

Machines That See

Teaching a machine to see — convolutional neural networks

Day 06 · Week 2

Unlocks  Photo Detective

1 Mission briefing

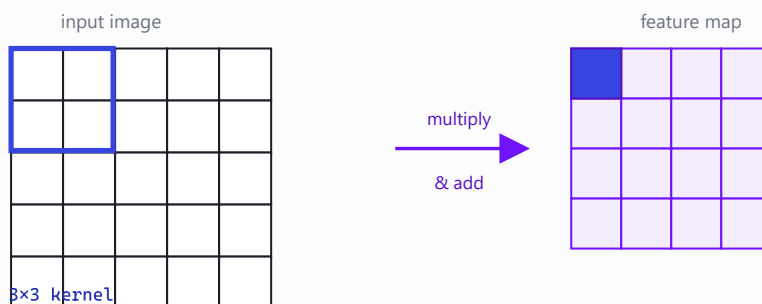
Our lab cameras are technically blind — they record **pixels**, not *things*. Yesterday you compared a flattened-image MLP with a CNN for handwriting, but real photos have edges, textures, and objects that can sit anywhere in the frame. Today you teach a machine to actually **see** — and turn it loose on 60,000 real photos. *Previously:* you unlocked  Digit Reader, a neural net that reads handwritten digits.

- ☐ See why **flattening** destroys a photo's structure
- ☐ Slide a 3×3 **kernel** to find edges, blur, and sharpen
- ☐ Stack convolutions: edges → shapes → objects
- ☐ Train a **CNN** on CIFAR-10 and read accuracy **honestly**
- ☐ Time a step on GPU vs CPU
- ☐ See that a convolution is really a matrix multiply

2 Field guide the concepts

A photo is a grid, not a row. Neighboring pixels belong together — they form an edge, a whisker, a wheel. Flattening removes explicit spatial axes, while preserving every pixel value. A CNN keeps the grid and studies small neighborhoods at a time.

Convolution is a sliding stencil. Take a tiny 3×3 grid of numbers (a **kernel**), slide it across the image, and at each stop multiply the nine pixels underneath and add them up. One kernel finds edges, another blurs, another sharpens. Each kernel's output is a **feature map**, and stacking layers lets them compose: edges become shapes, shapes become objects.



A 3×3 kernel slides across the image; at each stop it multiplies the nine pixels underneath and adds them into one number of the feature map.

Read accuracy honestly. **CIFAR-10** is 60,000 tiny 32×32 color photos in 10 classes. Around **70%** can feel low — until you remember random guessing is 10%. Compare this with the 10% random baseline, while examining class-specific errors. A real scientist reports the true number, not the flattering one.

 **Matrix Watch** — a convolution is secretly one big matrix multiply: unroll every 3×3 patch into a row (the `im2col` trick) and the whole layer becomes $C = A @ B$. High-performance computing makes that multiply fast.

A precise reading

Flattening preserves pixel values in a fixed order; it removes the explicit spatial structure from the layer interface. A CNN builds in local connectivity and shared weights. Day 5 trained both an MLP and a CNN. CIFAR-10 contains 50,000 training and 10,000 test images. Report the measured test score and failure cases rather than assuming a universal human baseline.

3 Lab missions check each off in the notebook

1 The vertical-edge kernel

Hand-build a 3×3 edge kernel and slide it with `F.conv2d`.

Expect: vertical outlines light up while flat areas stay dark.

2 A blur kernel

Fill a kernel that averages the 3×3 neighborhood.

Expect: the same photo comes back soft and smeared.

3 Assemble the CNN

Stack `Conv2d` + `ReLU` + `MaxPool` blocks, then `Flatten` into a linear head.

Expect: the feature-map size shrink $32 \rightarrow 16 \rightarrow 8$ as you pool.

4 Count the cost

Add up the model's trainable parameters.

Expect: a few hundred thousand weights — big, but trainable in class.

5 Build the confusion matrix

Plot `confusion_matrix` of predictions vs truth on the test set.

Expect: a bright diagonal, with cat and dog leaking into each other.

6 Look at a different class

Pick another class and read its row of mistakes.

Expect: the confusions make visual sense (deer ↔ horse, car ↔ truck).

7 Time a training step, GPU vs CPU

Clock one forward+backward step on each device.

Expect: the GPU finish many times faster — a preview of high-performance computing.

4 Cheat sheet convolutions & CNNs

<code>F.conv2d(x, kernel)</code>	Slide a kernel over x by hand (no learning).
<code>nn.Conv2d(3, 32, 3, padding=1)</code>	Learnable conv: $3 \rightarrow 32$ channels, 3×3 , keeps size.
<code>nn.MaxPool2d(2)</code>	Halve height & width, keep the strongest signal.
<code>nn.Flatten()</code>	Grid $\rightarrow$ row — only after the conv layers.
<code>nn.Dropout(0.3)</code>	Randomly switch off 30% of activations while training.
<code>Normalize((0.5,)*3, (0.5,)*3)</code>	Center each color channel to the range $[-1, 1]$.
<code>confusion_matrix(y_true, y_pred)</code>	Table of what got confused with what.
<code>torch.jit.trace(model, ex).save(path)</code>	Freeze the trained model into one portable file.

5 Unlock your Studio module

Photo Detective

Train the CNN, then trace and save it. The unlock cell must write exactly:

```
ai_studio/models/photo_detective.pt
```

```
ai_studio/models/photo_detective_classes.json
```

A new tab lights up — upload a photo and watch your detective investigate:

```
$ python ai_studio/app.py
```

6 Mini-glossary

convolution

Sliding a small kernel over an image, multiply-and-add at each spot.

feature map

The output image a single kernel produces.

stride / padding

How far the kernel jumps · a border that preserves size.

normalization

Rescaling pixel values so training stays stable.

kernel / filter

The little grid of weights that does the sliding.

pooling

Shrinking a feature map by keeping its strongest values.

dropout

Randomly switching off neurons in training to fight overfitting.

7 Show & tell

1. Show one photo your Detective nails and one it flubs. What's different about them?
2. Your accuracy vs the 10% random baseline — is 70% good? Defend your answer.
3. Which two classes does your confusion matrix mix up most, and does that make sense?