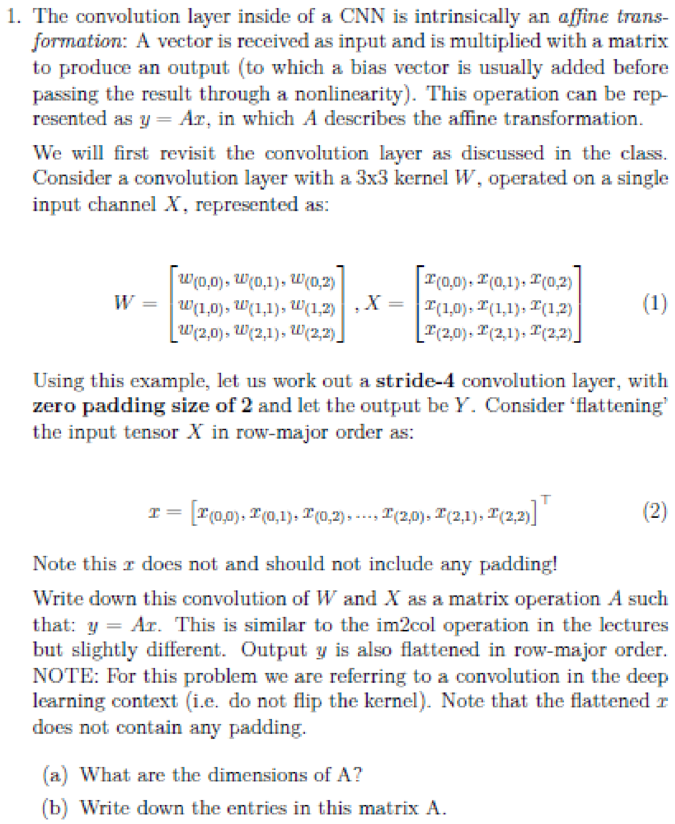
Assignment 2 Theory Problem Set

DO NOT TAG

Theory PS Q1. Must show your work for full credit. Feel free to add extra slides if needed.

Question:



1. The convolution layer inside of a CNN is intrinsically an *affine transformation*: A vector is received as input and is multiplied with a matrix to produce an output (to which a bias vector is usually added before passing the result through a nonlinearity). This operation can be represented as $y = Ax$, in which A describes the affine transformation.

   We will first revisit the convolution layer as discussed in the class. Consider a convolution layer with a 3x3 kernel W, operated on a single input channel X, represented as:

$$W = \begin{bmatrix} w_{(0,0)}, w_{(0,1)}, w_{(0,2)} \\ w_{(1,0)}, w_{(1,1)}, w_{(1,2)} \\ w_{(2,0)}, w_{(2,1)}, w_{(2,2)} \end{bmatrix}, X = \begin{bmatrix} x_{(0,0)}, x_{(0,1)}, x_{(0,2)} \\ x_{(1,0)}, x_{(1,1)}, x_{(1,2)} \\ x_{(2,0)}, x_{(2,1)}, x_{(2,2)} \end{bmatrix} \quad (1)$$

   Using this example, let us work out a **stride-4** convolution layer, with **zero padding size of 2** and let the output be Y. Consider 'flattening' the input tensor X in row-major order as:

$$x = \left[x_{(0,0)}, x_{(0,1)}, x_{(0,2)}, \ldots, x_{(2,0)}, x_{(2,1)}, x_{(2,2)}\right]^{\top} \quad (2)$$

   Note this x does not and should not include any padding!

   Write down this convolution of W and X as a matrix operation A such that: $y = Ax$. This is similar to the im2col operation in the lectures but slightly different. Output y is also flattened in row-major order. NOTE: For this problem we are referring to a convolution in the deep learning context (i.e. do not flip the kernel). Note that the flattened x does not contain any padding.

   (a) What are the dimensions of A?

   (b) Write down the entries in this matrix A.

(A) What are the dimensions of A

- We are performing a stride-4 convolution using a 3x3 kernel W on a 3x3 input X and padding the input with a padding size of 2.
- Need to express the convolution as a matrix multiplication y=Ax, where x is the flattened input and y is the flattened output.

Dimensions of input X

- The input X is flattened in row-major order and we need to do the same for the output y
- The input matrix X is 3x3. When flattened in row-major order, it becomes a 9-element vector x:
  x=[x(0,0),x(0,1),x(0,2),x(1,0),x(1,1),x(1,2),x(2,0),x(2,1),x(2,2)]
  Flattened matrix has a dimension of 9

Dimensions of Output Y

- We have a stride of 4 and padding of 2, so lets add that, the resulting matrix is 7x7

The original input is 3x3. With padding size 2, the padded input will be 7x7:

$$X_{\text{padded}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x(0,0) & x(0,1) & x(0,2) & 0 & 0 \\ 0 & 0 & x(1,0) & x(1,1) & x(1,2) & 0 & 0 \\ 0 & 0 & x(2,0) & x(2,1) & x(2,2) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Theory PS Q1. Must show your work for full credit. Feel free to add extra slides if needed. (continued)

Output Size computation:

- With a kernel size of 3x3, stride 4, and padded input size 7x7, the output size is calculated as:

$$\text{Output size} = \left(\frac{7-3}{4}+1\right) \times \left(\frac{7-3}{4}+1\right) = 2 \times 2$$

So the output Y will be 2x2. When flattened in row-major order, it becomes a vector y of size 4.

Dimensions of A

- Since y=Ax, the matrix A needs to map the 9-dimensional vector x to the 4-dimensional vector y. Therefore, A will have dimensions 4×9.

(B) Write down the entries in matrix A

- Matrix A represents how the flattened input vector x gets multiplied by the flattened kernel WW to produce the flattened output vector y. Each row of A corresponds to one element in the output y, and each column corresponds to an element in the input x. The entries in A are determined by the position of the kernel W on the padded input matrix
- Place the kernel W over the padded input matrix
  The following are the four kernel locations due to padding and stride in the input matrix and the corresponding row in A

1. Top Left Corner of padded input

$$W \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & x(0,0) & x(0,1) \\ 0 & x(1,0) & x(1,1) \end{bmatrix}$$

$$[w(1,1), w(1,2), 0, w(2,1), w(2,2), 0, 0, 0, 0]$$

2. Top Right Corner of padded input

$$W \cdot \begin{bmatrix} 0 & 0 & 0 \\ x(0,1) & x(0,2) & 0 \\ x(1,1) & x(1,2) & 0 \end{bmatrix}$$

$$[0, w(1,0), w(1,1), 0, w(2,0), w(2,1), 0, 0, 0]$$

3. Bottom Left Corner of padded input

$$W \cdot \begin{bmatrix} 0 & x(1,0) & x(1,1) \\ 0 & x(2,0) & x(2,1) \\ 0 & 0 & 0 \end{bmatrix}$$

$$[0, 0, 0, w(0,1), w(0,2), 0, w(1,1), w(1,2), 0]$$

4. Bottom Right Corner of padded input

$$W \cdot \begin{bmatrix} x(1,1) & x(1,2) & 0 \\ x(2,1) & x(2,2) & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

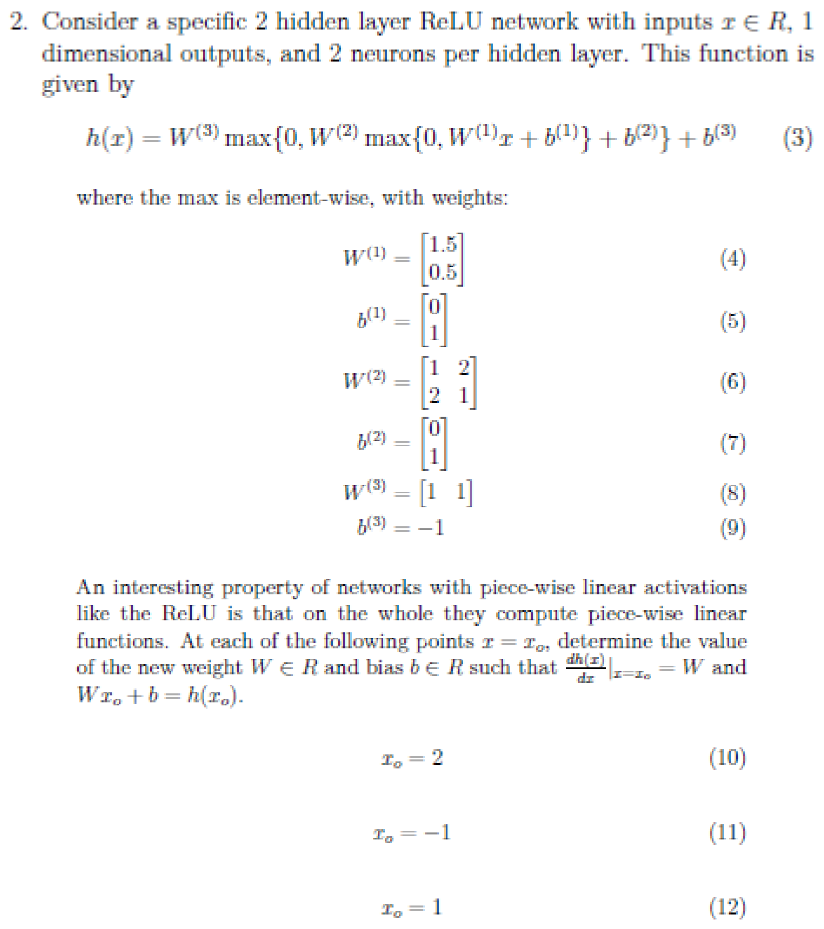
$$[0, 0, 0, 0, w(0,1), w(0,2), 0, w(1,1), w(1,2)]$$

Theory PS Q1. Must show your work for full credit. Feel free to add extra slides if needed.
(continued)

Thus the final form of Matrix A will be as follows:

$$A = \begin{bmatrix} w(1,1) & w(1,2) & 0 & w(2,1) & w(2,2) & 0 & 0 & 0 & 0 \\ 0 & w(1,0) & w(1,1) & 0 & w(2,0) & w(2,1) & 0 & 0 & 0 \\ 0 & 0 & 0 & w(0,1) & w(0,2) & 0 & w(1,1) & w(1,2) & 0 \\ 0 & 0 & 0 & 0 & w(0,1) & w(0,2) & 0 & w(1,1) & w(1,2) \end{bmatrix}$$

Theory PS Q2. Must show your work for full credit. Feel free to add extra slides if needed.



2. Consider a specific 2 hidden layer ReLU network with inputs $x \in R$, 1 dimensional outputs, and 2 neurons per hidden layer. This function is given by

$$h(x) = W^{(3)} \max\{0, W^{(2)} \max\{0, W^{(1)}x + b^{(1)}\} + b^{(2)}\} + b^{(3)} \tag{3}$$

where the max is element-wise, with weights:

$$W^{(1)} = \begin{bmatrix} 1.5 \\ 0.5 \end{bmatrix} \tag{4}$$

$$b^{(1)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{5}$$

$$W^{(2)} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \tag{6}$$

$$b^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{7}$$

$$W^{(3)} = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{8}$$

$$b^{(3)} = -1 \tag{9}$$

An interesting property of networks with piece-wise linear activations like the ReLU is that on the whole they compute piece-wise linear functions. At each of the following points $x = x_o$, determine the value of the new weight $W \in R$ and bias $b \in R$ such that $\frac{dh(x)}{dx}|_{x=x_o} = W$ and $Wx_o + b = h(x_o)$.

$$x_o = 2 \tag{10}$$

$$x_o = -1 \tag{11}$$

$$x_o = 1 \tag{12}$$

- The function is piece-wise linear due to the ReLU activation, which means it computes linear functions over different regions of the input space.
- You are working with three sets of weights and biases, and the function involves element-wise max operations, likely representing layers in a neural network with ReLU activation.

$$W(1) = \begin{pmatrix} 1.5 \\ 0.5 \end{pmatrix}, \quad b(1) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$W(2) = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \quad b(2) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$W(3) = \begin{pmatrix} 1 & 1 \end{pmatrix}, \quad b(3) = -1$$

$$h(x) = \text{ReLU}(W(1)g + b(1)) + \text{ReLU}(W(2)g + b(2)) + \text{ReLU}(W(3)g + b(3))$$

Theory PS Q2. Must show your work for full credit. Feel free to add extra slides if needed. (continued)

For x0=2

- We will compute the value of h(x0) and its derivative at x0=2
- First Layer Output:

$$W(1)\cdot 2+b(1)=\begin{pmatrix}1.5\times 2\\0.5\times 2\end{pmatrix}+\begin{pmatrix}0\\1\end{pmatrix}=\begin{pmatrix}3\\2\end{pmatrix}$$

After applying ReLU, the result is $\mathrm{ReLU}(3,2)=\begin{pmatrix}3\\2\end{pmatrix}$.

- Second Layer Output:

$$W(2)\cdot 2+b(2)=\begin{pmatrix}1\times 2+2\times 2\\2\times 2+1\times 2\end{pmatrix}+\begin{pmatrix}0\\1\end{pmatrix}=\begin{pmatrix}6\\7\end{pmatrix}$$

After applying ReLU, the result is $\mathrm{ReLU}(6,7)=\begin{pmatrix}6\\7\end{pmatrix}$.

- Third Layer Output:

$$W(3)\cdot 2+b(3)=(1\times 2+1\times 2)-1=4-1=3$$

After applying ReLU, the result is $\mathrm{ReLU}(3)=3$.

Now, the output of h(2) is the sum of all the ReLU results:

$$h(2)=\begin{pmatrix}3\\2\end{pmatrix}+\begin{pmatrix}6\\7\end{pmatrix}+3=\begin{pmatrix}9\\9\end{pmatrix}+3=12$$

The derivative is simply the sum of the gradients of the active linear functions. Since all components are active (i.e., not zeroed out by ReLU), the derivative is:

$$\left.\frac{dh(x)}{dx}\right|_{x=2}=W(1)+W(2)+W(3)=\begin{pmatrix}1.5\\0.5\end{pmatrix}+\begin{pmatrix}1&2\\2&1\end{pmatrix}+\begin{pmatrix}1&1\end{pmatrix}$$

Summing them:

$$W=1.5+(1,2)+1=(3.5,4.5)$$

Given:

- $W=\begin{pmatrix}3.5\\4.5\end{pmatrix}$,
- $h(x_0)=\begin{pmatrix}6.5\\7.5\end{pmatrix}$,
- $x_0=2$,

Now, we calculate Wx_0:

$$Wx_0=\begin{pmatrix}3.5\\4.5\end{pmatrix}\times 2=\begin{pmatrix}7\\9\end{pmatrix}$$

Now, calculate the bias b:

$$b=h(x_0)-Wx_0=\begin{pmatrix}6.5\\7.5\end{pmatrix}-\begin{pmatrix}7\\9\end{pmatrix}=\begin{pmatrix}-0.5\\-1.5\end{pmatrix}$$

Thus, the bias b is $\begin{pmatrix}-0.5\\-1.5\end{pmatrix}$.

So, the weight and bias terms are:

- For $x_0=2$: $W=\begin{pmatrix}3.5\\4.5\end{pmatrix}$, $b=\begin{pmatrix}-0.5\\-1.5\end{pmatrix}$

Theory PS Q2. Must show your work for full credit. Feel free to add extra slides if needed. (continued)

For x0=-1

- We will compute the value of h(x0) and its derivative at x0=-1
- First Layer Output:

$$W(1)\cdot(-1)+b(1)=\begin{pmatrix}1.5\times(-1)\\0.5\times(-1)\end{pmatrix}+\begin{pmatrix}0\\1\end{pmatrix}=\begin{pmatrix}-1.5\\0.5\end{pmatrix}$$

After applying ReLU, the result is $\mathrm{ReLU}(-1.5,0.5)=\begin{pmatrix}0\\0.5\end{pmatrix}$.

- Second Layer Output:

$$W(2)\cdot(-1)+b(2)=\begin{pmatrix}1\times(-1)+2\times(-1)\\2\times(-1)+1\times(-1)\end{pmatrix}+\begin{pmatrix}0\\1\end{pmatrix}=\begin{pmatrix}-3\\-2\end{pmatrix}$$

After applying ReLU, the result is $\mathrm{ReLU}(-3,-2)=\begin{pmatrix}0\\0\end{pmatrix}$.

- Third Layer Output:

$$W(3)\cdot(-1)+b(3)=(1\times(-1)+1\times(-1))-1=-2-1=-3$$

After applying ReLU, the result is $\mathrm{ReLU}(-3)=0$.

Now, the output of h(-1) is the sum of all the ReLU results:

$$h(-1)=\begin{pmatrix}0\\0.5\end{pmatrix}+\begin{pmatrix}0\\0\end{pmatrix}+0=\begin{pmatrix}0\\0.5\end{pmatrix}$$

The derivative is simply the sum of the gradients of the active linear functions. Since all components are active (i.e., not zeroed out by ReLU), the derivative is:

$$\left.\frac{dh(x)}{dx}\right|_{x=-1}=W(1)=\begin{pmatrix}1.5\\0.5\end{pmatrix}$$

So at $x_0=-1$, the new weight is $W=\begin{pmatrix}1.5\\0.5\end{pmatrix}$.

Now to calculate the bias term:

Using $W=\begin{pmatrix}1.5\\0.5\end{pmatrix}$ and $x_0=-1$, and $h(-1)=\begin{pmatrix}0\\0.5\end{pmatrix}$:

$$\begin{pmatrix}1.5\\0.5\end{pmatrix}\cdot(-1)+b=\begin{pmatrix}0\\0.5\end{pmatrix}$$

This gives:

$$\begin{pmatrix}-1.5\\-0.5\end{pmatrix}+b=\begin{pmatrix}0\\0.5\end{pmatrix}$$

Solving for B, we get:

$$b=\begin{pmatrix}1.5\\1\end{pmatrix}$$

Theory PS Q2. Must show your work for full credit. Feel free to add extra slides if needed. (continued)

For x0=1

- We will compute the value of h(x0) and its derivative at x0=1
- First Layer Output:

$$W(1)\cdot 1 + b(1) = \begin{pmatrix} 1.5\times 1 \\ 0.5\times 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$$

After applying ReLU, the result is $\mathrm{ReLU}(1.5, 1.5) = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$.

- Second Layer Output:

$$W(2)\cdot 1 + b(2) = \begin{pmatrix} 1\times 1 + 2\times 1 \\ 2\times 1 + 1\times 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

After applying ReLU, the result is $\mathrm{ReLU}(3, 4) = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$.

- Third Layer Output:

$$W(3)\cdot 1 + b(3) = (1\times 1 + 1\times 1) - 1 = 2 - 1 = 1$$

After applying ReLU, the result is $\mathrm{ReLU}(1) = 1$.

Now, the output of h(-1) is the sum of all the ReLU results:

$$h(1) = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix} + \begin{pmatrix} 3 \\ 4 \end{pmatrix} + 1 = \begin{pmatrix} 4.5 \\ 5.5 \end{pmatrix}$$

The derivative is simply the sum of the gradients of the active linear functions. Since all components are active (i.e., not zeroed out by ReLU), the derivative is:

$$\left.\frac{dh(x)}{dx}\right|_{x=1} = W(1) + W(2) + W(3) = \begin{pmatrix} 1.5 \\ 0.5 \end{pmatrix} + \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \end{pmatrix}$$

Summing them: $W = \begin{pmatrix} 3.5 \\ 4.5 \end{pmatrix}$

Now to calculate the bias term:

$$W \cdot x_0 + b = h(1)$$

Substitute $W = \begin{pmatrix} 3.5 \\ 4.5 \end{pmatrix}$, $x_0 = 1$, and $h(1) = \begin{pmatrix} 4.5 \\ 5.5 \end{pmatrix}$:

$$\begin{pmatrix} 3.5 \\ 4.5 \end{pmatrix} \cdot 1 + b = \begin{pmatrix} 4.5 \\ 5.5 \end{pmatrix}$$

This gives:

$$\begin{pmatrix} 3.5 \\ 4.5 \end{pmatrix} + b = \begin{pmatrix} 4.5 \\ 5.5 \end{pmatrix}$$

Solving for B, we get:

$$b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Final Results:

- For $x_0 = 2$:

$$W = \begin{pmatrix} 3.5 \\ 4.5 \end{pmatrix}, \quad b = \begin{pmatrix} -0.5 \\ -1.5 \end{pmatrix}$$

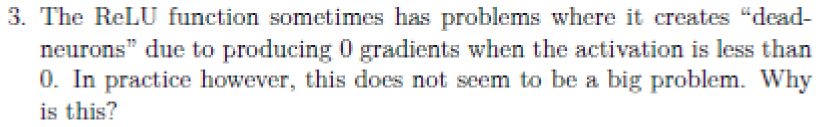
- For $x_0 = -1$:

$$W = \begin{pmatrix} 2.5 \\ 3.5 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

- For $x_0 = 1$:

$$W = \begin{pmatrix} 3.5 \\ 4.5 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Theory PS Q3. Feel free to add extra slides if needed.
Question:


3. The ReLU function sometimes has problems where it creates "dead-neurons" due to producing 0 gradients when the activation is less than 0. In practice however, this does not seem to be a big problem. Why is this?

The issue of "dead neurons" in ReLU (Rectified Linear Unit) activation functions occurs when a neuron outputs zero for all inputs due to the negative side of the ReLU function. This causes the gradients to be zero, preventing those neurons from updating their weights during backpropagation.

However, this problem is often not significant in reality for several reasons:

- Initialization and Input Distributions: When using proper weight initialization methods and appropriate input scaling, many neurons avoid falling into the dead zone, especially during the initial stages of training
- Sparsity is Beneficial: ReLU naturally introduces sparsity in the activations (many neurons output zero), which can be beneficial for efficiency and reducing overfitting. In practice, only a subset of neurons need to be active at any given time.
- Large Networks with Many Neurons: In deep networks, the large number of neurons compensates for the few that may die. Even if some neurons produce zero outputs, others continue learning and can carry the forward propagation process.
- Learning Rate Adjustments: Proper learning rate tuning ensures that neurons are less likely to become permanently inactive, as updates to weights can eventually lead neurons back into the active region.

Overall, while dead neurons can occur, their impact is typically minimal due to careful design and training techniques. This is certainly an issue to be aware of us but in practice it is usually not a considerable difficulty to overcome

Assignment 2 Paper Review

DO NOT TAG

Provide a short preview of the paper of your choice.
I chose: “Taskonomy: Disentangling Task Transfer Learning" by Amir R. Zamir et al

The paper *"Taskonomy: Disentangling Task Transfer Learning"* by Amir R. Zamir et al. (2018) introduces a novel framework for understanding task transferability in computer vision. Its main contribution is the creation of a comprehensive task transfer hierarchy, which maps how well various vision tasks (e.g., surface normal estimation, depth prediction, and object classification) transfer knowledge to each other. By constructing a large-scale dataset of 26 tasks, the authors empirically demonstrate that certain tasks act as better feature extractors and are more suitable for transfer learning. For example, low-level tasks like depth estimation transfer better to related tasks than high-level tasks like object classification.

One of the key strengths of this paper was that it was truly groundbreaking and had a significant impact on multi-task and transfer learning research in computer vision. Its systematic approach and the introduction of the task affinity map were a significant advancement, offering a clear, data-driven approach to identifying optimal task transfer pairs. And the large dataset built specifically for multiple vision tasks ensured that the results are generalizable across many computer vision problems. However, the framework does have some weaknesses. It might oversimplify task relationships by assuming that transferability is uniform across datasets and architectures, ignoring task-specific nuances that could affect transferability in different contexts. Also, the paper focuses on just visual task. How do we know if these findings extend to non-visual domains, which could benefit from similar task transfer insights? The paper opens the door to more research.

This latter point is my biggest takeaway. What does this say about all machine learning tasks? How many other areas can generic training create benefit for unrelated futures tasks. We are already seeing this take place in the NLP space where foundation models are created, doing the heavy lifting for models that are later finetuned for more specific use cases. I think it also makes us ask us about ourselves and learning in general. Are we as humans learning tasks that better prepare us for future tasks that only share cursory characteristics?

Paper specific Q1. Feel free to add extra slides if needed.
Question:
Do the task pairs with stronger arrows (better transfer) make sense in terms of why they would transfer better? Pick one positive pair (with good transfer) and one negative pair (with bad transfer) and conjecture why it might be the case. Note that there are several types of features in deep learning, including low-level (e.g. edges), mid-level (components), and high-level (abstract concepts and classification layer) that you might reason about.

In Taskonomy: Disentangling Task Transfer Learning, the paper explores how well different tasks in computer vision transfer knowledge to each other, and the strength of these transfers often makes sense based on the types of information each task processes. For example, a strong pair is depth estimation to surface normal prediction. Depth estimation is about understanding the 3D shape and distance of objects in a scene, which is very similar to surface normal prediction, a task that involves figuring out the direction each surface in the scene is facing. Both of these tasks rely on similar types of information, such as the edges and shapes of objects, making the features learned from one task (depth estimation) useful for the other (surface normals). The overlap in what they need to "see" in an image explains why the transfer between these two tasks works well. They are both trying to identify features that are similar in that they are both “low-level” as opposed to higher level abstract characteristics.

On the other hand, there are some tasks that don’t transfer well. A good example is object classification to depth estimation. Object classification is about recognizing and labeling objects (like determining if something is a cat or a chair), which involves understanding high-level, abstract information like shapes and textures. This type of task relies on the final, more semantic layers of a neural network, which are good for recognizing entire objects but not great at understanding their 3D structure. In contrast, depth estimation needs detailed information about the geometry of objects, like where their edges are and how far away they are from the camera. The high-level features used for object classification don’t help with the fine-grained, low-level details needed for depth estimation. That’s why transferring knowledge from object classification to depth estimation doesn’t work well—the two tasks focus on very different kinds of information. This mismatch between abstract object recognition and detailed geometric understanding explains the weak transferability between these tasks. These observations make intuitive sense to human understanding in terms of what tasks would and would not be transferable.

Paper specific Q2. Feel free to add extra slides if needed.
Question:
What does this say in terms of practical usage of deep learning across tasks? How might we use this information to guess where to transfer from if we have a new target task?

The insights from the paper have practical implications for how we approach deep learning across different tasks. Essentially, if we understand how tasks are related, we can make smarter decisions about which tasks to transfer knowledge from when working on a new task. For instance, if our new task involves understanding the 3D structure of a scene, we should look to transfer from tasks that deal with similar low-level geometric information like depth estimation or surface normal prediction. These tasks rely on the same types of features, like object edges and surface orientations, which are highly relevant for other tasks involving 3D spatial understanding.

On the flip side, if our target task is more abstract, like recognizing objects (identifying cars, trees, etc.), transferring knowledge from a task that focuses on geometry won't be very helpful. Object classification relies on high-level, abstract features, like shapes and textures, which aren't as useful for tasks that depend on detailed spatial relationships. So, if our target task requires abstract understanding, we should look to transfer from similar tasks that also focus on high-level features, such as other object classification tasks or scene understanding.

In practice, this means we need to think about the type of information a task relies on. Is it dependent on low-level, mid-level, or high-level features. And then we can choose a source task that has learned similar information. If our new task is low-level, like predicting edges or surface normals, we should transfer from tasks that also deal with geometry. But if it's a high-level task, like classifying objects or understanding the overall scene, we should transfer from tasks that have learned abstract, semantic features. By understanding these task relationships, we can save time and resources by pretraining our models on the most relevant tasks, improving performance in our new task without needing to start from scratch.

It can also help us set expectations. Some tasks are more amendable to transfer learning than others as shown in the paper. This can help us save time and resources directing transfer learning where it is most likely to achieve fruitful results.

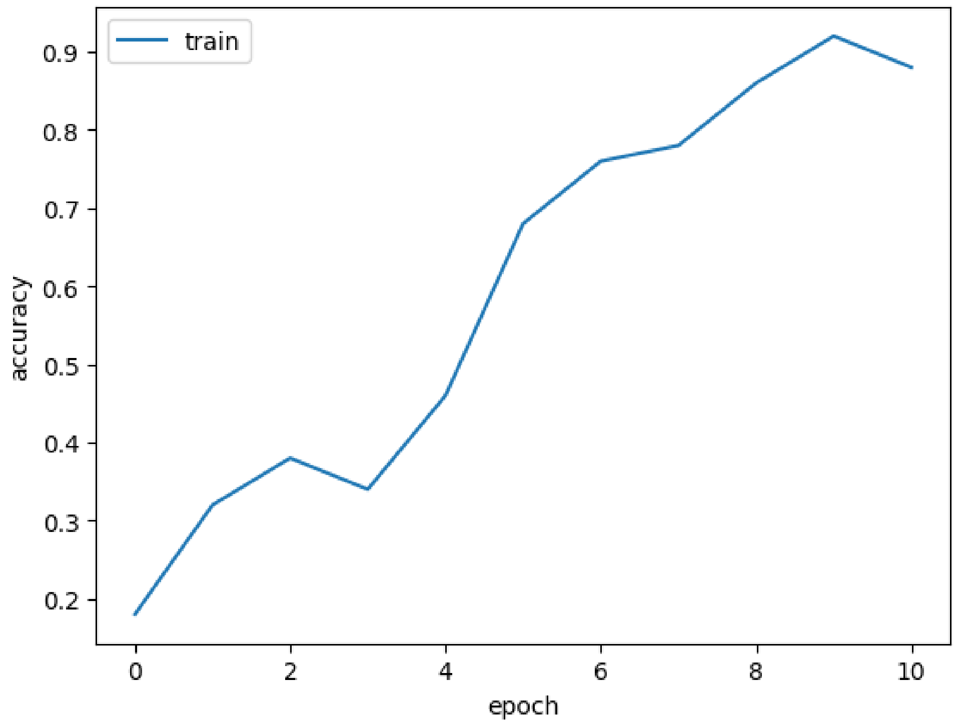
Assignment 2 Writeup

DO NOT TAG

Part-1 ConvNet

DO NOT TAG

Put your training curve here:



My CNN Model

DO NOT TAG

Describe and justify your model design in plain text here:

Our model has 4 convolutional layers with the last layer having 128 input channels and 256 output channels. We used batch normalization between each layer along with max pooling with a kernel size of 2 and a stride of 2. Each layer used ReLu for its activation function. After the four convolutional layers we had two more fully connected layers. We also used two dropout layers, one after all the convolution layers (5% dropout) and one after the fully connected layers (10% dropout).

Our model consists of four convolutional layers followed by two fully connected layers, carefully designed to balance feature extraction, computational efficiency, and classification performance. Here’s a breakdown with justifications for each choice:

Number of Convolutional Layers (4): We used four convolutional layers to build a hierarchical representation of the input data. With each successive layer, the model learns increasingly abstract and complex features. For instance, the first layer might detect edges, while deeper layers could identify shapes, textures, and higher-level patterns. Four layers allow sufficient depth for meaningful feature extraction without making the model too deep, which could lead to overfitting or excessive computational costs.
Kernel Size (2x2): The kernel size of 2x2 was used for max pooling, which reduces the spatial dimensions of the feature maps. This choice effectively halves the height and width of the feature maps with each pooling operation, preserving essential information while reducing computational load. A smaller kernel like 2x2 retains more local detail compared to larger kernels, which is important for preserving fine-grained information in early layers.
Stride of 2: A stride of 2 for max pooling ensures that we downsample the feature maps efficiently, reducing their spatial size while maintaining computational feasibility
Number of Output Channels (256): By increasing the number of channels as we progress deeper in the network, we allow the model to learn more complex patterns. The final layer has 256 output channels, which is sufficient for capturing a wide range of high-level features. More channels provide the model with a richer feature space, but we found that adding any more just slowed down training without improving performance

Describe and justify your model design in plain text here (continued):

ReLU Activation: ReLU is computationally efficient and helps avoid issues like the vanishing gradient problem, making it an ideal choice for deep networks. Standard best practice choice for activation.
Batch Normalization: Batch normalization after each convolutional layer normalizes the activations, reducing internal covariate shift. This helps speed up training, improves stability, and reduces sensitivity to the initialization of weights and learning rate. It also acts as a form of regularization, reducing the risk of overfitting.
Dropout: We introduced dropout layers (5% after the convolutional layers and 10% after the fully connected layers) to prevent overfitting. Dropout helps the network generalize better by randomly deactivating a fraction of neurons during training, forcing the model to learn more robust and distributed representations. The different dropout rates are tuned to the complexity of the respective layers—lower for convolutional layers where feature extraction is more structured, and higher for fully connected layers where there's more risk of overfitting due to the high number of parameters. These values and number of layers we had the best results with.
Fully Connected Layers: After feature extraction, we added two fully connected layers. These layers take the high-level features learned from the convolutional layers and combine them for classification. Fully connected layers are necessary for this final decision-making step because they allow the model to use the learned features to make accurate predictions.

This model design reflects a balance between extracting rich features through convolutional layers and making final predictions using fully connected layers, with dropout and batch normalization reducing the risk of overfitting and improving generalization.

Describe and justify your choice of hyper-parameters:

Here’s a breakdown of the hyperparameters used in our training process, along with a justification for each choice:

Batch Size (128): A batch size of 128 strikes a balance between computational efficiency and model performance. Smaller batch sizes tend to provide more updates and can sometimes offer better generalization, while larger batch sizes allow for faster training on GPUs due to parallelism. We saw similar performance from batch sizes 64 and up but really deteriorated below that so kept 128 for best performance

Learning Rate (0.1): A learning rate of 0.1 is a little high for what is typical for optimizers like SGD with momentum. It provides a good balance between converging quickly and avoiding overshooting the minimum though. Higher learning rates for the most part gave us comparable performance and trained quicker. We kept it at 0.01 for most of training to avoid overshooting and with no downside to overall accuracy in line with best practice and then increased to 0.1 for final model. Learning rates 0.0005 and below trained to slow and performance dropped off.

Regularization (reg = 0.0005): We apply an L2 regularization penalty of 0.0005 to help prevent overfitting by discouraging overly complex models. This regularization term penalizes large weights, encouraging the model to learn simpler patterns and improving generalization to unseen data. We kept his very low to not constrain the model and also paired it with out regularization techniques such as dropout

Epochs (10): Training for 10 epochs is a practical choice for balancing between underfitting and overfitting. We tried for higher (30+ epochs) but performance flat lined ~81% at roughly 8 epochs and increased epochs did not help so for our final model we used 10.

Steps ([6, 8]): The learning rate steps at epochs 6 and 8 indicate when we reduce the learning rate to encourage more precise convergence. Reducing the learning rate in stages allows the model to make large adjustments early on (when the learning rate is high) and fine-tune the learned parameters in later stages (when the learning rate is lower). We experimented with values here but ended up not using a warm up as it had no effect on overall end state performance

Describe and justify your choice of hyper-parameters:

Here’s a breakdown of the hyperparameters used in our training process, along with a justification for each choice:

Momentum (0.9): Momentum is used to accelerate convergence by smoothing out updates in the direction of the steepest descent. A momentum value of 0.9 is a standard choice and effectively dampens oscillations while accelerating the training process. It helps push the optimizer towards minima more efficiently than vanilla SGD by "remembering" past gradients, allowing the model to converge faster and with more stability. This had minimal impact on performance in either direction though.

Beta (1): Beta refers to a parameter in the Focal Loss function (if used), where it controls how much the loss focuses on hard-to-classify examples. In this case, a gamma value of 1 means no adjustment is made to the standard cross-entropy loss (since Focal Loss reduces to cross-entropy when gamma is 1). This indicates that we aren't explicitly focusing on hard examples and are treating all misclassifications equally, which is suitable when the data isn't too imbalanced. *(Note: We did not use focal loss for this portion of the assignment, including it for comprehensiveness)*

What’s your final accuracy on validation set?

Our final validation accuracy was 81.81%

Data Wrangling

DO NOT TAG

What’s your result of training with regular CE loss on imbalanced CIFAR-10?

Tune appropriate parameters and fill in your best per-class accuracy in the table

	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Acc.
CE Loss	91.8%	96.1%	63.2%	64.2%	50.3%	5.2%	15.1%	0.0%	0.0%	0.0%	38.59%

Overall Accuracy: 38.59%

What’s your result of training with CB-Focal loss on imbalanced CIFAR-10?

Additionally tune the hyper-parameter beta and fill in your per-class accuracy in the table; add more rows as needed

	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Acc.
beta=0.9	90.6%	85.6%	63.0%	48.0%	27.5%	10.4%	17.7	0.0%	0.0%	0.0%	34.28%
beta=0.95	38.88%	91.0%	89.6%	62.2%	56.4%	39.4%	16.7%	19.7%	13.8%	0.0%	38.73%
beta=0.99	86.9%	90.0%	63.4%	52.0%	36.6%	19.5%	21.8%	33.4%	4.0%	1.0%	40.86%
beta=0.997*	**74.6%**	**79.8%**	**54.1%**	**22.5%**	**17.4%**	**42.0%**	**47.0%**	**42.5%**	**36.6%**	**19.5%**	**43.60%**
beta=0.999	36.1%	10.3%	4.2%	0.7%	25.4%	43.0%	53.8%	66.3%	64.7%	52.9%	35.74%
beta=0.9999	1.5%	0.0%	0.6%	0.0%	0.9%	34.3%	66.0%	66.4%	67.2%	52.8%	28.97%

Put your results of CE loss and CB-Focal Loss(best) together:

	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Acc.
CE Loss	91.8%	96.1%	63.2%	64.2%	50.3%	5.2%	15.1%	0.0%	0.0%	0.0%	38.59%
CB-Focal	74.6%	79.8%	54.1%	22.5%	17.4%	42.0%	47.0%	42.5%	36.6%	19.5%	43.60%

Describe and explain your observation on the result: *Explanation should go into* ***WHY*** *things work the way they do in the context of Machine Learning theory/intuition, along with justification for your experimentation methodology.* ***DO NOT*** *just describe the results, you should explain the reasoning behind your choices and what behavior you expected. Also, be cognizant of the best way to mindful and show the results that best emphasizes your key observations. If you need more than one slide to answer the question, you are free to create new slides.*

The first thing to note is the ovrall accuracy has improved fairly significantly. We have increased 5% in raw accuracy which is a 13% increase to ~43% from our previous level ~38%. If we look at the distribution of errors it paints a more descript picture. The CE Loss is really focusing on getting the first two classes correct and to a lesser extent the 3 after that. These are the classes it sees the most of and it learns these classes at the detriment of learning the other 5. its average accuracy on the last 5 classes is around 4%. Worse than a random guess of 10%. This averages out to an accuracy not awful at 38.5% but the picture is actually bleaker than that. The model has a firm ceiling on its performance. Its close to maxing out performance on the classes it is focused on. It is really only trying to improve on those classes based on the design of CE loss and thus has to accept 4% accuracy on the other 5 leaving its top achievable performance very low and its already approaching it.

The focal loss results were an improvement on the CE results. While the top line number may not jump out, 43% is still a material improvement on 38%. More importantly, the distribution of accuracy is materially better. We see the results spread out much better among the classes. The worst performing class at 17.4% in these results would be the 6th best performing class in the CE results. This lends us to be optimistic that with deeper training and more fine tuning we can improve these numbers across the board. It also is in practice more useful. A random guess is 10% accurate. This model across the board is doing much better than that in every category. Compare that to our CE results. Here 5 of the classes are worse than a random guess. Is that useful? What if we care about those categories. To take the example to the extreme, medical data is often very unbalanced with few positive samples for diseases. A model can guess negative on every patient and get a score of 99.95% accurate. Its simply not a useful model though. Focal loss clearly allows us to do better.

(CONTINUED) Describe and explain your observation on the result:

Explanation should go into ***WHY*** *things work the way they do in the context of Machine Learning theory/intuition, along with justification for your experimentation methodology.* ***DO NOT*** *just describe the results, you should explain the reasoning behind your choices and what behavior you expected. Also, be cognizant of the best way to mindful and show the results that best emphasizes your key observations. If you need more than one slide to answer the question, you are free to create new slides.*

As for hyperparameter tuning. We went through our standard procedure for the CE Loss and found batch=64, learning rate=0.1, reg=0.0005, no warmup, momentum=0.9, and gamma=1.5. I imagined the gamma would help the model focus more on the samples it was getting wrong but that didn't have a huge impact on results. We found the biggest impact to our results came from adjusting the beta parameter. In the context of focal loss and class rebalancing, beta is often used for class reweighting when dealing with imbalanced datasets. Specifically, tuning the beta parameter can adjust the weighting scheme for each class based on its frequency in the dataset. Our best results came with a beta between 0.995 and 0.998. This helped the model weight heavier on classes with less samples. We found that tuning this was similar to a seesaw as we have to balance the emphasis across classes to improve performance. We could pick a given beta to maximize any particular class but the difficulty was balancing the emphasis to get the overall best result on the validation set. In this sense, the focal loss was superior to cross-entropy loss in both theory and practice as it allows us freedom to maneuver our model towards specific goals as well as achieve improved accuracy. Below is the focal loss calculation and how beta fits in to illustrate how it achieves the above goals.

$$\text{Weight}(c_i) = \frac{1-\beta}{1-\beta^{N_i}}$$

where N_i is the number of samples in class i, and β is a value between 0 and 1. As β approaches 1, more weight is placed on minority classes, rebalancing their influence during training.