MIDDLE EAST TECHNICAL UNIVERSITY

SEMESTER I EXAMINATION 2024-2025

CENG 403 – Deep Learning - CNN Visualization & Classic Architectures (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. CNN Visualization Techniques and CAM Variants (25 marks)
Based on Stanford CS231n and university computer vision course materials.

(a) Implement Class Activation Mapping (CAM) for a CNN with Global Average Pooling. Given a feature map F_k of size $H \times W$ for the k-th channel and weight w_k connecting to class c: (10 marks)

- Derive the mathematical formulation for CAM: $M_c(x, y) = \sum_k w_k \cdot F_k(x, y)$
- Explain why CAM requires Global Average Pooling (GAP) layer
- Calculate the computational complexity for generating CAM for a 512×512 image with 512 feature maps

Answer: Class Activation Mapping leverages the spatial information preserved in convolutional layers to visualize class-specific regions by weighted combination of feature maps using classification weights.

Mathematical Derivation of CAM:

Network Architecture with GAP:

(a) Final convolutional layer produces feature maps $F_k(x, y)$ for $k = 1, ..., K$ channels

(b) Global Average Pooling: $f_k = \frac{1}{H \times W} \sum_{x,y} F_k(x, y)$

(c) Classification layer: $S_c = \sum_{k=1}^{K} w_{k,c} \cdot f_k + b_c$

CAM Formula Derivation:

Substituting GAP into classification:

$$S_c = \sum_{k=1}^{K} w_{k,c} \cdot \frac{1}{H \times W} \sum_{x,y} F_k(x, y) + b_c$$

Rearranging:

$$S_c = \frac{1}{H \times W} \sum_{x,y} \left(\sum_{k=1}^{K} w_{k,c} \cdot F_k(x, y) \right) + b_c$$

The term in parentheses defines the Class Activation Map:

$$M_c(x, y) = \sum_{k=1}^{K} w_{k,c} \cdot F_k(x, y)$$

Therefore: $S_c = \frac{1}{H \times W} \sum_{x,y} M_c(x, y) + b_c$

Physical Interpretation:

- $M_c(x, y)$ represents the importance of spatial location (x, y) for class c
- High values indicate regions that strongly activate class c
- The classification score is the spatial average of this importance map

Why CAM Requires GAP:

1. Spatial Information Preservation:

- GAP maintains spatial structure: $H \times W \times K \to 1 \times 1 \times K$
- Each channel retains spatial activation patterns
- Classification weights directly correspond to spatial importance

2. Direct Weight Correspondence:

- Weights $w_{k,c}$ in classification layer have clear interpretation
- Each weight shows how much channel k contributes to class c
- This correspondence enables reversing the computation to get spatial maps

3. Alternative Architectures (FC layers) Break This:

- FC layers: $\text{flatten}(F) \to \mathbb{R}^{H \times W \times K} \to \mathbb{R}^{HWK}$
- Weight matrix: $W \in \mathbb{R}^{HWK \times C}$
- No clear spatial correspondence in weights
- Cannot trace back from weights to spatial locations

Computational Complexity Analysis:

Given: 512×512 image, 512 feature maps

Step-by-step computation:

(a) **Feature map storage:** $512 \times 512 \times 512 = 134,217,728$ values

(b) **Weight multiplication:** For each pixel (x, y): $\sum_{k=1}^{512} w_{k,c} \cdot F_k(x, y)$

(c) **Total multiplications:** $512 \times 512 \times 512 = 134,217,728$ operations

(d) **Total additions:** $512 \times 512 \times 511 = 133,693,440$ operations

Complexity Analysis:

- **Time Complexity:** $O(H \times W \times K)$ where H, W are spatial dimensions, K is channels
- **Space Complexity:** $O(H \times W \times K)$ for storing feature maps + $O(H \times W)$ for CAM
- **For given example:** $O(512^2 \times 512) = O(134M)$ operations

Efficiency Considerations:

- Can be parallelized across spatial dimensions
- Memory-bound rather than compute-bound
- Can generate CAMs for all classes simultaneously with minor overhead

(b) Compare CAM with Grad-CAM and Grad-CAM++. Explain the following improvements: (10 marks)

- How Grad-CAM generalizes CAM to any CNN architecture without GAP
- Why Grad-CAM++ provides better localization for objects with low spatial footprint
- Mathematical differences in weight calculation between the three methods

Answer: Grad-CAM and Grad-CAM++ progressively improve upon CAM by removing architectural constraints and enhancing localization accuracy through more sophisticated weight computation methods.

CAM Limitations and Grad-CAM Solution:

CAM Limitations:

- Requires specific architecture: GAP + single FC layer
- Cannot be applied to pre-trained networks with different architectures
- Limited to networks specifically designed for CAM
- Fixed weights regardless of input image

Grad-CAM Generalization:

Key Insight: Replace fixed learned weights with gradient-based importance scores

Mathematical Formulation:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{i,j}^k}$$

where:

- y^c is the score for class c (before softmax)
- A^k is the activation map for channel k at any convolutional layer
- α_k^c represents importance of channel k for class c
- $Z = H \times W$ for normalization

Grad-CAM Generation:

$$L_{\text{Grad-CAM}}^c = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right)$$

Advantages Over CAM:

- **Architecture Independence:** Works with any CNN (VGG, ResNet, DenseNet, etc.)

- **Layer Flexibility:** Can visualize any convolutional layer, not just the last
- **Input Dependence:** Weights adapt based on specific input image
- **No Retraining:** Works with existing pre-trained models

Grad-CAM++ Improvements:

Problem with Grad-CAM: Poor localization for objects with small spatial footprint

Grad-CAM++ Weight Calculation:

$$\alpha_{i,j,k}^{c} = \frac{\partial^2 y^c}{\partial (A_{i,j}^k)^2} \cdot \frac{2\frac{\partial^2 y^c}{\partial (A_{i,j}^k)^2}}{2\frac{\partial^2 y^c}{\partial (A_{i,j}^k)^2} + \sum_{a,b} A_{a,b}^k \frac{\partial^3 y^c}{\partial (A_{a,b}^k)^3}}$$

Key Improvements:

(a) **Pixel-wise Weights:** Different importance for each spatial location

(b) **Higher-order Gradients:** Uses second and third derivatives

(c) **Better Object Coverage:** Covers complete object extent, not just discriminative parts

(d) **Multi-instance Localization:** Better for multiple objects of same class

Mathematical Differences Summary:

CAM:

- Weight: $w_{k,c}$ (learned during training, fixed)
- Computation: $M_c(x, y) = \sum_k w_{k,c} F_k(x, y)$
- Requirements: GAP architecture

Grad-CAM:

- Weight: $\alpha_k^c = \frac{1}{HW} \sum_{i,j} \frac{\partial y^c}{\partial A_{i,j}^k}$ (input-dependent)
- Computation: $L^c = \text{ReLU}(\sum_k \alpha_k^c A^k)$
- Requirements: Any CNN architecture

Grad-CAM++:

- Weight: $\alpha_{i,j,k}^{c}$ (pixel-wise, using higher-order gradients)
- Computation: $L^c = \text{ReLU}(\sum_k \sum_{i,j} \alpha_{i,j,k}^{c} A_{i,j}^{k})$
- Requirements: Any CNN architecture, better for complex scenes

Performance Characteristics:

Method	Architecture Constraint	Localization Quality	Computational Cost
CAM	GAP required	Good	Low
Grad-CAM	None	Better	Medium
Grad-CAM++	None	Best	High

Use Case Recommendations:

- **CAM:** New architectures designed with GAP
- **Grad-CAM:** General-purpose visualization for any CNN
- **Grad-CAM++:** High-precision localization, multiple objects, small objects

(c) Design an evaluation protocol for visualization methods. Propose metrics for: (5 marks)

- Quantitative evaluation of localization accuracy
- Qualitative assessment of explanation quality
- Computational efficiency comparison

Answer: Comprehensive evaluation requires quantitative localization metrics, qualitative human assessment protocols, and computational efficiency benchmarks to ensure visualization methods are accurate, interpretable, and practical.

Quantitative Localization Accuracy:

1. Intersection over Union (IoU):

$$\text{IoU} = \frac{|\text{Predicted Region} \cap \text{Ground Truth}|}{|\text{Predicted Region} \cup \text{Ground Truth}|}$$

Implementation:

- Threshold heatmap at multiple levels: $\tau \in [0.1, 0.2, ..., 0.9]$
- Compute IoU for each threshold
- Report mean IoU (mIoU) across thresholds

2. Pointing Game:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\arg\max_{(x,y)} M_i(x,y) \in \text{Object}_i]$$

Check if maximum activation point falls within ground truth object.

3. Area Under ROC Curve (AUC):

- Treat heatmap as binary classifier for each pixel
- Ground truth: 1 for object pixels, 0 for background
- Compute ROC curve and AUC

4. Localization Error:

$$\text{Error} = \frac{1}{N}\sum_{i=1}^{N} ||\text{max\_loc}_i - \text{gt\_center}_i||_2$$

Distance between maximum activation and ground truth object center.

Qualitative Assessment of Explanation Quality:

1. Human Evaluation Protocol:

- **Participant Selection:** Domain experts + general users
- **Sample Size:** Minimum 100 images, 3+ evaluators per image
- **Evaluation Criteria:**
  - Semantic coherence (highlighted regions make sense)
  - Completeness (covers entire relevant object)
  - Precision (doesn't highlight irrelevant regions)

2. Rating Scales:

- 5-point Likert scale for each criterion

- Inter-rater reliability measurement (Cronbach's)
- Statistical significance testing (paired t-tests)

3. Comparative Studies:

- Side-by-side comparison of different methods
- Ranking tasks: order methods by explanation quality
- Forced choice: which explanation is more trustworthy?

4. Task-Based Evaluation:

- **Trust Calibration:** Can humans predict model failures using explanations?
- **Error Detection:** Do explanations help identify misclassifications?
- **Decision Support:** Do explanations improve human-AI collaboration?

Computational Efficiency Comparison:

1. Runtime Analysis:

$$\text{Runtime} = T_{\text{forward}} + T_{\text{backward}} + T_{\text{processing}}$$

Components:

- T_{forward}: Forward pass time
- T_{backward}: Gradient computation time
- $T_{\text{processing}}$: Post-processing (normalization, upsampling)

2. Memory Usage:

- Peak memory consumption during visualization
- Additional memory beyond normal inference
- Scalability with image resolution and network depth

3. Benchmark Protocol:

- **Hardware:** Standardized GPU (e.g., RTX 3080)

- **Batch Sizes:** 1, 8, 32 images
- **Resolutions:** 224×224, 512×512, 1024×1024
- **Networks:** ResNet-50, VGG-16, DenseNet-121
- **Metrics:** ms/image, GB memory, FLOPs

4. Scalability Analysis:

$$\text{Efficiency Score} = \frac{\text{Localization Accuracy}}{\text{Runtime} \times \text{Memory Usage}}$$

Comprehensive Evaluation Framework:

1. Multi-Dataset Validation:

- ImageNet (natural images)
- Medical imaging datasets (X-rays, MRI)
- Satellite imagery
- Fine-grained classification datasets

2. Cross-Architecture Testing:

- Test on multiple CNN architectures
- Evaluate consistency across different model types
- Assess robustness to architectural variations

3. Statistical Validation:

- Confidence intervals for all metrics
- Statistical significance testing
- Multiple comparison corrections (Bonferroni)
- Effect size measurements

This comprehensive evaluation protocol ensures that visualization methods are rigorously assessed across multiple dimensions of performance and usability.

Question 2. AlexNet Architecture Analysis (22 marks)
Based on D2L.ai and university deep learning course materials covering computational analysis.

(a) Analyze AlexNet's computational requirements. Given the architecture specifications: (12 marks)

- Input: 224×224×3 images
- Conv1: 96 filters, 11×11, stride 4, pad 0
- Conv2: 256 filters, 5×5, stride 1, pad 2
- FC6: 4096 neurons, FC7: 4096 neurons, FC8: 1000 neurons

Calculate:

- Memory footprint for each convolutional layer
- Number of parameters in fully connected layers vs. convolutional layers
- Which component dominates memory usage and why

Answer: AlexNet's computational analysis reveals that fully connected layers dominate both parameter count and memory usage, highlighting the inefficiency that led to later architectural innovations.

Detailed Architecture and Calculations:

Complete AlexNet Layer Specifications:

(a) Input: $224 \times 224 \times 3$

(b) Conv1: 96 filters, 11×11, stride=4, pad=0 $\rightarrow 55 \times 55 \times 96$

(c) MaxPool1: 3×3, stride=2 $\rightarrow 27 \times 27 \times 96$

(d) Conv2: 256 filters, 5×5, stride=1, pad=2 $\rightarrow 27 \times 27 \times 256$

(e) MaxPool2: 3×3, stride=2 $\rightarrow 13 \times 13 \times 256$

(f) Conv3: 384 filters, 3×3, stride=1, pad=1 $\rightarrow 13 \times 13 \times 384$

(g) Conv4: 384 filters, 3×3, stride=1, pad=1 $\rightarrow 13 \times 13 \times 384$

(h) Conv5: 256 filters, 3×3, stride=1, pad=1 $\rightarrow 13 \times 13 \times 256$

(i) MaxPool3: 3×3, stride=2 $\rightarrow 6 \times 6 \times 256$

(j) FC6: 4096 neurons

(k) FC7: 4096 neurons

(l) FC8: 1000 neurons

Memory Footprint Calculations:

Conv1 Memory Analysis:

- Input size: $224 \times 224 \times 3 = 150,528$ values
- Output size: $\frac{224-11+0}{4} + 1 = 55$, so $55 \times 55 \times 96 = 290,400$ values
- Memory (32-bit floats): $290,400 \times 4 = 1,161,600$ bytes $= 1.16$ MB
- Parameters: $(11 \times 11 \times 3) \times 96 + 96 = 34,944$ parameters
- Parameter memory: $34,944 \times 4 = 139,776$ bytes $= 0.14$ MB

Conv2 Memory Analysis:

- Input size: $27 \times 27 \times 96 = 69,984$ values (after pooling)
- Output size: $27 \times 27 \times 256 = 186,624$ values
- Memory: $186,624 \times 4 = 746,496$ bytes $= 0.75$ MB
- Parameters: $(5 \times 5 \times 96) \times 256 + 256 = 614,656$ parameters
- Parameter memory: $614,656 \times 4 = 2,458,624$ bytes $= 2.46$ MB

Remaining Convolutional Layers:

- Conv3: $13 \times 13 \times 384 = 64,896$ values, memory $= 0.26$ MB
- Conv4: $13 \times 13 \times 384 = 64,896$ values, memory $= 0.26$ MB
- Conv5: $13 \times 13 \times 256 = 43,264$ values, memory $= 0.17$ MB

Total Convolutional Layer Memory: $1.16+0.75+0.26+0.26+0.17 = 2.6$ MB

Parameter Count Analysis:

Convolutional Layer Parameters:

- Conv1: $(11^2 \times 3) \times 96 + 96 = 34,944$
- Conv2: $(5^2 \times 96) \times 256 + 256 = 614,656$
- Conv3: $(3^2 \times 256) \times 384 + 384 = 885,120$

- Conv4: $(3^2 \times 384) \times 384 + 384 = 1,327,488$
- Conv5: $(3^2 \times 384) \times 256 + 256 = 884,992$
- **Total Conv Parameters:** $3,747,200 \quad 3.75$M

Fully Connected Layer Parameters:

- FC6: $(6 \times 6 \times 256) \times 4096 + 4096 = 37,752,832 \quad 37.75$M
- FC7: $4096 \times 4096 + 4096 = 16,781,312 \quad 16.78$M
- FC8: $4096 \times 1000 + 1000 = 4,097,000 \quad 4.10$M
- **Total FC Parameters:** $58,631,144 \quad 58.63$M

Total Network Parameters: $3.75M + 58.63M = 62.38M$

Memory Usage Analysis:

Activation Memory (Forward Pass):

- Convolutional activations: 2.6 MB
- FC6 activations: $4096 \times 4 = 16,384$ bytes $= 0.016$ MB
- FC7 activations: $4096 \times 4 = 16,384$ bytes $= 0.016$ MB
- FC8 activations: $1000 \times 4 = 4,000$ bytes $= 0.004$ MB
- **Total Activation Memory:** 2.64 MB

Parameter Memory:

- Convolutional parameters: $3.75M \times 4 = 15$ MB
- FC parameters: $58.63M \times 4 = 234.5$ MB
- **Total Parameter Memory:** 249.5 MB

Training Memory (includes gradients):

- Parameters: 249.5 MB
- Gradients: 249.5 MB (same size as parameters)
- Optimizer states (Adam): 499 MB (2× parameters for momentum and variance)
- Activations: 2.64 MB
- **Total Training Memory:** 1000 MB = 1 GB per image

Dominant Components Analysis:

1. Parameter Distribution:

- Convolutional layers: 6% of parameters
- Fully connected layers: 94% of parameters
- FC6 alone: 60.5% of all parameters

2. Memory Distribution:

- Parameter storage: 249.5 MB (94% of static memory)
- Activation storage: 2.6 MB (6% of static memory)
- FC layers dominate parameter memory
- Conv layers dominate activation memory

3. Why FC Layers Dominate:

- **Dense Connectivity:** Every input connected to every output
- **Large Input Size:** $6 \times 6 \times 256 = 9,216$ inputs to FC6
- **Large Output Size:** 4,096 neurons in FC6 and FC7
- **Quadratic Growth:** Parameters scale as $O(n \times m)$ for n inputs, m outputs

Historical Impact: This analysis reveals why later architectures focused on:

- Global Average Pooling (eliminates FC6 entirely)
- Smaller spatial dimensions before FC layers
- More efficient architectural designs (ResNet, MobileNet)
- Attention mechanisms instead of FC layers

(b) Evaluate AlexNet's key innovations and their impact on deep learning: (10 marks)

- ReLU activation functions: advantages over sigmoid/tanh for training speed

- Dropout regularization: mathematical formulation and overfitting prevention
- GPU utilization: architectural modifications needed for parallel processing
- Performance improvement: quantify the error rate reduction from 26.2% to 15.3%

Answer: AlexNet's innovations fundamentally transformed deep learning by solving key training challenges and demonstrating the power of deep CNNs, establishing foundations still used today.

ReLU Activation Functions:

Mathematical Comparison:

Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$, $\sigma'(x) = \sigma(x)(1-\sigma(x))$

Tanh: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, $\tanh'(x) = 1 - \tanh^2(x)$

ReLU: $\text{ReLU}(x) = \max(0, x)$, $\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$

Training Speed Advantages:

1. Gradient Magnitude:

- Sigmoid: Maximum derivative = 0.25 (at $x = 0$)
- Tanh: Maximum derivative = 1.0 (at $x = 0$)
- ReLU: Derivative = 1.0 (for $x > 0$)

2. Vanishing Gradient Problem:

- Sigmoid/Tanh: Gradients vanish exponentially in deep networks
- Chain rule: $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_n} \prod_{i=2}^{n} \frac{\partial a_i}{\partial a_{i-1}}$
- With sigmoid: $\prod_{i=2}^{n} \sigma'(a_i) \leq 0.25^{n-1} \to 0$ rapidly
- With ReLU: $\prod_{i=2}^{n} \text{ReLU}'(a_i) = 1$ (for active units)

3. Computational Efficiency:

- Sigmoid/Tanh: Require expensive exponential operations
- ReLU: Simple thresholding operation ($\max(0, x)$)

- 6× faster training reported in original AlexNet paper

4. Biological Plausibility:

- Neurons either fire or don't fire (sparse activation)
- ReLU creates sparse representations (many zeros)
- Improved interpretability and reduced overfitting

Dropout Regularization:

Mathematical Formulation:

Training Phase:

$$y_i = \begin{cases} \frac{x_i}{p} & \text{with probability } p \\ 0 & \text{with probability } 1-p \end{cases}$$

Equivalently: $y_i = \frac{x_i \cdot B_i}{p}$ where $B_i \sim \text{Bernoulli}(p)$

Testing Phase:

$$y_i = x_i$$

(All neurons active, no scaling needed due to training-time scaling)

Overfitting Prevention Mechanism:

1. Co-adaptation Prevention:

- Prevents neurons from becoming dependent on specific other neurons
- Forces each neuron to learn useful features independently
- Reduces complex co-adaptations that don't generalize

2. Ensemble Effect:

- Each forward pass uses different subset of neurons
- Equivalent to training exponentially many different networks
- Test time uses expectation over all possible networks

3. Model Averaging:

$$\mathbb{E}[y_{\text{test}}] = \mathbb{E}[y_{\text{train}}]$$

The expected output at test time equals training time due to scaling factor $1/p$.

AlexNet Implementation:

- Applied to FC6 and FC7 layers only
- Dropout probability: $p = 0.5$
- Roughly doubled training time but significantly improved generalization

GPU Utilization and Parallel Processing:

Historical Context (2012):

- GPU memory: 3GB per GPU (GTX 580)
- CPU training: Weeks or months for large networks
- GPU training: Days for same networks

Architectural Modifications for Parallelism:

1. Dual-GPU Split:

- Network split across two GPUs
- Each GPU handles half the feature maps in each layer
- Cross-GPU communication only at specific layers (Conv2, Conv4, FC layers)
- Reduced communication overhead

2. Data Parallelism:

- Different mini-batches processed on different GPUs
- Gradients averaged across GPUs
- Enabled larger effective batch sizes

3. Memory Optimization:

- Careful memory management for large feature maps
- Gradient checkpointing to reduce memory usage
- Overlapped computation and communication

Performance Impact:

- Training time: Reduced from months to days
- Enabled experimentation with deeper networks
- Demonstrated feasibility of large-scale deep learning

Performance Improvement Analysis:

ImageNet ILSVRC 2012 Results:

- Previous best (non-deep): 26.2% top-5 error
- AlexNet: 15.3% top-5 error
- **Absolute improvement:** $26.2\% - 15.3\% = 10.9\%$
- **Relative improvement:** $\frac{26.2-15.3}{26.2} = 41.6\%$ reduction

Statistical Significance:

- Second place: 26.2% error
- AlexNet: 15.3% error
- Gap to second place: 10.9 percentage points
- This was unprecedented in computer vision history

Broader Impact:

1. Paradigm Shift:

- From hand-crafted features to learned features
- Demonstrated power of deep learning for computer vision
- Sparked ”deep learning revolution” starting 2012

2. Subsequent Improvements:

- 2013 ZFNet: 14.8% error
- 2014 VGGNet: 7.3% error

- 2014 GoogleNet: 6.7% error
- 2015 ResNet: 3.57% error (human-level performance)

3. Technical Legacy:

- ReLU became standard activation function
- Dropout became essential regularization technique
- GPU training became industry standard
- Established CNN architectures as dominant approach

AlexNet's innovations were not just incremental improvements but fundamental breakthroughs that enabled the modern deep learning era.

Question 4. ResNet and Residual Learning Theory (30 marks)
Based on Microsoft Research ResNet paper and university deep learning theory courses.

(a) Analyze the degradation problem in deep networks that ResNet addresses: (10 marks)

- Why do 56-layer CNNs perform worse than 20-layer CNNs on both training and test sets?
- Mathematical explanation of gradient vanishing in very deep networks
- Distinction between degradation problem and overfitting

Answer: The degradation problem is a fundamental optimization challenge in very deep networks where adding layers hurts both training and test performance, distinct from overfitting which only affects generalization.

Empirical Evidence of Degradation Problem:

Experimental Observations:

- 20-layer network: 91.25% training accuracy, 89.43% test accuracy
- 56-layer network: 90.16% training accuracy, 87.23% test accuracy
- Deeper network performs worse on **both** training and test sets
- This contradicts the intuition that more capacity should improve training performance

Why Degradation Occurs:

1. Optimization Difficulty: Deep networks create more complex loss landscapes that are harder to navigate:

- More parameters create higher-dimensional optimization spaces
- More local minima and saddle points
- Increased path-dependence in optimization trajectories
- Standard optimizers struggle with very deep architectures

2. Identity Mapping Problem: Theoretically, deeper networks should perform at least as well as shallow ones:

- Extra layers could learn identity mappings: $f(x) = x$
- In practice, learning perfect identity mappings is difficult
- Networks struggle to approximate $H(x) = x$ when needed
- Small deviations from identity accumulate through many layers

Mathematical Explanation of Gradient Vanishing:

Backpropagation Chain Rule: For a network with L layers, the gradient at layer ℓ is:

$$\frac{\partial \mathcal{L}}{\partial \theta_\ell} = \frac{\partial \mathcal{L}}{\partial a_L} \prod_{i=\ell+1}^{L} \frac{\partial a_i}{\partial a_{i-1}} \frac{\partial a_\ell}{\partial \theta_\ell}$$

The Product Problem: The gradient involves a product of $L - \ell$ Jacobian matrices. Each term $\frac{\partial a_i}{\partial a_{i-1}}$ typically has:

For ReLU Networks:

$$\frac{\partial a_i}{\partial a_{i-1}} = W_i \cdot \text{diag}(\mathbf{1}_{a_{i-1}>0})$$

where $\mathbf{1}_{a_{i-1}>0}$ is the indicator function for active ReLU units.

Gradient Magnitude Analysis:

$$\left\| \frac{\partial \mathcal{L}}{\partial \theta_\ell} \right\| \leq \left\| \frac{\partial \mathcal{L}}{\partial a_L} \right\| \prod_{i=\ell+1}^{L} \|W_i\| \prod_{i=\ell+1}^{L} \left\| \text{diag}(\mathbf{1}_{a_{i-1}>0}) \right\|$$

Problematic Cases:

- If $\|W_i\| < 1$ for most layers: $\prod \|W_i\| \to 0$ exponentially
- If many ReLU units are inactive: $\|\text{diag}(\mathbf{1}_{a_{i-1}>0})\| \ll 1$
- Product of many terms < 1 leads to vanishing gradients
- Product of many terms > 1 leads to exploding gradients

Spectral Analysis: The expected gradient magnitude can be approximated as:

$$\mathbb{E}[\|\nabla_{\theta_\ell}\mathcal{L}\|] \propto \gamma^{L-\ell}$$

where γ is related to the spectral radius of the weight matrices. For $\gamma < 1$, gradients vanish exponentially with depth.

Distinction from Overfitting:

Overfitting Characteristics:

- Training accuracy increases (or stays high)
- Test accuracy decreases
- Model memorizes training data
- Generalization gap increases
- More parameters typically lead to better training performance

Degradation Problem Characteristics:

- **Both** training and test accuracy decrease
- Optimization becomes harder, not easier
- Training loss may increase or plateau
- Clear indication of optimization failure
- More parameters lead to worse performance on all metrics

Formal Distinction:

Overfitting: TrainError $\downarrow$, TestError $\uparrow$ as model complexity increases

Degradation: TrainError $\uparrow$, TestError $\uparrow$ as model depth increases

Evidence Against Overfitting Explanation:

(a) **Training Performance:** If it were overfitting, training accuracy should be high

(b) **Regularization:** Standard techniques (dropout, weight decay) don't solve the problem

(c) **Dataset Independence:** Problem occurs across different datasets

(d) **Architecture Independence:** Multiple architectures show similar degradation

Implications: The degradation problem revealed that the fundamental challenge in very deep networks is optimization, not overfitting. This insight led to architectural innovations (skip connections) rather than just regularization techniques.

(b) Derive the mathematical foundation of residual learning: (12 marks)

- Given target mapping $H(x)$, show why learning $F(x) = H(x) - x$ is easier than learning $H(x)$
- Residual block formulation: $y = F(x, \{W_i\}) + x$
- Gradient flow analysis: $\frac{\partial loss}{\partial x} = \frac{\partial loss}{\partial y}(1 + \frac{\partial F}{\partial x})$
- Explain why the "+1" term prevents gradient vanishing

Answer: Residual learning reformulates the optimization problem to learn residual mappings rather than direct mappings, providing guaranteed gradient flow and easier optimization.

Residual Learning Reformulation:

Traditional Approach: Network layers directly learn the desired mapping:

$$y = H(x)$$

where $H(x)$ is the target function the layer should learn.

Residual Approach: Network layers learn the residual:

$$y = F(x) + x$$

where $F(x) = H(x) - x$ is the residual function.

Why Learning $F(x) = H(x) - x$ is Easier:

1. Identity Mapping Simplification:

When optimal mapping is identity: $H(x) = x$

- Traditional: Learn $H(x) = x$ (difficult to approximate perfectly)
- Residual: Learn $F(x) = H(x) - x = x - x = 0$ (much easier)
- Learning to output zero is simpler than learning identity

2. Near-Identity Mapping:

When optimal mapping is close to identity: $H(x) = x + \epsilon(x)$ where $\epsilon(x)$ is small

- Traditional: Learn entire mapping $H(x) = x + \epsilon(x)$
- Residual: Learn only the residual $F(x) = \epsilon(x)$
- Focusing on small residuals is more efficient

3. Optimization Landscape:

- Identity mapping provides good baseline performance
- Network can gradually improve from this baseline
- Easier to optimize small deviations than large transformations
- More stable training dynamics

Residual Block Mathematical Formulation:

Standard Block:

$$y = F(x, \{W_i\}) + x$$

where:

- x is the input to the residual block
- $F(x, \{W_i\})$ is the residual function with parameters $\{W_i\}$
- y is the output of the residual block
- The $+$ represents element-wise addition

Typical Implementation:

$$F(x, \{W_1, W_2\}) = W_2\sigma(W_1x)$$

where σ is ReLU activation.

Complete Forward Pass:

$$y = W_2\sigma(W_1x) + x$$

Gradient Flow Analysis:

Chain Rule Application: Using the chain rule on $y = F(x) + x$:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial x}$$

Partial Derivative Calculation:

$$\frac{\partial y}{\partial x} = \frac{\partial}{\partial x}(F(x) + x) = \frac{\partial F(x)}{\partial x} + \frac{\partial x}{\partial x} = \frac{\partial F(x)}{\partial x} + I$$

where I is the identity matrix.

Final Gradient Expression:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y}\left(\frac{\partial F(x)}{\partial x} + I\right) = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial F(x)}{\partial x} + \frac{\partial \mathcal{L}}{\partial y}$$

Why the "+1" Term Prevents Gradient Vanishing:

1. Guaranteed Gradient Flow:

- The term $\frac{\partial \mathcal{L}}{\partial y}$ always provides a direct gradient path
- Even if $\frac{\partial F(x)}{\partial x} \to 0$, gradients still flow through the "+1" term
- Minimum gradient magnitude: $\left\|\frac{\partial \mathcal{L}}{\partial x}\right\| \geq \left\|\frac{\partial \mathcal{L}}{\partial y}\right\|$

2. Mathematical Analysis:

Traditional Network:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial F(x)}{\partial x}$$

If $\left\|\frac{\partial F(x)}{\partial x}\right\| \ll 1$, then $\left\|\frac{\partial \mathcal{L}}{\partial x}\right\| \ll \left\|\frac{\partial \mathcal{L}}{\partial y}\right\|$

ResNet:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y}\left(\frac{\partial F(x)}{\partial x} + I\right)$$

Even if $\left\|\frac{\partial F(x)}{\partial x}\right\| \ll 1$, we have $\left\|\frac{\partial \mathcal{L}}{\partial x}\right\| \approx \left\|\frac{\partial \mathcal{L}}{\partial y}\right\|$

3. Multi-Layer Analysis:

For a network with n residual blocks:

$$\frac{\partial \mathcal{L}}{\partial x_0} = \frac{\partial \mathcal{L}}{\partial x_n}\prod_{i=1}^{n}\left(\frac{\partial F_i}{\partial x_{i-1}} + I\right)$$

Traditional Network:

$$\prod_{i=1}^{n} \left\| \frac{\partial F_i}{\partial x_{i-1}} \right\| \text{ can vanish exponentially}$$

ResNet:

$$\prod_{i=1}^{n} \left\| \frac{\partial F_i}{\partial x_{i-1}} + I \right\| \geq 1 \text{ always}$$

4. Practical Implications:

Training Dynamics:

- Early layers receive meaningful gradients throughout training
- Network can learn both fine-grained and coarse-grained features
- Stable optimization even for very deep networks (100+ layers)
- Faster convergence due to better gradient flow

Feature Learning:

- Identity path preserves low-level features
- Residual path learns incremental improvements
- Network adaptively balances preservation and transformation
- Hierarchical feature learning becomes more effective

Theoretical Guarantees: The "+1" term provides theoretical guarantees that:

- Gradients cannot vanish completely
- Adding layers cannot hurt performance in the worst case
- Network maintains learning capability at all depths
- Optimization landscapes become smoother and more navigable

This mathematical foundation explains why ResNet enabled successful training of networks with 100+ layers, revolutionizing deep learning.

(c) Compare ResNet variants and their performance characteristics: (8 marks)

- ResNet-18, ResNet-34: basic blocks vs. deeper architectures
- ResNet-50, ResNet-101, ResNet-152: bottleneck blocks for efficiency
- Performance scaling: how accuracy improves with depth up to 152 layers
- Computational complexity comparison with VGG architectures

Answer: ResNet variants demonstrate systematic scaling from basic to bottleneck blocks, achieving consistent performance improvements with depth while maintaining computational efficiency compared to traditional architectures.

ResNet Architecture Variants:

Basic Block (ResNet-18, ResNet-34):

Architecture:

- Two 3×3 convolutional layers per block
- ReLU activations after each convolution
- Batch normalization before each activation
- Skip connection adds input to output

Mathematical Formulation:

$$y = F(x) + x$$

where $F(x) = W_2\sigma(\text{BN}(W_1\sigma(\text{BN}(x))))$

Parameter Count: For channels C: $(3^2 \times C \times C) \times 2 = 18C^2$ parameters per block

ResNet-18 Structure:

- Conv1: 7×7, 64 filters, stride 2
- Layer 1: 2 basic blocks, 64 channels
- Layer 2: 2 basic blocks, 128 channels
- Layer 3: 2 basic blocks, 256 channels
- Layer 4: 2 basic blocks, 512 channels
- Total: $2 \times 4 \times 2 + 2 = 18$ layers

ResNet-34 Structure:

- Same structure but more blocks per layer: [3, 4, 6, 3]
- Total: $3 + 4 + 6 + 3 = 16$ basic blocks $= 34$ layers

Bottleneck Block (ResNet-50, ResNet-101, ResNet-152):

Motivation:

- Deeper networks need computational efficiency
- Reduce parameters while maintaining expressiveness
- Enable training of 100+ layer networks

Architecture:

- Three convolutions: $1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$
- First 1×1: Reduce channels (bottleneck)
- Middle 3×3: Process spatial information
- Last 1×1: Expand channels back

Mathematical Formulation:

$$F(x) = W_3\sigma(\text{BN}(W_2\sigma(\text{BN}(W_1\sigma(\text{BN}(x))))))$$

Bottleneck Design: If input has $4C$ channels:

- 1×1 conv: $4C \rightarrow C$ (reduce)
- 3×3 conv: $C \rightarrow C$ (process)
- 1×1 conv: $C \rightarrow 4C$ (expand)

Parameter Comparison:

Basic Block: $2 \times (3^2 \times C \times C) = 18C^2$ parameters

Bottleneck Block: $(1^2 \times 4C \times C) + (3^2 \times C \times C) + (1^2 \times C \times 4C) = 4C^2 + 9C^2 + 4C^2 = 17C^2$ parameters

Efficiency Gain: Similar parameter count but more layers and depth

Performance Scaling Analysis:

ImageNet Results (Top-1 Error):

Architecture	Layers	Parameters	Top-1 Error
ResNet-18	18	11.7M	30.24%
ResNet-34	34	21.8M	26.70%
ResNet-50	50	25.6M	23.85%
ResNet-101	101	44.5M	22.63%
ResNet-152	152	60.2M	21.69%

Key Observations:

1. Consistent Improvement:

- Performance monotonically improves with depth
- No degradation problem observed
- Diminishing returns but continued improvement

2. Scaling Efficiency:

- ResNet-34 vs ResNet-18: 1.9× layers, 3.54 percentage point improvement
- ResNet-50 vs ResNet-34: 1.47× layers, 2.85 percentage point improvement
- ResNet-152 vs ResNet-50: 3.04× layers, 2.16 percentage point improvement

3. Parameter Efficiency: ResNet-50 with bottlenecks has fewer parameters than ResNet-34 with basic blocks while being deeper and more accurate.

Computational Complexity Comparison with VGG:

VGG-16 Analysis:

- Parameters: 138M (mostly in FC layers)
- FLOPs: 15.5B per forward pass
- Memory: Heavy due to large FC layers
- No skip connections → gradient issues in deeper variants

ResNet-50 Analysis:

- Parameters: 25.6M (no FC layers, only GAP)

- FLOPs: 4.1B per forward pass
- Memory: Much more efficient
- Skip connections → trainable to 152+ layers

Efficiency Comparison:

Parameter Efficiency:

- VGG-16: 138M parameters, 28.41% top-1 error
- ResNet-50: 25.6M parameters, 23.85% top-1 error
- ResNet achieves 4.56 percentage points better with 5.4× fewer parameters

Computational Efficiency:

- VGG-16: 15.5B FLOPs
- ResNet-50: 4.1B FLOPs
- ResNet is 3.78× more FLOP-efficient while being more accurate

Memory Efficiency:

- VGG: Dominated by FC layers (>90
- ResNet: All convolutional, uses GAP
- ResNet models are much smaller and mobile-friendly

Training Efficiency:

- VGG: Gradient vanishing limits depth scaling
- ResNet: Can scale to 1000+ layers with proper training
- Skip connections enable much deeper and more powerful models

Practical Impact:

Architecture Design Principles:

- Skip connections enable depth scaling
- Bottleneck designs improve parameter efficiency
- GAP eliminates parameter-heavy FC layers

- Systematic depth scaling improves performance

Industry Adoption:

- ResNet-50 became standard backbone for many applications
- Enabled modern computer vision systems
- Foundation for subsequent innovations (DenseNet, EfficientNet, etc.)
- Still widely used in production systems today

ResNet's systematic approach to depth scaling with efficiency considerations established the blueprint for modern deep learning architectures.

Question 5. Adversarial Attacks and Defenses (25 marks)
Based on cybersecurity research and university machine learning security courses.

(a) Formulate adversarial attack optimization problems for image classification: (10 marks)

- Targeted attack: $\min_\delta |||\delta|||_p$ subject to $f(x+\delta) = t$ and $|||\delta|||_\infty \leq \epsilon$
- Untargeted attack: $\min_\delta |||\delta|||_p$ subject to $f(x+\delta) \neq y$ and $|||\delta|||_\infty \leq \epsilon$
- Explain the role of L_p norms in constraint formulation

Answer: Adversarial attacks are formulated as constrained optimization problems that find minimal perturbations to fool neural networks while remaining imperceptible to humans.

General Adversarial Attack Framework:

Given a classifier $f : \mathbb{R}^n \to \{1, 2, ..., C\}$, input x with true label y, find perturbation δ such that the adversarial example $x' = x + \delta$ fools the classifier while remaining imperceptible.

Targeted Attack Formulation:

Objective: Force the model to predict a specific target class $t \neq y$

Optimization Problem:

$$\min_\delta \|\delta\|_p \text{ subject to } f(x+\delta) = t \text{ and } \|\delta\|_\infty \leq \epsilon$$

Alternative Formulation (Lagrangian):

$$\min_\delta \|\delta\|_p + \lambda \mathcal{L}(f(x+\delta), t) \text{ subject to } \|\delta\|_\infty \leq \epsilon$$

where $\mathcal{L}$ is the loss function (e.g., cross-entropy).

Practical Implementation:

$$\min_\delta \lambda \|\delta\|_p - \log P(y = t|x+\delta) \text{ subject to } x + \delta \in [0,1]^n$$

Untargeted Attack Formulation:

Objective: Force the model to predict any class except the true class y

Optimization Problem:

$$\min_{\delta} \|\delta\|_p \text{ subject to } f(x+\delta) \neq y \text{ and } \|\delta\|_\infty \leq \epsilon$$

Alternative Formulation:

$$\max_{\delta} \mathcal{L}(f(x+\delta), y) \text{ subject to } \|\delta\|_p \leq \epsilon$$

This maximizes the loss for the true class, making the model less confident about the correct prediction.

Practical Implementation:

$$\min_{\delta} \lambda\|\delta\|_p + \log P(y = y|x+\delta) \text{ subject to } x+\delta \in [0,1]^n$$

Role of L_p Norms in Constraint Formulation:

1. L_∞ Norm (Chebyshev Distance):

$$\|\delta\|_\infty = \max_i |\delta_i|$$

Interpretation:

- Bounds the maximum change to any single pixel
- $\|\delta\|_\infty \leq \epsilon$ means each pixel changed by at most ϵ
- Most commonly used in image attacks
- Perceptually meaningful: uniform brightness change bound

Example: For 8-bit images (0-255), $\epsilon = 8$ means each pixel changes by at most 8 intensity levels.

2. L_2 Norm (Euclidean Distance):

$$\|\delta\|_2 = \sqrt{\sum_i \delta_i^2}$$

Interpretation:

- Bounds the total energy of the perturbation
- Allows larger changes to fewer pixels
- More concentrated perturbations
- Natural distance metric in high-dimensional spaces

3. L_0 Norm (Hamming Distance):

$$\|\delta\|_0 = |\{i : \delta_i \neq 0\}|$$

Interpretation:

- Counts number of pixels that are changed
- Sparse attacks: modify only few pixels
- Computationally harder (non-convex)
- Realistic attack scenario: change minimal pixels

4. L_1 Norm (Manhattan Distance):

$$\|\delta\|_1 = \sum_i |\delta_i|$$

Interpretation:

- Promotes sparsity (fewer non-zero perturbations)
- Total absolute change across all pixels
- Balance between L_0 and L_2 properties

Constraint Comparison:

Norm	Constraint Shape	Practical Meaning
L_∞	Hypercube	Uniform pixel change bound
L_2	Hypersphere	Total energy constraint
L_1	Diamond	Sparse total change
L_0	Discrete set	Number of changed pixels

Choice of Norm Depends on:

- **Threat Model:** What kind of perturbations are realistic?
- **Perceptual Constraints:** What changes are imperceptible?
- **Computational Efficiency:** Which norms enable efficient optimization?
- **Application Domain:** Different domains have different natural metrics

(b) Implement the Fast Gradient Sign Method (FGSM): (10 marks)

- Derive FGSM formula: $x' = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))$
- Explain why FGSM is effective against linear behavior in neural networks
- Calculate perturbation for a binary classification problem with cross-entropy loss
- Discuss computational efficiency compared to iterative methods

Answer: FGSM exploits the linear nature of neural networks in high-dimensional spaces by taking a single step in the direction of maximum gradient to efficiently generate adversarial examples.

FGSM Derivation:

Starting Point: Untargeted attack optimization

$$\max_{\delta} \mathcal{L}(f(x+\delta), y) \text{ subject to } \|\delta\|_\infty \leq \epsilon$$

Linear Approximation: Using first-order Taylor expansion around x:

$$\mathcal{L}(f(x+\delta), y) \approx \mathcal{L}(f(x), y) + \delta^T \nabla_x \mathcal{L}(f(x), y)$$

Optimization Problem:

$$\max_{\delta} \delta^T \nabla_x \mathcal{L} \text{ subject to } \|\delta\|_\infty \leq \epsilon$$

Analytical Solution: To maximize $\delta^T g$ where $g = \nabla_x \mathcal{L}$ subject to $\|\delta\|_\infty \leq \epsilon$:

The optimal δ aligns with the sign of g:

$$\delta_i^* = \epsilon \cdot \text{sign}(g_i) = \epsilon \cdot \text{sign}\left(\frac{\partial \mathcal{L}}{\partial x_i}\right)$$

Final FGSM Formula:

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(\theta, x, y))$$

where θ represents model parameters.

Why FGSM is Effective Against Linear Behavior:

1. High-Dimensional Linear Approximation:

In high-dimensional spaces (e.g., images with millions of pixels):

- Local behavior of neural networks is approximately linear
- Small perturbations don't trigger significant non-linearities
- Linear approximation becomes more accurate in higher dimensions

2. Curse of Dimensionality for Defense:

Accumulated Perturbation Effect: Even small perturbations per dimension accumulate:

Total perturbation effect: $\sum_{i=1}^{n} \epsilon \cdot \text{sign}(g_i) \cdot g_i = \epsilon \sum_{i=1}^{n} |g_i| = \epsilon \|\nabla_x \mathcal{L}\|_1$

For high-dimensional inputs, $\|\nabla_x \mathcal{L}\|_1$ can be very large even if individual gradients are small.

3. Gradient Information Exploitation:

- Gradients reveal the direction of maximum loss increase
- Sign function extracts only directional information
- Ignores gradient magnitudes, focuses on directions
- Efficient way to find worst-case perturbations

Binary Classification Example:

Setup:

- Input: $x \in \mathbb{R}^n$

- True label: $y \in \{0, 1\}$
- Model output: $f(x) \in [0, 1]$ (probability of class 1)
- Loss: Cross-entropy $\mathcal{L} = -y \log f(x) - (1 - y) \log(1 - f(x))$

Gradient Calculation:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial f} \frac{\partial f}{\partial x}$$

Cross-entropy gradient w.r.t. output:

$$\frac{\partial \mathcal{L}}{\partial f} = -\frac{y}{f} + \frac{1 - y}{1 - f} = \frac{f - y}{f(1 - f)}$$

Complete gradient:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{f(x) - y}{f(x)(1 - f(x))} \frac{\partial f(x)}{\partial x}$$

FGSM Perturbation:

$$\delta = \epsilon \cdot \text{sign} \left(\frac{f(x) - y}{f(x)(1 - f(x))} \frac{\partial f(x)}{\partial x} \right)$$

Case Analysis:

For $y = 1$ (true class 1):

- If $f(x) > 0.5$ (correct prediction): $f(x) - y < 0$, so perturbation opposes gradient direction
- Goal: Reduce confidence in class 1

For $y = 0$ (true class 0):

- If $f(x) < 0.5$ (correct prediction): $f(x) - y < 0$, so perturbation opposes gradient direction
- Goal: Increase false confidence in class 1

Computational Efficiency Analysis:

FGSM Computational Cost:

- **Forward pass:** $O(N)$ where N is number of operations in network
- **Backward pass:** $O(N)$ to compute gradients
- **Sign operation:** $O(d)$ where d is input dimensionality
- **Total:** $O(N + d) \approx O(N)$ since typically $N \gg d$

Iterative Methods (e.g., PGD) Computational Cost:

- k **iterations of gradient computation:** $k \times O(N)$
- **Projection steps:** $k \times O(d)$
- **Total:** $O(k \times N)$ where typically $k = 10 - 100$

Efficiency Comparison:

Method	Forward Passes	Backward Passes	Relative Cost
FGSM	1	1	1×
PGD-10	10	10	10×
PGD-100	100	100	100×
C&W	1000+	1000+	1000×+

FGSM Advantages:

- **Speed:** Single gradient computation
- **Simplicity:** Easy to implement and understand
- **Baseline:** Good starting point for more sophisticated attacks
- **Real-time:** Fast enough for online attack generation

FGSM Limitations:

- **Linear assumption:** May not be optimal for highly non-linear regions
- **Single step:** Cannot refine perturbations iteratively
- **Success rate:** Lower than iterative methods
- **Transferability:** May not transfer well across different models

FGSM provides an excellent balance between computational efficiency and attack effectiveness, making it a popular choice for both attack generation and adversarial training.

(c) Analyze defense strategies against adversarial attacks: (5 marks)

- Adversarial training: training with adversarial examples
- Defensive distillation: temperature-based softmax smoothing
- Detection methods: statistical analysis of input distributions
- Trade-offs between robustness and accuracy

Answer: Defense strategies aim to improve model robustness through various approaches, each with specific strengths and limitations, while facing fundamental trade-offs between robustness and standard accuracy.

1. Adversarial Training:

Basic Principle: Include adversarial examples in the training process to improve robustness.

Mathematical Formulation:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\|\delta\|\leq\epsilon} \mathcal{L}(\theta, x+\delta, y) \right]$$

This is a min-max optimization: minimize worst-case loss over perturbations.

Implementation:

(a) Generate adversarial examples: $x_{adv} = x + \arg\max_{\|\delta\|\leq\epsilon} \mathcal{L}(\theta, x+\delta, y)$

(b) Train on mixture: $\mathcal{L}_{\text{total}} = \lambda\mathcal{L}(\theta, x, y) + (1-\lambda)\mathcal{L}(\theta, x_{adv}, y)$

Advantages:

- Principled approach to robustness
- Directly optimizes for worst-case performance
- Effective against known attack methods
- Theoretically motivated

Limitations:

- Computationally expensive (2-10× training time)
- May overfit to specific attack methods

- Reduced clean accuracy
- Gradient masking potential

2. Defensive Distillation:

Basic Principle: Train networks to produce smoother probability distributions using knowledge distillation with temperature scaling.

Mathematical Framework:

Temperature Softmax:

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

where $T > 1$ is the temperature parameter.

Two-Stage Process:

(a) **Teacher Training:** Train initial network with temperature T

(b) **Student Training:** Train final network to match teacher's soft targets at temperature T

Student Loss:

$$\mathcal{L}_{\text{distill}} = \text{KL}(\text{softmax}(z_{\text{teacher}}/T), \text{softmax}(z_{\text{student}}/T))$$

Robustness Mechanism:

- Higher temperature creates smoother gradients
- Reduces gradient magnitude: $\|\nabla_x \mathcal{L}\|$ becomes smaller
- Makes FGSM and gradient-based attacks less effective

Limitations:

- Broken by adaptive attacks (C&W attack)
- Primarily defends against specific gradient-based methods
- May not improve robustness against other attack types
- Effectiveness questioned by later research

3. Detection Methods:

Statistical Detection:

Input Statistics:

- Monitor input distribution changes
- Detect outliers using statistical tests
- Compare against training data distribution

Activation Pattern Analysis:

- Analyze intermediate layer activations
- Detect unusual activation patterns
- Use reconstruction errors as indicators

Mathematical Approaches:

Kernel Density Estimation:

$$p(x) = \frac{1}{n} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Flag inputs with $p(x) < \tau$ as potential adversarial examples.

Mahalanobis Distance:

$$M(x) = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)}$$

where μ and Σ are estimated from training data.

Advantages:

- Can be added to existing models
- Computationally efficient at inference
- Provides interpretable rejection mechanism

Limitations:

- Vulnerable to adaptive attacks designed to evade detection
- High false positive rates

- May not generalize to new attack methods
- Can be circumvented by distributional mimicry

4. Robustness vs. Accuracy Trade-offs:

Fundamental Tension:

Theoretical Analysis: There appears to be a fundamental trade-off between robustness and accuracy:

$$\text{Clean Accuracy} + \text{Robustness} \leq \text{Constant}$$

Empirical Evidence:

- Adversarial training typically reduces clean accuracy by 5-15%
- More robust models often have lower standard accuracy
- Perfect robustness may be incompatible with high accuracy

Causes of Trade-off:

1. Feature Learning Conflicts:

- Standard training learns discriminative but fragile features
- Robust training learns stable but potentially less discriminative features
- These objectives can be conflicting

2. Capacity Limitations:

- Model capacity must be split between clean and robust performance
- Finite capacity creates fundamental constraints
- Larger models can partially mitigate but not eliminate trade-offs

3. Data Distribution Effects:

- Training data may not cover adversarial regions adequately
- Robust features may be harder to learn from natural data
- Distribution mismatch between training and attack scenarios

Mitigation Strategies:

- **Architecture Design:** Use architectures naturally more robust
- **Data Augmentation:** Improve natural data diversity
- **Multi-task Learning:** Balance multiple objectives
- **Certified Defense:** Provide guarantees within specific bounds

Practical Implications:

- Need to choose appropriate robustness-accuracy balance for application
- Defense effectiveness depends on threat model
- No single defense strategy is universally effective
- Combination of multiple defenses often most practical

The ongoing arms race between attacks and defenses continues to drive research in both directions, with each new defense spurring development of adaptive attacks.

Question 6. Architecture Comparison and Evolution (20 marks)
Based on comprehensive analysis from multiple university computer vision courses.

(a) Create a comparative analysis table for CNN architectures: (12 marks)

Architecture	Depth	Parameters	Top-5 Error	Key Innovation
AlexNet (2012)	8	60M	15.3%	**ReLU + GPU + Dropout**
VGG-16 (2014)	16	138M	**7.3%**	**Small 3×3 filters**
GoogleNet (2014)	22	4M	**6.7%**	**Inception modules**
ResNet-152 (2015)	152	**60M**	**3.57%**	**Skip connections**

Complete the table and analyze:

- Parameter efficiency trends over time
- Relationship between depth and performance
- Trade-offs between accuracy and computational cost

Answer: CNN architecture evolution demonstrates increasing efficiency through architectural innovations, with performance improvements achieved through depth scaling and parameter optimization rather than just increasing model size.

Parameter Efficiency Trends Analysis:

Evolution Pattern:

- **AlexNet (2012):** 60M parameters → 15.3% error = 0.255% error per million parameters
- **VGG-16 (2014):** 138M parameters → 7.3% error = 0.053% error per million parameters
- **GoogleNet (2014):** 4M parameters → 6.7% error = 1.675% error per million parameters
- **ResNet-152 (2015):** 60M parameters → 3.57% error = 0.0595% error per million parameters

Key Observations:

1. GoogleNet's Revolutionary Efficiency:

- Achieved near-VGG performance with 97% fewer parameters
- Demonstrated that architectural innovation > parameter scaling
- Inception modules enabled efficient multi-scale processing
- Global Average Pooling eliminated parameter-heavy FC layers

2. Parameter Efficiency Metrics:

Efficiency Score = Performance / Parameters:

- GoogleNet: Most parameter efficient (6.7% error with 4M parameters)
- ResNet-152: Best absolute performance with reasonable parameter count
- VGG-16: Least efficient (high parameters for modest performance gain)
- AlexNet: Baseline efficiency for its era

3. Innovation Impact: Each architecture introduced paradigm shifts:

- **AlexNet:** Proved deep learning viability for computer vision
- **VGG:** Showed benefits of smaller, deeper filters
- **GoogleNet:** Demonstrated multi-scale processing efficiency
- **ResNet:** Enabled ultra-deep networks through skip connections

Relationship Between Depth and Performance:

Depth Scaling Analysis:

Performance vs. Depth Correlation:

- AlexNet (8 layers): 15.3% error
- VGG-16 (16 layers): 7.3% error (7.8 point improvement for 2× depth)
- GoogleNet (22 layers): 6.7% error (0.6 point improvement for 1.4× depth)
- ResNet-152 (152 layers): 3.57% error (3.13 point improvement for 6.9× depth)

Diminishing Returns Pattern:

$$\text{Performance Gain} \propto \log(\text{Depth Increase})$$

Each doubling of depth provides smaller improvements, but ResNet broke this pattern by enabling much deeper networks.

Architectural Enablers of Depth:

Before ResNet: Depth limited by vanishing gradients

- VGG struggled beyond 19 layers
- GoogleNet needed auxiliary classifiers for 22 layers
- Training instability increased with depth

After ResNet: Skip connections enabled arbitrary depth

- ResNet-152: 152 layers trained successfully
- ResNet-1001: Experimental ultra-deep networks
- Consistent performance improvement with depth

Trade-offs Between Accuracy and Computational Cost:

Computational Complexity Analysis:

Architecture	FLOPs (B)	Memory (MB)	Inference (ms)	Accuracy/Cost
AlexNet	0.7	249	2.3	High cost, low accuracy
VGG-16	15.5	528	69.5	Highest cost, medium accuracy
GoogleNet	1.6	16	6.8	Low cost, high accuracy
ResNet-152	11.3	230	29.4	Medium cost, highest accuracy

Efficiency Frontiers:

1. GoogleNet: The Efficiency Champion

- Best accuracy-per-FLOP ratio
- Smallest memory footprint
- Fastest inference while maintaining high accuracy
- Demonstrated that smart architecture > brute force scaling

2. ResNet: The Performance Leader

- Best absolute accuracy
- Reasonable computational cost for performance gained
- Scalable architecture (can trade depth for accuracy)
- Good balance of performance and practicality

3. VGG: The Computational Burden

- Highest computational cost across all metrics
- Diminishing returns for computational investment
- Useful for feature extraction but impractical for deployment
- Historical importance but poor efficiency

Design Philosophy Evolution:

Era 1 (AlexNet): "Bigger is better"

- Focus on raw parameter count
- Computational constraints secondary
- Proof of concept for deep learning

Era 2 (VGG): "Deeper is better"

- Systematic depth scaling
- Uniform architecture design
- Computational cost becomes concern

Era 3 (GoogleNet): "Smarter is better"

- Architectural innovation over scaling
- Multi-scale processing
- Efficiency as primary concern

Era 4 (ResNet): "Deeper and smarter"

- Fundamental training problem solved
- Both depth scaling and efficiency

- Balance of performance and practicality

Practical Implications:

Architecture Selection Criteria:

- **High Accuracy Applications:** ResNet variants
- **Resource-Constrained Environments:** GoogleNet-inspired designs
- **Feature Extraction:** VGG (despite inefficiency, good features)
- **Historical Baseline:** AlexNet (educational purposes)

This analysis shows that architectural innovation, not just parameter scaling, drives progress in deep learning efficiency and performance.

(b) Evaluate architectural design principles: (8 marks)

- Why smaller filter sizes (3×3) became preferred over larger ones (11×11, 7×7)
- Role of skip connections in enabling very deep networks
- Impact of global average pooling vs. fully connected layers
- Evolution from hand-crafted to learnable architectures

Answer: Architectural design principles evolved through empirical discoveries and theoretical insights, leading to more efficient, trainable, and generalizable networks through systematic design improvements.

Preference for Smaller Filter Sizes (3×3):

1. Representational Power vs. Efficiency:

Receptive Field Equivalence: Two 3×3 convolutions achieve the same receptive field as one 5×5:

- Layer 1: 3×3 receptive field
- Layer 2: $(3 - 1) + 3 = 5 \times 5$ effective receptive field
- Same coverage with more non-linearity

Parameter Comparison:

- Single 5×5 layer: $25C^2$ parameters

- Two 3×3 layers: $2 \times 9C^2 = 18C^2$ parameters
- Savings: $(25 - 18)/25 = 28\%$ parameter reduction

2. Increased Non-linearity:

- Two layers $\rightarrow$ Two ReLU activations
- More complex decision boundaries
- Better feature learning capability
- Improved approximation of complex functions

3. Better Gradient Flow:

- Shorter paths reduce vanishing gradients
- Each layer easier to train individually
- More frequent gradient updates per receptive field

4. Computational Efficiency:

- Better memory access patterns
- More efficient GPU utilization
- Easier to parallelize smaller convolutions
- Reduced computational complexity

Role of Skip Connections in Deep Networks:

1. Gradient Flow Preservation:

Mathematical Guarantee:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} \left(\frac{\partial F(x)}{\partial x} + I \right)$$

The identity term I ensures gradients never completely vanish.

2. Identity Mapping Learning:

- Easy to learn identity when no transformation needed
- Network can choose between transformation and preservation
- Adaptive feature processing based on depth requirements

3. Feature Reuse and Combination:

- Low-level features preserved through skip connections
- High-level processing doesn't destroy useful low-level information
- Multi-scale feature combinations naturally emerge

4. Training Stability:

- Reduced sensitivity to initialization
- More stable optimization landscapes
- Lower risk of gradient explosion/vanishing
- Faster convergence in practice

Impact of Global Average Pooling vs. FC Layers:

Parameter Reduction Analysis:

Traditional FC Approach (AlexNet-style):

- Final conv: $6 \times 6 \times 256$ feature maps
- FC layer: $(6 \times 6 \times 256) \times 4096 = 37.75M$ parameters
- Dominates total parameter count (>90%)

GAP Approach (GoogleNet-style):

- Same conv layer: $6 \times 6 \times 256$ feature maps
- GAP: 256×1 vector (no parameters)
- Classification: $256 \times 1000 = 256K$ parameters
- Reduction: $\frac{37.75M - 0.256M}{37.75M} = 99.3\%$

Benefits Beyond Parameter Reduction:

1. Spatial Invariance:

- Object location doesn't affect classification
- Natural translation invariance
- Robust to spatial variations

2. Interpretability:

- Each channel becomes semantic detector
- Direct correspondence to class activation maps
- Easier visualization of learned features

3. Overfitting Resistance:

- Fewer parameters → less overfitting risk
- No complex weight interactions between spatial locations
- More robust generalization

Evolution from Hand-crafted to Learnable Architectures:

Hand-crafted Era (Pre-2012):

Design Process:

- Manual feature engineering (SIFT, HOG, etc.)
- Fixed architectural patterns
- Domain expert knowledge required
- Limited adaptability to new domains

Early Deep Learning (2012-2015):

Empirical Architecture Design:

- Trial-and-error architecture search
- Incremental improvements on successful designs
- Limited systematic understanding
- Heavy reliance on expert intuition

Modern Era (2016+):

Neural Architecture Search (NAS):

- Automated architecture discovery
- Search spaces defined by humans, architectures found by AI
- Multi-objective optimization (accuracy, efficiency, latency)
- Data-driven design decisions

Key Principles Discovered:

1. Modularity:

- Repeatable building blocks (ResNet blocks, Inception modules)
- Composable architectural elements
- Systematic scaling laws

2. Multi-objective Optimization:

- Balance accuracy, speed, memory, energy
- Pareto-optimal architecture families
- Task-specific architecture adaptation

3. Search Space Design:

- Well-designed search spaces more important than search algorithms
- Hierarchical search (macro $\rightarrow$ micro architecture)
- Transfer of architectural insights across domains

Future Directions:

- Fully automated end-to-end architecture design
- Hardware-aware architecture optimization
- Multi-domain architecture transfer
- Theoretical understanding of optimal architectures

The evolution demonstrates a shift from human intuition to data-driven design, while maintaining the importance of fundamental architectural principles discovered through empirical research.

Question 7. Feature Visualization and Interpretability (20 marks) Based on interpretable AI research and university courses on explainable deep learning.

(a) Design feature inversion techniques for understanding CNN representations: (10 marks)

- Formulate optimization: $\min_x |||f(x) - f_0|||^2 + \lambda R(x)$
- Explain different regularization terms $R(x)$: total variation, L_2 norm
- Implement gradient-based optimization for feature reconstruction
- Analyze why early layers preserve more spatial information than later layers

Answer: Feature inversion reconstructs images from neural network representations, revealing what information is preserved at different network depths through constrained optimization with natural image priors.

Feature Inversion Optimization Formulation:

Core Problem: Given a target feature representation f_0 from a specific layer of a pre-trained CNN, find an input image x^* that produces a similar feature representation.

Mathematical Formulation:

$$x^* = \arg\min_x |||f(x) - f_0|||^2 + \lambda R(x)$$

where:

- $f(x)$ is the feature representation of image x at the chosen layer
- f_0 is the target feature representation
- $R(x)$ is the regularization term enforcing natural image properties
- λ controls the regularization strength

Detailed Component Analysis:

1. Feature Matching Term:

$$\mathcal{L}_{\text{feature}} = |||f(x) - f_0|||^2 = \sum_{i,j,k} (f_{i,j,k}(x) - f_{0,i,j,k})^2$$

This ensures the reconstructed image activates the network similarly to the original.

2. Regularization Terms $R(x)$:

Total Variation (TV) Regularization:

$$R_{TV}(x) = \sum_{i,j,c} \left[(x_{i+1,j,c} - x_{i,j,c})^2 + (x_{i,j+1,c} - x_{i,j,c})^2 \right]^{\beta/2}$$

where $\beta \in \{1, 2\}$ controls the penalty type.

Purpose:

- Promotes spatial smoothness
- Reduces high-frequency noise
- Encourages natural image statistics
- Prevents checkerboard artifacts

L2 Norm Regularization:

$$R_{L2}(x) = |||x|||^2 = \sum_{i,j,c} x_{i,j,c}^2$$

Purpose:

- Controls overall image magnitude
- Prevents extreme pixel values
- Encourages sparse/low-energy solutions
- Avoids saturation artifacts

Alpha Norm Regularization:

$$R_\alpha(x) = \sum_{i,j,c} |x_{i,j,c}|^\alpha$$

where $\alpha \in (1, 2)$ (commonly $\alpha = 1.5$ or $\alpha = 6$).

Purpose:

- Promotes natural image intensity distributions
- Balances sparsity and smoothness
- Matches empirical pixel statistics

Combined Regularization:

$$R(x) = \lambda_1 R_{TV}(x) + \lambda_2 R_{L2}(x) + \lambda_3 R_\alpha(x)$$

Gradient-based Optimization Implementation:

Algorithm Overview:

(a) Initialize $x^{(0)}$ (usually random noise or mean image)

(b) For $t = 1, 2, ..., T$:

- Compute feature representation: $f(x^{(t-1)})$
- Compute loss: $\mathcal{L} = |||f(x^{(t-1)}) - f_0|||^2 + \lambda R(x^{(t-1)})$
- Compute gradient: $\nabla_x \mathcal{L}$
- Update: $x^{(t)} = x^{(t-1)} - \eta \nabla_x \mathcal{L}$
- Project to valid range: $x^{(t)} = \text{clip}(x^{(t)}, 0, 1)$

(c) Return $x^{(T)}$

Gradient Computation:

Feature Loss Gradient:

$$\frac{\partial \mathcal{L}_{\text{feature}}}{\partial x} = 2 \sum_{i,j,k} (f_{i,j,k}(x) - f_{0,i,j,k}) \frac{\partial f_{i,j,k}(x)}{\partial x}$$

This requires backpropagation through the CNN to the input layer.

TV Regularization Gradient:

$$\frac{\partial R_{TV}}{\partial x_{i,j,c}} = \beta \left[(x_{i+1,j,c} - x_{i,j,c}) + (x_{i,j+1,c} - x_{i,j,c}) \right] \left[\sum \text{differences}^2 \right]^{\beta/2-1}$$

Implementation Details:

1. Learning Rate Scheduling:

- Start with higher learning rate (0.1-1.0)

- Decay exponentially or use adaptive methods (Adam)
- Fine-tune with lower rates for detail preservation

2. Multi-scale Optimization:

- Start with low resolution, progressively increase
- Helps avoid local minima
- Captures both coarse and fine details

3. Initialization Strategies:

- Random noise: Unbiased but may get stuck in poor local minima
- Mean image: Stable but may bias toward average features
- Multiple random restarts: Best results but computationally expensive

Layer-wise Information Preservation Analysis:

Early Layer Reconstruction (e.g., Conv1-Conv2):

Characteristics:

- Nearly perfect pixel-level reconstruction
- Preserves fine textures and exact colors
- Maintains sharp edges and detailed patterns
- High spatial resolution information retained

Mathematical Explanation: Early layers have small receptive fields and preserve local information:

- Receptive field size: $\sim 3 \times 3$ to 11×11
- Direct correspondence between input pixels and features
- Minimal information loss through pooling
- Features are essentially local texture descriptors

Middle Layer Reconstruction (e.g., Conv3-Conv4):

Characteristics:

- Good shape and structure preservation

- Some texture detail loss
- Colors may be approximate
- Spatial layout maintained but simplified

Mathematical Explanation:

- Receptive field: $\sim 51 \times 51$ to 131×131
- Pooling operations reduce spatial resolution
- Features represent mid-level patterns (object parts)
- Information becomes more semantic, less pixel-specific

Late Layer Reconstruction (e.g., Conv5, FC layers):

Characteristics:

- Semantic content preserved
- Significant spatial detail loss
- Colors and textures highly approximate
- Object shapes recognizable but simplified

Mathematical Explanation:

- Large receptive fields: entire image
- Multiple pooling operations accumulate information loss
- Features represent high-level semantic concepts
- Spatial information heavily downsampled

Information Theory Perspective:

Mutual Information Analysis:

$$I(\text{Image}; \text{Features}) \text{ decreases with depth}$$

$$I(\text{Class}; \text{Features}) \text{ increases with depth}$$

This trade-off explains why reconstruction quality decreases but classification performance improves with depth.

Practical Applications:

1. Understanding Network Representations:

- Visualize what different layers learn
- Debug network training problems
- Compare different architectures

2. Artistic Applications:

- Style transfer algorithms
- Deep dream generation
- Creative image synthesis

3. Adversarial Analysis:

- Understand vulnerability sources
- Analyze feature robustness
- Design better defenses

(b) Evaluate saliency map generation methods: (10 marks)

- Vanilla gradients: $S_i = \left|\frac{\partial f_c}{\partial x_i}\right|$
- Integrated gradients: addressing gradient saturation problems
- LIME (Local Interpretable Model-agnostic Explanations): local approximation approach
- Quantitative evaluation metrics for explanation quality

Answer: Saliency map methods provide different approaches to understanding model decisions, each with specific strengths in handling various limitations of gradient-based explanations.

Vanilla Gradients Method:

Mathematical Formulation:

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

where $f_c(x)$ is the model's output for class c and x_i is the i-th input feature.

Implementation:

(a) Forward pass: Compute $f_c(x)$ for target class

(b) Backward pass: Compute gradients $\nabla_x f_c(x)$

(c) Generate saliency: $S = |\nabla_x f_c(x)|$

(d) Optionally normalize: $S = \frac{S - \min(S)}{\max(S) - \min(S)}$

Advantages:

- Computationally efficient (single forward + backward pass)
- Direct interpretation: measures local sensitivity
- Easy to implement and understand
- Real-time generation possible

Limitations:

- **Gradient Saturation:** ReLU networks can have zero gradients in saturated regions
- **Noise Sensitivity:** Can highlight irrelevant high-frequency components
- **Shattered Gradients:** Modern deep networks often have noisy gradient landscapes
- **Limited Scope:** Only captures first-order local information

Integrated Gradients Method:

Motivation: Address gradient saturation and provide more stable attributions

Mathematical Formulation:

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

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

Practical Implementation:

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

with m typically 20-300 steps.

Theoretical Foundation:

Axioms Satisfied:

(a) **Sensitivity:** If inputs differ only in feature i and outputs differ, then $IG_i \neq 0$

(b) **Implementation Invariance:** Identical attributions for functionally equivalent networks

(c) **Linearity:** Attribution for sum of functions equals sum of attributions

(d) **Completeness:** $\sum_i IG_i = f(x) - f(x')$

Advantages over Vanilla Gradients:

- **Reduced Saturation:** Integrates gradients along path, avoids local saturation
- **Stable Attributions:** Less sensitive to irrelevant features
- **Theoretical Guarantees:** Satisfies desirable axioms
- **Baseline Independence:** Results relatively stable across baseline choices

Limitations:

- **Computational Cost:** Requires m gradient computations (20-300× slower)
- **Baseline Selection:** Results can depend on baseline choice
- **Path Dependence:** Different integration paths may give different results
- **Approximation Errors:** Numerical integration introduces errors

LIME (Local Interpretable Model-agnostic Explanations):

Core Concept: Approximate the model locally around the instance using an interpretable model.

Mathematical Framework:

$$\text{explanation}(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

where:

- g is an interpretable model from class G (e.g., linear model)
- $\mathcal{L}$ measures how well g approximates f in locality π_x
- $\Omega(g)$ is complexity penalty for g
- π_x defines neighborhood around x

Algorithm for Images:

(a) **Superpixel Segmentation:** Divide image into interpretable segments

(b) **Perturbation Generation:** Create samples by turning superpixels on/off

(c) **Distance Weighting:** Weight samples by distance from original

(d) **Model Training:** Fit linear model to predict $f(x)$ from binary features

(e) **Feature Selection:** Select top-k most important superpixels

Detailed Implementation:

1. Locality Definition:

$$\pi_x(z) = \exp\left(-\frac{D(x,z)^2}{\sigma^2}\right)$$

where $D(x, z)$ measures distance between original and perturbed instances.

2. Local Linear Model:

$$g(z) = w_0 + \sum_{i=1}^{d} w_i z_i$$

where $z_i \in \{0, 1\}$ indicates whether superpixel i is present.

3. Optimization:

$$\min_w \sum_{z,z' \in Z} \pi_x(z)(f(z) - g(z'))^2 + \lambda ||w||_1$$

Advantages:

- **Model Agnostic:** Works with any ML model (CNN, RNN, etc.)
- **Interpretable Features:** Uses meaningful segments rather than pixels
- **Local Fidelity:** Accurate approximation in neighborhood
- **User Control:** Can specify desired explanation complexity

Limitations:

- **Segmentation Dependence:** Results depend on superpixel quality
- **Locality Definition:** Choice of neighborhood affects explanations
- **Linear Assumption:** May not capture complex local interactions
- **Computational Cost:** Requires many model evaluations

Quantitative Evaluation Metrics:

1. Deletion/Insertion Metrics:

Deletion Curve:

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

where M_k masks the top-k most salient pixels.

Insertion Curve:

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

Area Under Curve (AUC): Lower deletion AUC and higher insertion AUC indicate better explanations.

2. Pointing Game:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[\arg\max_p S_i(p) \in \text{BBox}_i]$$

Measures if maximum saliency point falls within ground truth object.

3. Localization Metrics:

IoU with Ground Truth:

$$\text{IoU} = \frac{|\text{Saliency Region} \cap \text{GT Object}|}{|\text{Saliency Region} \cup \text{GT Object}|}$$

4. Stability Metrics:

Input Perturbation Stability:

$$\text{Stability} = 1 - \frac{||S(x+\epsilon) - S(x)||_2}{||S(x)||_2}$$

where ϵ is small random noise.

5. Faithfulness Metrics:

Correlation with Model Gradients:

$$\text{Faithfulness} = \text{Corr}(S, |\nabla_x f|)$$

Incremental Deletion Consistency: Measure how well saliency predicts performance drops when features are removed.

6. Human Evaluation Metrics:

Subjective Assessment:

- Semantic coherence ratings
- Trust and confidence measures
- Task performance improvement
- Time to make decisions with explanations

Comparative Analysis:

Method	Computational Cost	Faithfulness	Stability	Interpretability
Vanilla Gradients	Low	Medium	Low	High
Integrated Gradients	High	High	High	High
LIME	Medium	Medium	Medium	Very High

Method Selection Guidelines:

- **Real-time applications:** Vanilla gradients
- **High-stakes decisions:** Integrated gradients
- **Non-expert users:** LIME

- **Research/debugging:** Multiple methods for comparison

The choice of explanation method should depend on the specific requirements of computational efficiency, explanation quality, and target audience interpretability needs.

END OF PAPER

1. Natural Image Statistics:

- Objects appear at different scales in images
- Same object type varies in size (near vs. far)
- Important features exist at multiple spatial frequencies
- Fine details and coarse structures both matter for recognition

2. Information Distribution Across Scales:
Fine Scale (1×1, 3×3 filters):

- Local texture patterns
- Edge orientations and junctions
- Color variations
- Small discriminative features

Medium Scale (5×5 filters):

- Object parts (eyes, wheels, leaves)
- Medium-sized patterns and shapes
- Spatial relationships between local features
- Regional context information

Coarse Scale (pooling operations):

- Global shape information
- Spatial layout of major components

- Background context
- Overall object structure

3. Traditional CNN Limitation:

- Fixed kernel size per layer restricts information extraction
- Forces choice between detail preservation and context aggregation
- Suboptimal for objects with both fine and coarse important features
- Sequential processing may lose information at intermediate scales

Naive Inception Module Analysis:
Architecture Components:

1. **1×1 Convolution:** Point-wise feature combinations, C_1 filters
2. **3×3 Convolution:** Local pattern detection, C_2 filters
3. **5×5 Convolution:** Larger pattern detection, C_3 filters
4. **3×3 Max Pooling + 1×1 Conv:** Spatial reduction + feature extraction, C filters

Channel Growth Problem:
Output Channel Calculation:

$$\text{Output Channels} = C_1 + C_2 + C_3 + C$$

Growth Analysis: For meaningful representation, each branch needs sufficient channels:

- If input has $C = 256$ channels
- Reasonable branch outputs: $C_1 = 128, C_2 = 192, C_3 = 96, C = 64$
- Total output: $128 + 192 + 96 + 64 = 480$ channels
- Growth factor: $\frac{480}{256} = 1.875$ per layer

Exponential Growth Problem: After n layers: $C_n = C_0 \times 1.875^n$

- Layer 0: 256 channels
- Layer 3: $256 \times 1.875^3 1,687$ channels
- Layer 6: $256 \times 1.875^6 11,118$ channels
- Becomes computationally infeasible

Computational Cost Analysis:
For input $H \times W \times C$:

- 3×3 conv cost: $H \times W \times C \times C_2 \times 9$
- 5×5 conv cost: $H \times W \times C \times C_3 \times 25$
- Total cost: $O(HWC(9C_2 + 25C_3))$
- Grows quadratically with channel count

Improved Inception Module with Bottlenecks:
1×1 Convolution Benefits:
1. Dimensionality Reduction:

- Reduces channels before expensive operations
- Acts as learned linear combination across channels
- Minimal spatial computation (1×1 receptive field)
- Introduces non-linearity through activation functions

2. Channel Control Strategy:
Bottleneck Design:

1. **Reduce:** $C \to C/4$ using 1×1 conv
2. **Process:** Apply 3×3 or 5×5 conv on reduced channels
3. **Maintain:** Output $C/4$ channels per branch
4. **Concatenate:** $4 \times (C/4) = C$ total output channels

Mathematical Analysis:
Computational Savings:
Without Bottleneck (5×5 branch):

$$\text{Cost} = H \times W \times C \times (C/4) \times 25 = 6.25 \times HWC^2/4$$

With Bottleneck:

- 1×1 reduction: $H \times W \times C \times (C/4) = HWC^2/4$
- 5×5 conv: $H \times W \times (C/4) \times (C/4) \times 25 = 25HWC^2/16$
- Total: $HWC^2/4 + 25HWC^2/16 = (4 + 25)HWC^2/16 = 1.8125 \times HWC^2$

Speedup Factor:

$$\frac{6.25 \times HWC^2/4}{1.8125 \times HWC^2} = \frac{1.5625}{1.8125}0.86$$

Wait, let me recalculate:

$$\frac{6.25 \times HWC^2/4}{1.8125 \times HWC^2} = \frac{1.5625 \times HWC^2}{1.8125 \times HWC^2}0.86$$

Actually: $\frac{6.25}{1.8125}3.45$ speedup

4. Complete Module Design:
Branch Structure:

1. **Branch 1:** Input → 1×1 conv (C/4) → 3×3 conv (C/4)
2. **Branch 2:** Input → 1×1 conv (C/4) → 5×5 conv (C/4)
3. **Branch 3:** Input → 3×3 pool → 1×1 conv (C/4)
4. **Branch 4:** Input → 1×1 conv (C/4)

Key Properties:

- **Channel Conservation:** Output channels = Input channels
- **Multi-scale Processing:** Parallel extraction at different scales