

King of Dumbs

Level: ♠ Technical

How does a computer turn a photo into numbers?

A photo is millions of numbers, and almost everything AI does with it is multiply and add. Vectors, the dot product and matrices at three levels, with two widgets.

English Published October 17, 2026



Scan to open the interactive lesson

<https://kingofdumbs.com/en/lessons/vectors-and-matrices>

What you'll learn

- A photo is a list of numbers: vectors and dimension
- The dot product: multiply and add, and how much two arrows point the same way
- A matrix moves the plane: its columns, the area and the dimensions that are left
- Multiplying matrices: chaining changes, why order matters and what a neural network does

Symbols you'll meet

E_a subscript: the label underneath (♦♠) · Σ sum: add them all up (♠) · x_i index i: item number i (♠) · $\approx$ approximately equal (♦) · $\leq \geq$ less or equal / greater or equal (♠) · $|x|$ absolute value: drop the sign (♠) · · dot: multiply (or dot product) (♦♠) · σ sigma: depends on the lesson (♠) · θ theta: an angle (almost always) (♠) · λ lambda: a stretching number (♠) · $\|x\|$ norm: the length of the arrow (♠) · x^T transpose: lay the column down (♠) · $\in$ belongs to (♠) · $\mathbb{R} \mathbb{Z}$ sets of numbers (♠)

You take a photo of your cat and, if you search for “cat” in your phone’s gallery, it often finds it. But a computer doesn’t see cats: all it knows how to do is add and multiply numbers, very fast.

This lesson’s dumb question is: **how does a computer turn a photo into numbers?** And while we’re at it, what does it do with them? The answer fits into three ideas that sound like an exam, vector, dot product and matrix, and a single sum: multiply and add.

Pick your card: ♣ if you’re starting from zero, ♦ if a simple formula doesn’t scare you, ♠ if you want the notation. You can switch at any time.

A photo is a list of numbers

♦ → ♠ From intermediate to technical

In ♦ we said	In ♠ we write
vector	$x \in \mathbb{R}^n$, a list of n real numbers
components	$x = (x_1, \dots, x_n)$
length	Euclidean norm $\ x\ = \sqrt{(\sum_i x_i^2)}$
dimension	$\dim \mathbb{R}^n = n$: the vectors in a basis, $e_1, \dots, e_n$
channel	C ; in RGB, $C = 3$, 8-bit integers in $\{0, \dots, 255\}$
height $\times$ width $\times 3$	tensor $H \times W \times C$ (HWC); torchvision uses $C \times H \times W$
it's all put in a row	flatten: $x \in \mathbb{R}^n$ with $n = H \cdot W \cdot C$

$$x \in \mathbb{R}^n, \quad x = (x_1, \dots, x_n)$$

$$\|x\| = \sqrt{(\sum_i x_i^2)}$$

$$\text{img} \in \{0, \dots, 255\}^{(H \times W \times C)}, \quad C = 3$$

$$\text{ToTensor}: H \times W \times C \rightarrow C \times H \times W, \quad /255$$

$$n = H \cdot W \cdot C = 4000 \cdot 3000 \cdot 3 = 3.6 \times 10^7$$

$$\dim \mathbb{R}^n = n \quad (\text{basis } e_1, \dots, e_n)$$

An 8-bit RGB image is a tensor of integers in $\{0, \dots, 255\}^{(H \times W \times C)}$ with $C = 3$: Pillow 13.0 describes the mode «RGB (3x8-bit pixels, true color)» and says that «an 8-bit pixel has a range of 0-255». torchvision 0.29 converts it with `ToTensor`, from $(H \times W \times C)$ in $[0, 255]$ to a floating-point tensor of shape $(C \times H \times W)$ in $[0.0, 1.0]$. Flattened, it's a vector in $\mathbb{R}^n$ with $n = H \cdot W \cdot C$; for 4000×3000 , $n = 3.6 \times 10^7$.

The dimension of a space is the number of vectors in any basis (Austin, *Understanding Linear Algebra*, def. 3.5.6). The standard basis $e_1, \dots, e_n$ has n vectors, so $\dim \mathbb{R}^n = n$: the number of components. The norm $\|x\| = \sqrt{(\sum_i x_i^2)}$ is Pythagoras in n dimensions.



The dot product: multiply and add

♦ → ♠ From intermediate to technical

In ♦ we said	In ♠ we write
dot product	$a \cdot b = \sum_i a_i b_i$ (also $\langle a, b \rangle$ or $a^T b$)
36 million multiplications	cost $O(n)$: n products and $n - 1$ sums
length of $a \times$ length of $b \times$ cosine of the angle	$a \cdot b = \ a\ \ b\ \cos \theta$, with $\theta \in [0, \pi]$
The cosine is 1 if they point the same way, 0 if they make a right angle and -1 if they're opposite	$-1 \leq \cos \theta \leq 1$, that is, $ a \cdot b \leq \ a\ \ b\ $ (Cauchy–Schwarz)
perpendicular	orthogonal: $a \cdot b = 0$ ($a \perp b$)
cosine similarity	$\cos \theta = a \cdot b / (\ a\ \ b\)$
the arrow gets longer but doesn't turn	$\cos(x, \lambda x) = 1$ for every $\lambda > 0$

$$a \cdot b = \sum_i a_i b_i = \|a\| \|b\| \cos \theta$$

$$\cos \theta = a \cdot b / (\|a\| \|b\|) \in [-1, 1]$$

$$a \perp b \Leftrightarrow a \cdot b = 0$$

$$|a \cdot b| \leq \|a\| \|b\| \quad (\text{Cauchy–Schwarz})$$

$$\cos(x, 1.2 \cdot x) = 1: \text{brighter, same direction}$$

$$\text{cost: } n \text{ products, } n - 1 \text{ sums: } O(n)$$

In $\mathbb{R}^n$, $a \cdot b = \sum_i a_i b_i$. OpenStax (*Calculus Volume 3*, 2.3) proves with the law of cosines that $a \cdot b = \|a\| \|b\| \cos \theta$ (theorem 2.4), and that two nonzero vectors are orthogonal if and only if $a \cdot b = 0$ (theorem 2.5). In more than three dimensions it goes the other way round: the angle is defined by $\cos \theta = a \cdot b / (\|a\| \|b\|)$, and the Cauchy–Schwarz inequality, $|a \cdot b| \leq \|a\| \|b\|$, ensures the ratio falls in $[-1, 1]$.

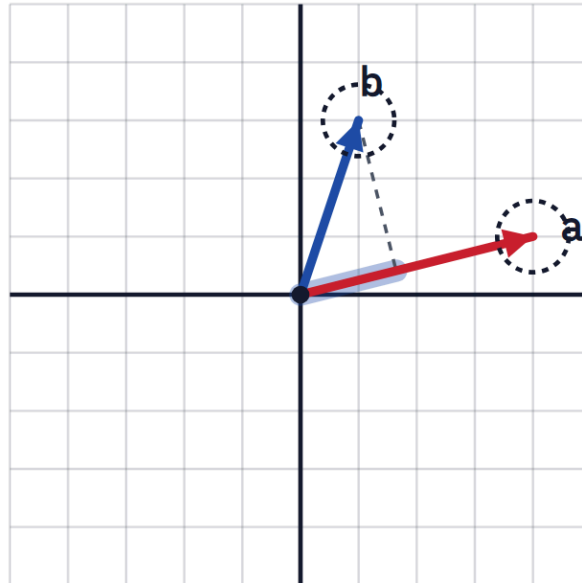
PyTorch 2.14 computes cosine similarity with `torch.nn.functional.cosine_similarity`, which divides by $\max(\|x_1\| \|x_2\|, \epsilon)$ to avoid dividing by zero. Since $\cos(x, \lambda x) = 1$ for $\lambda > 0$, cosine similarity can't tell a photo from a lighter or darker one that only scales its values. The cost is linear: n products and $n - 1$ sums.

The King of Dumbs says:

Multiply and add. That's all there is. With that I compare prices, arrows and photos, and still have time left for a nap.

Try it: two arrows

Drag the tips of the arrows, or move them with the sliders or the arrow keys. Find a position where the dot product is 0: the arrows will end up at a right angle. The blue band is the shadow of b on a: it gets longer the more they point the same way.



a, across	<input type="range" value="4"/>	4	a, up	<input type="range" value="1"/>	1
b, across	<input type="range" value="1"/>	1	b, up	<input type="range" value="3"/>	3

Dot product $a \cdot b$ $4 \cdot 1 + 1 \cdot 3 = 7$

Lengths $\|a\| = 4.12 \cdot \|b\| = 3.16$

Angle 57.5°

Cosine (how alike the directions are) 0.54

$a = (4, 1)$, $b = (1, 3)$, $a \cdot b = 7$. They point the same way: the dot product is positive.



Interactive widget. Try it on the web: <https://kingofdumbs.com/en/lessons/vectors-and-matrices>

A matrix moves the plane

♦ → ♠ From intermediate to technical

In ♦ we said	In ♠ we write
matrix	$A \in \mathbb{R}^{(m \times n)}$, entries a_{ij} (row i , column j)
each row	$(Ax)_i = \sum_j a_{ij}x_j = (\text{row } i \text{ of } A) \cdot x$
the first column is where $(1, 0)$ goes and the second, where $(0, 1)$ goes	$A = [Ae_1 \ Ae_2]$: column j is $T(e_j)$
any point is a mix of those two	linearity: $A(c_1e_1 + c_2e_2) = c_1Ae_1 + c_2Ae_2$
slants	shear, for example $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$
$ad - bc$	$\det A$, the signed area of the parallelogram of the columns
squashes	$\det A = 0$: A isn't invertible; the rank drops and the image is a line
as in a mirror	$\det A < 0$: the orientation flips
rotate	rotation $R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$
which point of the original it comes from	uses A^{-1} : it walks the output and looks up the input
(plus a shift)	affine transformation $x \mapsto Ax + t$

$$(Ax)_i = \sum_j a_{ij} x_j$$

$$A = [Ae_1 \ Ae_2], \quad A(cu + v) = cAu + Av$$

$$R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

$\det A = ad - bc$: signed area

$\det A = 0 \Leftrightarrow$ not invertible $\Leftrightarrow$ rank < 2

$\begin{pmatrix} 1 & -1 \\ 2 & -2 \end{pmatrix}$: $\det 0$, rank 1, a line

An $m \times n$ matrix A defines $T(x) = Ax$ from $\mathbb{R}^n$ to $\mathbb{R}^m$, with $(Ax)_i = \sum_j a_{ij}x_j$. Its columns are the images of the basis: $A = [T(e_1) \ \cdots \ T(e_n)]$ (Austin, prop. 2.5.8). Since T is linear, $T(\sum_j c_j e_j) = \sum_j c_j T(e_j)$, and the columns tell you everything.

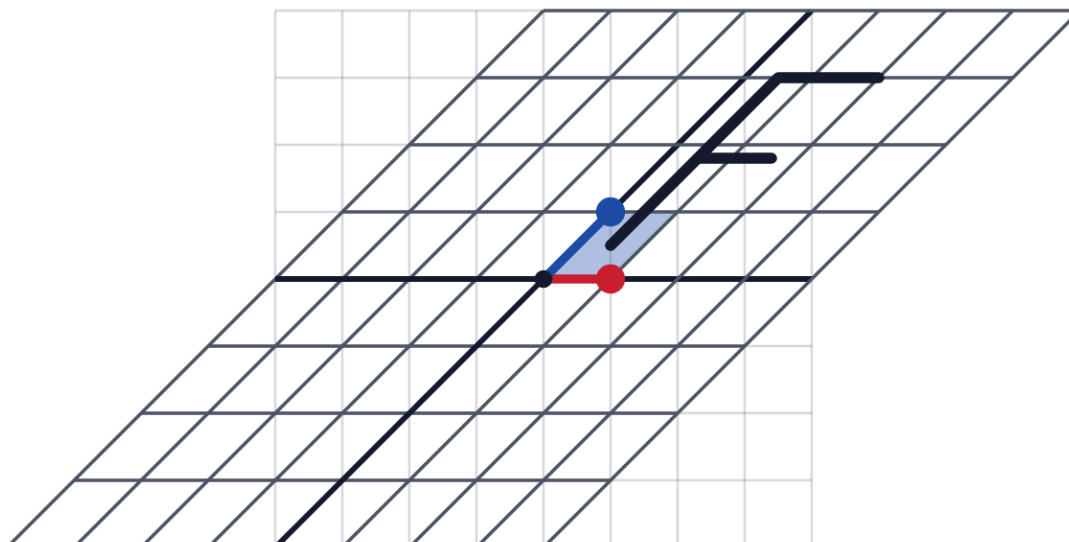
In 2×2 , the rotation by an angle θ is $(\cos \theta \ -\sin \theta / \sin \theta \ \cos \theta)$, and $(1 \ 1 / 0 \ 1)$ is a shear (Austin, 2.6). $\det A = ad - bc$ is the signed area of the parallelogram of the columns (3.4), and A is invertible if and only if $\det A \neq 0$ (prop. 3.4.12). With $\det A = 0$ the rank drops: the image $\text{Col}(A)$ has dimension rank A (prop. 3.5.12), and with $(1 \ -1 / 2 \ -2)$ it's a line.

Pillow 13.0 (`ImageTransform.AffineTransform`) takes «the first two rows from the inverse of an affine transform matrix»: for each output pixel (x, y) , it reads the input at $(ax + by + c, dx + ey + f)$. It uses the inverse because it walks the output photo, and the shift (c, f) makes the transformation affine, not linear.



Try it: bend the grid

Move the four numbers of the matrix or try the buttons. The red arrow is the first column, where $(1, 0)$ goes; the blue one, the second. Find a matrix that squashes the grid onto a line: the area will become 0. And see what happens to the F with the mirror.



Try one

Do nothing

Stretch

Turn 90°

Slant

Mirror

Squash

a (top left)



1

b (top right)



1

c (bottom left)



0

d (bottom right)



1

(1, 0) goes to

(1, 0)

(0, 1) goes to

(1, 1)

The area is multiplied by $(ad - bc)$

1

Dimensions left

2: the plane is still a plane

Matrix $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$: the area is multiplied by 1 and 2 dimensions are left.



Interactive widget. Try it on the web: <https://kingofdumbs.com/en/lessons/vectors-and-matrices>



Multiplying matrices: order matters

♦ → ♠ From intermediate to technical

In ♦ we said	In ♠ we write
multiply two matrices	$(AB)_{ij} = \sum_k a_{ik}b_{kj}$
a row of the first with a column of the second	(row i of A) $\cdot$ (column j of B)
an $m \times k$ one times a $k \times n$ one gives an $m \times n$ one	$(m \times k)(k \times n) \rightarrow m \times n$: the inner dimensions match
AB	composition: $A(Bx) = (AB)x$; if $T \mapsto A$ and $S \mapsto B$, $S \circ T \mapsto BA$
AB usually isn't equal to BA	not commutative; associative: $(AB)C = A(BC)$
layer	linear layer (<code>nn.Linear</code>): $y = xA^T + b$
weights	$A \in \mathbb{R}^{(\text{out} \times \text{in})}$, learned; b , the bias
10 million multiplications	$\text{out} \cdot \text{in} = 10^3 \cdot 10^4 = 10^7$ fused multiply-adds (FMA) per photo
a step that isn't a multiplication	nonlinear activation σ ; without it, $A_2(A_1x) = (A_2A_1)x$ is a single layer
lots of photos at once	batch of N : one GEMM of $(M \times K)(K \times N)$, $2 \cdot M \cdot N \cdot K$ FLOP

$$(AB)_{ij} = \sum_k a_{ik} b_{kj}$$

$$(m \times k)(k \times n) \rightarrow m \times n; \quad AB \neq BA$$

$$x \mapsto A(Bx) = (AB)x$$

$$\text{cost: } m \cdot n \cdot k \text{ FMA} = 2 \cdot m \cdot n \cdot k \text{ FLOP}$$

$$\text{nn.Linear: } y = xA^T + b, \quad A: (\text{out}, \text{in})$$

$$2 \cdot 1,000 \cdot 64 \cdot 10,000 = 1.28 \times 10^9 \text{ FLOP}$$

$(AB)_{ij} = \sum_k a_{ik}b_{kj}$, defined when the inner dimensions match (OpenStax, *College Algebra 2e*, 7.5). It's associative and distributive, but not commutative. If $T(x) = Ax$ and $S(x) = Bx$, $S \circ T$ has matrix BA (Austin, prop. 2.5.11): the one applied first goes on the right.

Each of the $m \cdot n$ entries is a dot product of k terms: $m \cdot n \cdot k$ fused multiply-adds (FMA), that is, $2 \cdot m \cdot n \cdot k$ FLOP. NVIDIA counts it the same way in its GEMM guide, where each entry is «a dot-product of K -element vectors». For the $1,000 \times 10,000$ layer with a batch of 64 photos: $2 \cdot 1,000 \cdot 64 \cdot 10,000 = 1.28 \times 10^9$ FLOP (our own calculation).

PyTorch 2.14 writes the linear layer as $y = xA^T + b$, with one example per row and the weight A of shape (out_features, in_features). Because of the bias b it's affine, not linear. NVIDIA puts the weights in A and the activations in B , one example per column: it's the same sum transposed. Without a nonlinear σ between layers, associativity would make several layers in a row collapse into one.

The King of Dumbs says:

I don't see cats: I see 36 million numbers. The hard part is knowing which matrices to multiply them by.



Final challenge

1. A colour photo of 100×100 pixels. How many numbers does the computer store when it opens it?

- a. 100
- b. 10,000, one per pixel
- c. 30,000: three per pixel, red, green and blue

2. Two arrows make a right angle. What's their dot product?

- a. 1
- b. 0
- c. It depends on how long they are

3. You rotate a photo by 90° and then stretch it sideways. Is that the same as stretching it first and rotating it afterwards?

- a. Yes: multiplying gives the same in any order
- b. No: with matrices, order matters
- c. Only if the photo is square



In three sentences

1. To a computer, a photo is a list of numbers (three per pixel, from 0 to 255): a vector, whose dimension is how many numbers it has.
2. The dot product multiplies two lists term by term and adds: it gives a total, as in the fruit seller's sum, and says how much two arrows point the same way (0 if they're perpendicular).
3. A matrix is a table that moves every point of the plane, and multiplying matrices chains those changes, with the order mattering: it's the sum a neural network does millions of times with each photo.



Sources

- Vectors, length and the dot product: OpenStax, *Calculus Volume 3*, [2.1](#) and [2.3](#) (the fruit seller, theorems 2.4 and 2.5). Matrices and their product: OpenStax, *College Algebra 2e*, [7.5](#).
- Transformations, determinant and dimension: D. Austin, *Understanding Linear Algebra*, [2.2](#), [2.5](#), [2.6](#), [3.4](#) and [3.5](#).
- Pixels and image transforms: Pillow 13.0, [Concepts](#) and [ImageTransform](#). From photo to tensor: torchvision 0.29, [ToTensor](#).
- Neural networks: PyTorch 2.14, [nn.Linear](#) and [cosine_similarity](#). NVIDIA, [Matrix Multiplication Background](#) and [Linear/Fully-Connected Layers](#).

How this lesson was made: researched and written with help from Claude from the sources above, consulted on 17 October 2026. Figures without their own link are calculations from their definitions. The examples (the 4,000 × 3,000 photo, the two shops, the layer with 1,000 outputs) are made up. The drawings are our own.

Answers

1. **c)** 30,000: three per pixel, red, green and blue. That's it. $100 \times 100 \times 3$: a vector of dimension 30,000.
2. **b)** 0. Right. Perpendicular, dot product 0, however long they are.
3. **b)** No: with matrices, order matters. Exactly. Like the sock and the shoe.

A King of Dumbs lesson, kingofdumbs.com. Offline version; the original, with widgets you can play with, is at:
<https://kingofdumbs.com/en/lessons/vectors-and-matrices>

Updated on October 17, 2026.