# EECS 182 Homework 5 Report

## Question 1

Hi Deepseek I am working on a problem on Deep Learning with regards to CNN practice, will you be able to help me with that?

## Answer

Of course! I'd be happy to help you with your Convolutional Neural Network (CNN) problem.

Please go ahead and share the specific problem you're working on. You can:

- **Paste the problem statement**
- **Show me your code** and describe where you're stuck
- **Ask a conceptual question** about how CNNs work
- **Describe the error message** you're receiving

I can assist with a wide range of CNN topics, including:

- **Architecture Design**: Choosing layers, filter sizes, pooling, etc.
- **Implementation**: Coding in frameworks like TensorFlow/Keras or PyTorch.
- **Debugging**: Fixing errors, dealing with overfitting/underfitting, vanishing gradients.
- **Data Preparation**: Image preprocessing, augmentation, data loaders.
- **Theory**: How convolutions, pooling, and backpropagation work in CNNs.
- **Model Interpretation**: Understanding what your model has learned.

**Just paste your problem or code below, and we'll start working on it together.**

*Looking forward to helping you deepen your deep learning knowledge!*

## Question 2

We are now having the first problem, throughout this problem, we will use the convention of NOT flipping the filter before dragging it over the signal. This is the standard notation with neural networks (ie, we assume the filter given to us is already flipped)

1. List two reasons we typically prefer convolutional layers instead of fully connected layers when working with image data.
2. Consider the following 1D signal: [1,4,0,–2,3]. After convolution with a length-3 filter, no padding, stride=1, we get the following sequence: [–2,2,11]. What was the filter?
3. Transpose convolution is an operation to help us upsample a signal (increase the resolution). For example, if our original signal were [a,b,c] and we perform transpose convolution with pad=0 and stride=2, with the filter [x,y,z], the output would be [ax,ay,az+bx,by,bz+cx,cy,cz]. Notice that the entries of the input are multiplied by each of the entries of the filter. Overlaps are summed. Also notice how for a fixed filter size and stride, the dimensions of the input and output are swapped compared to standard convolution. (For example, if we did standard convolution on a length-7 sequence with filter size of 3 and stride=2, we would output a length-3 sequence). If our 2D input is [–1 2; 3 1] and the 2D filter is [+1–1; 0 +1]. What is the output of transpose convolution with pad=0 and stride=1?

# Answer

Of course! Let's work through each part of this problem step by step.

# 1. Two reasons we typically prefer convolutional layers over fully connected layers for image data:

**Reason 1: Parameter Efficiency & Spatial Locality**
Convolutional layers use weight sharing - the same filter is applied across all spatial positions of the input. This dramatically reduces the number of parameters compared to fully connected layers where each input pixel would connect to each neuron. For example, a 1000×1000 image would require 1 billion parameters in a single fully connected layer, while a convolutional layer might use only thousands.

**Reason 2: Translation Equivariance**
Convolutional layers naturally capture the property that features (like edges, corners, textures) are important regardless of their position in the image. A filter that detects vertical edges will detect them anywhere in the image, making CNNs inherently translation-equivariant.

# 2. Finding the filter:

Given:

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

Let the filter be [$w_1$, $w_2$, $w_3$]. The convolution operation (without flipping) gives:

Position 1: $1 \cdot w_1 + 4 \cdot w_2 + 0 \cdot w_3 = -2$
Position 2: $4 \cdot w_1 + 0 \cdot w_2 + (-2) \cdot w_3 = 2$
Position 3: $0 \cdot w_1 + (-2) \cdot w_2 + 3 \cdot w_3 = 11$

Solving this system:
From equation 1: $w_1 + 4w_2 = -2$
From equation 2: $4w_1 - 2w_3 = 2$
From equation 3: $-2w_2 + 3w_3 = 11$

Solving: $w_1 = 2$, $w_2 = -1$, $w_3 = 3$

**Filter: [2, -1, 3]**

# 3. Transpose convolution:

Given:

- Input: `[−1 2; 3 1]` (2×2 matrix)
- Filter: `[+1 −1; 0 +1]` (2×2 matrix)
- Pad=0, Stride=1

For transpose convolution with stride=1 and no padding on a 2×2 input with 2×2 filter, the output size is:
Output size = (Input size - 1) × stride + filter size = (2-1)×1 + 2 = 3

So we need a 3×3 output.

Let's compute step by step. Each input element gets multiplied by the entire filter and placed in the output with overlaps summed:

**From input element (-1) at position (0,0):**

```
[−1×1  −1×(−1)   0 ]   [ −1   1   0 ]
[−1×0  −1×1     0 ] = [  0  −1   0 ]
[  0     0      0 ]   [  0   0   0 ]
```

**From input element (2) at position (0,1):**

```
[  0   2×1    2×(−1)]   [  0   2  −2 ]
[  0   2×0    2×1   ] = [  0   0   2 ]
[  0    0      0    ]   [  0   0   0 ]
```

**From input element (3) at position (1,0):**

```
[ 3×0   3×1    0 ]   [  0   3   0 ]
[ 3×0   3×(−1) 0 ] = [  0  −3   0 ]
[  0     0     0 ]   [  0   0   0 ]
```

> As we can see here, the inference is incorrect, thus leading to false result in the end.

**From input element (1) at position (1,1):**

```
[  0     0      0    ]   [  0   0   0 ]
[  0   1×1    1×(−1)] = [  0   1  −1 ]
[  0   1×0    1×1   ]   [  0   0   1 ]
```

Now sum all contributions:

```
Position (0,0): -1 + 0 + 0 + 0 = -1
Position (0,1): 1 + 2 + 3 + 0 = 6
Position (0,2): 0 + (-2) + 0 + 0 = -2
Position (1,0): 0 + 0 + 0 + 0 = 0
Position (1,1): -1 + 0 + (-3) + 1 = -3
Position (1,2): 0 + 2 + 0 + (-1) = 1
Position (2,0): 0 + 0 + 0 + 0 = 0
Position (2,1): 0 + 0 + 0 + 0 = 0
Position (2,2): 0 + 0 + 0 + 1 = 1
```

**Final output:**

```
[-1  6 -2]
[ 0 -3  1]
[ 0  0  1]
```

Let me know if you'd like me to explain any of these steps in more detail!

> The last calculation is incorrect! As the third input is incorrect.

# Question 3

We will move on to the next question. Consider the following digram where the shaded blocks are the entries participating in one normalization step for a CNN-type architecture. N represents the mini-batch, H,W represent the different pixels of the “image” at this layer, and C represents different channels. Which one denotes the process of batch normalization and Which one denotes layer normalization?

Attached is the diagram for the question.

# Answer

Based on the description of the diagram where shaded blocks represent the entries participating in one normalization step, and given the dimensions N (mini-batch), H and W (spatial dimensions), and C (channels), the identification of batch normalization and layer normalization is as follows:

- **Batch normalization** normalizes across the batch dimension N for each feature (i.e., for each fixed channel C and spatial location H, W). Therefore, in the diagram, batch normalization corresponds to the option where the shaded blocks are along the N dimension, meaning that for a single spatial location and channel, all batch elements are included in the normalization step.
- **Layer normalization** normalizes across the feature dimensions C, H, and W for each batch element N. Therefore, in the diagram, layer normalization corresponds to the option where the shaded blocks are along the C, H, and W dimensions, meaning that for a single batch element, all channels and spatial locations are included in the normalization step.

From the provided text, it is not explicitly stated which option (A, B, or C) corresponds to which normalization type. However, based on the standard conventions:

- If an option shows shading over the N dimension, it denotes batch normalization.
- If an option shows shading over the C, H, and W dimensions, it denotes layer normalization.

You should examine the diagram in the screenshot to identify which option has shading along N for batch normalization and which has shading along C, H, W for layer normalization. If the diagram does not clearly show one option with shading over C, H, W, it may be that layer normalization is represented by an option where all channels and spatial locations are shaded for a single batch element.

> Deepseek model does not have multimodal capabilities, therefore it cannot read the diagram and can only provide a general answer.

# Question 4

Consider a simplified BN where we do not divide by the standard deviation of the data batch. Instead, we just de-mean our data batch before applying the scaling factor γand shifting factor β. For simplicity, consider scalar data in an n-sized batch: [x1,x2,...,xn]. Specifically, we let xi = xi−μ where μis the average across the batch and output [y1,y2,...,yn] where yi = gamma xi + β to the next layer. Assume we have a final loss Lsomewhere downstream. Calculate ∂L in terms of ∂L/∂xi and ∂L/∂yj for j = 1,...,n as well as γ and β as needed. Numerically, what is ∂L/∂x1 when n = 1 and our input batch just consists of [x1] with an output batch of [y1]? (Your answer should be a real number. No need to justify.) What happens when n→∞? (Feel free to assume here that all relevant quantities are bounded.)

# Answer

The general expression for the gradient of the loss with respect to the input $x_i$ is:

$$\frac{\partial L}{\partial x_i} = \gamma \left( \frac{\partial L}{\partial y_i} - \frac{1}{n} \sum_{j=1}^{n} \frac{\partial L}{\partial y_j} \right)$$

For $n = 1$ and input batch $[x_1]$, output batch $[y_1]$, the numerical value of $\frac{\partial L}{\partial x_1}$ is:

$$0$$

As $n \to \infty$, assuming all relevant quantities are bounded, $\frac{\partial L}{\partial x_i}$ approaches $\gamma \frac{\partial L}{\partial y_i}$.

> Correct!

# Question 5

We are moving on to question 3

Depthwise separable convolutions are a type of convolutional operation used in deep learning for image processing tasks. Unlike traditional convolutional operations, which perform both spatial and channel-wise convolutions simultaneously, depthwise separable convolutions decompose the convolution operation into two separate operations: Depthwise convolution and Pointwise convolution. This can be viewed as a low-rank approximation to a traditional convolution. For simplicity, throughout this problem, we will ignore biases while counting learnable parameters.

(a) Suppose the input is a three-channel 224 ×224-resolution image, the kernel size of the convolutional layer is 3 ×3, and the number of output channels is 4. What is the number of learnable parameters in the traditional convolution layer?

(b) Depthwise separable convolution consists of two parts: depthwise convolutions (Fig.2) followed by pointwise convolutions. Suppose the input is still a three-channel 224 ×224-resolution image. The input first goes through depthwise convolutions, where the number of output channels is the same as the number of input channels, and there is no “cross talk” between different channels. Then, this intermediate output goes through pointwise convolutions, which is basically a traditional convolution with the filter size being 1 ×1. Assume that we have 4 output channels. What is the total number of learnable parameters of the depthwise separable convolution layer which consists of both depthwise and pointwise convolutions?

# Answer

## 🔍 Step 1: Traditional Convolutional Layer

**Given:**

- Input channels: $C_{\text{in}} = 3$
- Output channels: $C_{\text{out}} = 4$
- Kernel size: $3 \times 3$

Each filter has:

$$3 \times 3 \times 3 = 27 \text{ parameters}$$

With 4 filters:

$$4 \times 27 = 108$$

**Answer:**

$$\boxed{108}$$

---

## 🔍 Step 2: Depthwise Separable Convolution

### (a) Depthwise Convolution

- Input channels: $C_{\text{in}} = 3$
- Output channels: $C_{\text{out}} = 3$
- Kernel size: $3 \times 3$

Each channel uses a separate filter:

$$3 \times (3 \times 3) = 27 \text{ parameters}$$

### (b) Pointwise Convolution

- Input channels: $3$
- Output channels: $4$
- Kernel size: $1 \times 1$

Each filter has:

$$1 \times 1 \times 3 = 3 \text{ parameters}$$

With 4 filters:

$$4 \times 3 = 12 \text{ parameters}$$

**Total parameters:**

$$27 + 12 = 39$$

**Answer:**

$$\boxed{39}$$

> Correct!

# Question 6

We are moving to the next question:

Recall that linear regression optimizes the following learning objective:

$$\mathcal{L}(w) = |y - Xw|_2^2 \tag{1}$$

One way of using dropout during SGD on the ( d )-dimensional input features ( x_i ) involves keeping each feature at random ( \sim i.i.d.\ Bernoulli(p) ) (and zeroing it out if not kept) and then performing a traditional SGD step.

It turns out that such dropout makes our learning objective effectively become

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

where ( \odot ) is the element-wise product and the random binary matrix ( R \in {0,1}^{n \times d} ) is such that ( R_{i,j} \sim i.i.d.\ Bernoulli(p) ).
We use ( \tilde{w} ) to remind you that this is learned by dropout.

Recalling how Tikhonov-regularized (generalized ridge-regression) least-squares problems involve solving:

$$\mathcal{L}(w) = |y - Xw|_2^2 + |\Gamma w|_2^2 \tag{3}$$

for some suitable matrix ( \Gamma ), it turns out we can manipulate (2) to eliminate the expectations and get:

$$\mathcal{L}(\tilde{w}) = |y - pX\tilde{w}|_2^2 + p(1-p)|\tilde{\Gamma}\tilde{w}|_2^2 \tag{4}$$

with ( \tilde{\Gamma} ) being a diagonal matrix whose ( j )-th diagonal entry is the norm of the ( j )-th column of the training matrix ( X ).

(a) Show that we can manipulate (2) to eliminate the expectations and get:

$$\mathcal{L}(\tilde{w}) = |y - pX\tilde{w}|_2^2 + p(1-p)|\tilde{\Gamma}\tilde{w}|_2^2 \tag{5}$$

with ( \tilde{\Gamma} ) being a diagonal matrix whose ( j )-th diagonal entry is the norm of the ( j )-th column of the training matrix ( X ).

(b) How should we transform the ( \tilde{w} ) we learn using (5) (i.e., with dropout) to get something that looks like a solution to the traditionally regularized problem (3)?

> Hint: This is related to how we adjust weights learned using dropout training for using them at inference time. PyTorch by default does this adjustment during training itself, but here, we are doing dropout slightly differently with no adjustments during training.

(c) With the understanding that the ( \Gamma ) in (3) is an invertible matrix, change variables in (3) to make the problem look like classical ridge regression:

$$\mathcal{L}(\tilde{w}) = |y - \tilde{X}\tilde{w}|_2^2 + \lambda|\tilde{w}|_2^2 \tag{6}$$

Explicitly, what is the changed data matrix ( \tilde{X} ) in terms of the original data matrix ( X ) and ( \Gamma )?

(d) Continuing the previous part, with the further understanding that ( \Gamma ) is a diagonal invertible matrix with the ( j )-th diagonal entry proportional to the norm of the ( j )-th column in ( X ), what can you say about the norms of the columns of the effective training matrix ( \tilde{X} ) and speculate briefly on the relationship between dropout and batch-normalization.

# Answer

## 🧠 Step 1: Problem Overview

We consider linear regression with dropout regularization. The standard least-squares objective is:

$$\mathcal{L}(w) = \|y - Xw\|_2^2$$

With dropout applied during stochastic gradient descent (SGD), the objective becomes:

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

where $R \in \{0,1\}^{n\times d}$ has i.i.d. Bernoulli($p$) entries, and $\odot$ denotes element-wise multiplication.

We aim to show that this expectation simplifies to:

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

where $\tilde{\Gamma}$ is a diagonal matrix with the $j$-th diagonal entry equal to the norm of the $j$-th column of $X$.

---

## 🔍 Step 2: Part (a) — Deriving the Simplified Objective

Let $R$ be a random matrix with entries $R_{i,j} \sim \text{Bernoulli}(p)$. Then:

$$(R \odot X)\tilde{w} = \sum_j R_{i,j}X_{i,j}\tilde{w}_j$$

The loss becomes:

$$\mathcal{L}(\tilde{w}) = \mathbb{E}\left[\sum_i \left(y_i - \sum_j R_{i,j} X_{i,j} \tilde{w}_j\right)^2\right]$$

For fixed $i$, let $A_i = \sum_j R_{i,j} X_{i,j} \tilde{w}_j$. Then:

- $\mathbb{E}[A_i] = p \sum_j X_{i,j} \tilde{w}_j = p(X\tilde{w})_i$
- $\mathrm{Var}(A_i) = p(1-p) \sum_j X_{i,j}^2 \tilde{w}_j^2$

Using the identity $\mathbb{E}[(y_i - A_i)^2] = (y_i - \mathbb{E}[A_i])^2 + \mathrm{Var}(A_i)$, we get:

$$\mathbb{E}[(y_i - A_i)^2] = (y_i - p(X\tilde{w})_i)^2 + p(1-p) \sum_j X_{i,j}^2 \tilde{w}_j^2$$

Summing over $i$:

$$\mathcal{L}(\tilde{w}) = \sum_i (y_i - p(X\tilde{w})_i)^2 + p(1-p) \sum_i \sum_j X_{i,j}^2 \tilde{w}_j^2$$

Note that:

- $\sum_i (y_i - p(X\tilde{w})_i)^2 = \|y - pX\tilde{w}\|_2^2$
- $\sum_i \sum_j X_{i,j}^2 \tilde{w}_j^2 = \sum_j \left(\sum_i X_{i,j}^2\right) \tilde{w}_j^2 = \sum_j \|X_{:,j}\|_2^2 \tilde{w}_j^2$

Define $\tilde{\Gamma}$ as the diagonal matrix with $\tilde{\Gamma}_{jj} = \|X_{:,j}\|_2$. Then:

$$\sum_j \|X_{:,j}\|_2^2 \tilde{w}_j^2 = \|\tilde{\Gamma}\tilde{w}\|_2^2$$

Thus:

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

---

## 🔁 Step 3: Part (b) — Transforming the Learned Weights

The dropout objective is:

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

The traditional regularized objective is:

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

Let $w = p\tilde{w}$. Then:

- $\|y - pX\tilde{w}\|_2^2 = \|y - Xw\|_2^2$
- $p(1-p)\|\tilde{\Gamma}\tilde{w}\|_2^2 = \frac{1-p}{p}\|\tilde{\Gamma}w\|_2^2$

So the transformed objective becomes:

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

This matches the traditional form with $\Gamma = \sqrt{\frac{1-p}{p}}\tilde{\Gamma}$. Thus, the learned weights $\tilde{w}$ should be scaled by $p$ for inference:

$$\boxed{p}$$

---

## 🔄 Step 4: Part (c) — Change of Variables for Ridge Regression

The regularized objective is:

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

Let $w = \Gamma^{-1}\tilde{w}$. Then:

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

So the objective becomes:

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

where $\tilde{X} = X\Gamma^{-1}$. This is classical ridge regression with $\lambda = 1$.

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

---

## 📊 Step 5: Part (d) — Column Norms and Connection to Batch Normalization

Assume $\Gamma$ is diagonal with $\Gamma_{jj} = \sqrt{\frac{1-p}{p}}\|X_{:,j}\|_2$. Then:

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

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

$$\tilde{X}_{:,j} = \frac{X_{:,j}}{\Gamma_{jj}} = \sqrt{\frac{p}{1-p}} \cdot \frac{X_{:,j}}{\|X_{:,j}\|_2}$$

So the norm of each column is:

$$\|\tilde{X}_{:,j}\|_2 = \sqrt{\frac{p}{1-p}}$$

All columns of $\tilde{X}$ have the same norm. This is reminiscent of batch normalization, which normalizes activations to have zero mean and unit variance. Dropout implicitly normalizes feature scales, similar to batch normalization.

$$\boxed{\sqrt{\frac{p}{1-p}}}$$

---

## ✅ Final Answers

- **(a)** Derived above.
- **(b)** Scale $\tilde{w}$ by $p$: $\boxed{p}$
- **(c)** Transformed data matrix: $\boxed{X\Gamma^{-1}}$
- **(d)** Column norms: $\boxed{\sqrt{\frac{p}{1-p}}}$

# Follow up question

Can you go back to the first question and see that in the question 1(c), your calculation for the third input seems a little bit wrong!

# Answer

I apologize for the error in my previous calculation for question 1(c) regarding the transpose convolution. Let me provide the corrected solution.

## 🔍 Corrected Calculation for Transpose Convolution (Question 1(c))

**Given:**

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

**Output size calculation:**
For transpose convolution with stride=1 and no padding, the output size is $(\text{input size} - 1) \times \text{stride} + \text{filter size} = (2-1) \times 1 + 2 = 3$, so the output is a $3 \times 3$ matrix.

**Step-by-step computation:**
Each input element is multiplied by the entire filter and placed in the output matrix starting at position $(i \times \text{stride}, j \times \text{stride})$ with overlaps summed.

1. **Input element $-1$ at (0,0):**
   Contribution: $-1 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix}$
   Placed at: (0,0) to (1,1)
   Matrix:

$$\begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2. **Input element $2$ at (0,1):**
   Contribution: $2 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 0 & 2 \end{bmatrix}$
   Placed at: (0,1) to (1,2)
   Matrix:

$$\begin{bmatrix} 0 & 2 & -2 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

3. **Input element $3$ at (1,0):**
   Contribution: $3 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & -3 \\ 0 & 3 \end{bmatrix}$
   Placed at: (1,0) to (2,1)
   Matrix:

$$\begin{bmatrix} 0 & 0 & 0 \\ 3 & -3 & 0 \\ 0 & 3 & 0 \end{bmatrix}$$

4. **Input element $1$ at (1,1):**
   Contribution: $1 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$
   Placed at: (1,1) to (2,2)
   Matrix:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

**Summing all contributions:**

$$
\begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 2 & -2 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 3 & -3 & 0 \\ 0 & 3 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} -1 & 3 & -2 \\ 3 & -3 & 1 \\ 0 & 3 & 1 \end{bmatrix}
$$

**Correct output:**

$$
\boxed{\begin{bmatrix} -1 & 3 & -2 \\ 3 & -3 & 1 \\ 0 & 3 & 1 \end{bmatrix}}
$$

Thank you for pointing out the error. If you have any further questions, feel free to ask!

> CorrecT!