[00:00:00] Let's say that this is our original image,

and it's a colour image,

and it's three different colours.

There's red, green, and blue,

which I've drawn here as three different layers.

Then I've got what's called a convolutional filter.

[00:00:30] L is used to index these different channels.

We call the red, green, and blue three different channels.

Then I'm summing over the channels l. This is the input layer.

I imagine that at the first hidden layer I have an array of nodes like this.

The pink thing is the inputs.

This blue thing is the first hidden layer.

The neurons in the hidden layer are [00:01:00] also arranged in a rectangular grid like this.

I'm going to compute the activation of

each of these nodes in this hidden layer, one by one.

I'm going to start by computing this one in the corner,

and it's not going to be dependent on all of the inputs.

It's only going to be dependent on the pixels [00:01:30] in a small neighbourhood.

The activation of this node in the hidden layer only depends on this part of the image.

This three-by-three components of the image is represented by this green square.

This is what I'm computing.

This green thing represents the receptive field of that neuron.

[00:02:00] I have weights connecting only those nine pixels to that hidden node.

But actually there's 27 inputs altogether,

because there's three different channels.

There's the red component, the green component, and the blue component in each of those pixels.

To compute the activation of that blue node,

I keep j and k fixed.

Because j and k [00:02:30] tell me the location of the blue node.

But I use m and n to scan over this green area.

That's what these two summations here are doing.

That n goes, in this case,

from 0, 1, and 2,

and m also goes 0, 1, and 2.

The weights for this value [00:03:00] is different from the weight for,

each of these has a corresponding weight.

I've got 27 different weights that I'm using to calculate this activation,

I have a bias as well,

and then after I've computed that linear sum,

I've put it through the activation function.

That completes the computation for that hidden node.

Then I move on to the next hidden node,

and I repeat the process.

[00:03:30] K has now increased.

The key point here is that I use exactly the same weights in computing.

I've shifted along my inputs,

but I'm using the same weights to compute

the next hidden node that I used to compute the first hidden node.

This is called weight sharing,

because the same weights are being used in every place.

Again, I follow through [00:04:00] these two sums with the m and the n,

to compute that hidden node,

and then I move on to the next one.

I keep going like this for all the nodes in that top row.

Now notice this, the number of hidden nodes in this row shrinks a little bit.

Because by the time I get to here,

I've bang into the edge.

[00:04:30] Now in a minute we'll talk about how to do padding and stuff.

But just in this situation,

in the original input,

this had a width of 7.

The hidden nodes will only have a width of 5.

J stays still and k moves up to here.

Then when that's all done,

then I come down and start doing the next row.

J comes down here, and so on.

[00:05:00] I compute that hidden node and then move along.

I continue this process until the thing gets down to here,

at which point I've computed all the nodes in that hidden layer.

Any questions about this? Does this make sense?

Yes? If you're just trying to fill in one [00:05:30] at a time,

why do you need a three-by-three?

This three-by-three thing is meant to show a subset of these pink nodes.

These are not green nodes.

The green is meant to show a subset of the pink nodes,

which are being used to compute the blue node.

[00:06:00] That's right, to compute one node in the hidden layer,

you use nine nodes in the input layer.

There is a good question. The question was, should the blue thing be here or should it be here?

The answer is, I guess conceptually we think of it as being here in the middle,

[00:06:30] but computationally it's more

convenient if this index n goes from 0 to something,

rather than going from minus.

It's better if n goes 0, 1, 2,

rather than like minus 1, 0,

1, it's just cleaner when we write the program.

When we write the program it's actually like this,

but we do [00:07:00] think of it conceptually as being in the middle.

Very good question.

Basically this is the way the convolutional networks work.

Then this is from the input to the hidden layer.

Then the same thing could happen from one hidden layer to another.

We had three channels in the inputs.

We could have multiple channels in the [00:07:30] hidden layer.

Where are we? We could have three different channels in this hidden layer,

we could have five, we could have 10.

It doesn't have to be the same number as we're in the input, it's totally independent.

Often we have multiple channels,

and they're all computed in exactly the same way,

but each channel or filter has its own set of weights.

[00:08:00] In the inputs we call them channels,

in the hidden units we sometimes call them filters.

Per channel and filter is interchangeable.