Lecture 12 · Part III · Lecture notes

Convolutional networks

Dataset: Flowers102

How to use these notes

The primary text for this lecture is Géron, Chapter 12. These notes are not a replacement for it and do not repeat it: they are the lecture's own argument written out at length — the derivation done slowly, the numbers with the runs that produced them, and the reasoning between one slide and the next that a deck can only gesture at. Read the chapter for the method; read these for why the lecture makes the choices it does. Every figure quoted here is printed by this lecture's notebook, and the course checks that mechanically rather than trusting it.

Contents

1 What breaks when the image gets bigger **2**

1.1 The brief 2

2 Derivation: weight sharing and equivariance **3**

2.1 The parameter count, on the same image 3

2.2 Equivariance, precisely 3

2.3 Equivariance is not invariance 3

3 Convolution, pooling, and an architecture **4**

4 What it learned **5**

5 The notebook **5**

6 Exercises **6**

Summary **6**

1 What breaks when the image gets bigger

Three lectures of neural networks, on flattened greyscale images. The obvious next step is to resize a colour photograph to 128 by 128, flatten it, and feed it to the multilayer perceptron: 49,152 inputs, and a fully connected layer producing 32 values at every pixel position needs 25,769,803,776 weights — 96 GB of float32 for one layer. Every one has to be learned from data, and none of them knows that two adjacent pixels are related.

Today's method fixes both problems at once, and the derivation says exactly how much it saves.

1.1 The brief

A horticultural wholesaler receives mixed pallets of cut stems and must sort them by species before the auction floor. One overhead camera per lane, one photograph per stem, and a human inspector who is the bottleneck.

In their words: *"Tell us which of our species it is. If you cannot tell, say so and we will look at it ourselves — but do not tell us the wrong one."* Two requirements hiding in one sentence: a classifier over a fixed list, and a usable notion of confidence. We build the first today; the second waits until the numbers are good enough to be worth calibrating. A softmax always returns a class, and its largest probability is not a confidence anyone has calibrated — deliver it as one and you have answered a different requirement while appearing to meet this one.

The split shipped with the dataset is 1,020 train, 1,020 val and 6,149 test — six times as many test images as training images, and the training split is balanced at exactly ten per class where the test split is not.

The standing constraint of applied vision

Images are cheap and labels are not. Somebody had to look at every photograph and name the species. Every application in this part of the course has more pixels than it can use and fewer labels than it wants — so a method that needs thousands of labelled examples per class is not a method you can deploy here.

What is hard here is position (the flower is not centred and not at a fixed scale), background, between-class similarity (several species separated by petal count and little else), and within-class variation. *Only the first has a structural answer*, and it is the one this chapter is about.

Every photograph is resized to 128 by 128 and standardised per colour channel. The per-channel means and standard deviations are computed from the *training split only*. They are fitted, exactly like Lecture 2's scaler, even though they look like a property of the dataset; the validation and test images are normalised with the training split's numbers, not with their own. Computing them over all 8,189 images instead is a leak, and a quiet one: the statistics come out close, nothing warns you, and the score cannot tell you which case you are in — so the rule is procedural. Split first.

2 Derivation: weight sharing and equivariance

You replace a matrix multiplication with a convolution. What exactly did you buy? Two answers: a parameter count, and a symmetry.

2.1 The parameter count, on the same image

Fix the input and output shapes so the comparison is between two ways of computing the same sized thing: input 3 by 128 by 128, which is 49,152 values, and output 32 by 128 by 128, which is 524,288.

A dense layer connects every output to every input,

$$\#\text{weights} = n_{\text{in}} n_{\text{out}} = (3HW)(32HW) = 96H^2W^2,$$

which is 25,769,803,776 weights — 96 GB of float32 for one layer, before any gradient or optimiser state.

A convolution connects every output to a k by k window of every input channel and *uses the same weights at every position*,

$$\#\text{weights} = C_{\text{out}} C_{\text{in}} k^2 = 32 \cdot 3 \cdot 7 \cdot 7,$$

which is 4,704. Note that H and W do not appear: the image size is genuinely absent from the count. The ratio is H^2W^2/k^2, which here is 5,478,275 times fewer weights for the same output shape — and with the 3 by 3 filters used everywhere else in the network, larger again by $(7/3)^2$, because the dense count never contained k.

2.2 Equivariance, precisely

Sharing the weights assumes that a pattern means the same thing wherever it appears. Let $T_{\mathbf{v}}$ shift an image by $\mathbf{v}$, so $(T_{\mathbf{v}}x)(\mathbf{p}) = x(\mathbf{p} - \mathbf{v})$. A map f is **equivariant** to translation when $f(T_{\mathbf{v}}x) = T_{\mathbf{v}} f(x)$ for every $\mathbf{v}$: shift then compute, or compute then shift, is the same answer.

A convolution has this property because its weights do not depend on the position. The equality holds exactly away from the border: at the border, zero padding invents input that was not there, so a shift moves real content into invented content.

2.3 Equivariance is not invariance

A map is **invariant** when $g(T_{\mathbf{v}}x) = g(x)$: under a shift the output does not change, and the position is not kept. Invariance is strictly the stronger and strictly the lossier of the two, and it is what you get by discarding the equivariant structure.

Pooling manufactures it. A 2 by 2 max pool is invariant to any shift that keeps the maximum inside its own window; stack four stages and the tolerated shift grows to about eight pixels in the last feature map; a global pool over the whole map is invariant to every shift, because there is no position left to be sensitive to. The invariance is approximate and it is *built*, one factor of two at a time — nothing in the architecture declares it.

Classification wants invariance: "this is a sunflower" is true wherever the sunflower is, so a representation that still carries the position is carrying a nuisance variable, and discarding it reduces variance at no cost in bias *provided the assumption really holds*.

3 Convolution, pooling, and an architecture

A feature worth detecting in the top left corner is worth detecting in the bottom right, so use the same weights everywhere. That is the whole idea.

For an input with C channels and a kernel of size k by k,

$$z_{ij}^{(f)} = b^{(f)} + \sum_{c=1}^{C} \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w_{c,u,v}^{(f)} \, x_{c,\, i+u,\, j+v}$$

— a dot product between the kernel and a window, repeated at every position, with the sum over c being why a filter reads all input channels at once.

The first layer uses $k = 7$ and every later one $k = 3$, with stride 1 and enough zero padding, $\lfloor k/2 \rfloor$, that the output keeps the input's height and width. A filter sweeps the whole image and produces one two-dimensional map; 32 filters produce 32 maps, stacked into a $32 \times H \times W$ tensor whose channels the next layer's filters all read at once. "Channel" and "feature map" are the same object seen from the two sides of a layer.

Failure condition — the default nobody asked for

`padding` defaults to *zero*, which silently shrinks every feature map by $k - 1$ and will eventually give you a shape error twelve layers later. Ask what the default is, every time.

Pooling halves the height and width by keeping the largest value in each 2 by 2 block. Three quarters of the activations disappear, and with them three quarters of the memory and arithmetic in every later layer; it has no parameters; and each later unit sees a larger piece of the original image. The fourth consequence is the invariance the derivation ended on.

Every convolution is followed by a batch normalisation, the layer Lecture 8 met, and a ReLU. `BatchNorm2d` normalises across the batch and both spatial axes, so it holds one mean and one variance per channel — $2C_{\text{out}}$ parameters, not $2C_{\text{out}}HW$ — and the convolution before it takes `bias=False`, because the batch norm has its own shift.

Stage	Layers	Output
1	conv 7 by 7 ×32, conv 3 by 3 ×32, pool	32 by 64 by 64
2	conv ×64, conv ×64, pool	64 by 32 by 32
3	conv ×128, conv ×128, pool	128 by 16 by 16
4	conv ×256, pool	256 by 8 by 8
head	flatten, dense 256, dropout, dense 102	102

Part	Parameters	Share
Every convolution and batch-norm layer together	586,720	12.2%
One dense layer, 16,384 to 256	4,194,560	87.3%
Output layer, 256 to 102	26,214	0.5%
Total	4,807,494	100%

Channels double as the map halves — the textbook's rule of thumb: what pooling takes away in resolution, the extra filters give back in channels.

Nine parameters in ten are in the layer that is *not* convolutional.

The training loop is exactly Lecture 8's five lines. Convolution changed the model; it did not change the loop, because autograd does not care what the modules are.

4 What it learned

The first layer's weights are 4,704 numbers as 32 filters of 7 by 7 by 3 — small enough to look at. Against the initialisation they are still almost the same vector — cosine similarity 0.9880 between all 4,704 weights before and after, the mean absolute change is 14.1% of the mean weight, and 160 of the 4,704 changed sign.

The weights moved. *No structure appeared* — and thirty epochs of training bought that. Two reasons, both worth knowing. The gradient reaching this layer is small, because ten layers of chain rule separate it from the loss and 1,020 images give only 960 optimiser steps. And its *scale does not matter*: a batch norm follows immediately, so normalising the output makes the overall size of these weights irrelevant to the function, and there is no gradient pressure on it at all.

The textbook picture of a first layer — colour blobs, oriented edges — is real, and you will see it next lecture. It comes from a corpus a thousand times this size.

A feature map is read the same way at every depth: bright means *this filter fired strongly here*, and the spatial layout survives, so a bright patch is where in the image the pattern was found. First-layer maps still look like the photograph; four layers down they do not. After four pooling stages the map is 8 by 8, so one cell answers for a 256-pixel square of the original — a loss of resolution chosen on purpose, right for naming a species and wrong for any task that must say *where*.

5 The notebook

What to change

Set `padding=0` on every convolution and push one image through the network, as the notebook's network cell suggests. Every feature map shrinks, the flattened size changes, and the first linear layer no longer matches — the default nobody asked for, found by breaking it.

6 Exercises

The five set at the end of this lecture's deck, in the style of the written paper. The solutions are on Lecture 13's deck and in the exercise book.

1. A dense layer mapping a $3 \times 128 \times 128$ input to a $32 \times 128 \times 128$ output has $n_{\text{in}} n_{\text{out}}$ weights. Give the convolutional count for a 7×7 kernel, and say which symbols are absent from it. [5]

2. State the difference between equivariance and invariance, say which of the two pooling builds, and why a species classifier wants it. [4]

3. Convolution is equivariant to translation on the interior of an image but not at the border. Give the reason. [3]

4. The lecture's network has 4,807,494 parameters. Name the layer that holds most of them, give its count from its shapes, and say why the ten convolutions together hold so few. [4]

5. After thirty epochs the first layer's 32 filters are still almost the vector they started as: cosine similarity 0.9880. Give the two reasons, and say what a first layer needs before colour blobs and oriented edges appear. [4]

Summary

Weight sharing costs 5,478,275-fold fewer parameters than the dense layer with the same output shape, and the image size is absent from the convolutional count. Sharing the weights is also what makes a convolution translation equivariant, exactly so away from the border; pooling builds invariance on top of it, which classification wants.

The network — convolutions with batch norm, four pooling stages and a dense head that holds nine parameters in ten — trained from scratch on 1,020 images for thirty epochs leaves its first-layer filters still noise, for two reasons: little gradient reaches them, and a batch norm downstream removes any gradient pressure on their scale. Its feature maps keep the layout of the photograph at the first layer and lose it four layers down, where one cell answers for a 256-pixel square.