} - \text{Filter_size} + 2 \times \text{Padding}) / \text{Stride} \rfloor + 1$

Note: Depth of each filter must match the depth of the input (e.g., 3 for RGB). The number of filters is a hyperparameter and sets the depth of the output feature map.

NOV-DEC 2023

Q1a Explain stride Convolution with example.

[6 Marks]

Stride in Convolution

Stride is the step size by which the convolution filter moves across the input. It directly controls the spatial dimensions of the output feature map.

Effect of Stride

Stride Value	Effect
Stride = 1	Filter moves 1 pixel at a time. Output is nearly the same size as input (with appropriate padding). Maximum information retention.
Stride = 2	Filter jumps 2 pixels at a time. Output is approximately halved in each dimension. Acts like pooling.
Stride > 2	Aggressive down-sampling; reduces computation but may lose fine detail.

Numerical Example — Stride 2

Input (5×5):
 1 2 3 4 5
 6 7 8 9 10
 11 12 13 14 15
 16 17 18 19 20
 21 22 23 24 25

Filter (3×3): Stride = 2

Filter applied at: Output (2×2)

top-left (0,0) → O[0,0]
 top-right(0,2) → O[0,1]
 bot-left (2,0) → O[1,0] bot-right(2,2) → O[1,1]

Output size = $\lfloor (5-3)/2 \rfloor + 1 = 2$

Advantages of Strided Convolution

- Replaces the need for a separate pooling layer, reducing depth of the network.
- The network can learn an optimal down-sampling rather than using a fixed pooling operation.
- Reduces computational cost and memory usage.

Note: Modern architectures (e.g., ResNet, DCGAN) prefer strided convolution over max-pooling for down-sampling, as it is learnable and thus more flexible.

Q1b Explain Padding and its types.

[6 Marks]

Padding in CNN

Padding refers to adding extra values (typically zeros) around the border of an input before applying a convolution filter. Without padding, each convolution reduces the spatial dimensions of the feature map, and pixels at the edges of the image are processed far fewer times than central pixels, losing information.

Types of Padding

- **Valid Padding (No Padding):** No extra pixels are added. The filter only slides where it fully fits inside the input. Output size = $\lfloor (W - F)/S \rfloor + 1$, where W = input size, F = filter size, S = stride. Information at borders is lost.
- **Same Padding (Zero Padding):** Zeros are added around the input so the output has the same spatial size as the input (when stride = 1). Amount of padding $P = (F - 1) / 2$. Preserves spatial dimensions and edge information.
- **Full Padding:** Enough zeros are added so the filter can slide over every position where it has at least one input pixel overlapping. Output is larger than input. Rarely used in classification networks.
- **Reflect Padding:** Pads with mirrored input values instead of zeros. Reduces border artifacts for image generation tasks.

Visual Comparison — Same vs Valid Padding

Input: 4×4 Filter: 3×3 Stride: 1

Valid Padding → Output: 2×2 (shrinks) Same Padding → Output: 4×4 (preserved, P=1 zero-padding on each side)

Note: Same Padding is used extensively in ResNets and any network where consistent spatial dimensions between layers are important (e.g., skip connections require matching sizes).

Q1c Explain Local Response Normalization and need of it.

[6 Marks]

Local Response Normalization (LRN)

Local Response Normalization is a normalization technique introduced in AlexNet (2012) that promotes competition among neurons activated by neighboring feature maps. It mimics the lateral inhibition observed in real neurons in the brain.

Mathematical Formula

$$b^i_{(x,y)} = a^i_{(x,y)} / (k + \alpha \cdot \sum (a^j_{(x,y)})^2)^{\beta}$$

where the sum is over j from $\max(0, i-n/2)$ to $\min(N-1, i+n/2)$

$a^i_{(x,y)}$ = activation at position (x,y) in feature map i

N = total number of feature maps

n = number of adjacent feature maps to normalize over k, α, β = hyperparameters ($k=2, n=5, \alpha=1e-4, \beta=0.75$ in AlexNet)

Need / Purpose of LRN

- Normalizes activations across adjacent feature maps at the same spatial position.
- Encourages feature maps with large activations to suppress neighboring maps — creates lateral inhibition.
- Provides a form of local contrast normalization, making highly activated neurons more pronounced relative to neighbors.
- Helps the network generalize better, especially for unnormalized (unbounded) activations like ReLU.

Limitations

- LRN has largely been replaced by Batch Normalization (BN), which is more principled, faster to train, and consistently more effective.
- LRN normalizes across feature maps (channel dimension), while BN normalizes across the batch and spatial dimensions.

Note: Batch Normalization (Ioffe & Szegedy, 2015) made LRN obsolete in practice. BN normalizes layer inputs using batch statistics (mean and variance) and introduces learnable scale (γ) and shift (β) parameters, allowing much faster training with higher learning rates.

Q2a Explain ReLU Layer and its advantages.

[6 Marks]

[REPEATED] Covered comprehensively in Q1c of May-June 2023. Key advantages summary:

- Avoids vanishing gradient problem — gradient is either 0 or 1 for negative/positive inputs.
- Computationally trivial: $f(x) = \max(0, x)$ — no exponentiation.
- Produces sparse activations — efficient and more biologically plausible.
- Networks converge significantly faster compared to sigmoid/tanh.

Q2b Explain Pooling layers and its types with examples. [6 Marks]

[REPEATED] Pooling Layer covered in detail in Q1a of May-June 2023. See that section for full explanation and types.

Q2c What are the applications of Convolution with examples?

[6 Marks]

Applications of Convolution in Deep Learning

- **Image Classification:** CNNs classify images into categories. Example: ResNet classifying 1000 ImageNet classes; VGG achieving human-level performance.
- **Object Detection:** Detect and localise multiple objects in a scene. Example: YOLO (You Only Look Once) for real-time detection; Faster-RCNN using region proposal networks.
- **Semantic Segmentation:** Assign a class label to every pixel. Example: FCN (Fully Convolutional Network) for scene parsing; DeepLab for medical imaging.
- **Face Recognition:** Identify individuals from facial images. Example: FaceNet, DeepFace (Facebook).
- **Medical Image Analysis:** Detect tumours, segment organs, analyse X-rays. Example: U-Net architecture for biomedical image segmentation.
- **Natural Language Processing:** 1-D convolutions over word embeddings for text classification. Example: TextCNN for sentiment analysis.
- **Autonomous Driving:** Road segmentation, pedestrian detection, lane detection. Example: CNNs in Tesla's Autopilot, Waymo.
- **Image Generation:** Transposed convolutions (deconvolution) in GANs to upsample and generate images. Example: DCGAN.

Note: The key insight enabling all these applications is that convolution exploits local spatial structure and parameter sharing, making it data-efficient and powerful for any data with grid-like topology (images, audio spectrograms, video frames, time-series).

MAY-JUNE 2024

Q1a Explain CNN architecture with its application.

[6 Marks]

[REPEATED] CNN Architecture is covered comprehensively in Q1b of May-June 2023. Refer to that section for the detailed layered diagram and explanations. Applications are also listed there.

Q1b What is Padding? Enlist and explain types of padding.

[6 Marks]

[REPEATED] Padding is covered in full in Q1b of Nov-Dec 2023. Refer to that section for all types (Valid, Same, Full, Reflect) with formulas.

Q1c Explain Dropout Layer in Convolutional Neural Network.

[6 Marks]

[REPEATED] Dropout Layer is explained in detail in Q2b of May-June 2023. Refer to that section.

Q2a Define ReLU. Explain disadvantages of ReLU.

[6 Marks]

ReLU — Definition

ReLU (Rectified Linear Unit) is a piecewise linear activation function defined as $f(x) = \max(0, x)$. It outputs the input directly if positive, otherwise outputs zero. It is the most widely used activation function in hidden layers of deep neural networks due to its simplicity and effectiveness.

Disadvantages of ReLU

- **Dying ReLU Problem:** If a neuron's weights update such that it always receives negative input, it permanently outputs 0 and its gradients are also 0 — the neuron 'dies' and stops learning. This is especially problematic with high learning rates.
- **Not Zero-Centred:** ReLU outputs are always ≥ 0 . When inputs to the next layer are all positive, gradients are always positive or always negative for a given weight, causing zig-zag updates (inefficient convergence).
- **Unbounded Output:** ReLU has no upper bound, which can cause very large activations if not managed (e.g., with batch normalisation).
- **Not Differentiable at 0:** The derivative is undefined at $x = 0$ (though in practice, it is set to 0 or 0.5 and does not cause issues).
- **No Negative Information:** Negative inputs are completely discarded, losing potentially useful information.

Variants that Address These Issues

Variant	How It Fixes the Issue
Leaky ReLU	$f(x) = \max(0.01x, x)$ — allows small negative gradient, preventing dying neurons.
Parametric ReLU (PReLU)	Learns the slope for negative part as a parameter.
ELU	Exponential for negatives; smooth, zero-centred-like outputs.
SELU	Self-normalising — maintains mean=0 and std=1 across layers.
GELU	Gaussian-weighted; used in transformers (BERT, GPT).

Note: Despite its disadvantages, ReLU remains the default activation in CNNs due to its simplicity and empirical effectiveness. The dying ReLU problem is mitigated by using appropriate weight initialisation (He initialisation), lower learning rates, and batch normalisation.

Q2b What is Strides in CNN? Explain in brief. [6 Marks] **[REPEATED]** Stride is covered in detail in Q1a of Nov-Dec 2023. Refer to that section for full explanation with numerical example and comparison table.

Q2c Explain Pooling Layer with its different types. [6 Marks] **[REPEATED]** Pooling Layer is covered exhaustively in Q1a of May-June 2023. Refer to that section.

MAY-JUNE 2025 [REPEATED] All questions were repeated.

NOV-DEC 2025

Q1a List the main steps involved in training a CNN for image classification. Why is normalisation important before training a CNN. **[6 Marks]**

Steps in Training a CNN for Image Classification

Step 1 — Data Collection and Preparation

- Gather a large, representative labeled dataset (e.g., 50,000 images across 10 classes for CIFAR-10).
- Split into training, validation, and test sets (common split: 70 / 15 / 15%).

Step 2 — Data Preprocessing and Normalisation

- Resize all images to a fixed input size (e.g., 224×224 for ImageNet-trained networks).
- Normalise pixel values (detailed explanation below).
- Apply data augmentation: random crops, horizontal flips, colour jitter, rotation — artificially enlarges the dataset and reduces overfitting.

Step 3 — Model Architecture Design

- Choose or design the CNN architecture: number of conv layers, filter sizes, pooling strategy, number of FC layers, and output neurons (one per class).
- Select activation functions (ReLU for hidden layers, Softmax at output for multi-class classification).

Step 4 — Initialise Weights

- Use a principled initialisation strategy: He initialisation for ReLU networks (weights drawn from $N(0, \sqrt{2/n_{in}})$), Xavier for tanh/sigmoid.
- Poor initialisation causes vanishing or exploding activations from the very first forward pass.

Step 5 — Define Loss Function and Optimiser

- **Loss:** Cross-entropy loss for classification: $L = -\sum y_i \cdot \log(\hat{y}_i)$.
- **Optimiser:** Adam (adaptive learning rates, fast convergence) or SGD with momentum (often better final accuracy with careful tuning).
- **Learning rate scheduler:** Step decay, cosine annealing, or warm-up strategies improve convergence.

Step 6 — Forward Pass

- Feed a mini-batch of images through all layers to obtain predictions \hat{y} .
- Compute the loss L between predictions and ground-truth labels.

Step 7 — Backward Pass (Backpropagation)

- Compute gradients $\partial L / \partial W$ for every learnable weight W using the chain rule.
- Gradients flow back through the network from the output layer to the input layer.

Step 8 — Weight Update

Adam update (simplified):

$$\begin{aligned} m &\leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot g && \text{(first moment — momentum)} \\ v &\leftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot g^2 && \text{(second moment — adaptive scale)} \quad W \leftarrow W - \eta \cdot \hat{m} / (\sqrt{\hat{v}} + \epsilon) \quad \text{(parameter update)} \end{aligned}$$

Step 9 — Validation and Hyperparameter Tuning

- Evaluate on the validation set after each epoch to monitor for overfitting.
- Tune learning rate, batch size, weight decay, dropout rate based on validation accuracy.

Step 10 — Test Evaluation

- Run the final trained model on the held-out test set once — this gives the unbiased performance estimate.

Why Normalisation is Important Before Training

Normalisation transforms raw pixel values (typically 0–255) into a standardised range. The standard approach subtracts the dataset mean (per channel) and divides by the standard deviation:

$$x_{\text{normalised}} = (x - \mu_{\text{channel}}) / \sigma_{\text{channel}}$$

Example (ImageNet mean/std per RGB channel): $\mu = [0.485, 0.456, 0.406]$ $\sigma = [0.229, 0.224, 0.225]$

- **Prevents gradient scale mismatch:** Without normalisation, pixels in channel R (mean ≈ 100) and channel B (mean ≈ 40) have very different magnitudes. Gradients for filters connected to different channels scale proportionally, causing uneven and unstable updates.
- **Speeds up convergence:** Normalised inputs place all features in a similar range, allowing the loss surface to be more isotropic (circular contours rather than elongated ellipses). Gradient descent can then take direct steps toward the minimum rather than zig-zagging.
- **Enables higher learning rates:** With unnormalised data, using a high learning rate causes the large-magnitude gradients for high-value channels to overshoot. Normalisation makes the network robust to a wider range of learning rates.
- **Helps weight initialisation work correctly:** Schemes like Xavier and He initialisation assume inputs have zero mean and unit variance. Violating this assumption negates their benefit.

Note: Batch Normalisation (BN), applied inside the network after conv layers, provides a related but distinct benefit — it normalises the activations at each layer dynamically during training, not just the input. BN allows much higher learning rates and acts as a regulariser, often eliminating the need for dropout in many architectures.

Q1b Given input 64×64, kernel 5×5, stride=2, 'same' padding: (i) output feature map size; (ii) how does padding help retain spatial size. **[6 Marks]**

Output Feature Map Size — Worked Calculation

(i) Output Size with Same Padding

The general formula for output size after a convolution is:

$$\text{Output_size} = \text{floor}((\text{Input_size} - \text{Kernel_size} + 2 \times \text{Padding}) / \text{Stride}) + 1$$

For 'Same' padding the goal is: $\text{Output_size} = \text{ceil}(\text{Input_size} / \text{Stride})$

Given: Input = 64, Kernel = 5, Stride = 2

Step 1 — Required output size (same padding, stride 2):

$$\text{Output} = \text{ceil}(64 / 2) = 32$$

Step 2 — Solve for required padding P:

$$32 = \text{floor}((64 - 5 + 2P) / 2) + 1$$

$$31 = \text{floor}((59 + 2P) / 2)$$

To satisfy: $(59 + 2P) / 2 \geq 31 \rightarrow 59 + 2P \geq 62 \rightarrow P \geq 1.5$
 $\rightarrow P = 2$ (rounded up, applied as `pad_top=1`, `pad_bottom=2`, etc.)

Step 3 — Verify:

Output = $\text{floor}((64 - 5 + 2 \times 2) / 2) + 1$
 $= \text{floor}(63 / 2) + 1$
 $= 31 + 1 = 32$

✓ Output feature map size = 32×32

(ii) How Same Padding Retains Spatial Size

- **Without padding (valid padding):** Every convolution shrinks the feature map. With a 5×5 kernel and stride 1, a 64×64 input becomes 60×60 — losing 4 pixels on each dimension. After multiple layers, spatial information near the edges is discarded entirely.
- **With same padding:** Zero-rows and zero-columns are added around the input border so that filters can be centred on every input pixel, including the edge pixels. This ensures the output is exactly $\lceil \text{Input} / \text{Stride} \rceil$ — preserving spatial resolution at stride 1, or halving it cleanly at stride 2.
- **Edge pixel benefit:** Without padding, the corner and edge pixels of an image are covered by far fewer filter positions than central pixels, creating an effective 'blind spot'. Same padding equalises coverage.
- **Skip connections:** In architectures like ResNet, skip connections add the input of a block directly to its output. Same padding is essential here so the spatial dimensions match exactly for the element-wise addition.

Same Padding Visual (3×3 kernel, 5×5 input):

```
0 0 0 0 0 0 0
0 [x x x x x] 0  ← one ring of zeros added
0 [x x x x x] 0
0 [x x x x x] 0
0 [x x x x x] 0
0 [x x x x x] 0
0 0 0 0 0 0 0
```

The filter can now be centred on every cell of the 5×5 input. Output (stride 1) = 5×5 — spatial size fully preserved.

Note: In TensorFlow/Keras, `padding='same'` automatically computes and applies the correct padding. In PyTorch, you must specify `padding=(kernel_size-1)//2` manually for a square kernel with stride 1. For stride > 1, the relationship is more complex and TensorFlow's 'same' padding logic may pad asymmetrically.

Q1c What are CNNs primarily used for in deep learning? List at least four real-world applications.

[6 Marks]

[REPEATED] CNN applications are listed in Q1b of May-June 2023 (CNN architecture section) and Q2c of Nov-Dec 2023 (Applications of Convolution). Six applications are covered there: image classification, object detection, semantic segmentation, face recognition, medical imaging, NLP, autonomous driving, and image generation.

Q2a What is Interleaving Between Layers in CNN? Why is Interleaving Important? Explain the role of each interleaving layer.

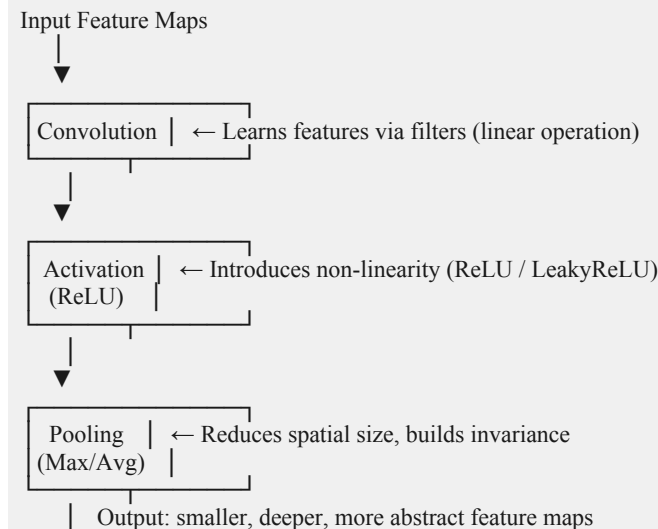
[6 Marks]

Interleaving Between Layers in CNN

'Interleaving' in a CNN refers to the deliberate alternating pattern of layer types — specifically the sequence of Convolution → Activation → Pooling that is repeated as a building block throughout the network. Rather than stacking all convolutions together or all activations together, these layers are interleaved so each type performs its distinct role in the correct order and at the correct stage of processing.

The Standard Interleaved Block

One Interleaved Block (repeated N times):



Why Interleaving is Important

- **Separation of concerns:** Each layer has a specific mathematical role. Mixing their order breaks the intended information processing pipeline — e.g., pooling before activation discards potentially useful activated features.
- **Progressive abstraction:** Each interleaved block compresses spatial information (pooling) while deepening the feature channels (convolution). Stacking N blocks creates a hierarchy: block 1 detects edges, block 2 detects shapes, block 3 detects object parts, etc.
- **Stable gradient flow:** The activation function immediately after convolution ensures that non-linearity is applied before pooling, keeping gradient magnitudes healthy during backpropagation.
- **Computational efficiency:** By pooling after each conv+activation pair, the spatial dimensions shrink progressively, so the next conv layer operates on a smaller map — reducing the total computation quadratically.

Role of Each Interleaved Layer

- **Convolutional Layer:** Performs the core feature extraction using learnable filters. Produces a linear combination of local input regions — identifies what patterns are present and where. Without this, there are no features to learn.
- **Activation Layer (ReLU):** Applies the non-linear transformation $f(x)=\max(0,x)$ element-wise. Without this non-linearity, stacking multiple conv layers is mathematically equivalent to a single conv layer (the composition of linear functions is linear). Non-linearity is what gives deep networks their expressive power.
- **Pooling Layer:** Down-samples feature maps — most commonly max-pooling takes the

maximum in each local region. This provides: (1) spatial invariance to small shifts/distortions, (2) reduction of spatial dimensions for fewer parameters in downstream layers, and (3) a broader receptive field for subsequent conv layers.

- **Batch Normalisation (often interleaved after conv):** Normalises the distribution of activations within a mini-batch. Placed between conv and activation, it stabilises training, allows higher learning rates, and reduces sensitivity to weight initialisation.

Note: Modern architectures like ResNet and DenseNet modify the interleaving order. ResNet uses BatchNorm → ReLU → Conv (pre-activation) rather than Conv → BN → ReLU. Some architectures (e.g., MobileNet) replace standard convolutions with depthwise-separable convolutions within the same interleaved pattern to dramatically reduce parameters while maintaining accuracy.

Q2b What is the ReLU activation function? Write its mathematical expression and describe how it transforms $x = [-3, -2, -1, 0, 2, 5, 8]$. **[6 Marks]**

ReLU — Definition and Numerical Application

The Rectified Linear Unit (ReLU) is the most widely used activation function in deep learning.

Mathematical Expression

$$f(x) = \max(0, x)$$

Equivalently:

$$\begin{aligned} f(x) &= x & \text{if } x > 0 \\ f(x) &= 0 & \text{if } x \leq 0 \end{aligned}$$

Derivative:

$$f'(x) = 1 \quad \text{if } x > 0 \quad f'(x) = 0 \quad \text{if } x \leq 0 \quad (\text{undefined at } x=0, \text{ set to 0 by convention})$$

Applying ReLU to $x = [-3, -2, -1, 0, 2, 5, 8]$

Input x	ReLU Output $f(x) = \max(0, x)$
-3	0 (negative → clamped to 0)
-2	0 (negative → clamped to 0)
-1	0 (negative → clamped to 0)
0	0 (zero → stays 0)
2	2 (positive → passed through unchanged)
5	5 (positive → passed through unchanged)
8	8 (positive → passed through unchanged)

Result vector: $f(x) = [0, 0, 0, 0, 2, 5, 8]$

Key observation: All negative values are zeroed out. This creates sparse activations (3 out of 4 negative values become 0). Only the three positive values are passed unchanged to the next layer, which is computationally efficient and biologically plausible (neurons either fire or they don't).

[REPEATED] Full advantages and disadvantages of ReLU are covered in Q1c of May-June 2023 and Q2a of May-June 2024. Refer to those sections.

Q2c Explain how input data flows through a typical CNN architecture from the raw image to the final output layer. **[6 Marks]**

[REPEATED] Data flow through CNN architecture is covered in Q1b of May-June 2023 with the full architecture flowchart (Input → Conv+ReLU → Pooling → Flatten → FC → Softmax) and layer-by-layer explanation.