Grand Canyon University

Course: AIT-204 **Instructor:** Professor Artzi

Authors: Owen Lindsey & Tyler Friesen **Date:** February 15, 2026

CNN Convolution & Max Pooling — Interactive Walkthrough

Problem A — Constructing the Convolution Output Matrix M (General Form)

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \qquad K = \begin{bmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{bmatrix}$$

STEP 1

Determine the Output Dimensions

Before any computation begins, you need to know how large the result will be. The input matrix A has 3 rows and 3 columns, making it a 3×3 matrix. The kernel K has 2 rows and 2 columns, making it 2×2.

When performing convolution with *no padding* and a *stride of 1*, the output dimension is calculated with the formula:

$$\text{output size} = (n - f + 1) \times (n - f + 1)$$

Here, n is the dimension of the input and f is the dimension of the kernel. Substituting the values gives $(3 - 2 + 1) = 2$, so the output matrix M will be 2×2. This tells you that the kernel will land

in exactly four distinct positions as it slides across A.

> **OUTCOME**
>
> The output matrix M is 2×2, meaning there are four values to compute.

STEP 2

Identify the Receptive Fields

A **receptive field** is the specific region of the input that the kernel overlaps at a given position. Think of the kernel as a small window that slides over the input. At each stop, it "sees" a submatrix of A that matches its own size.

Since the kernel is 2×2, each receptive field is a 2×2 block pulled from A. The kernel starts at the top-left corner and slides one column to the right, then drops down one row and repeats. With stride 1, each move shifts the window by exactly one position.

> **How to read these diagrams:** The green-highlighted cells show which elements of A the kernel covers at that position. These are the values that will be paired with the kernel for element-wise multiplication.

Position (1,1) — Top Left

The kernel sits over the first two rows and first two columns of A.

Input A (receptive field highlighted)

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Kernel K

$k_{1,1}$	$k_{1,2}$
$k_{2,1}$	$k_{2,2}$

Position (1,2) — Top Right

The kernel slides one column to the right, now covering columns 2 and 3 of the first two rows.

Input A (receptive field highlighted)

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$

Kernel K

$k_{1,1}$	$k_{1,2}$

$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$k_{2,1}$	$k_{2,2}$

Position (2,1) — Bottom Left

The kernel moves back to column 1 and drops down one row, covering rows 2–3 and columns 1–2.

Input A (receptive field highlighted)

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Kernel K

$k_{1,1}$	$k_{1,2}$
$k_{2,1}$	$k_{2,2}$

Position (2,2) — Bottom Right

The kernel slides right once more, now covering the bottom-right 2×2 corner of A.

Input A (receptive field highlighted)

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Kernel K

$k_{1,1}$	$k_{1,2}$
$k_{2,1}$	$k_{2,2}$

> **OUTCOME**
>
> You now know exactly which four 2×2 submatrices of A will be used. Each one maps to a single entry in the output matrix M.

STEP 3

Compute Each Output Element

At every position, the operation is the same: take the receptive field and the kernel, multiply them *element-wise* (each cell in the receptive field times the corresponding cell in the kernel), then **sum all the products** into a single number. This is sometimes written using the Hadamard product symbol $\odot$:

$$M_{i,j} = \sum (\text{receptive field} \odot K)$$

Expanding this for each of the four positions gives you four equations. In each one, you pair the top-left of the receptive field with $k_{1,1}$, the top-right with $k_{1,2}$, the bottom-left with $k_{2,1}$, and the bottom-right with $k_{2,2}$.

$M_{1,1}$

$$M_{1,1} = (a_{1,1} \cdot k_{1,1}) + (a_{1,2} \cdot k_{1,2}) + (a_{2,1} \cdot k_{2,1}) + (a_{2,2} \cdot k_{2,2})$$

$M_{1,2}$

$$M_{1,2} = (a_{1,2} \cdot k_{1,1}) + (a_{1,3} \cdot k_{1,2}) + (a_{2,2} \cdot k_{2,1}) + (a_{2,3} \cdot k_{2,2})$$

$M_{2,1}$

$$M_{2,1} = (a_{2,1} \cdot k_{1,1}) + (a_{2,2} \cdot k_{1,2}) + (a_{3,1} \cdot k_{2,1}) + (a_{3,2} \cdot k_{2,2})$$

$M_{2,2}$

$$M_{2,2} = (a_{2,2} \cdot k_{1,1}) + (a_{2,3} \cdot k_{1,2}) + (a_{3,2} \cdot k_{2,1}) + (a_{3,3} \cdot k_{2,2})$$

> **OUTCOME**
>
> Each equation produces one scalar value. Together they fill every cell of the 2×2 output.

STEP 4

Assemble the Output Matrix

Place each computed value into its corresponding position to form M:

$$M = \begin{bmatrix} M_{1,1} & M_{1,2} \\ M_{2,1} & M_{2,2} \end{bmatrix}$$

OUTCOME

This is the complete convolution result for the general case. In Problem B you will plug in real numbers and carry out the arithmetic.

Problem B — Convolution with Specific Values + Max Pooling

$$A = \begin{bmatrix} 14 & 15 & 16 \\ 17 & 18 & 19 \\ 20 & 21 & 22 \end{bmatrix} \qquad K = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Interactive Kernel Slider — Watch the kernel slide across A

14	**15**	**16**
17	**18**	**19**
20	**21**	**22**

(18 × 1) + (19 × 2) + (21 × 3) + (22 × 4)

◂ Previous Position (2,2) — Bottom Right Next ▸

▶ Auto-Play

STEP 1

Confirm the Output Dimensions

The same formula from Problem A applies. A is 3×3 and K is 2×2, so the output M will again be $(3 - 2 + 1) \times (3 - 2 + 1) = 2 \times 2$.

> **OUTCOME**
>
> You will compute exactly four values to fill the 2×2 output.

STEP 2

Compute Each Convolution Element

Follow the same sliding-window process from Problem A, but now with actual numbers. At each kernel position, identify the 2×2 receptive field from A, multiply each element by its kernel counterpart, and add the four products.

$M_{1,1}$ — Top Left

The kernel overlaps the top-left 2×2 region of A:

Receptive Field			Kernel K	
14	**15**	⊙	**1**	**2**
17	**18**		**3**	**4**

$$M_{1,1} = (14 \times 1) + (15 \times 2) + (17 \times 3) + (18 \times 4) = ?$$

Intermediate products: $14 + 30 + 51 + 72$. Now add them together.

Your answer: 167 Check Hint ✓ Correct

$M_{1,2}$ — Top Right

The kernel slides one column right, now covering columns 2–3 of the top two rows:

Receptive Field		⊙	Kernel K	
15	**16**		**1**	**2**
18	**19**		**3**	**4**

$$M_{1,2} = (15 \times 1) + (16 \times 2) + (18 \times 3) + (19 \times 4) = ?$$

Intermediate products: $15 + 32 + 54 + 76$. Now add them together.

Your answer: 177 Check Hint ✓ Correct

$M_{2,1}$ — Bottom Left

The kernel returns to column 1 and drops down one row, covering rows 2–3:

Receptive Field		⊙	Kernel K	
17	**18**		**1**	**2**
20	**21**		**3**	**4**

$$M_{2,1} = (17 \times 1) + (18 \times 2) + (20 \times 3) + (21 \times 4) = ?$$

Intermediate products: $17 + 36 + 60 + 84$. Now add them together.

Your answer: 197 Check Hint ✓ Correct

$M_{2,2}$ — Bottom Right

The kernel slides right to its final position at the bottom-right corner of A:

Receptive Field		⊙	Kernel K	
18	**19**		**1**	**2**
21	**22**		**3**	**4**

$$M_{2,2} = (18 \times 1) + (19 \times 2) + (21 \times 3) + (22 \times 4) = ?$$

Intermediate products: $18 + 38 + 63 + 88$. Now add them together.

Your answer: 207 Check Hint ✓ Correct

OUTCOME

After performing the arithmetic for each position, you have four numerical values ready to place into the output matrix.

STEP 3

Assemble the Convolution Result

Place your four computed values into the 2×2 output matrix. Type each value into the corresponding cell:

Output M

167	177
197	207

Check Matrix ✓ All correct!

OUTCOME

M is now a complete 2×2 feature map — the result of convolving A with K.

STEP 4

Apply Max Pooling

Max pooling is a down-sampling operation. It takes a region of the feature map and reduces it to a single value by keeping only the **largest** element. The purpose is to retain the most prominent feature while reducing spatial dimensions.

In this problem, the pooling window covers the entire 2×2 matrix M. That means you look at all four values and select the maximum:

$$M_p = \max(M_{1,1},\ M_{1,2},\ M_{2,1},\ M_{2,2}) = ?$$

Your answer for M_p: 207 Check ✓ Correct

OUTCOME

The max pooling operation collapses the 2×2 matrix into a single scalar value M_p.

STEP 5

Transpose the Result

The problem asks for M_p^T, the transpose of the max-pooled result. Transposing a matrix swaps its rows and columns — the element at row i, column j moves to row j, column i.

However, since max pooling over the full 2×2 window produced a **scalar** (a 1×1 matrix), the transpose of a scalar is simply itself. There are no rows and columns to swap.

$$M_p^T = M_p$$

Your answer for M_p^T: 207 Check ✓ Correct

OUTCOME

> M_p^T equals M_p. The final answer is a single number — the largest value found in the convolution output.

Quick Reference

Concept	Description
Convolution Output Size	$(n - f + 1) \times (n - f + 1)$ where n = input dimension, f = kernel dimension. Assumes no padding and stride = 1.
Receptive Field	The subregion of the input that the kernel overlaps at a given position. Its size always matches the kernel.
Convolution Operation	Element-wise multiplication of the receptive field and the kernel, followed by summation of all products into one value.
Stride	The number of positions the kernel moves between each computation. Stride 1 means the window shifts by one row or column at a time.
Padding	Zeros added around the border of the input to control the output size. "Valid" (no padding) shrinks the output; "Same" padding preserves dimensions.
Max Pooling	A down-sampling technique that selects the maximum value from each pooling window, reducing spatial size while retaining dominant features.
Transpose	Swaps rows and columns of a matrix. For a scalar, the transpose is itself.

STEP 6

Fully Connected (Dense) Layer

After convolution and pooling, the CNN has extracted the most important *features* from the input. These features are then passed into a **fully connected (FC) layer**, which performs the final decision-making step of the network.

A fully connected layer works like a traditional neural network layer: every input value is connected to every neuron in the next layer using learned weights.

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

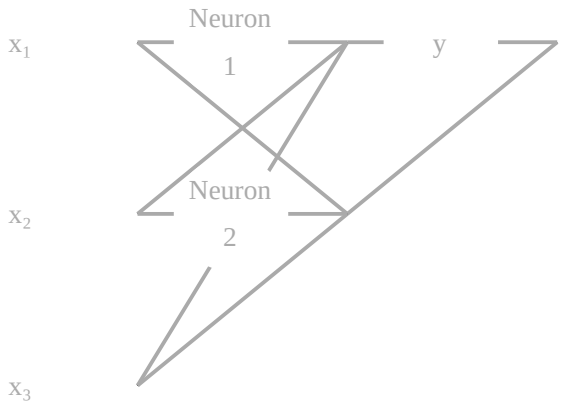
- $\mathbf{x}$: flattened input feature vector
- $\mathbf{W}$: weight matrix
- $\mathbf{b}$: bias vector
- $\mathbf{y}$: output scores (logits)

In this problem, max pooling produced a single scalar value M_p. Before entering a fully connected layer, this value would be *flattened* (converted into a 1D vector), even though it already contains only one element.

> **OUTCOME**
>
> The fully connected layer combines extracted features to produce the final prediction, such as a class score or probability.

VISUAL

Fully Connected Layer Diagram



Every input feature connects to every neuron in the next layer, allowing the network to combine learned features into a final prediction.

> **CNN PIPELINE SUMMARY**
>
> Convolutional layers extract local patterns from the input, pooling layers reduce spatial size while preserving dominant features, and fully connected layers combine these features to produce the final output. Together, these components allow CNNs to learn hierarchical representations of data.

STEP 7

Image Dimensions

A colored RGB image has **3 dimensions**: height, width, and color channels (Red, Green, Blue). A black and white (grayscale) image has only **2 dimensions**: height and width, with a single intensity channel.

In machine learning, understanding the number of dimensions is important because it affects how convolutional layers process the data.

RGB images have 3 color channels (R, G, B) while black and white images have only a single intensity channel.

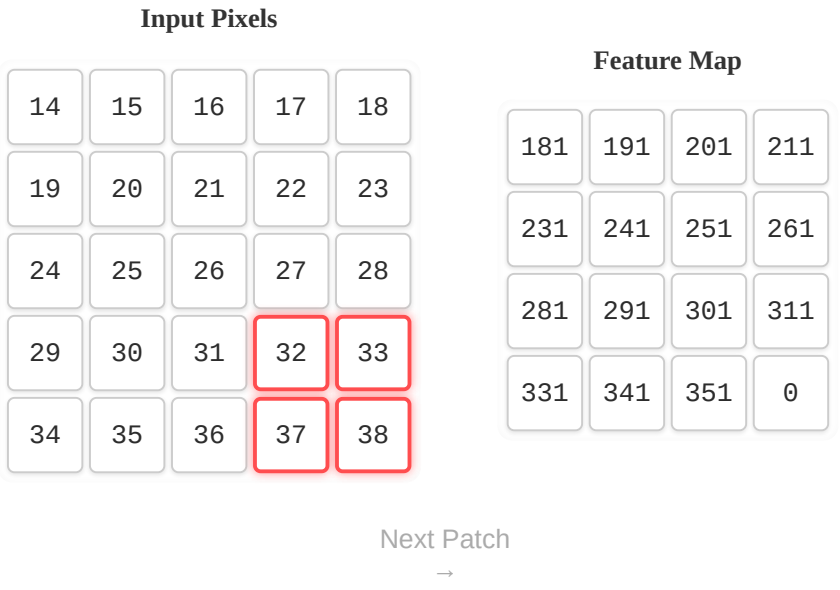
Steps in a Convolutional Neural Network (CNN)

CNNs are composed of layers that extract and process features from input images. Below we detail each step, along with diagrams to illustrate the flow.

1

Convolutional Layer

Convolutional layers apply learnable **filters** (kernels) across the input image. Each filter generates a **feature map** that highlights specific patterns such as edges or textures.



Each filter slides over the input image producing a feature map that captures a specific pattern.

2

Activation Function (ReLU)

Activation functions introduce nonlinearity. The ReLU function outputs 0 for negative inputs and outputs the input itself for positive values, allowing the network to learn complex patterns.

ReLU graph showing input vs output: negative inputs are zeroed, positive inputs pass through.

3

Pooling Layer (Max Pooling)

Pooling reduces the spatial dimensions of feature maps while preserving the strongest activations. In **max pooling**, the maximum value in each local region is selected.

4×4 Feature Map

3	1	2	4
5	6	1	2
0	2	7	3
1	4	2	6

2×2 Pooled Map

6	4
4	7

Next Pool
→

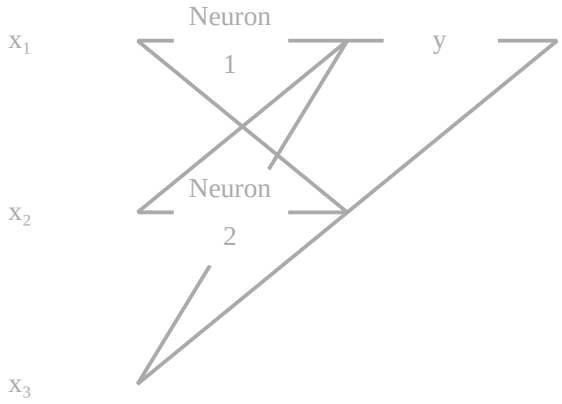
Each 2×2 region is replaced by its maximum value, reducing spatial size while keeping dominant features.

4

Fully Connected Layer

After convolution and pooling, the feature maps are **flattened** into a 1D vector and fed into the fully connected (dense) layer. In this layer, every input feature is connected to every neuron, allowing the network to combine and weigh the extracted features in complex ways. Each neuron computes a weighted sum of the inputs plus a

bias and applies an optional activation function, capturing high-level patterns that help the model make final decisions. Essentially, the fully connected layer interprets all the learned features together and transforms them into the final prediction scores or class probabilities.

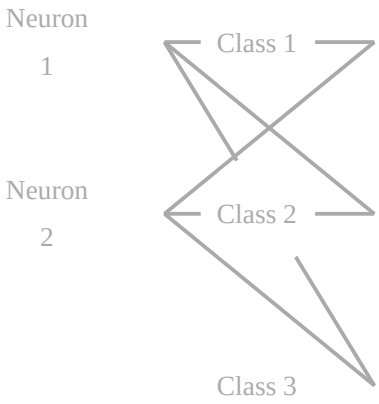


Fully connected layers combine extracted features to produce predictions.

5

Output Layer

The output layer takes the high-level features extracted and combined by the previous layers (especially the fully connected layer) and transforms them into the model's final predictions. Each neuron in this layer represents a possible class (in classification tasks) or a target value (in regression tasks). The network applies the learned weights and biases to compute a score for each class, which can then be normalized using a function like **softmax** to produce probabilities. The class with the highest probability is selected as the model's predicted output. Essentially, the output layer translates the abstract feature representations learned by the network into actionable, human-interpretable results.



Each neuron in the last hidden layer connects to every output node. The output layer produces the final model predictions (e.g., class probabilities).

ETHICS

Ethical Considerations in Convolutional Neural Networks

One major ethical concern when working with convolutional neural networks (CNNs) is **bias in image datasets**. CNNs learn patterns directly from the data they are trained on, so if a dataset overrepresents certain groups, environments, or visual features, the resulting model may perform unevenly across populations. For example, a facial recognition model trained primarily on images of one demographic group may exhibit significantly higher error rates for others. These biases can lead to unfair outcomes, particularly in high-stakes applications such as hiring, law enforcement, or medical diagnosis.

Another important concern is **data security and confidentiality**. Images used to train CNNs often contain sensitive personal or proprietary information. Improper handling of this data—such as inadequate anonymization, insecure storage, or unauthorized reuse—can violate privacy rights and legal protections. Ensuring secure data pipelines, limiting access, and complying with data protection regulations are essential responsibilities when developing CNN-based systems.

The **lack of interpretability and transparency** in CNN decision-making is also ethically significant. CNNs are often described as “black box” models because their internal representations are difficult to understand. This opacity makes it challenging to explain why a particular prediction was made, which can undermine trust and accountability. In domains such as healthcare or criminal justice, clear explanations of model behavior are often necessary to support human oversight and informed decision-making.

Responsible deployment of CNNs is especially critical in applications such as surveillance systems and autonomous technologies. These systems can influence real-world actions that affect safety, civil liberties, and human rights. Deploying CNNs without proper oversight, clearly defined use cases, and safeguards against misuse can result in harmful or unethical outcomes.

Finally, **errors and inaccuracies in CNN predictions** can have serious consequences. Misclassifications in medical imaging may delay or prevent appropriate treatment, while errors in autonomous systems could lead to physical harm or property damage. Ethical use of CNNs therefore requires extensive testing, ongoing

monitoring, and an understanding of the limitations of the model. CNN outputs should support, rather than replace, critical human judgment.