Deep Learning

Problem Set 2

Authors:
Lonny Chen
L.Chen@students.hertie-school.org
Dominik Allen
Do.Allen@students.hertie-school.org
Aditi Joshi
A.Joshi@students.hertie-school.org

Oct 2025 – Deep Learning
Instructor: Dr. Lynn Kaack

1.1 Kernels

(a) Design a kernel

A 3×3 kernel we can design that leaves an image unchanged is the identity kernel:

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

When applied with stride = 1 and zero-padding of one pixel on each side, the output keeps the same dimensions as the input and each output pixel equals the corresponding input pixel.

(b) How does it modify

Matrix:

$$K = \frac{1}{16} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

This kernel modifies an input image by producing a blurring effect. Local details and high frequency edges are smoothed out. Depending on what stride it uses. It does this by taking the average of the 4×4 area it filters over and therefore a blurring effect.

(c) Custom kernel

Sobel vertical edge detector:

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

This kernel responds strongly to vertical intensity changes, producing high magnitude where vertical edges occur based on the weights assigned in the left and right columns.

1.2 Kernel interpretation

1.2.1

$$K = \begin{bmatrix} -0.89 & -0.92 & -0.90 \\ 0.01 & 0.02 & 0.005 \\ 0.90 & 0.92 & 0.89 \end{bmatrix}.$$

T his kernel detects and activates strongly for horizontal edges where intensity increases from top to bottom (dark above, bright below). The strong negative weights at the top and positive weights at the bottom make the output large and positive for downward brightness increases.

So high positive values would appear wherever a darker region lies directly above a brighter one.

1.2.2

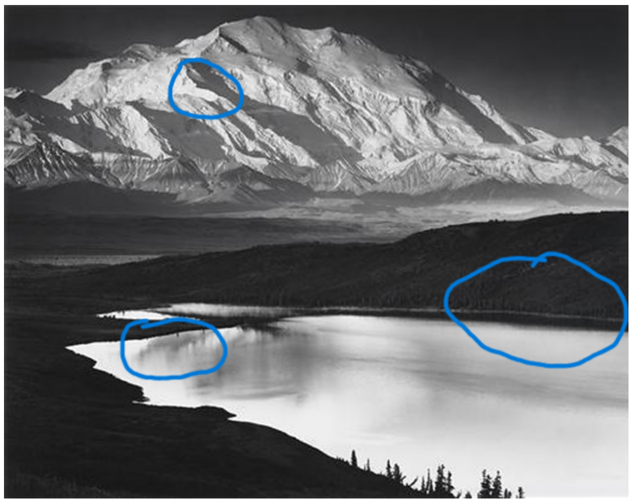

Figure 1: Areas in the image where the learned kernel produces high positive activation.

These are areas where the image transitions from darker regions above to brighter regions below, which corresponds to a strong horizontal boundary — exactly what this kernel is sensitive to.

Sources for 1.1 (a-c) and 1.2.1 "Kernels (Filters) in convolutional neural network" (https://www.geeksfor
learning/kernels-filters-in-convolutional-neural-network/) and "All about convolutions, kernels, features in CNN" Abhishek Jain (https://medium.com/@abhishekjainindore24/all-about-convolutions-kernels-features-in-cnn-c656616390a1)

1.3 Convolution and pooling

Given Input and Kernel

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0.5 & 1 & 0.5 \\ 0 & 1 & 0 & 1 \end{bmatrix} \qquad K = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 1 & 1 \\ 0 & 2 & 0 \end{bmatrix}$$

Task

Compute feature map after convolution and sigmoid activation, maintaining same dimensions as input.

Solution

Step 1: Modify Input for Same Dimensions

Problem: Without padding, output size =

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

Solution: Apply **zero padding = 1** (add one layer of zeros around the image)
Formula:

$$\text{Output size} = \frac{\text{Input size} - \text{Kernel size} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

$$\text{Output size} = \frac{4-3+2\times 1}{1} + 1 = 4 \times 4$$

Padded Input (6×6)

$$X_{\text{padded}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0.5 & 1 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 2: Flip the Kernel 180°

Flipped Kernel K' (rotated 180°):

$$K' = \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$

Step 3: Perform Convolution with Flipped Kernel

Position (0,0):

$$\text{Window} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Flipped Kernel } K' = \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$

Calculation: $(0 \cdot 0) + (0 \cdot 2) + (0 \cdot 0) + (0 \cdot 1) + (0 \cdot 1) + (0 \cdot 2) + (0 \cdot 0) + (0 \cdot 2) + (1 \cdot 0) = 0$

$$\text{Sum} = 0$$

Position (0,1):

$$\text{Window} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad \text{Flipped Kernel } K' = \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$

Calculation: $(0 \cdot 0) + (0 \cdot 2) + (0 \cdot 0) + (0 \cdot 1) + (0 \cdot 1) + (0 \cdot 2) + (0 \cdot 0) + (1 \cdot 2) + (0 \cdot 0) = 2$

$$\text{Sum} = 2$$

Similarly we calculated other values.

Step 4: Intermediate Result (Before Activation Function)

Complete Convolution Output:

$$Z = \begin{bmatrix} 0 & 2 & 0 & 2 \\ 4 & 2 & 5 & 2 \\ 2 & 7.5 & 2.5 & 5.5 \\ 4 & 2 & 5 & 2 \end{bmatrix}$$

Step 5: Apply Sigmoid Activation Function

Sigmoid Formula:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Calculations:

- $\sigma(0) = \frac{1}{1+e^{0}} = \mathbf{0.5000}$
- $\sigma(2) = \frac{1}{1+e^{-2}} = \mathbf{0.8808}$
- $\sigma(2.5) = \frac{1}{1+e^{-2.5}} = \mathbf{0.9241}$
- $\sigma(4) = \frac{1}{1+e^{-4}} = \mathbf{0.9820}$
- $\sigma(5) = \frac{1}{1+e^{-5}} = \mathbf{0.9933}$
- $\sigma(5.5) = \frac{1}{1+e^{-5.5}} = \mathbf{0.9959}$
- $\sigma(7.5) = \frac{1}{1+e^{-7.5}} = \mathbf{0.9994}$

Final Answer Modification Made:
Zero padding of 1 pixel added on all sides (from 4×4 to 6×6 padded input)

Flipped Kernel (180° rotation):

$$K' = \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$

Intermediate Result (After Convolution, Before Activation):

$$Z = \begin{bmatrix} 0 & 2 & 0 & 2 \\ 4 & 2 & 5 & 2 \\ 2 & 7.5 & 2.5 & 5.5 \\ 4 & 2 & 5 & 2 \end{bmatrix}$$

Final Feature Map (After Sigmoid Activation):

$$A = \begin{bmatrix} 0.5000 & 0.8808 & 0.5000 & 0.8808 \\ 0.9820 & 0.8808 & 0.9933 & 0.8808 \\ 0.8808 & 0.9994 & 0.9241 & 0.9959 \\ 0.9820 & 0.8808 & 0.9933 & 0.8808 \end{bmatrix}$$

(2) 2×2 max pooling

Step-by-Step Solution

Step 1: Divide Feature Map into 2×2 Regions

With stride = 2, we have **4 non-overlapping regions**:

Region 1 (Top-Left):

$$\begin{bmatrix} 0.5000 & 0.8808 \\ 0.9820 & 0.8808 \end{bmatrix}$$

Maximum value = 0.9820

Region 2 (Top-Right):

$$\begin{bmatrix} 0.5000 & 0.8808 \\ 0.9933 & 0.8808 \end{bmatrix}$$

Maximum value = 0.9933

Region 3 (Bottom-Left):

$$\begin{bmatrix} 0.8808 & 0.9994 \\ 0.9820 & 0.8808 \end{bmatrix}$$

Maximum value = 0.9994

Region 4 (Bottom-Right):

$$\begin{bmatrix} 0.9241 & 0.9959 \\ 0.9933 & 0.8808 \end{bmatrix}$$

Maximum value = 0.9959

Step 2: Construct Output

$$\begin{aligned} &\text{Position } (0,0) \leftarrow \text{Region 1: } 0.9820 \\ &\text{Position } (0,1) \leftarrow \text{Region 2: } 0.9933 \\ &\text{Position } (1,0) \leftarrow \text{Region 3: } 0.9994 \\ &\text{Position } (1,1) \leftarrow \text{Region 4: } 0.9959 \end{aligned}$$

Final Answer

Output After 2×2 Max Pooling (Stride 2):

$$\begin{bmatrix} 0.9820 & 0.9933 \\ 0.9994 & 0.9959 \end{bmatrix}$$

(3) Discussion

When we use the sigmoid activation function, one big problem that can happen during training is called saturation, which leads to vanishing gradients. The sigmoid squashes any input into a value between 0 and 1, but when the input is very positive or very negative, the output becomes almost flat or very close to 1 or 0. In those flat regions, the slope (or derivative) of the sigmoid is almost zero, which means that during backpropagation, the gradient passed to earlier layers becomes extremely small. When this keeps happening layer after layer, the model basically "stops learning" because the weights barely update. This is why deep nets with sigmoids can be slow to train or get stuck, and why RELUs are usually used.

1.4 Average pooling as convolution

How to represent 2×2 average pooling as a convolution:

Parameters

$$\text{Kernel size} = 2 \times 2, \quad K = \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \text{stride} = 2, \quad \text{padding} = 0.$$

With a 2×2 kernel, you can use a Kernel with values of $K = \frac{1}{4}$.

This kernel averages every 2×2 block and moves two pixels each step which is identical to 2×2 average pooling.

Example: 4×4 input

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}.$$

Applying a 2×2 average pooling (similar to the convolution above) gives us:

$$P = \begin{bmatrix} 3.5 & 5.5 \\ 11.5 & 13.5 \end{bmatrix}.$$

Therefore average pooling is a special case of convolution with fixed weights 1/4 and stride equal to the pool size.

1.5 Dimensions of CNN layers

Computation formula

For any convolution or pooling layer:

$$\text{output\_size} = \left\lfloor \frac{\text{input\_size} + 2P - K}{S} \right\rfloor + 1,$$

where K = kernel size, S = stride, P = padding.

Layer-by-layer sizes

Layer	Parameters	Output formula	Output size (C×H×W)
Input	–	–	(3, 227, 227)
Conv1	11×11, 96, S=4, P=0	$(227 - 11)/4 + 1 = 55$	(96, 55, 55)
Pool1	3×3, S=2	$(55 - 3)/2 + 1 = 27$	(96, 27, 27)
Conv2	5×5, 256, S=1, P=2	$(27 + 4 - 5)/1 + 1 = 27$	(256, 27, 27)
Pool2	3×3, S=2	$(27 - 3)/2 + 1 = 13$	(256, 13, 13)
Conv3	3×3, 384, S=1, P=1	$(13 + 2 - 3)/1 + 1 = 13$	(384, 13, 13)
Conv4	3×3, 384, S=1, P=1	same	(384, 13, 13)
Conv5	3×3, 256, S=1, P=1	same	(256, 13, 13)
Pool3	3×3, S=2	$(13 - 3)/2 + 1 = 6$	(256, 6, 6)
Flatten	–	$256 \times 6 \times 6 = 9216$	9216
FC1	4096	–	4096
FC2	4096	–	4096
FC3	1000	–	1000

Summary

$$\begin{aligned}
\text{Input: } & (3, 227, 227) \\
\text{Conv1: } & (96, 55, 55) \\
\text{Pool1: } & (96, 27, 27) \\
\text{Conv2: } & (256, 27, 27) \\
\text{Pool2: } & (256, 13, 13) \\
\text{Conv3: } & (384, 13, 13) \\
\text{Conv4: } & (384, 13, 13) \\
\text{Conv5: } & (256, 13, 13) \\
\text{Pool3: } & (256, 6, 6) \\
\text{Flatten: } & 9216 \\
\text{FC layers: } & 4096 \to 4096 \to 1000
\end{aligned}$$

0.1 source

https://setosa.io/ev/image-kernels/

"Calculating Parameters of Convolutional and Fully Connected Layers with Keras" by Yan Ding (https://dingyan89.medium.com/calculating-parameters-of-convolutional-and-fully-connected-layers-with-keras-186590df36c6) for computation of layer sizes and ChatGPT for doing the math and charts.