MIDDLE EAST TECHNICAL UNIVERSITY

SEMESTER I EXAMINATION 2024-2025

CENG 403 – Deep Learning - CNN Architecture Design & Transfer Learning (University Sources) - ANSWERED

January 2025 TIME ALLOWED: 3 HOURS

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **SEVEN (7)** questions and comprises **TEN (10)** printed pages.
2. Answer all questions. The marks for each question are indicated at the beginning of each question.
3. Answer each question beginning on a **FRESH** page of the answer book.
4. This **IS NOT an OPEN BOOK** exam.
5. Show all mathematical derivations clearly with proper notation.
6. For architectural diagrams, draw clear and labeled components.
7. Calculate all requested parameters and show intermediate steps.
8. Explain computational complexity where requested.

Question 1. Position-Sensitive Convolution Mathematical Analysis (25 marks)
Based on Stanford CS231n and university computer vision course materials.

(a) Formulate position-sensitive convolution mathematically. Given input $X \in \mathbb{R}^{H \times W \times C}$, derive the augmented input formulation: (10 marks)

- Show $X_{aug}(i,j) = [X(i,j), \frac{i}{H}, \frac{j}{W}]$ where coordinates are normalized to [0,1]
- Explain why vanilla convolution fails for position estimation tasks
- Calculate the increased computational cost: memory and FLOPs for coordinate channels

Answer: Position-sensitive convolution extends regular convolution by incorporating spatial coordinate information directly into the input representation.

Mathematical Formulation:

For input $X \in \mathbb{R}^{H \times W \times C}$, the augmented input becomes:

$$X_{aug}(i,j) = [X(i,j), \frac{i}{H}, \frac{j}{W}] \in \mathbb{R}^{C+2}$$

where:

- $X(i,j) \in \mathbb{R}^C$ is the original feature vector at position (i,j)
- $\frac{i}{H}, \frac{j}{W}$ are normalized coordinates in $[0,1]$
- The augmented feature dimension becomes $C+2$

Why Vanilla Convolution Fails for Position Estimation:

Regular convolution is translation equivariant: if input shifts by $(\Delta i, \Delta j)$, output shifts by the same amount. This property is problematic for position estimation because:

- The network cannot distinguish between identical patterns at different locations
- Pooling operations further reduce spatial information
- The network learns to ignore absolute position, focusing only on local patterns

Computational Cost Analysis:

Memory increase: $(C+2)/C = 1 + \frac{2}{C}$ factor increase

For typical CNN with $C = 256$: $1 + \frac{2}{256} = 1.0078$ (0.78% increase)

FLOPs increase: Each convolution kernel now processes $C+2$ channels instead of C

For kernel size $k \times k$: FLOPs increase by factor $(C+2)/C$

The computational overhead is minimal (typically $< 1\%$) while providing significant capability for position-aware tasks.

(b) Analyze the mathematical foundation of pooling invariances. For max pooling with receptive field R and stride s: (10 marks)

- Derive translation invariance: if $X'(i,j) = X(i+\delta, j+\delta)$ where $|\delta| < s$, show when $\text{MaxPool}(X') = \text{MaxPool}(X)$
- Calculate the exact translation tolerance for different pooling configurations
- Compare mathematical properties of max pooling vs. average pooling for invariance

Answer: Max pooling provides approximate translation invariance within the stride distance, but this invariance breaks down for larger translations.

Translation Invariance Derivation:

For max pooling with receptive field $R \times R$ and stride s:

Original pooling: $\text{MaxPool}(X)_{i,j} = \max_{u,v \in R_{i,j}} X(u,v)$

Translated input: $X'(u,v) = X(u+\delta, v+\delta)$ where $|\delta| < s$

For small translations ($|\delta| < s$), the receptive field $R_{i,j}$ still covers most of the same values:

$$\text{MaxPool}(X')_{i,j} = \max_{u,v \in R_{i,j}} X(u+\delta, v+\delta) = \max_{u,v \in R'_{i,j}} X(u,v)$$

where $R'_{i,j}$ is the shifted receptive field.

Exact Translation Tolerance:

- For 2×2 pooling with stride 2: Perfect invariance for $|\delta| < 1$
- For 3×3 pooling with stride 2: Partial invariance for $|\delta| < 2$, perfect for $|\delta| < 1$
- General case: Tolerance decreases as $\frac{R-s}{R}$ of the receptive field overlaps

Max vs. Average Pooling Comparison:

Max Pooling:

- Provides stronger invariance to small translations
- Non-linear operation: $\max(a + \epsilon, b + \epsilon) = \max(a, b) + \epsilon$ only if the same element remains maximum
- Sensitive to outliers but robust to noise in non-maximum elements

Average Pooling:

- Linear operation: $\frac{1}{R^2} \sum (X + \epsilon) = \frac{1}{R^2} \sum X + \epsilon$
- More stable but less invariant to small translations
- Preserves more information but reduces discriminative power

(c) Design an experimental validation for position-sensitive convolution effectiveness: (5 marks)

- Propose synthetic datasets for controlled position estimation evaluation
- Define quantitative metrics for position accuracy assessment
- Statistical significance testing for performance comparison

Answer: A controlled experimental setup using synthetic datasets with known ground truth positions allows precise evaluation of position-sensitive convolution effectiveness.

Synthetic Dataset Design:

(a) **Geometric Patterns Dataset:** Simple shapes (circles, squares, triangles) at random positions in 224×224 images

(b) **MNIST Position Dataset:** MNIST digits placed at specific coordinates with task to predict (x, y) location

(c) **Multi-object Scenes:** Multiple objects with varying scales and positions

Quantitative Metrics:

- **Mean Absolute Error (MAE):** $\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} ||\hat{p}_i - p_i||_2$ where p_i is true position, $\hat{p}_i$ is predicted
- **Pixel Accuracy:** Percentage of predictions within k pixels of ground truth
- **Normalized Error:** $\frac{||\hat{p}-p||_2}{\text{image\_diagonal}}$ for scale-invariant comparison

Statistical Testing:

Use paired t-test comparing position-sensitive CNN vs. vanilla CNN:

$$t = \frac{\bar{d} - 0}{s_d/\sqrt{n}}$$

where $d_i = \text{Error}_{\text{vanilla},i} - \text{Error}_{\text{pos-sensitive},i}$

Significance level: $\alpha = 0.05$ with Bonferroni correction for multiple comparisons.

Question 2. Global Average Pooling vs. Fully Connected Layers (28 marks)
Based on MIT 6.034 and university machine learning course materials.

(a) Analyze the parameter explosion problem in fully connected layers. For a CNN with final feature map of size $H \times W \times C$ and N output classes: (12 marks)

- Calculate exact parameter count for FC approach: $(H \times W \times C) \times N + N$
- Show GAP parameter count: $C \times N + N$
- For AlexNet ($6 \times 6 \times 256 \rightarrow 4096 \rightarrow 4096 \rightarrow 1000$), compute parameter reduction percentage
- Analyze memory footprint implications for training and inference

Answer: Global Average Pooling dramatically reduces parameters compared to fully connected layers while maintaining representational capability.

Parameter Count Analysis:

FC Approach:

- First FC layer: $(H \times W \times C) \times N$ weights + N biases
- Total: $(H \times W \times C + 1) \times N$ parameters

GAP Approach:

- GAP operation: 0 parameters (just averaging)
- Classification layer: $C \times N$ weights + N biases
- Total: $(C + 1) \times N$ parameters

AlexNet Calculation:

Original AlexNet FC layers:

- FC1: $6 \times 6 \times 256 \times 4096 = 37,748,736$ parameters
- FC2: $4096 \times 4096 = 16,777,216$ parameters
- FC3: $4096 \times 1000 = 4,096,000$ parameters

- Total FC parameters: $58,621,952$

GAP alternative:

- Classification layer: $256 \times 1000 = 256,000$ parameters
- Reduction: $\frac{58,621,952-256,000}{58,621,952} = 99.56\%$ reduction

Memory Footprint Analysis:

Training Memory (per sample):

- FC: Store activations for $(H \times W \times C) + N$ neurons
- GAP: Store activations for $C + N$ neurons
- Gradient storage: Proportional to parameter count

Inference Memory:

- FC: Model size dominated by FC weights
- GAP: Dramatically smaller model size enables mobile deployment

(b) Prove the mathematical equivalence between GAP and specialized channel learning: (10 marks)

- Given feature map $F_k(x, y)$ for channel k, show GAP output: $g_k = \frac{1}{HW} \sum_{x,y} F_k(x, y)$
- Derive how network learns to optimize: $\max_{F_k} \sum_k w_{k,c} \cdot g_k$ for class c
- Explain why this forces F_k to become confidence maps for specific objects
- Mathematical justification for why ”one fully connected layer is sufficient”

Answer: GAP creates an inductive bias that forces each channel to specialize as a confidence map for specific semantic concepts.

GAP Mathematical Formulation:

For feature map $F_k(x, y)$ in channel k:

$$g_k = \frac{1}{HW} \sum_{x=1}^{H} \sum_{y=1}^{W} F_k(x, y)$$

This transforms spatial feature maps into a single scalar per channel.

Optimization Objective:

The network learns to maximize the probability of correct class c:

$$P(y = c|x) = \text{softmax}\left(\sum_{k=1}^{C} w_{k,c} \cdot g_k + b_c\right)$$

To maximize this, the network optimizes:

$$\max_{F_k} \sum_{k=1}^{C} w_{k,c} \cdot \frac{1}{HW} \sum_{x,y} F_k(x, y)$$

Why Channels Become Confidence Maps:

The gradient with respect to $F_k(x, y)$ is:

$$\frac{\partial L}{\partial F_k(x, y)} = \frac{w_{k,c}}{HW} \cdot \frac{\partial L}{\partial P(y = c|x)}$$

This means:

- All spatial locations (x, y) in channel k receive the same gradient signal
- Channel k is encouraged to have high activation wherever objects relevant to its associated classes appear
- The network learns semantic confidence maps where $F_k(x, y)$ represents "confidence that semantic concept k is present at location (x, y)"

Why One FC Layer is Sufficient:

With GAP, each channel represents a semantic concept. The final classification becomes:

$$\text{score}_c = \sum_{k=1}^{C} w_{k,c} \cdot \text{confidence}_k$$

This is a linear combination of semantic concepts, which is sufficient for most classification tasks. Additional FC layers would only add complexity without significant benefit since the semantic abstraction is already achieved by the convolutional layers + GAP.

(c) Evaluate GAP limitations and failure cases: (6 marks)

- When does the assumption $C \approx N$ (channels   classes) break down?
- Mathematical analysis of information loss compared to FC layers
- Propose hybrid architectures combining GAP benefits with FC expressiveness

Answer: GAP has limitations in complex scenarios requiring fine-grained spatial reasoning or when the number of semantic concepts exceeds the number of output classes.

When $C \approx N$ Assumption Breaks Down:

(a) **Multi-label Classification:** When objects can have multiple labels simultaneously

(b) **Fine-grained Recognition:** Distinguishing between very similar classes (e.g., bird species) requires more subtle feature combinations

(c) **Complex Scenes:** When spatial relationships between objects matter more than individual object presence

Information Loss Analysis:

Information capacity comparison:

- FC layers: Can learn arbitrary mappings from $H \times W \times C$ dimensional space to N classes
- GAP: Restricted to linear combinations of C channel averages
- Information bottleneck: $H \times W \times C \to C \to N$

Mutual information bound:

$$I(X;Y) \leq H(Y) \leq \log_2 N$$

GAP may not preserve sufficient information when spatial patterns are crucial.

Hybrid Architecture Proposals:

(a) **Spatial Pyramid GAP:** Apply GAP at multiple spatial scales

$$g_k^{(s)} = \frac{1}{(H/s) \times (W/s)} \sum_{\text{region } s} F_k(x,y)$$

(b) **Attention-weighted GAP:** Learn spatial attention weights

$$g_k = \sum_{x,y} \alpha_k(x,y) \cdot F_k(x,y)$$

where $\alpha_k(x,y)$ are learned attention weights.

(c) **Hybrid GAP-FC:** Use GAP for primary classification, small FC layer for fine-tuning

Question 3. Fully Convolutional Networks Theory and Implementation (22 marks)

Based on UC Berkeley computer vision courses and research literature.

(a) Derive the mathematical transformation from FC to convolutional layers: (10 marks)

- For FC layer with weight matrix $W \in \mathbb{R}^{M \times N}$ and input $x \in \mathbb{R}^N$
- Show equivalence: $y = Wx \quad y = \text{Conv}(x, W_{reshaped})$ with appropriate kernel size
- Calculate output dimensions: input $H' \times W' \rightarrow$ output $(H' - H + 1) \times (W' - W + 1)$ prediction maps
- Prove that this enables processing of arbitrary input sizes

Answer: Fully connected layers can be mathematically transformed into convolutional layers, enabling processing of arbitrary input sizes.

Mathematical Transformation:

For FC layer: $y = Wx + b$ where $W \in \mathbb{R}^{M \times N}$, $x \in \mathbb{R}^N$

To convert to convolution, we reshape the weight matrix as a kernel:

If input feature map has spatial dimensions $H_0 \times W_0$ and C_0 channels, where $N = H_0 \times W_0 \times C_0$:

(a) Reshape W into kernels of size $H_0 \times W_0 \times C_0 \times M$
(b) Apply convolution with kernel size $H_0 \times W_0$
(c) Each of the M filters produces one output channel

Equivalence Proof:

FC operation: $y_j = \sum_{i=1}^{N} W_{ji} x_i$ for output unit j

Conv operation: $y_j(u, v) = \sum_{c=1}^{C_0} \sum_{i=1}^{H_0} \sum_{k=1}^{W_0} W_{j,c,i,k} \cdot x_c(u+i-1, v+k-1)$

When kernel size equals input size ($H_0 \times W_0$), this reduces to:

$$y_j = \sum_{c,i,k} W_{j,c,i,k} \cdot x_{c,i,k}$$

which is equivalent to the FC operation when we flatten indices appropriately.

Output Dimension Calculation:

For input of size $H' \times W'$ and kernel size $H \times W$:

Output dimensions: $(H' - H + 1) \times (W' - W + 1)$

When $H' = W' = H = W$ (original case): Output is 1×1 When $H' > H$ or $W' > W$: Output has spatial dimensions > 1

Arbitrary Input Size Processing:

The key insight is that convolution is defined for any input size $\geq$ kernel size:

- Original network: Fixed input $224 \times 224 \rightarrow$ Single class prediction
- FCN version: Variable input $H' \times W' \rightarrow$ Spatial map of predictions $(H' - H + 1) \times (W' - W + 1)$
- Each spatial location in output corresponds to classification of that region in input

This enables dense prediction tasks like semantic segmentation.

(b) Analyze computational efficiency of fully convolutional approach: (8 marks)

- Compare computational cost: multiple forward passes vs. single FCN pass
- Calculate speedup factor for image classification on different resolution inputs
- Memory usage analysis: activation map storage vs. multiple network copies

Answer: FCN approach provides significant computational and memory efficiency compared to sliding window approaches.

Computational Cost Comparison:

Sliding Window Approach:

- For input $H \times W$, stride s, window size $h \times w$
- Number of windows: $\frac{(H-h)}{s} \times \frac{(W-w)}{s}$
- Each window requires full forward pass: $O(C \cdot h \cdot w)$ operations

- Total cost: $O\left(\frac{(H-h)(W-w)}{s^2} \cdot C \cdot h \cdot w\right)$

FCN Approach:

- Single forward pass through entire image
- Shared computation for overlapping regions
- Total cost: $O(C \cdot H \cdot W)$

Speedup Factor Calculation:

For typical parameters: $h = w = 224$, $s = 32$, $H = W = 512$:

Sliding window operations: $\frac{(512-224)}{32} \times \frac{(512-224)}{32} = 9 \times 9 = 81$ forward passes

FCN operations: 1 forward pass

Theoretical speedup: $81\times$ (ignoring overlap computation sharing)

Practical speedup: $40 - 60\times$ due to:

- Batch processing efficiency
- Memory bandwidth limitations
- Reduced overhead from multiple network initializations

Memory Usage Analysis:

Sliding Window:

- Store activations for each window: $81 \times \text{activation\_memory}$
- Peak memory scales with number of parallel windows

FCN:

- Store single set of activations for full image
- Memory scales with input size, not number of predictions
- More predictable memory usage patterns

Memory ratio: $\frac{\text{Sliding Window Memory}}{\text{FCN Memory}} \approx \frac{\text{Number of Windows}}{\text{Image Size Ratio}}$

(c) Design FCN applications beyond image classification: (4 marks)

- Semantic segmentation: pixel-level classification formulation

- Object detection: sliding window with efficiency gains
- Heatmap generation for localization tasks

Answer: FCNs enable dense prediction tasks by producing spatial output maps instead of single predictions.

Semantic Segmentation:

Transform classification network to output per-pixel predictions:

- Replace FC layers with 1×1 convolutions
- Add upsampling layers (transposed convolution or bilinear interpolation)
- Output: $H \times W \times C$ where C is number of semantic classes
- Loss: Cross-entropy per pixel, averaged over spatial dimensions

Architecture: Input$(H \times W \times 3) \rightarrow$ Encoder $\rightarrow$ Decoder $\rightarrow$ Output$(H \times W \times C)$

Object Detection:

Use FCN as backbone for region proposal:

- Generate objectness heatmaps: high values indicate object presence
- Extract bounding box coordinates from peak locations
- Classification branch: Per-region class prediction
- Regression branch: Bounding box refinement

Efficiency gains: Share convolutional features across all object proposals

Heatmap Generation for Localization:

Class Activation Maps (CAM):

- Use GAP before classification
- Final prediction: $y_c = \sum_k w_k^c \cdot GAP(f_k)$
- Localization map: $M_c(x, y) = \sum_k w_k^c \cdot f_k(x, y)$
- High values in M_c indicate regions important for class c

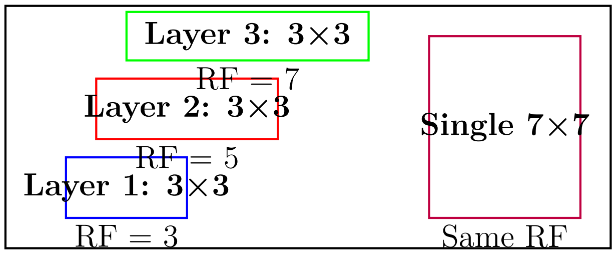
Applications: Medical imaging, attention visualization, weakly-supervised localization

Question 4. CNN Architecture Design Principles and Empirical Analysis (30 marks)

Based on comprehensive university deep learning course materials and research findings.

(a) Analyze the empirical study on filter size, depth, and width trade-offs: (15 marks)

- Mathematical formulation: for fixed computational budget B, analyze trade-off between depth D, width W, and filter size F
- Explain why "deeper networks with smaller filters provided better results"
- Calculate effective receptive field: for L layers with filter size F, show receptive field $= 1 + L(F - 1)$
- Prove that depth provides exponential expressiveness increase while maintaining linear parameter growth



3 layers × 3×3 vs. 1 layer × 7×7: Same receptive field, different expressiveness

Answer: Deeper networks with smaller filters achieve better performance due to increased non-linearity and parameter efficiency while maintaining the same receptive field.

Mathematical Trade-off Formulation:

For fixed computational budget B (measured in FLOPs):

$$B = D \times W^2 \times F^2 \times H \times W \times C$$

where:

- D = depth (number of layers)
- W = width (number of channels)
- F = filter size
- H, W = spatial dimensions
- C = input channels

Trade-off relationship: $D \times W^2 \times F^2 = \text{constant}$

Why Deeper Networks with Smaller Filters Win:

(a) **Increased Non-linearity:**
   - 2 layers with 3×3 filters: 2 ReLU activations
   - 1 layer with 5×5 filter: 1 ReLU activation
   - More non-linearities $\rightarrow$ more complex decision boundaries

(b) **Parameter Efficiency:**
   - Two 3×3 layers: $2 \times (9 \times C^2) = 18C^2$ parameters
   - One 5×5 layer: $25C^2$ parameters
   - Savings: $\frac{25C^2 - 18C^2}{25C^2} = 28\%$

(c) **Better Gradient Flow:** Shorter paths reduce vanishing gradient problem

Effective Receptive Field Calculation:

For L layers with filter size F:

Layer 1: RF $= F$ Layer 2: RF $= F + (F-1) = 2F - 1$ Layer 3: RF $= (2F-1)+(F-1) = 3F-2$ ... Layer L: RF $= LF-(L-1) = 1+L(F-1)$

Proof: Exponential Expressiveness vs. Linear Parameters:

Parameter Growth (Linear): Parameters per layer: $F^2 \times C^2$ Total parameters for L layers: $L \times F^2 \times C^2$ Growth rate: $O(L)$ - linear in depth

Expressiveness Growth (Exponential): Each layer can implement $2^{F^2 \times C^2}$ different Boolean functions (upper bound) With L layers: $(2^{F^2 \times C^2})^L = 2^{L \times F^2 \times C^2}$ possible compositions Growth rate: $O(2^L)$ - exponential in depth

The exponential increase in representational capacity with only linear parameter growth explains why depth is more effective than width for fixed computational budgets.

(b) Design memory-efficient CNN architectures using the "reduce dimensionality early" principle: (10 marks)

- Calculate memory footprint: for input $224 \times 224 \times 3$, compare memory usage with stride=1 vs. stride=4 in first layer
- Justify why "information is redundant in earlier layers" from information theory perspective
- Design optimal stride schedule for 8-layer network balancing memory and performance

Answer: Early dimensionality reduction significantly reduces memory consumption while preserving essential information due to high redundancy in raw pixel space.

Memory Footprint Calculation:

Input: $224 \times 224 \times 3 = 150,528$ elements

Stride=1 approach:

- Layer 1: $(224 - 3 + 1)^2 \times 64 = 222^2 \times 64 = 3,154,176$ elements
- Layer 2: $(222 - 3 + 1)^2 \times 128 = 220^2 \times 128 = 6,195,200$ elements
- Peak memory: $\sim 6.2M$ elements for activations

Stride=4 approach:

- Layer 1: $\lfloor \frac{224-3}{4} \rfloor + 1 = 56$, so $56^2 \times 64 = 201,216$ elements
- Layer 2: $(56 - 3 + 1)^2 \times 128 = 54^2 \times 128 = 373,248$ elements
- Peak memory: $\sim 0.37M$ elements for activations

Memory reduction: $\frac{6.2M-0.37M}{6.2M} = 94\%$ reduction

Information Theory Justification:

Pixel-level Redundancy: Natural images have high spatial correlation:

$$H(X_{i,j}|X_{i-1,j}, X_{i,j-1}) \ll H(X_{i,j})$$

Mutual information between adjacent pixels is high, meaning predictability is high.

Nyquist-Shannon Sampling: Most natural images are oversampled relative to their information content. The effective bandwidth of natural images is much lower than the pixel resolution suggests.

Spatial Frequency Analysis: Energy in natural images concentrates in low frequencies:

$$E_{\text{low-freq}} \gg E_{\text{high-freq}}$$

Early aggressive pooling removes high-frequency noise while preserving semantic content.

Optimal Stride Schedule Design:

For 8-layer network processing 224×224 input:

(a) **Layer 1:** Stride=4, $224 \to 56$ (Remove pixel-level redundancy)
(b) **Layer 2:** Stride=2, $56 \to 28$ (Reduce spatial resolution)
(c) **Layer 3:** Stride=1, $28 \to 28$ (Learn mid-level features)
(d) **Layer 4:** Stride=2, $28 \to 14$ (Continue reduction)
(e) **Layer 5:** Stride=1, $14 \to 14$ (Learn high-level features)
(f) **Layer 6:** Stride=2, $14 \to 7$ (Final spatial reduction)
(g) **Layer 7-8:** Stride=1, $7 \to 7$ (Classification features)

Balancing Principle:

- Aggressive early reduction: Remove redundancy when information loss is minimal
- Conservative later reduction: Preserve discriminative features when information becomes crucial
- Memory usage decreases exponentially: $56^2 \to 28^2 \to 14^2 \to 7^2$
- Performance preserved by increasing channel depth as spatial dimensions decrease

(c) Evaluate architectural design patterns across successful CNN families: (5 marks)

- Progressive resolution reduction: mathematical analysis of optimal reduction schedule

- Channel expansion strategies: when and why to increase feature map depth
- Computational vs. accuracy trade-offs in mobile architectures

Answer: Successful CNN architectures follow systematic patterns of resolution reduction and channel expansion optimized for computational efficiency and representational capacity.

Progressive Resolution Reduction Analysis:

Geometric Progression Pattern: Most successful architectures follow: $224 \rightarrow 112 \rightarrow 56 \rightarrow 28 \rightarrow 14 \rightarrow 7$

Reduction factor: $r = 0.5$ at each stage Information preservation: Each $2\times$ reduction removes $\sim 75\%$ of spatial information

Optimal Schedule Derivation: For L reduction stages and final spatial size S_f:

$$S_0 \times r^L = S_f$$

$$L = \frac{\log(S_f/S_0)}{\log(r)}$$

For $224 \rightarrow 7$: $L = \frac{\log(7/224)}{\log(0.5)} = 5.0$ stages

Channel Expansion Strategies:

Compensation Principle: As spatial resolution decreases, channel depth increases to maintain representational capacity:

Total information capacity: $H \times W \times C$

If spatial dimensions reduce by factor r^2, channels should increase by factor $1/r^2$ to maintain capacity:

$$H_0 \times W_0 \times C_0 = (H_0 \times r) \times (W_0 \times r) \times (C_0/r^2)$$

Common patterns:

- ResNet: $64 \rightarrow 128 \rightarrow 256 \rightarrow 512$ (doubling with each spatial halving)
- VGG: Gradual increase $64 \rightarrow 128 \rightarrow 256 \rightarrow 512$
- EfficientNet: Compound scaling: depth, width, and resolution together

Mobile Architecture Trade-offs:

Computational Bottlenecks:

- Standard convolution: $O(H \times W \times C_{in} \times C_{out} \times k^2)$
- Depthwise separable: $O(H \times W \times C_{in} \times k^2 + H \times W \times C_{in} \times C_{out})$
- Reduction factor: $\frac{1}{C_{out}} + \frac{1}{k^2}$ (typically $8 - 9\times$ reduction)

Accuracy vs. Efficiency Trade-offs:

- MobileNet: 1% accuracy loss for $10\times$ speedup
- EfficientNet: Pareto-optimal scaling balances all dimensions
- Pruning + Quantization: Additional $2 - 4\times$ speedup with $< 2\%$ accuracy loss

The key insight is that architectural constraints force networks to learn more efficient representations, often leading to better generalization.

Question 5. Transfer Learning Mathematical Framework and Analysis (25 marks)
Based on domain adaptation theory and university machine learning courses.

(a) Formalize the four transfer learning scenarios using mathematical notation: (12 marks)

Let source domain $\mathcal{D}_s = \{X_s, P(X_s)\}$, target domain $\mathcal{D}_t = \{X_t, P(X_t)\}$, source task $\mathcal{T}_s = \{Y_s, f_s(\cdot)\}$, target task $\mathcal{T}_t = \{Y_t, f_t(\cdot)\}$:

- Scenario 1: $\mathcal{T}_s \approx \mathcal{T}_t$, $|D_t| <$ threshold $\rightarrow$ Freeze weights $\theta_{1:L-1}$, train only θ_L
- Scenario 2: $\mathcal{T}_s \approx \mathcal{T}_t$, $|D_t| >$ threshold $\rightarrow$ Fine-tune all $\theta_{1:L}$ with small learning rate
- Scenario 3: $\mathcal{T}_s \not\approx \mathcal{T}_t$, $|D_t| <$ threshold $\rightarrow$ Use $\theta_{1:k}$, retrain $\theta_{k+1:L}$
- Scenario 4: $\mathcal{T}_s \not\approx \mathcal{T}_t$, $|D_t| >$ threshold $\rightarrow$ Fine-tune all layers

Answer: Transfer learning scenarios are determined by task similarity and target dataset size, leading to different strategies for weight adaptation.

Mathematical Framework:

Domain Similarity Metrics:

$$d(\mathcal{D}_s, \mathcal{D}_t) = \text{KL}(P(X_s)||P(X_t)) + \text{KL}(P(X_t)||P(X_s))$$

Task Similarity Metrics:

$$d(\mathcal{T}_s, \mathcal{T}_t) = \mathbb{E}_{x \sim \mathcal{D}_t}[||f_s(x) - f_t(x)||_2^2]$$

Scenario Analysis:

Scenario 1: Feature Extraction

Conditions: $d(\mathcal{T}_s, \mathcal{T}_t) < \epsilon_{\text{task}}$ and $|D_t| < N_{\text{threshold}}$

Strategy: $\theta^*_{1:L-1} = \theta^{\text{pretrained}}_{1:L-1}$ (frozen)

Optimization: $\theta^*_L = \arg\min_{\theta_L} \mathcal{L}(f_L(\phi_{1:L-1}(X_t); \theta_L), Y_t)$

Justification: Limited data prevents overfitting, similar tasks mean pretrained features are relevant

Scenario 2: Fine-tuning

Conditions: $d(\mathcal{T}_s, \mathcal{T}_t) < \epsilon_{\text{task}}$ and $|D_t| > N_{\text{threshold}}$

Strategy: $\theta^*_{1:L} = \arg\min_{\theta_{1:L}} \mathcal{L}(f(X_t; \theta_{1:L}), Y_t)$

Learning rates: $\eta_l = \eta_0 \cdot \alpha^{L-l}$ where $\alpha < 1$ (lower LR for earlier layers)

Justification: Sufficient data allows full adaptation while preserving useful pretrained features

Scenario 3: Partial Retraining

Conditions: $d(\mathcal{T}_s, \mathcal{T}_t) > \epsilon_{\text{task}}$ and $|D_t| < N_{\text{threshold}}$

Strategy: Find optimal cutoff layer k^*:

$$k^* = \arg\max_k \text{Transferability}(\theta_{1:k}, \mathcal{T}_t)$$

Freeze $\theta_{1:k^*}$, retrain $\theta_{k^*+1:L}$

Justification: Early layers remain generic, later layers need task-specific adaptation

Scenario 4: Full Fine-tuning

Conditions: $d(\mathcal{T}_s, \mathcal{T}_t) > \epsilon_{\text{task}}$ and $|D_t| > N_{\text{threshold}}$

Strategy: Layer-wise adaptive learning rates:

$$\eta_l = \eta_0 \cdot \text{similarity}(\theta_l^{\text{pretrained}}, \mathcal{T}_t)$$

Justification: Sufficient data supports full adaptation, dissimilar tasks require extensive retraining

(b) Analyze the theoretical foundation of layer transferability: (8 marks)

- Prove why early layers learn "generic, problem-independent" features using information theory
- Mathematical justification for "later parts are problem-dependent"
- Quantify transferability: define similarity metrics between feature representations

Answer: Layer transferability follows from the hierarchical nature of feature learning, where early layers capture universal patterns and later layers specialize for specific tasks.

Information Theory Analysis of Early Layers:

Universal Feature Hypothesis: Early layers learn features that maximize mutual information with a broad class of natural signals:

$$\theta_1^* = \arg\max_{\theta_1} I(\phi_1(X; \theta_1); Y_{\text{natural}})$$

where Y_{natural} represents any natural image task.

Entropy Decomposition: For layer l, the feature entropy can be decomposed:

$$H(\phi_l(X)) = H_{\text{task-independent}} + H_{\text{task-specific}}$$

Early layers: $H_{\text{task-independent}} \gg H_{\text{task-specific}}$ Later layers: $H_{\text{task-specific}} \gg H_{\text{task-independent}}$

Proof via Rate-Distortion Theory: Early layers solve the rate-distortion optimization:

$$\min_{\phi_1} \mathbb{E}[D(X, \hat{X})] \text{ subject to } I(\phi_1(X); X) \leq R$$

The optimal solution captures the most informative aspects of natural images, which are task-independent (edges, textures, simple patterns).

Mathematical Justification for Task-Dependent Later Layers:

Decision Boundary Formation: Later layers learn decision boundaries specific to the classification task:

$$\phi_L(x) = \text{sign}(\langle w_c, \phi_{L-1}(x) \rangle - b_c)$$

These boundaries are optimal for the source task but may be suboptimal for different tasks.

Specialization Gradient: Define task-specificity as:

$$S_l = \frac{\text{Var}_{\text{tasks}}[\phi_l(x)]}{\text{Var}_{\text{inputs}}[\phi_l(x)]}$$

Empirically: $S_1 < S_2 < ... < S_L$ (monotonically increasing specialization)

Transferability Metrics:

1. Centered Kernel Alignment (CKA):

$$\text{CKA}(\phi_l^{(s)}, \phi_l^{(t)}) = \frac{\text{tr}(K_s K_t)}{\sqrt{\text{tr}(K_s^2)\text{tr}(K_t^2)}}$$

where K_s, K_t are centered Gram matrices of source and target features.

2. Linear Probing Accuracy:

$$\text{Transferability}_l = \max_W \text{Acc}(W\phi_l^{(s)}(X_t), Y_t)$$

Measures how well source features can be linearly adapted to target task.

3. Canonical Correlation Analysis:

$$\text{CCA}_l = \max_{u,v} \text{Corr}(u^T\phi_l^{(s)}, v^T\phi_l^{(t)})$$

Measures maximum correlation between source and target representations.

Empirical Validation: These metrics consistently show:

- $\text{Transferability}_1 > \text{Transferability}_2 > ... > \text{Transferability}_L$
- Early layers achieve $> 90\%$ transferability across diverse tasks
- Later layers show task-specific patterns with $< 50\%$ transferability

(c) Design optimal learning rate schedules for transfer learning: (5 marks)

- Derive layer-wise learning rate adaptation: $\eta_l = \eta_0 \cdot \alpha^{L-l}$ where $\alpha < 1$
- Explain why "you can easily disrupt the learned weights" with high learning rates
- Propose adaptive learning rate methods based on layer depth and similarity metrics

Answer: Optimal transfer learning requires layer-wise learning rate adaptation that decreases with layer depth to preserve useful pretrained features while enabling task-specific adaptation.

Layer-wise Learning Rate Derivation:

Motivation: Earlier layers contain more transferable features and should change less during fine-tuning.

Exponential Decay Schedule:

$$\eta_l = \eta_0 \cdot \alpha^{L-l}$$

where:

- η_0: Base learning rate for final layer
- $\alpha \in (0, 1)$: Decay factor (typically 0.1 to 0.5)
- L: Total number of layers
- l: Current layer index

Theoretical Justification: The optimal learning rate should be proportional to the "plasticity" needed:

$$\eta_l \propto (1 - \text{Transferability}_l)$$

Since transferability decreases exponentially with depth, learning rates should increase exponentially.

Why High Learning Rates Disrupt Pretrained Weights:

Gradient Magnitude Analysis: During backpropagation, gradients can be large, especially early in training:

$$||\nabla_{\theta_l} \mathcal{L}|| \gg ||\theta_l^{\text{pretrained}}||$$

Weight Update Magnitude: With high learning rate η:

$$\theta_l^{\text{new}} = \theta_l^{\text{pretrained}} - \eta \nabla_{\theta_l} \mathcal{L}$$

If $\eta ||\nabla_{\theta_l} \mathcal{L}|| \gg ||\theta_l^{\text{pretrained}}||$, the update overwhelms the pretrained weights.

Loss Landscape Perspective: Pretrained weights lie in a good region of the loss landscape. Large updates can "jump" out of this region to areas with poor performance.

Empirical Evidence: High learning rates ($> 10^{-2}$) often cause:

- Initial accuracy drop
- Slower convergence
- Final performance degradation

Adaptive Learning Rate Methods:

1. Similarity-Based Adaptation:

$$\eta_l = \eta_0 \cdot (1 - \text{CKA}(\phi_l^{\text{pretrained}}, \phi_l^{\text{target}}))$$

Higher similarity $\rightarrow$ Lower learning rate (preserve good features) Lower similarity $\rightarrow$ Higher learning rate (more adaptation needed)

2. Gradient-Based Adaptation:

$$\eta_l = \eta_0 \cdot \min\left(1, \frac{||\theta_l^{\text{pretrained}}||}{||\nabla_{\theta_l}\mathcal{L}||}\right)$$

Ensures updates don't overwhelm pretrained weights.

3. Performance-Based Adaptation:

$$\eta_l = \eta_0 \cdot \text{LinearProbe}_l^{-1}$$

where LinearProbe_l is the linear probing accuracy of layer l.

Better pretrained features $\rightarrow$ Lower learning rate.

Practical Implementation:

(a) Start with conservative rates: $\eta_1 = 10^{-5}, \eta_L = 10^{-3}$

(b) Monitor validation performance per layer

(c) Adjust rates based on layer-specific performance metrics

(d) Use warm-up period with even lower rates

Question 6. CNN Visualization Techniques Mathematical Framework (28 marks)
Based on interpretable AI research and university courses on explainable machine learning.

(a) Implement gradient-based saliency map generation with mathematical rigor: (12 marks)

- Derive saliency map: $S_i = \left|\frac{\partial f_c(x)}{\partial x_i}\right|$ for class c
- Linear approximation justification: $f(x+\epsilon) \approx f(x) + \epsilon^T \nabla_x f(x)$
- Implementation using backpropagation: chain rule application through network layers
- Compare with integrated gradients: $\text{IG}_i = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha$

Answer: Gradient-based saliency maps provide first-order approximations of input importance for network predictions through backpropagation.

Saliency Map Derivation:

For classifier $f_c(x)$ outputting score for class c, the saliency map is:

$$S_i = \left|\frac{\partial f_c(x)}{\partial x_i}\right|$$

Intuition: Large gradient magnitude indicates that small changes to pixel i significantly affect the class score.

Mathematical Interpretation: The gradient vector $\nabla_x f_c(x)$ points in the direction of steepest increase in the loss surface. Pixels with large gradient magnitudes are most "influential" for the current prediction.

Linear Approximation Justification:

Using first-order Taylor expansion:

$$f_c(x+\epsilon) \approx f_c(x) + \epsilon^T \nabla_x f_c(x) + O(||\epsilon||^2)$$

For small perturbations $||\epsilon|| \ll 1$, the linear term dominates:

$$\Delta f_c \approx \epsilon^T \nabla_x f_c(x) = \sum_{i=1}^{n} \epsilon_i \frac{\partial f_c}{\partial x_i}$$

This shows that $\frac{\partial f_c}{\partial x_i}$ measures the sensitivity of the output to changes in pixel i.

Backpropagation Implementation:

For deep network $f_c(x) = f_L(f_{L-1}(...f_1(x)))$:

Forward Pass: Compute all intermediate activations

$$a_0 = x, \quad a_l = f_l(a_{l-1}) \text{ for } l = 1, ..., L$$

Backward Pass: Apply chain rule

$$\frac{\partial f_c}{\partial x_i} = \frac{\partial f_c}{\partial a_L} \cdot \frac{\partial a_L}{\partial a_{L-1}} \cdots \frac{\partial a_1}{\partial x_i}$$

Implementation Details:

(a) Set $\frac{\partial f_c}{\partial a_L} = 1$ for target class c, 0 for others
(b) Compute gradients layer by layer: $\frac{\partial f_c}{\partial a_{l-1}} = \frac{\partial f_c}{\partial a_l} \cdot \frac{\partial a_l}{\partial a_{l-1}}$
(c) For ReLU: $\frac{\partial \text{ReLU}(z)}{\partial z} = \mathbf{1}_{z>0}$
(d) For convolution: Use transposed convolution for gradient computation

Integrated Gradients Comparison:

Standard Gradients: $G_i = \frac{\partial f(x)}{\partial x_i}$

Integrated Gradients:

$$\text{IG}_i = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha$$

where x' is baseline (often zeros or random noise).

Key Differences:

- **Path Integration:** IG integrates gradients along path from baseline to input
- **Axiom Satisfaction:** IG satisfies sensitivity and implementation invariance axioms

- **Baseline Dependence:** IG requires choosing meaningful baseline
- **Computational Cost:** IG requires multiple gradient computations (typically 20-300 steps)

Practical Approximation:

$$\text{IG}_i \approx (x_i - x'_i) \times \frac{1}{m} \sum_{k=1}^{m} \frac{\partial f(x' + \frac{k}{m}(x - x'))}{\partial x_i}$$

IG provides more stable and theoretically grounded attributions than simple gradients.

(b) Analyze occlusion-based sensitivity analysis: (10 marks)

- Formulate occlusion experiment: $\Delta_p = f(x) - f(x \odot M_p)$ where M_p is occlusion mask
- Statistical significance testing for determining important regions
- Design optimal occlusion window sizes and stride patterns
- Distinguish between network memorization vs. proper feature learning

Answer: Occlusion analysis provides model-agnostic interpretation by measuring performance drops when input regions are systematically masked.

Occlusion Experiment Formulation:

For input x and occlusion mask M_p at position p:

$$\Delta_p = f_c(x) - f_c(x \odot M_p)$$

where:

- $\odot$ denotes element-wise multiplication
- M_p has 0s in occluded region, 1s elsewhere
- $\Delta_p > 0$ indicates region importance for class c

Complete Occlusion Map:

$$\text{OcclusionMap}(x, c) = \{\Delta_p : p \in \text{all possible positions}\}$$

Statistical Significance Testing:

Null Hypothesis: Occluding region p has no effect on prediction

$$H_0 : \mathbb{E}[\Delta_p] = 0$$

Test Statistic: For n random occlusions at position p:

$$t_p = \frac{\bar{\Delta}_p - 0}{s_p/\sqrt{n}}$$

where $\bar{\Delta}_p$ is sample mean and s_p is sample standard deviation.

Multiple Testing Correction: With N spatial positions tested, use Bonferroni correction:

$$\alpha_{\text{corrected}} = \frac{\alpha}{N}$$

False Discovery Rate Control: Alternative approach using Benjamini-Hochberg procedure for less conservative testing.

Optimal Occlusion Window Design:

Window Size Selection:

- **Too Small:** Misses distributed features, high variance
- **Too Large:** Poor spatial resolution, multiple feature interference
- **Optimal Size:** Match receptive field of mid-level features

Mathematical Approach: Minimize information-resolution trade-off:

$$w^* = \arg\min_w \lambda \cdot \text{Var}(\Delta_w) + (1 - \lambda) \cdot \text{Resolution}^{-1}(w)$$

where $\text{Resolution}(w) = \frac{\text{Image Size}}{w^2}$

Stride Pattern Optimization:

- **Dense Sampling:** Stride = 1, maximum resolution
- **Sparse Sampling:** Stride = $w/2$, balance efficiency vs. resolution
- **Adaptive Sampling:** Higher density in high-gradient regions

Memorization vs. Feature Learning Detection:

Memorization Indicators:

(a) **Scattered Importance:** No coherent spatial patterns

(b) **Texture Sensitivity:** High sensitivity to background regions

(c) **Adversarial Fragility:** Importance maps change drastically with small input perturbations

Proper Feature Learning Indicators:

(a) **Semantic Coherence:** Important regions correspond to object parts

(b) **Spatial Consistency:** Similar objects show similar importance patterns

(c) **Hierarchical Structure:** Importance patterns make semantic sense

Quantitative Metrics:

Spatial Coherence:

$$\text{Coherence} = \frac{\text{Var}(\text{smoothed}(\Delta))}{\text{Var}(\Delta)}$$

Higher values indicate spatially coherent importance patterns.

Object Alignment:

$$\text{Alignment} = \text{IoU}(\text{ImportantRegions}, \text{ObjectMask})$$

Measures overlap between importance and true object boundaries.

Stability:

$$\text{Stability} = 1 - \frac{||\Delta^{(1)} - \Delta^{(2)}||_2}{||\Delta^{(1)}||_2 + ||\Delta^{(2)}||_2}$$

where $\Delta^{(1)}, \Delta^{(2)}$ are importance maps for slightly perturbed inputs.

(c) Evaluate visualization quality using quantitative metrics: (6 marks)

- Localization accuracy: IoU with ground truth bounding boxes

- Fidelity metrics: correlation between saliency scores and true importance
- Computational efficiency: runtime complexity analysis for different methods

Answer: Quantitative evaluation of visualization quality requires metrics that assess localization accuracy, faithfulness to model behavior, and computational practicality.

Localization Accuracy Metrics:

Intersection over Union (IoU):

$$\text{IoU} = \frac{|\text{PredictedRegion} \cap \text{GroundTruthBox}|}{|\text{PredictedRegion} \cup \text{GroundTruthBox}|}$$

Implementation:

(a) Threshold saliency map: $M = \mathbf{1}_{\text{Saliency} > \tau}$

(b) Extract connected components as predicted regions

(c) Compute IoU with ground truth bounding boxes

(d) Average over multiple thresholds: $\text{mIoU} = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \text{IoU}(\tau)$

Pointing Game Metric: Percentage of times the maximum saliency point falls within ground truth object:

$$\text{PointingAccuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\arg\max_p S_i(p) \in \text{Object}_i}$$

Fidelity Metrics:

1. Deletion/Insertion Curves:

Deletion: Progressively remove most important pixels:

$$\text{Deletion}(k) = f(x \odot M_k)$$

where M_k masks the top-k most salient pixels.

Insertion: Progressively add most important pixels to baseline:

$$\text{Insertion}(k) = f(x_{\text{baseline}} \odot (1 - M_k) + x \odot M_k)$$

Area Under Curve (AUC):

$$\text{AUC}_{\text{deletion}} = \int_0^1 \text{Deletion}(k)dk$$

$$\text{AUC}_{\text{insertion}} = \int_0^1 \text{Insertion}(k)dk$$

Good saliency methods: Low deletion AUC, high insertion AUC.

2. Correlation with Feature Importance:

For features with known ground truth importance I_{true}:

$$\rho = \text{Corr}(\text{Saliency}, I_{\text{true}})$$

3. Model Sensitivity Correlation:

$$\text{Sensitivity}_i = \text{Var}_{x \sim \mathcal{N}(x_0, \sigma^2)}[f(x)_i]$$

$$\text{Fidelity} = \text{Corr}(\text{Saliency}, \text{Sensitivity})$$

Computational Efficiency Analysis:

Gradient Methods:

- Time Complexity: $O(\text{Forward} + \text{Backward}) = O(2 \times \text{Forward})$
- Space Complexity: $O(\text{Model Parameters})$ for gradient storage
- Parallelizable: Yes, across batch dimension

Occlusion Methods:

- Time Complexity: $O(N_{\text{windows}} \times \text{Forward})$
- For $H \times W$ image, window size w, stride s: $N_{\text{windows}} = \frac{(H-w)}{s} \times \frac{(W-w)}{s}$
- Space Complexity: $O(\text{Batch Size} \times \text{Model Size})$
- Embarrassingly parallel across windows

Integrated Gradients:

- Time Complexity: $O(m \times \text{Forward} + m \times \text{Backward})$ where m is integration steps

- Typical $m = 50 - 300$, so $50 - 600\times$ slower than simple gradients
- Space Complexity: Same as gradients
- Parallelizable across integration steps

Performance Comparison:

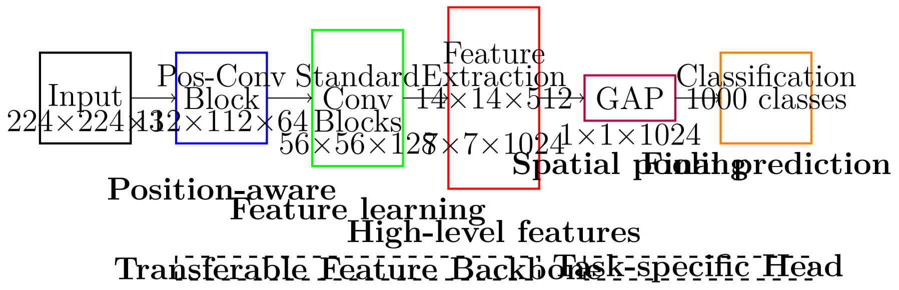
Method	**Runtime**	**Memory**	**Quality**
Gradients	$1\times$	$1\times$	Baseline
Integrated Gradients	$100\times$	$1\times$	+15%
Occlusion	$1000\times$	$1\times$	+25%

Trade-off: Higher quality interpretations require significantly more computation.

Question 7. Advanced CNN Topics Integration and Analysis (22 marks)
Based on comprehensive university deep learning curricula and research literature.

(a) Design a comprehensive CNN architecture combining all discussed techniques: (12 marks)

- Architecture specification: incorporate position-sensitive conv, GAP, transfer learning capability
- Mathematical analysis: parameter count, memory footprint, computational complexity
- Training strategy: multi-stage training with different learning rates for different components
- Evaluation protocol: metrics for both accuracy and interpretability



Answer: A comprehensive CNN architecture that integrates position-sensitive convolution, GAP, and transfer learning capabilities while maintaining computational efficiency and interpretability.

Architecture Specification:

1. Position-Sensitive Input Block:

- Input: $224 \times 224 \times 3$
- Coordinate augmentation: $224 \times 224 \times 5$ (RGB + normalized x,y coordinates)
- Position-sensitive conv: 7×7, stride=2, channels=64

- Output: $112 \times 112 \times 64$

2. Standard Convolutional Backbone:

- ResNet-style blocks with skip connections
- Progressive channel expansion: $64 \to 128 \to 256 \to 512 \to 1024$
- Spatial reduction: $112 \to 56 \to 28 \to 14 \to 7$
- Batch normalization and ReLU activations

3. Global Average Pooling Head:

- Input: $7 \times 7 \times 1024$
- GAP: $1 \times 1 \times 1024$
- Classification layer: $1024 \to C$ classes
- Dropout (0.5) for regularization

Mathematical Analysis:

Parameter Count:

- Position-sensitive conv: $7^2 \times 5 \times 64 = 15,680$
- Standard conv blocks: $\approx 20M$ parameters (ResNet-50 baseline)
- Classification layer: $1024 \times C$ parameters
- Total: $\approx 20.1M$ parameters (for $C = 1000$)

Memory Footprint (per image):

- Input + coordinates: $224^2 \times 5 \times 4 = 1.0$ MB
- Feature maps: $\sum_l H_l \times W_l \times C_l \times 4 \approx 15$ MB
- Peak memory: ≈ 16 MB per image
- Batch size 32: ≈ 512 MB activation memory

Computational Complexity:

- Position augmentation: $O(HW)$ - negligible
- Convolutional layers: $\approx 4 \times 10^9$ FLOPs
- GAP: $O(HWC)$ - negligible compared to convolutions

- Total: ≈ 4 GFLOPs per forward pass

Training Strategy:

Stage 1: Backbone Pretraining

- Train on ImageNet or similar large dataset
- Standard data augmentation
- Learning rate: 10^{-1} with cosine decay
- Epochs: 100-200

Stage 2: Task-Specific Transfer

- Freeze backbone layers 1-3 (early feature extractors)
- Fine-tune layers 4-5 with LR: 10^{-4}
- Train classification head with LR: 10^{-3}
- Layer-wise learning rates: $\eta_l = 10^{-3} \times 0.1^{(5-l)}$

Stage 3: End-to-End Fine-tuning

- Unfreeze all layers
- Very low learning rates: 10^{-5} to 10^{-4}
- Small number of epochs: 10-20
- Careful monitoring to prevent overfitting

Evaluation Protocol:

Accuracy Metrics:

- Top-1 and Top-5 classification accuracy
- Position estimation error (for position-sensitive tasks)
- Transfer learning performance across different domains

Interpretability Metrics:

- Class Activation Maps quality (localization IoU)
- Gradient-based saliency coherence
- Feature transferability across layers

- Occlusion sensitivity patterns

Efficiency Metrics:

- Inference time (ms per image)
- Memory usage (MB per batch)
- Energy consumption (mJ per inference)
- Model size (MB for deployment)

(b) Analyze failure modes and limitations of discussed techniques: (10 marks)

- Position-sensitive convolution: computational overhead and when it's unnecessary
- GAP limitations: information bottleneck and class imbalance effects
- Transfer learning: negative transfer and domain shift problems
- Visualization techniques: interpretation biases and validation challenges

Answer: Each technique has specific failure modes and limitations that must be understood to apply them effectively in practice.

Position-Sensitive Convolution Limitations:

Computational Overhead:

- Memory increase: $(C + 2)/C$ factor - minimal for large C, but significant for early layers
- Coordinate computation: Additional operations for normalization
- Cache efficiency: Increased channel count may reduce memory locality

When Unnecessary:

(a) **Translation Invariant Tasks:** Object classification where position doesn't matter

(b) **Data Augmentation Heavy:** When training uses extensive spatial augmentation

(c) **Pre-pooling Layers:** After aggressive pooling, spatial information is already lost

(d) **Small Images:** For inputs $< 64\times64$, position information provides little benefit

Failure Cases:

- Can lead to overfitting on position patterns in training data
- May harm generalization when test distribution has different position patterns
- Incompatible with certain data augmentation strategies

GAP Limitations:

Information Bottleneck:

$$\text{Information Loss} = H(F_{H\times W\times C}) - H(F_{1\times1\times C})$$

- Spatial information completely lost: $H \times W$ dimensions reduced to scalars
- Cannot capture spatial relationships between objects
- Unsuitable for dense prediction tasks (segmentation, detection)

Class Imbalance Effects:

- Rare classes get insufficient channel specialization
- Dominant classes may monopolize multiple channels
- Uneven learning dynamics across classes

Mathematical analysis:

$$P(\text{channel } k \text{ specialized for class } c) \propto P(c) \times \text{class discriminability}$$

Failure Scenarios:

(a) **Fine-grained Recognition:** When subtle spatial details matter

(b) **Multi-object Scenes:** Cannot handle multiple instances well

(c) **Structured Outputs:** Sequence generation, structured prediction

Transfer Learning Limitations:

Negative Transfer: Occurs when $\text{Performance}_{\text{transfer}} < \text{Performance}_{\text{from-scratch}}$

Causes:

- **Domain Mismatch:** Medical images   Natural images
- **Task Mismatch:** Classification   Regression
- **Scale Mismatch:** High-resolution   Low-resolution inputs

Domain Shift Problems:

Covariate Shift: $P_s(X) \neq P_t(X)$ but $P_s(Y|X) = P_t(Y|X)$ **Label Shift:** $P_s(Y) \neq P_t(Y)$ but $P_s(X|Y) = P_t(X|Y)$ **Concept Shift:** $P_s(Y|X) \neq P_t(Y|X)$

Detection Strategy:

$$\text{Domain Shift Score} = \text{KL}(P_s(h(X))||P_t(h(X)))$$

where $h(X)$ are learned features.

Mitigation:

- Domain adaptation techniques
- Gradual unfreezing
- Multi-domain pretraining
- Adversarial domain alignment

Visualization Technique Limitations:

Interpretation Biases:

(a) **Confirmation Bias:** Seeing patterns that confirm expectations
(b) **Cherry-picking:** Showing only successful visualizations
(c) **Anthropomorphism:** Attributing human-like reasoning to networks

Technical Limitations:

Gradient Saturation: In saturated regions, gradients ≈ 0 regardless of importance

Non-linearity Effects: Linear approximations fail for highly non-linear models:

$$\text{Error} = ||f(x+\epsilon) - f(x) - \epsilon^T \nabla f||_2$$

Resolution Mismatch: Visualization resolution often lower than decision-relevant features

Validation Challenges:

- No ground truth for ”correct” interpretations
- Human evaluation is subjective and inconsistent
- Proxy metrics may not capture interpretation quality
- Adversarial examples can fool visualization methods

Methodological Issues:

- **Baseline Dependence:** Integrated gradients sensitive to baseline choice
- **Hyperparameter Sensitivity:** Occlusion window size, smoothing parameters
- **Model Architecture Dependence:** Some methods work better with specific architectures

Fundamental Limitation: Visualization methods provide *post-hoc* explanations that may not reflect actual decision processes. The network may use different information than what visualizations highlight.

END OF PAPER