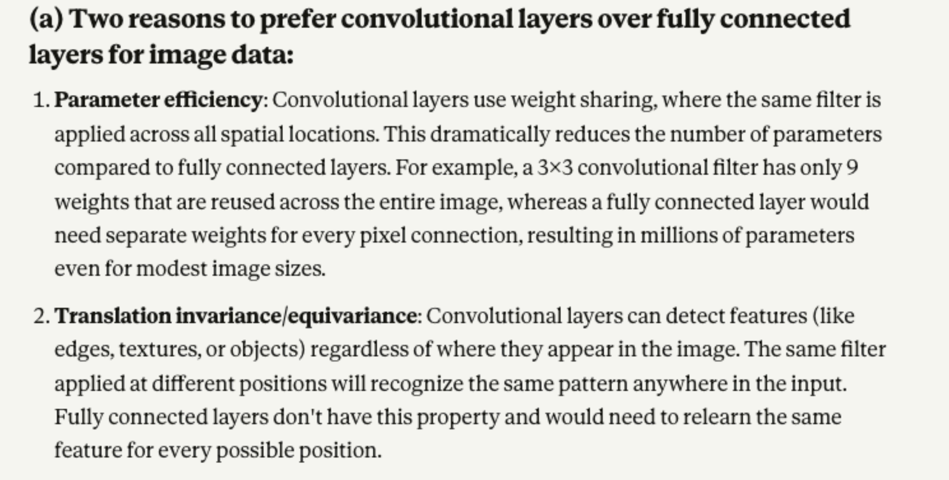
In this document, I will guide Claude AI to walk through the written problems for HW5, showing its misconceptions and hallucinations.

Question 1:

Part(a)

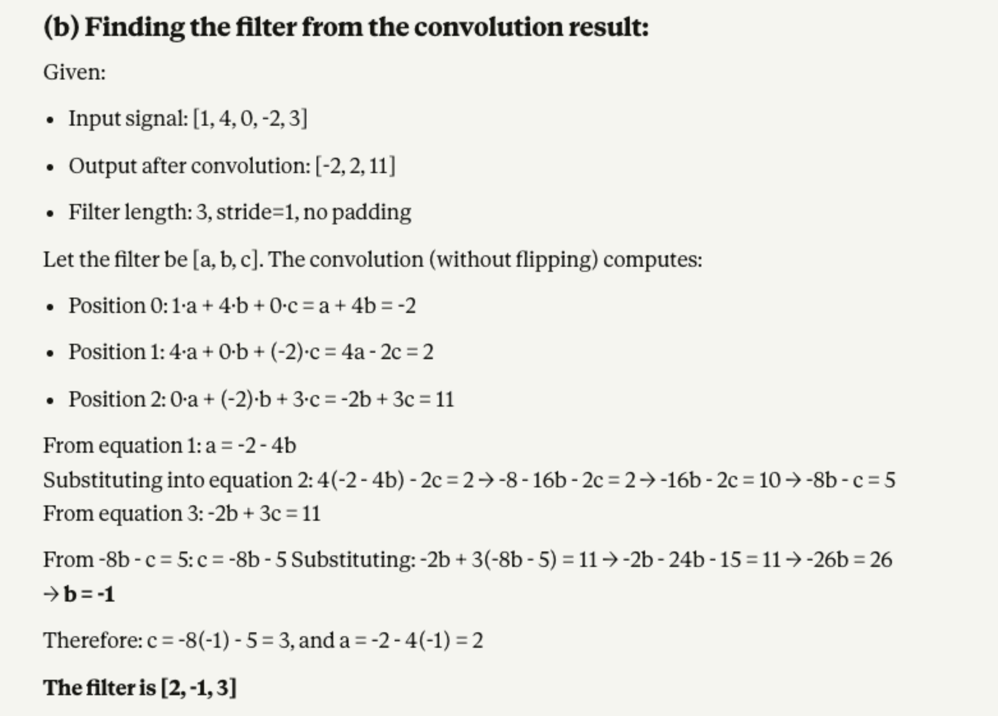


(a) Two reasons to prefer convolutional layers over fully connected layers for image data:

1. **Parameter efficiency**: Convolutional layers use weight sharing, where the same filter is applied across all spatial locations. This dramatically reduces the number of parameters compared to fully connected layers. For example, a 3×3 convolutional filter has only 9 weights that are reused across the entire image, whereas a fully connected layer would need separate weights for every pixel connection, resulting in millions of parameters even for modest image sizes.
2. **Translation invariance/equivariance**: Convolutional layers can detect features (like edges, textures, or objects) regardless of where they appear in the image. The same filter applied at different positions will recognize the same pattern anywhere in the input. Fully connected layers don't have this property and would need to relearn the same feature for every possible position.

I would say it does a really good job here. The arguments it made are sensible.

Part(b):



(b) Finding the filter from the convolution result:

Given:

- Input signal: [1, 4, 0, -2, 3]
- Output after convolution: [-2, 2, 11]
- Filter length: 3, stride=1, no padding

Let the filter be [a, b, c]. The convolution (without flipping) computes:

- Position 0: $1 \cdot a + 4 \cdot b + 0 \cdot c = a + 4b = -2$
- Position 1: $4 \cdot a + 0 \cdot b + (-2) \cdot c = 4a - 2c = 2$
- Position 2: $0 \cdot a + (-2) \cdot b + 3 \cdot c = -2b + 3c = 11$

From equation 1: $a = -2 - 4b$
Substituting into equation 2: $4(-2 - 4b) - 2c = 2 \rightarrow -8 - 16b - 2c = 2 \rightarrow -16b - 2c = 10 \rightarrow -8b - c = 5$
From equation 3: $-2b + 3c = 11$

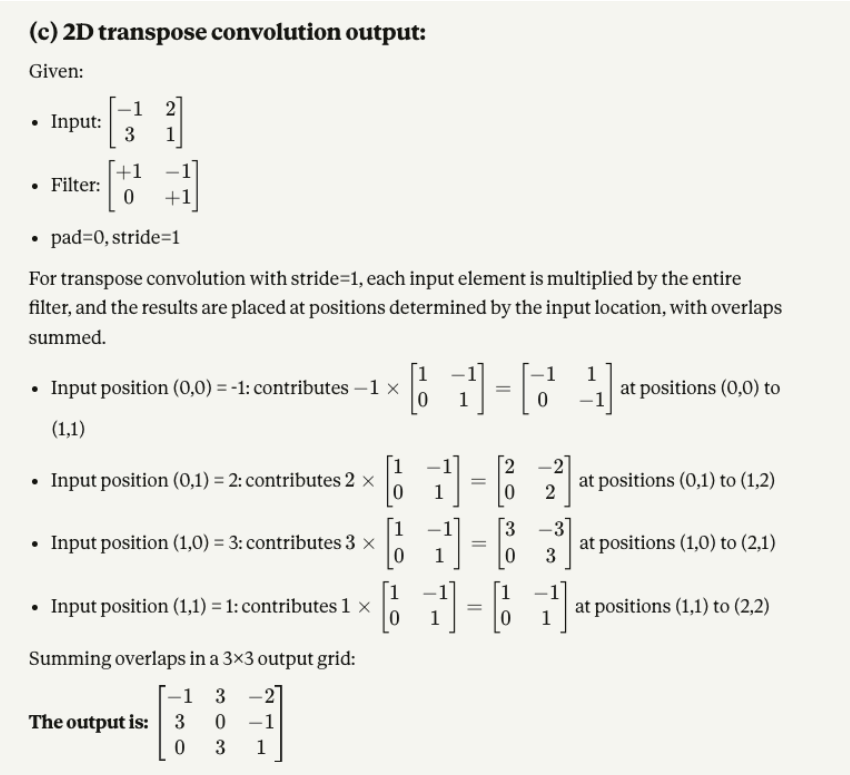
From $-8b - c = 5$: $c = -8b - 5$ Substituting: $-2b + 3(-8b - 5) = 11 \rightarrow -2b - 24b - 15 = 11 \rightarrow -26b = 26$ $\rightarrow$ **$b = -1$**

Therefore: $c = -8(-1) - 5 = 3$, and $a = -2 - 4(-1) = 2$

The filter is [2, -1, 3]

It gives the right solution with the correct procedure.

Part(c):



(c) 2D transpose convolution output:

Given:

- Input: $\begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}$
- Filter: $\begin{bmatrix} +1 & -1 \\ 0 & +1 \end{bmatrix}$
- pad=0, stride=1

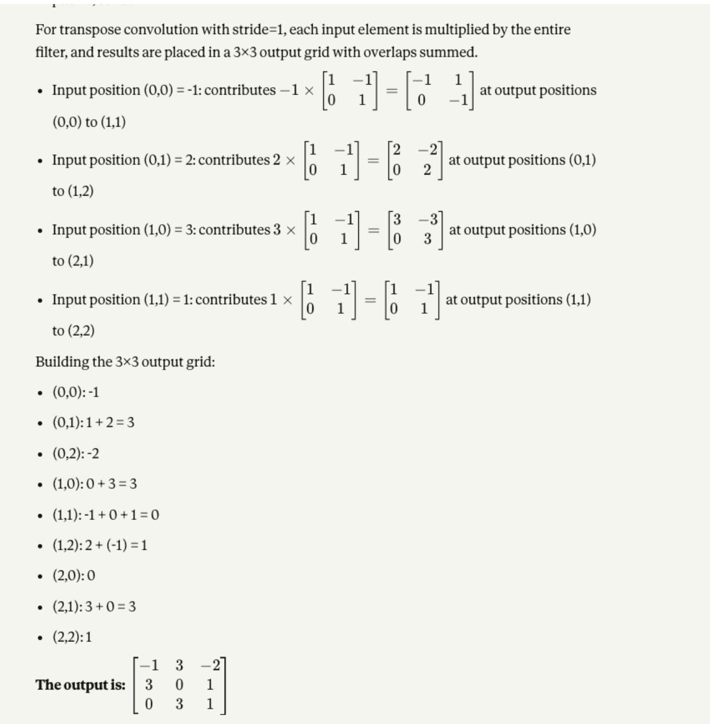
For transpose convolution with stride=1, each input element is multiplied by the entire filter, and the results are placed at positions determined by the input location, with overlaps summed.

- Input position (0,0) = -1: contributes $-1 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix}$ at positions (0,0) to (1,1)
- Input position (0,1) = 2: contributes $2 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 0 & 2 \end{bmatrix}$ at positions (0,1) to (1,2)
- Input position (1,0) = 3: contributes $3 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & -3 \\ 0 & 3 \end{bmatrix}$ at positions (1,0) to (2,1)
- Input position (1,1) = 1: contributes $1 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$ at positions (1,1) to (2,2)

Summing overlaps in a 3×3 output grid:

The output is: $\begin{bmatrix} -1 & 3 & -2 \\ 3 & 0 & -1 \\ 0 & 3 & 1 \end{bmatrix}$

It correctly understands the nature of transpose convolution output and generalizes the idea from the 1-dimensional example to the 2-dimensional case. However, it made a minor calculation error. The center entry of the square matrix should be -3 instead of 0.



For transpose convolution with stride=1, each input element is multiplied by the entire filter, and results are placed in a 3×3 output grid with overlaps summed.

- Input position (0,0) = -1: contributes $-1 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix}$ at output positions (0,0) to (1,1)
- Input position (0,1) = 2: contributes $2 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 0 & 2 \end{bmatrix}$ at output positions (0,1) to (1,2)
- Input position (1,0) = 3: contributes $3 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & -3 \\ 0 & 3 \end{bmatrix}$ at output positions (1,0) to (2,1)
- Input position (1,1) = 1: contributes $1 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$ at output positions (1,1) to (2,2)

Building the 3×3 output grid:

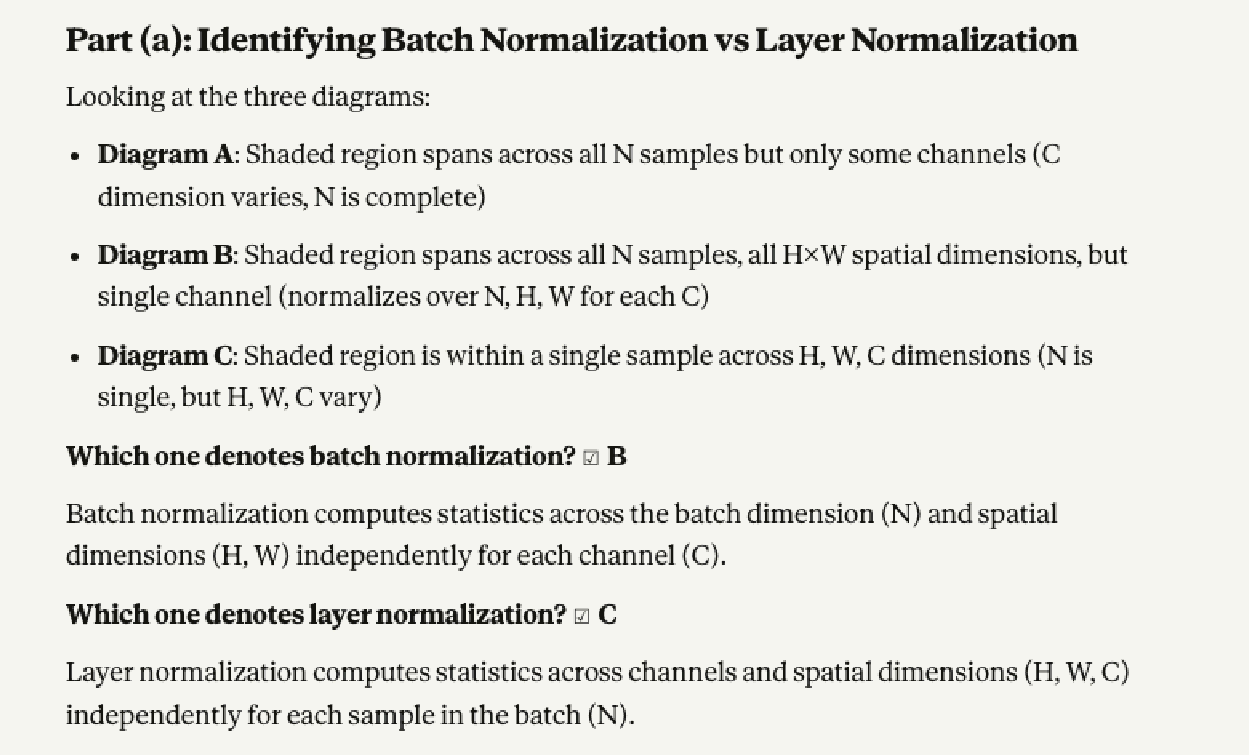
- (0,0): -1
- (0,1): 1 + 2 = 3
- (0,2): -2
- (1,0): 0 + 3 = 3
- (1,1): -1 + 0 + 1 = 0
- (1,2): 2 + (-1) = 1
- (2,0): 0
- (2,1): 3 + 0 = 3
- (2,2): 1

The output is: $\begin{bmatrix} -1 & 3 & -2 \\ 3 & 0 & 1 \\ 0 & 3 & 1 \end{bmatrix}$

Then I asked it to recalculate part c by telling it that there is a mistake. Unfortunately, it still gives the wrong solution. Obviously, once it makes a small mistake, it will remember it and get stuck on previous misconceptions even with explicit prompts telling it about the mistake.

Question 2

Part (a):
Its response is very interesting. Though it demonstrates the right conceptual understanding about batch normalization and layer normalization, it fails to apply such understanding to recognize the right visual representation from figures.



Part (a): Identifying Batch Normalization vs Layer Normalization

Looking at the three diagrams:

- **Diagram A**: Shaded region spans across all N samples but only some channels (C dimension varies, N is complete)
- **Diagram B**: Shaded region spans across all N samples, all H×W spatial dimensions, but single channel (normalizes over N, H, W for each C)
- **Diagram C**: Shaded region is within a single sample across H, W, C dimensions (N is single, but H, W, C vary)

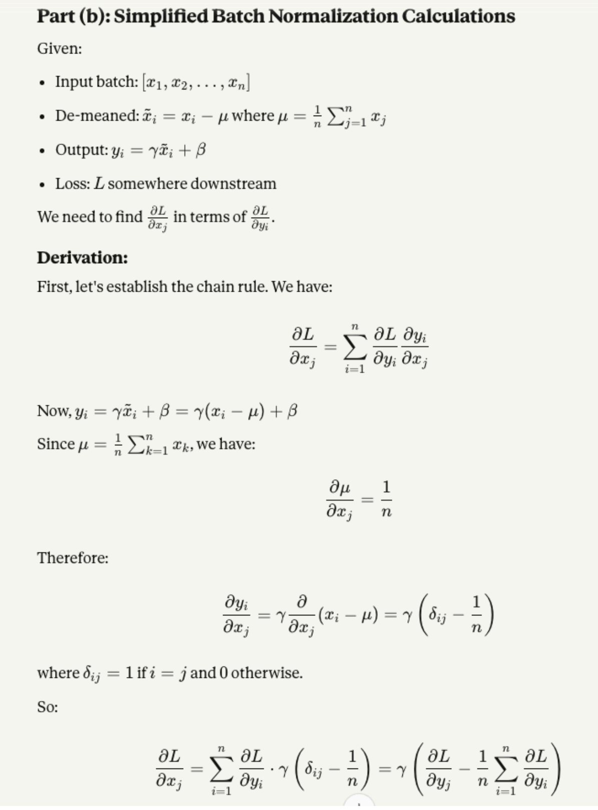
Which one denotes batch normalization? ☑ B

Batch normalization computes statistics across the batch dimension (N) and spatial dimensions (H, W) independently for each channel (C).

Which one denotes layer normalization? ☑ C

Layer normalization computes statistics across channels and spatial dimensions (H, W, C) independently for each sample in the batch (N).

Part (b):
It gets the right algebraic representation using chain rule. For the sanity check, it calculates the right solution for n = 1 case with the right reasoning. However, for the infinity case, it fails to recognize the most obvious reasoning that could be made to argue that the summation part will go to 0. Instead, it tries to make some more meaningless complicated arguments that interpret the summation part as expectation.



Part (b): Simplified Batch Normalization Calculations

Given:

- Input batch: $[x_1, x_2, \dots, x_n]$
- De-meaned: $\tilde{x}_i = x_i - \mu$ where $\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$
- Output: $y_i = \gamma \tilde{x}_i + \beta$
- Loss: L somewhere downstream

We need to find $\frac{\partial L}{\partial x_j}$ in terms of $\frac{\partial L}{\partial y_i}$.

Derivation:

First, let's establish the chain rule. We have:

$$\frac{\partial L}{\partial x_j} = \sum_{i=1}^{n} \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial x_j}$$

Now, $y_i = \gamma \tilde{x}_i + \beta = \gamma(x_i - \mu) + \beta$

Since $\mu = \frac{1}{n}\sum_{k=1}^{n} x_k$, we have:

$$\frac{\partial \mu}{\partial x_j} = \frac{1}{n}$$

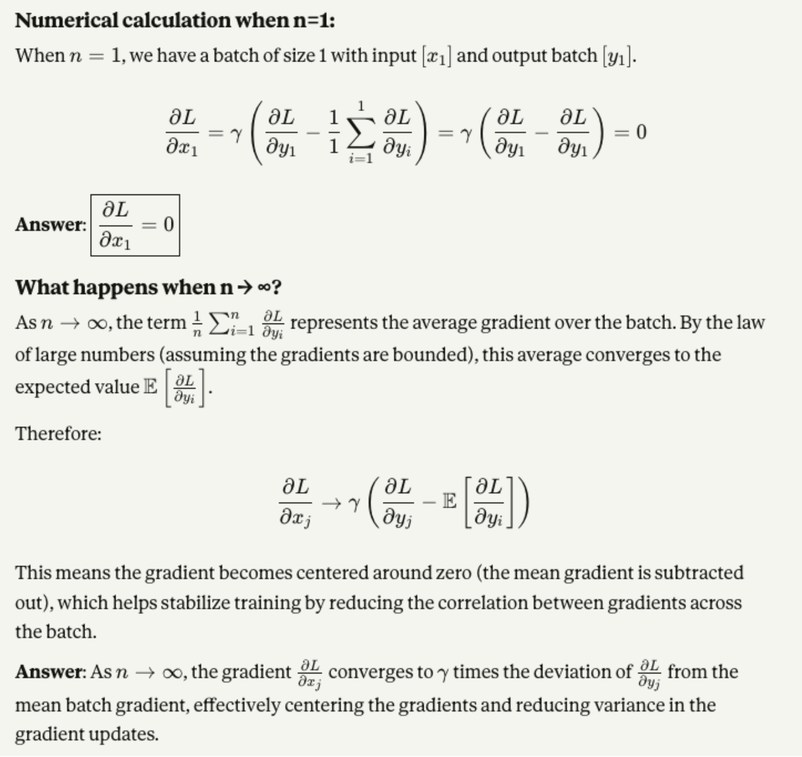
Therefore:

$$\frac{\partial y_i}{\partial x_j} = \gamma \frac{\partial}{\partial x_j}(x_i - \mu) = \gamma\left(\delta_{ij} - \frac{1}{n}\right)$$

where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise.

So:

$$\frac{\partial L}{\partial x_j} = \sum_{i=1}^{n} \frac{\partial L}{\partial y_i} \cdot \gamma\left(\delta_{ij} - \frac{1}{n}\right) = \gamma\left(\frac{\partial L}{\partial y_j} - \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L}{\partial y_i}\right)$$



Numerical calculation when n=1:

When $n = 1$, we have a batch of size 1 with input $[x_1]$ and output batch $[y_1]$.

$$\frac{\partial L}{\partial x_1} = \gamma\left(\frac{\partial L}{\partial y_1} - \frac{1}{1}\sum_{i=1}^{1} \frac{\partial L}{\partial y_i}\right) = \gamma\left(\frac{\partial L}{\partial y_1} - \frac{\partial L}{\partial y_1}\right) = 0$$

Answer: $\boxed{\frac{\partial L}{\partial x_1} = 0}$

What happens when n → ∞?

As $n \to \infty$, the term $\frac{1}{n}\sum_{i=1}^{n} \frac{\partial L}{\partial y_i}$ represents the average gradient over the batch. By the law of large numbers (assuming the gradients are bounded), this average converges to the expected value $\mathbb{E}\left[\frac{\partial L}{\partial y_i}\right]$.

Therefore:

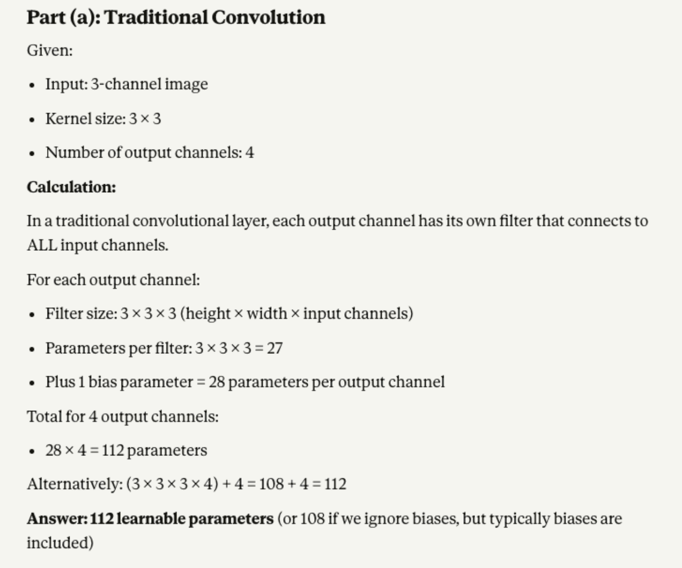
$$\frac{\partial L}{\partial x_j} \to \gamma\left(\frac{\partial L}{\partial y_j} - \mathbb{E}\left[\frac{\partial L}{\partial y_i}\right]\right)$$

This means the gradient becomes centered around zero (the mean gradient is subtracted out), which helps stabilize training by reducing the correlation between gradients across the batch.

Answer: As $n \to \infty$, the gradient $\frac{\partial L}{\partial x_j}$ converges to γ times the deviation of $\frac{\partial L}{\partial y_j}$ from the mean batch gradient, effectively centering the gradients and reducing variance in the gradient updates.

Question 3

It correctly reasons about both part (a) and part (b).



Part (a): Traditional Convolution

Given:

- Input: 3-channel image
- Kernel size: 3 × 3
- Number of output channels: 4

Calculation:

In a traditional convolutional layer, each output channel has its own filter that connects to ALL input channels.

For each output channel:

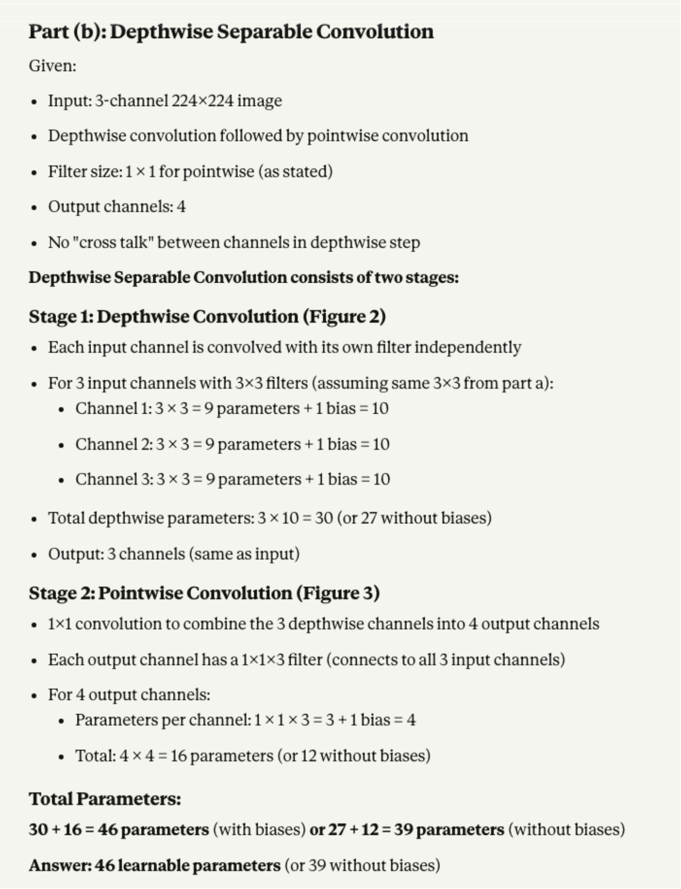
- Filter size: 3 × 3 × 3 (height × width × input channels)
- Parameters per filter: 3 × 3 × 3 = 27
- Plus 1 bias parameter = 28 parameters per output channel

Total for 4 output channels:

- 28 × 4 = 112 parameters

Alternatively: (3 × 3 × 3 × 4) + 4 = 108 + 4 = 112

Answer: 112 learnable parameters (or 108 if we ignore biases, but typically biases are included)



Part (b): Depthwise Separable Convolution

Given:

- Input: 3-channel 224×224 image
- Depthwise convolution followed by pointwise convolution
- Filter size: 1 × 1 for pointwise (as stated)
- Output channels: 4
- No "cross talk" between channels in depthwise step

Depthwise Separable Convolution consists of two stages:

Stage 1: Depthwise Convolution (Figure 2)

- Each input channel is convolved with its own filter independently
- For 3 input channels with 3×3 filters (assuming same 3×3 from part a):
  - Channel 1: 3 × 3 = 9 parameters + 1 bias = 10
  - Channel 2: 3 × 3 = 9 parameters + 1 bias = 10
  - Channel 3: 3 × 3 = 9 parameters + 1 bias = 10
- Total depthwise parameters: 3 × 10 = 30 (or 27 without biases)
- Output: 3 channels (same as input)

Stage 2: Pointwise Convolution (Figure 3)

- 1×1 convolution to combine the 3 depthwise channels into 4 output channels
- Each output channel has a 1×1×3 filter (connects to all 3 input channels)
- For 4 output channels:
  - Parameters per channel: 1 × 1 × 3 = 3 + 1 bias = 4
  - Total: 4 × 4 = 16 parameters (or 12 without biases)

Total Parameters:

30 + 16 = 46 parameters (with biases) **or 27 + 12 = 39 parameters** (without biases)

Answer: 46 learnable parameters (or 39 without biases)

Question 4

Part (a): It almost arrives at a perfect solution except step 6. From step 1 to 5, it puts the right argument regarding algebraic expansions and matrix expectation calculations. At step 6, it omits the most obvious quadratic pattern and does something meaningless and more complicated.

Derivation: From Dropout Objective to Tikhonov-Regularized Form

Starting point (equation 2):

$$\mathcal{L}(\tilde{\mathbf{w}}) = \mathbb{E}_{R\sim Bernoulli(p)}\left[||\mathbf{y} - (R \odot X)\tilde{\mathbf{w}}||_2^2\right]$$

Step 1: Expand the squared norm

$$\mathbb{E}_R\left[||\mathbf{y} - (R \odot X)\tilde{\mathbf{w}}||_2^2\right] = \mathbb{E}_R\left[(\mathbf{y} - (R \odot X)\tilde{\mathbf{w}})^T(\mathbf{y} - (R \odot X)\tilde{\mathbf{w}})\right]$$

Expanding:

$$= \mathbb{E}_R\left[\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T(R \odot X)\tilde{\mathbf{w}} + \tilde{\mathbf{w}}^T(R \odot X)^T(R \odot X)\tilde{\mathbf{w}}\right]$$

Step 2: Apply linearity of expectation

$$= \mathbf{y}^T\mathbf{y} - 2\mathbb{E}_R[\mathbf{y}^T(R \odot X)]\tilde{\mathbf{w}} + \tilde{\mathbf{w}}^T\mathbb{E}_R[(R \odot X)^T(R \odot X)]\tilde{\mathbf{w}}$$

Step 3: Compute $\mathbb{E}_R[\mathbf{y}^T(R \odot X)]$

Since R is element-wise independent with $R_{i,j} \sim Bernoulli(p)$:

- $\mathbb{E}[R_{i,j}] = p$

For each column j of X, the j-th column of $(R \odot X)$ has entries multiplied by independent $R_{i,j}$ values.

$$\mathbb{E}_R[R \odot X] = p \cdot X$$

Therefore:

$$\mathbb{E}_R[\mathbf{y}^T(R \odot X)] = \mathbf{y}^T(pX) = p\mathbf{y}^TX$$

Step 4: Compute $\mathbb{E}_R[(R \odot X)^T(R \odot X)]$

Let's denote $X = [x_1, x_2, \ldots, x_d]$ where x_j is the j-th column.

The (j,k) entry of $(R \odot X)^T(R \odot X)$ is:

$$(R \odot X)^T(R \odot X)]_{j,k} = \sum_{i=1}^{n} R_{i,j}R_{i,k}X_{i,j}X_{i,k}$$

Taking expectation:

- If $j \neq k$: $\mathbb{E}[R_{i,j}R_{i,k}] = \mathbb{E}[R_{i,j}]\mathbb{E}[R_{i,k}] = p^2$ (independence)
- If $j = k$: $\mathbb{E}[R_{i,j}^2] = \mathbb{E}[R_{i,j}] = p$ (since $R_{i,j} \in \{0,1\}$)

Therefore: $$\mathbb{E}*R[(R \odot X)^T(R \odot X)]*{j,k} = \begin{cases} p \sum_{i=1}^n X_{i,j}^2 = p \cdot ||x_j||*2^2 & \text{if } j=k \ p^2 \sum*{i=1}^n X_{i,j}X_{i,k} = p^2 (X^TX)_{j,k} & \text{if } j \neq k \end{cases}$$

This can be written as:

$$\mathbb{E}_R[(R \odot X)^T(R \odot X)] = p^2X^TX + p(1-p)\tilde{\Gamma}$$

where $\tilde{\Gamma}$ is a diagonal matrix with $\tilde{\Gamma}_{j,j} = ||x_j||_2^2$ (the norm of the j-th column of X).

Step 5: Substitute back

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

Step 6: Reorganize

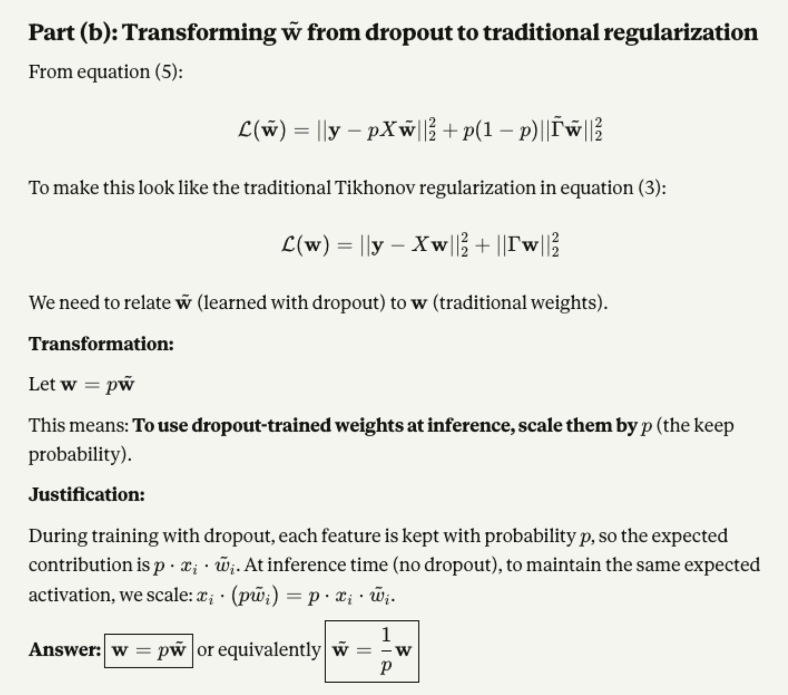
$$= p^2||\mathbf{y}||_2^2 + (1-p^2)||\mathbf{y}||_2^2 - 2p\mathbf{y}^TX\tilde{\mathbf{w}} + p^2\tilde{\mathbf{w}}^TX^TX\tilde{\mathbf{w}} + p(1-p)\tilde{\mathbf{w}}^T\tilde{\Gamma}\tilde{\mathbf{w}}$$

$$= p^2(||\mathbf{y}||_2^2 - 2\mathbf{y}^TX\tilde{\mathbf{w}} + \tilde{\mathbf{w}}^TX^TX\tilde{\mathbf{w}}) + \text{const} + p(1-p)||\tilde{\Gamma}\tilde{\mathbf{w}}||_2^2$$

$$= p^2||\mathbf{y} - X\tilde{\mathbf{w}}||_2^2 + p(1-p)||\tilde{\Gamma}\tilde{\mathbf{w}}||_2^2 + \text{const}$$

Part (b):

It makes a sensible argument about inference time scaling when dropouts are involved during training time. However, it fails to give a mathematical explanation about how the such scaling can be absorbed by w and tau.



Part (b): Transforming $\tilde{\mathbf{w}}$ from dropout to traditional regularization

From equation (5):

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

To make this look like the traditional Tikhonov regularization in equation (3):

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

We need to relate $\tilde{\mathbf{w}}$ (learned with dropout) to $\mathbf{w}$ (traditional weights).

Transformation:

Let $\mathbf{w} = p\tilde{\mathbf{w}}$

This means: **To use dropout-trained weights at inference, scale them by** p (the keep probability).

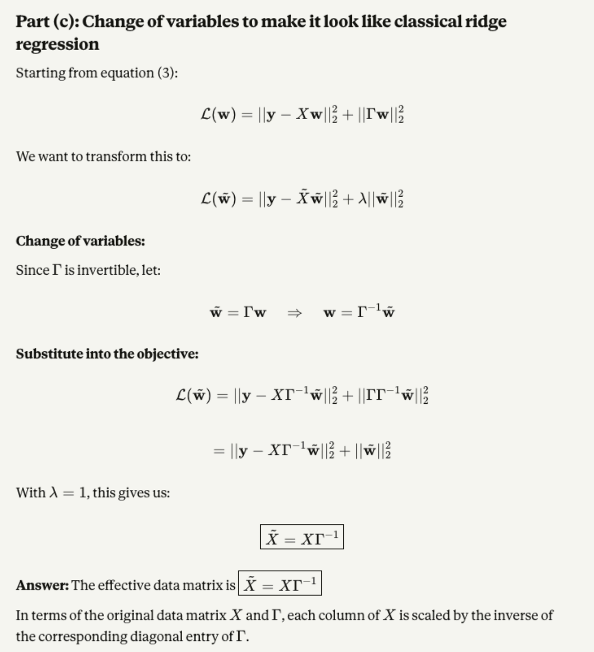
Justification:

During training with dropout, each feature is kept with probability p, so the expected contribution is $p \cdot x_i \cdot \tilde{w}_i$. At inference time (no dropout), to maintain the same expected activation, we scale: $x_i \cdot (p\tilde{w}_i) = p \cdot x_i \cdot \tilde{w}_i$.

Answer: $\boxed{\mathbf{w} = p\tilde{\mathbf{w}}}$ or equivalently $\boxed{\tilde{\mathbf{w}} = \frac{1}{p}\mathbf{w}}$

Part (c)

It is doing the right reparameterization.



Part (c): Change of variables to make it look like classical ridge regression

Starting from equation (3):

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

We want to transform this to:

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

Change of variables:

Since Γ is invertible, let:

$$\tilde{\mathbf{w}} = \Gamma\mathbf{w} \quad \Rightarrow \quad \mathbf{w} = \Gamma^{-1}\tilde{\mathbf{w}}$$

Substitute into the objective:

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

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

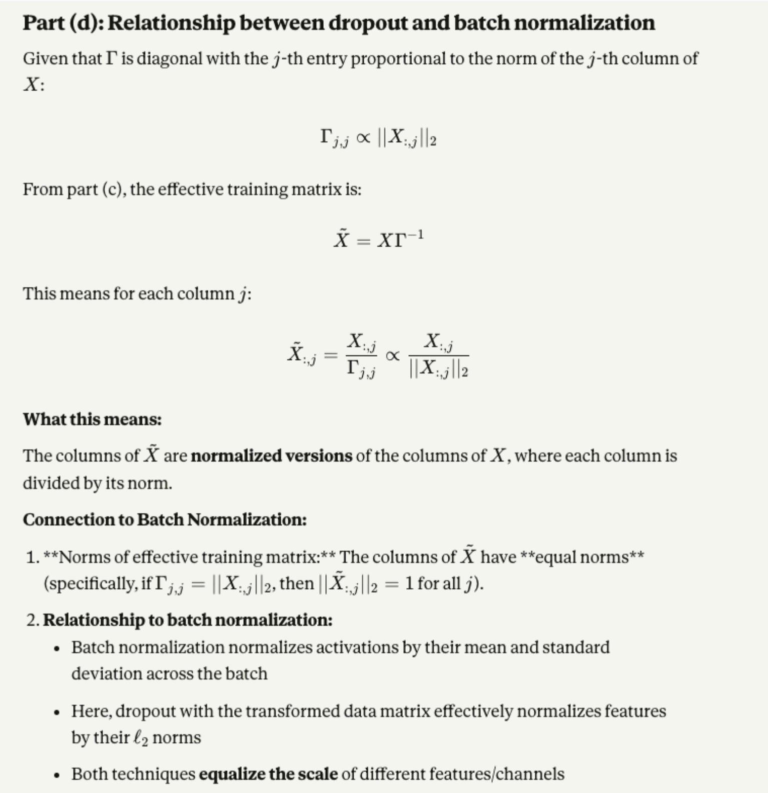
With $\lambda = 1$, this gives us:

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

Answer: The effective data matrix is $\boxed{\tilde{X} = X\Gamma^{-1}}$

In terms of the original data matrix X and Γ, each column of X is scaled by the inverse of the corresponding diagonal entry of Γ.

Part (d)
It makes exactly the right argument about the implicit impacts of dropouts as a kind of normalization.



Part (d): Relationship between dropout and batch normalization

Given that Γ is diagonal with the j-th entry proportional to the norm of the j-th column of X:

$$\Gamma_{j,j} \propto ||X_{:,j}||_2$$

From part (c), the effective training matrix is:

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

This means for each column j:

$$\tilde{X}_{:,j} = \frac{X_{:,j}}{\Gamma_{j,j}} \propto \frac{X_{:,j}}{||X_{:,j}||_2}$$

What this means:

The columns of $\tilde{X}$ are **normalized versions** of the columns of X, where each column is divided by its norm.

Connection to Batch Normalization:

1. **Norms of effective training matrix:** The columns of $\tilde{X}$ have **equal norms** (specifically, if $\Gamma_{j,j} = ||X_{:,j}||_2$, then $||\tilde{X}_{:,j}||_2 = 1$ for all j).
2. **Relationship to batch normalization:**
   - Batch normalization normalizes activations by their mean and standard deviation across the batch
   - Here, dropout with the transformed data matrix effectively normalizes features by their ℓ_2 norms
   - Both techniques **equalize the scale** of different features/channels

Summarization for using Claude AI as a homework helper:
I have to acknowledge that Claude AI is a very powerful tool that can help us walk through the homework. I rarely see any hallucinations or misconceptions. Most of the arguments it made are consistent with the right solution especially when the questions are fairly straightforward. However, when it is asked to derive something that requires many intermediate steps, it will sometimes fail to recognize the most obvious thing to do at some point. That is to say, for long-horizon tasks, it tends to make its solutions more complex even though such complex reasoning is not ideal.