EECS 182 Homework 5 Solutions

Problems 1, 2, 3, 4

Problem 1: Convolutional Networks

Part (a): Two reasons to prefer convolutional layers over fully connected layers

Answer:

1. Weight-sharing and translation equivariance: Convolutional layers use the same filter weights across different spatial locations in the image. This drastically reduces the number of parameters compared to fully connected layers. More importantly, convolutions build in translation invariance/equivariance - a feature detected in one part of an image can be detected anywhere else using the same learned weights. This is reasonable for images since the desired output for a translated image is closely related to the original image (e.g., a cat moved 10 pixels to the right is still a cat).

For example, a fully connected layer connecting a 224×224×3 RGB image to a 1000-neuron hidden layer would require: - Parameters: 224 × 224 × 3 × 1000 = 150,528,000 parameters

A convolutional layer with 64 filters of size 3×3 would only require: - Parameters: 3 × 3 × 3 × 64 = 1,728 parameters

This is a reduction by a factor of over 86,000!

2. Locality and hierarchical structure: Convolutions with finite-sized filters respect the spatial locality inherent in images, where nearby pixels are typically more related than distant pixels. By stacking convolutional layers, we can capture hierarchical part-whole relationships naturally found in images: - Layer 1: edges and simple textures - Layer 2: corners and more complex patterns - Layer 3: object parts - Layer 4: objects

This hierarchical composition allows the network to build complex features from simple ones in a structured way that matches the compositional nature of visual scenes.

Additional valid reasons include: - There is biological inspiration from how the visual cortex works, with receptive fields and edge-detecting responses - Historical success of filterbanks and convolutional structures in hand-designed computer vision approaches - Convolutions naturally handle different horizontal and vertical spatial relationships - Proven empirical success in the deep learning community

---

Part (b): Finding the filter from convolution output

Given: - Input signal: [1, 4, 0, -2, 3] - After convolution with length-3 filter, no padding, stride=1 - Output: [-2, 2, 11] - Find the filter: [h_1, h_2, h_3]

Solution:

The convolution operation (without flipping, as specified) at each position gives: - Position 0: $1 \cdot h_1 + 4 \cdot h_2 + 0 \cdot h_3 = -2$ - Position 1: $4 \cdot h_1 + 0 \cdot h_2 + (-2) \cdot h_3 = 2$ - Position 2: $0 \cdot h_1 + (-2) \cdot h_2 + 3 \cdot h_3 = 11$

This gives us the system of equations:

$$
\begin{aligned}
h_1 + 4h_2 &= -2 \quad \ldots (1)\\
4h_1 - 2h_3 &= 2 \quad \ldots (2)
\end{aligned}
$$

$$-2h_2 + 3h_3 = 11 \quad \ldots (3)$$

Solving by substitution:

From equation (1):

$$h_1 = -2 - 4h_2$$

From equation (3):

$$-2h_2 + 3h_3 = 11$$
$$3h_3 = 11 + 2h_2$$
$$h_3 = (11 + 2h_2)/3$$

Substitute both into equation (2):

$$4(-2 - 4h_2) - 2((11 + 2h_2)/3) = 2$$
$$-8 - 16h_2 - (22 + 4h_2)/3 = 2$$

Multiply through by 3:

$$-24 - 48h_2 - 22 - 4h_2 = 6$$
$$-46 - 52h_2 = 6$$
$$-52h_2 = 52$$
$$h_2 = -1$$

Back-substituting:

$$h_1 = -2 - 4(-1) = -2 + 4 = 2$$
$$h_3 = (11 + 2(-1))/3 = (11 - 2)/3 = 9/3 = 3$$

Answer: [2, -1, 3]

Verification: - Position 0: 1(2) + 4(-1) + 0(3) = 2 - 4 + 0 = -2 ✓ - Position 1: 4(2) + 0(-1) + (-2)(3) = 8 + 0 - 6 = 2 ✓ - Position 2: 0(2) + (-2)(-1) + 3(3) = 0 + 2 + 9 = 11 ✓

Part (c): Transpose convolution

Given: - Input: [[-1, 2], [3, 1]] - Filter: [[+1, -1], [0, +1]] - Parameters: pad=0, stride=1

Understanding transpose convolution:

Transpose convolution is designed to upsample a signal. The key idea is that each input element is multiplied by the entire filter, and these responses are placed at shifted positions corresponding to the input location. Where responses overlap, they are summed.

For a 2×2 input with a 2×2 filter, pad=0, stride=1: - Standard convolution would take a 3×3 input and produce a 2×2 output - Transpose convolution reverses this: 2×2 input → 3×3 output

Calculation:

Each input element contributes the filter shifted to its position:

Contribution from input element (-1) at position (0,0):

$$-1 \times \begin{bmatrix} +1 & -1 \\ 0 & +1 \end{bmatrix} = \begin{bmatrix} -1 & +1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

(Filter placed at top-left, extended to 3×3)

Contribution from input element (2) at position (0,1):

$$2 \times \begin{bmatrix} +1 & -1 \\ 0 & +1 \end{bmatrix} = \begin{bmatrix} 0 & +2 & -2 \\ 0 & 0 & +2 \\ 0 & 0 & 0 \end{bmatrix}$$

(Filter shifted one column right)

Contribution from input element (3) at position (1,0):

$$3 \times \begin{bmatrix} +1 & -1 \\ 0 & +1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ +3 & -3 & 0 \end{bmatrix}$$

```
                    [0, +3, 0]]
```

(Filter shifted one row down)

Contribution from input element (1) at position (1,1):

```
1 × [[+1, -1],  =  [[0, 0, 0],
     [0, +1]]       [0, +1, -1],
                    [0, 0, +1]]
```

(Filter shifted one row down and one column right)

Summing all contributions:

```
  [[-1, +1, 0],     [[0, +2, -2],     [[0, 0, 0],      [[0, 0, 0],
   [0, -1, 0],   +   [0, 0, +2],   +   [+3, -3, 0],  +  [0, +1, -1],
   [0, 0, 0]]        [0, 0, 0]]        [0, +3, 0]]      [0, 0, +1]]

=  [[-1, 3, -2],
    [3, -3, 1],
    [0, 3, 1]]
```

Answer:

```
[[-1,  3, -2],
 [ 3, -3,  1],
 [ 0,  3,  1]]
```

Problem 2: Batch Normalization for CNN

Part (a): Identifying normalization types

Given diagram shows three different normalization patterns: - Dimension N: mini-batch - Dimensions H, W: spatial dimensions (height, width) - Dimension C: channels

Batch Normalization (Answer: A): - Normalizes across the batch dimension N - For each channel c and each spatial position (h, w), compute statistics over all N samples - This means all samples in the batch contribute to the mean and variance for that specific channel and position - Results in independent normalization for each (channel, spatial position) combination

Layer Normalization (Answer: B): - Normalizes across channels C and spatial dimensions H, W - For each individual sample in the batch, compute statistics over all channels and all spatial positions - This means each sample is normalized independently using its own statistics - Results in normalization within each sample, independent of batch

Why these assignments? - Batch norm was designed for CNNs to normalize activations across the batch, maintaining spatial and channel structure - Layer norm normalizes all activations for a single sample, making it batch-size independent (useful for RNNs and transformers)

Part (b): Simplified batch normalization gradient

Setup: - Input batch: $[x_1, x_2, \ldots, x_n]$ - Mean: $\mu = (1/n)\Sigma_j x_j$ - De-meaned: $\hat{x}_i = x_i - \mu$ - Output: $y_i = \gamma\hat{x}_i + \beta$ - Loss: L (somewhere downstream)

Goal: Calculate $\partial L/\partial x_i$ in terms of $\partial L/\partial y_j$, γ, and β

Solution:

Since x_i affects all outputs y_j through the mean μ, we need to sum over all output gradients:

$$\partial L/\partial x_i = \Sigma_j \, (\partial L/\partial y_j)(\partial y_j/\partial x_i)$$

Calculate $\partial y_j/\partial x_i$:

The output y_j can be written as:

$$
\begin{aligned}
y_j &= \gamma(x_j - \mu) + \beta \\
&= \gamma(x_j - (1/n)\Sigma_k x_k) + \beta \\
&= \gamma x_j - (\gamma/n)\Sigma_k x_k + \beta
\end{aligned}
$$

Taking the derivative with respect to x_i: - The term γx_j contributes γ when i = j, and 0 otherwise - The term $-(\gamma/n)\Sigma_k x_k$ contributes $-\gamma/n$ for all i (since x_i appears in the sum) - The constant β contributes 0

Therefore:

$$
\partial y_j/\partial x_i = \begin{cases} \gamma(1 - 1/n), & \text{if } i = j \\ \gamma(-1/n), & \text{if } i \neq j \end{cases}
$$

Combining the sum:

$$
\begin{aligned}
\partial L/\partial x_i &= \Sigma_j \, (\partial L/\partial y_j)(\partial y_j/\partial x_i) \\
&= (\partial L/\partial y_i)\cdot\gamma(1 - 1/n) + \Sigma_{j\neq i} \, (\partial L/\partial y_j)\cdot\gamma(-1/n) \\
&= \gamma(\partial L/\partial y_i) - (\gamma/n)(\partial L/\partial y_i) - (\gamma/n)\Sigma_{j\neq i} \, (\partial L/\partial y_j) \\
&= \gamma(\partial L/\partial y_i) - (\gamma/n)[\partial L/\partial y_i + \Sigma_{j\neq i} \, (\partial L/\partial y_j)] \\
&= \gamma(\partial L/\partial y_i) - (\gamma/n)\Sigma_j \, (\partial L/\partial y_j)
\end{aligned}
$$

Final Answer:

$$
\partial L/\partial x_i = \gamma[\partial L/\partial y_i - (1/n)\Sigma_j(\partial L/\partial y_j)]
$$

Special case when n = 1:

$$
\partial L/\partial x_1 = \gamma[\partial L/\partial y_1 - \partial L/\partial y_1] = 0
$$

With a batch size of 1, the gradient with respect to the input is zero! This is because subtracting the mean from a single value always gives zero, so the operation has no gradient.

Limiting case when n → ∞:

As n approaches infinity, the term $(1/n)\Sigma_j(\partial L/\partial y_j)$ becomes negligible (assuming bounded gradients), so:

$$
\partial L/\partial x_i \to \gamma\cdot(\partial L/\partial y_i)
$$

Interpretation: With larger batch sizes, batch normalization has the ideal effect of isolating the gradients through each sample. The gradient for sample i depends primarily on its own downstream gradient, with decreasing influence from other samples in the batch as n grows. This helps stabilize training and reduces the coupling between samples in the batch.

Problem 3: Depthwise Separable Convolutions

Part (a): Traditional convolution parameters

Given: - Input: 3 channels, 224×224 resolution - Kernel size: 3×3 - Output: 4 channels

Calculation:

For traditional convolution, each output channel requires a full 3D filter that processes all input channels: - Each filter has shape: (kernel_height × kernel_width × input_channels) - Each filter has: 3 × 3 × 3 = 27 weights - We need 4 such filters (one per output channel) - Total weights: 27 × 4 = 108

Formula:

$$
\begin{aligned}
&\text{Parameters} = (\text{output\_channels}) \times (\text{input\_channels}) \times (\text{kernel\_height}) \times (\text{kernel\_width}) \\
&\qquad = 4 \times 3 \times 3 \times 3
\end{aligned}
$$

```
= 108
```

Note: If we included biases, we would add 4 more parameters (one per output channel) for a total of 112 parameters. However, the problem states to ignore biases.

Answer: 108 parameters

Part (b): Depthwise separable convolution parameters

Architecture: Depthwise separable convolution consists of two operations:

1. **Depthwise convolution**: Apply a separate spatial filter to each input channel
2. **Pointwise convolution**: Use 1×1 convolutions to combine channels

Given: - Input: 3 channels, 224×224 - Depthwise: 3×3 spatial filters - Pointwise: 1×1 convolutions - Output: 4 channels

Depthwise Convolution: - Each input channel gets its own 3×3 filter - No cross-talk between channels - Each filter has 3×3 = 9 weights - For 3 input channels: 3 × 9 = 27 weights - Output: still 3 channels (same as input)

Pointwise Convolution: - Takes the 3-channel depthwise output - Uses 1×1 filters to combine channels - Each output channel needs weights to combine all 3 input channels: 3 × 1 × 1 = 3 weights - For 4 output channels: 4 × 3 = 12 weights

Total Parameters:

```
Depthwise:  3 × 3 × 3 = 27
Pointwise:  4 × 3 × 1 × 1 = 12
Total:      27 + 12 = 39 parameters
```

Answer: 39 parameters

Comparison: - Traditional convolution: 108 parameters - Depthwise separable: 39 parameters - Reduction: 108/39 ≈ 2.77x fewer parameters

This dramatic parameter reduction is why depthwise separable convolutions are popular in efficient architectures like MobileNet and EfficientNet. They can be viewed as a low-rank approximation to traditional convolution, factoring the spatial and channel-wise operations.

Note on biases: If we included biases: - Depthwise would add 3 biases (one per channel) - Pointwise would add 4 biases (one per output channel) - However, since everything is linear, we only need biases on the final output: 4 biases - Total with biases: 39 + 4 = 43 parameters

Alternatively, if we included biases at both stages: 39 + 3 + 4 = 46 parameters.

Problem 4: Regularization and Dropout

Part (a): Manipulating the expectation to eliminate randomness

Given: - Dropout objective: $L(\hat{w}) = E_R[||y - (R \odot X)\hat{w}||^2_2]$ - R is a random binary matrix with $R_{ij} \sim$ i.i.d. Bernoulli(p) - $\odot$ denotes element-wise product

Goal: Show that

$$L(\hat{w}) = ||y - pX\hat{w}||^2_2 + p(1-p)||\Gamma\hat{w}||^2_2$$

where Γ̂ is a diagonal matrix with j-th diagonal entry equal to the norm of the j-th column of X.

Solution:

Let P = R ⊙ X. Then:

$$||y - P\hat{w}||_2^2 = y^T y - 2\hat{w}^T P^T y + \hat{w}^T P^T P \hat{w}$$

Taking expectations:

$$L(\hat{w}) = E[y^T y - 2\hat{w}^T P^T y + \hat{w}^T P^T P \hat{w}]$$
$$= y^T y - 2\hat{w}^T E[P^T]y + \hat{w}^T E[P^T P]\hat{w}$$

Step 1: Calculate E[P]

Since P = R ⊙ X:

$$E[P_{ij}] = E[R_{ij} \cdot X_{ij}]$$
$$= E[R_{ij}] \cdot X_{ij} \quad (R_{ij} \text{ and } X_{ij} \text{ are the same entry, not independent random variables})$$
$$= p \cdot X_{ij}$$

Therefore: E[P] = pX, and E[P^T] = pX^T

This gives us:

$$E[2\hat{w}^T P^T y] = 2p\hat{w}^T X^T y$$

Step 2: Calculate E[P^T P]

The (i,j)-th entry of P^T P is:

$$(P^T P)_{ij} = \Sigma_k P_{ki} P_{kj}$$
$$= \Sigma_k (R_{ki} X_{ki})(R_{kj} X_{kj})$$
$$= \Sigma_k R_{ki} R_{kj} X_{ki} X_{kj}$$

Taking expectations:

$$E[(P^T P)_{ij}] = \Sigma_k E[R_{ki} R_{kj}] X_{ki} X_{kj}$$

We need to consider two cases:

Case 1: i ≠ j Since R_{ki} and R_{kj} are independent Bernoulli(p):

$$E[R_{ki} R_{kj}] = E[R_{ki}]E[R_{kj}] = p \cdot p = p^2$$

Therefore:

$$E[(P^T P)_{ij}] = p^2 \Sigma_k X_{ki} X_{kj} = p^2(X^T X)_{ij}$$

Case 2: i = j Now R_{ki} and R_{kj} are the same random variable:

$$E[R_{ki} R_{ki}] = E[R_{ki}^2]$$

For Bernoulli(p): $R_{ki}^2 = R_{ki}$ (since it's 0 or 1), so:

$$E[R_{ki}^2] = E[R_{ki}] = p$$

Therefore:

$$E[(P^T P)_{ii}] = p \Sigma_k X_{ki}^2 = p(X^T X)_{ii}$$

Step 3: Separate the diagonal

We can write:

$$E[P^T P] = p^2(X^T X) + (p - p^2)\text{diag}(X^T X)$$

where diag(M) denotes the matrix with the same diagonal as M but zeros elsewhere.

Step 4: Combine everything

$$L(\hat{w}) = y^T y - 2p\hat{w}^T X^T y + \hat{w}^T[p^2(X^T X) + (p - p^2)\text{diag}(X^T X)]\hat{w}$$

$$= y^T y - 2p\hat{w}^T X^T y + p^2\hat{w}^T X^T X \hat{w} + (p - p^2)\hat{w}^T \text{diag}(X^T X)\hat{w}$$

Recognizing the first three terms as a squared norm:

$$y^T y - 2p\hat{w}^T X^T y + p^2\hat{w}^T X^T X \hat{w} = ||y - pX\hat{w}||_2^2$$

For the last term, note that $\hat{w}^T \text{diag}(X^T X)\hat{w} = ||\Gamma\hat{w}||_2^2$ where Γ is the diagonal matrix with:

$$\Gamma_{jj} = \sqrt{[(X^T X)_{jj}]} = \sqrt{[\Sigma_i X_{ij}^2]} = ||X_j||_2$$

(the norm of the j-th column of X).

Final Result:

$$L(\hat{w}) = ||y - pX\hat{w}||_2^2 + p(1-p)||\Gamma\hat{w}||_2^2$$

This shows that dropout is equivalent to adding a special form of regularization where the regularization strength on each weight is proportional to the norm of its corresponding feature in the training data. ✓

Part (b): Transform $\hat{w}$ to standard form

Goal: Find a transformation of $\hat{w}$ that makes the dropout objective look like a standard Tikhonov regularized problem.

From part (a):

$$L(\hat{w}) = ||y - pX\hat{w}||_2^2 + p(1-p)||\Gamma\hat{w}||_2^2$$

Standard Tikhonov form:

$$L(w) = ||y - Xw||_2^2 + ||\Gamma w||_2^2$$

Solution:

Let $w = p\hat{w}$, which means $\hat{w} = w/p$.

Substituting into the dropout objective:

$$L(w) = ||y - pX(w/p)||_2^2 + p(1-p)||\Gamma(w/p)||_2^2$$

$$= ||y - Xw||_2^2 + p(1-p)/p^2 \cdot ||\Gamma\hat{w}||_2^2$$

$$= ||y - Xw||_2^2 + (1-p)/p \cdot ||\Gamma\hat{w}||_2^2$$

Factoring out (1-p)/p:

$$L(w) = ||y - Xw||_2^2 + ||\sqrt{[(1-p)/p]} \cdot \Gamma\hat{w}||_2^2$$

This is in standard Tikhonov form with:

$$\Gamma = \sqrt{[(1-p)/p]} \cdot \Gamma\hat{}$$

Answer: $w = p\hat{w}$

Interpretation: This transformation is exactly what's done at inference time in dropout! During training with dropout probability p, we randomly keep each neuron with probability p. At inference, we scale the weights by p to compensate for the fact that all neurons are now active. This mathematical analysis shows that this scaling is not just a heuristic - it's the correct transformation to relate the dropout objective to standard regularized regression.

For example, if p = 0.5 (keeping each feature with 50% probability): - During training, we learn $\hat{w}$ with dropout - At inference, we use $w = 0.5\hat{w}$ - The regularization strength is $\Gamma = \sqrt{[(1-0.5)/0.5]} \cdot \Gamma\hat{} = \Gamma\hat{}$

Part (c): Change variables for ridge regression

Given:

$$L(w) = ||y - Xw||_2^2 + ||\Gamma w||_2^2$$

where Γ is invertible.

Goal: Change variables to get classical ridge regression form:

$$L(\tilde{w}) = ||y - \tilde{X}\tilde{w}||_2^2 + \lambda||\tilde{w}||_2^2$$

Solution:

Let $\tilde{w} = \Gamma w$, which means $w = \Gamma^{-1}\tilde{w}$.

Substituting into the objective:

$$L(\tilde{w}) = ||y - X(\Gamma^{-1}\tilde{w})||_2^2 + ||\Gamma(\Gamma^{-1}\tilde{w})||_2^2$$

$$= ||y - X\Gamma^{-1}\tilde{w}||_2^2 + ||\tilde{w}||_2^2$$

This is exactly ridge regression form with $\lambda = 1$ and:

$$\tilde{X} = X\Gamma^{-1}$$

Answer: $\tilde{X} = X\Gamma^{-1}$

Alternative with rescaling: If we want an arbitrary λ, we can include a constant factor:

$$\tilde{X} = (1/\sqrt{\lambda})X\Gamma^{-1}$$

Then:

$$L(\tilde{w}) = ||y - (1/\sqrt{\lambda})X\Gamma^{-1}\tilde{w}||_2^2 + ||\tilde{w}||_2^2$$
$$= (1/\lambda)||y - X\Gamma^{-1}\tilde{w}||_2^2 + ||\tilde{w}||_2^2$$

Multiplying through by λ:

$$\lambda L(\tilde{w}) = ||y - X\Gamma^{-1}\tilde{w}||_2^2 + \lambda||\tilde{w}||_2^2$$

So any positive rescaling of $\tilde{X} = X\Gamma^{-1}$ is acceptable.

Interpretation: This variable change "pre-conditions" the features by Γ^{-1}. Features that had large regularization penalties (large entries in Γ) are scaled down, while features with small penalties are scaled up. This makes the regularization uniform across all transformed features.

Part (d): Column norms of $\tilde{X}$ and relation to batch normalization

Given: - Γ is diagonal and invertible - The j-th diagonal entry of Γ is proportional to $||f_j||_2$ (norm of j-th column of X)

From part (c): $\tilde{X} = X\Gamma^{-1}$

Analysis:

Let f_j denote the j-th column of X. Since Γ is diagonal:

$$\Gamma = \text{diag}(\gamma_1, \gamma_2, ..., \gamma_d)$$

where $\gamma_j \propto ||f_j||_2$.

The j-th column of $\tilde{X}$ is:

$$\tilde{f}_j = (X\Gamma^{-1})_j = f_j \cdot (1/\gamma_j) = f_j / \gamma_j$$

Since γ_j is proportional to $||f_j||_2$, we can write $\gamma_j = c||f_j||_2$ for some constant $c > 0$.

Therefore:

$$\tilde{f}_j = f_j / (c||f_j||_2)$$

The norm of this transformed column is:

$$||\tilde{f}_j||_2 = ||f_j / (c||f_j||_2)||_2$$
$$= (1/(c||f_j||_2)) \cdot ||f_j||_2$$
$$= 1/c$$

Conclusion: All columns of $\tilde{X}$ have the same norm: $||\tilde{f}_j||_2 = 1/c$ for all j.

Answer: All columns of $\tilde{X}$ have constant norm (specifically, 1/c where c is the proportionality constant).

Relationship to Batch Normalization:

This is very similar to what batch normalization does:

1. **Feature Normalization:**
   - Dropout's Γ transformation: scales features inversely to their magnitude
   - Batch norm: standardizes each feature to have zero mean and unit variance
2. **Equalized Feature Scales:**
   - Dropout with Γ^{-1}: makes all features have the same ℓ^2 norm
   - Batch norm: makes all features have the same variance (and mean)
3. **Adaptive Rescaling:**
   - Dropout: the regularization penalty $||\Gamma w||^2$ adapts to feature magnitudes
   - Batch norm: the scale γ and shift β parameters adapt after normalization
4. **Training Stability:**
   - Both techniques prevent features with large magnitudes from dominating
   - Both help optimization by working with normalized/scaled features
   - Both reduce the dependence of gradient magnitudes on feature scales

Key Insight: Dropout acts as an implicit feature normalization scheme. By regularizing features proportionally to their training data norms, dropout encourages the model to give equal "attention" to all features, similar to how batch normalization explicitly standardizes features. This helps explain why both techniques improve generalization and training stability.

In fact, both dropout and batch normalization: - Reduce internal covariate shift - Make the loss landscape smoother - Help prevent certain features from dominating - Act as forms of regularization

The mathematical connection revealed here shows that dropout's regularization effect has a geometric interpretation: it "spherizes" the effective feature space.

Summary

This completes the detailed solutions for Problems 1, 2, 3, and 4:

1. **Convolutional Networks**: Explained advantages of convolutions, solved for filter coefficients, and computed transpose convolution output with detailed matrix calculations.

2. **Batch Normalization**: Identified normalization types, derived gradients through batch norm showing how batch size affects gradient coupling, with special cases for $n=1$ and $n\to\infty$.

3. **Depthwise Separable Convolutions**: Calculated parameters for both traditional and depthwise separable convolutions, showing the dramatic parameter reduction (108 → 39 parameters).

4. **Regularization and Dropout**:
   - Showed dropout is equivalent to Tikhonov regularization with data-dependent penalties
   - Derived the weight transformation $w = p\hat{w}$ for inference
   - Changed variables to get ridge regression form
   - Revealed dropout's connection to feature normalization and batch normalization

All solutions include detailed mathematical derivations, step-by-step calculations, and interpretations of the results.