[SQUEAKING]

[RUSTLING]

[CLICKING]

RAMA RAMAKRISHNAN: All right, folks, good morning. Welcome back. I hope you all had a nice weekend and I hope you had a chance to watch the video walkthrough I posted yesterday. It's going to save us some time today, so let's get right in. Today is going to be super packed. You're going to go from not knowing anything about convolutions perhaps for some of you, to actually knowing how convolutional networks work, and actually to build one, and demo it in class.

And this demo has actually worked pretty well for the last few years that I've taught the class. But you never know because it's a live demo. It may not work. We'll see. Valentine's Day gods, may they be with us. OK, so let's get going. So Fashion MNIST, we saw previously, i.e. as in, in the walkthrough, the video walkthrough, that a neural network with a single hidden layer can get us to an accuracy in the high 80s.

And that thing, that network actually didn't know what was coming in was an image. It literally took this table of numbers, and just took each row, and then concatenated all the rows into one giant, long vector, and then sent it in. So the neural network did exploit the fact that the input data was known to be of a certain type, which is the clue for, how can we do better?

So let's just spend a few minutes on, what is it about images that we have to really pay attention to, as opposed to any arbitrary vector of numbers that's coming in? So when we flatten the image into a long vector and feed it into a dense layer, several undesirable things can actually happen. What are some of them. Any guesses? Yeah?

AUDIENCE: Did you lose the proximity of one pixel to other ones that would be around it?

RAMA RAMAKRISHNAN: Right. So if you take a particular pixel, then let's say that the picture shows a t-shirt. If there's a little pixel in the center of the t-shirt, knowing that the surrounding pixels are related to the pixel in a way, because they are all part of this concept called a t-shirt, would certainly be helpful. So to put it more technically, spatial adjacency information is very important. And we need to somehow take that into account. All right. What else? What else might be going on here?

AUDIENCE: Yeah, doing some metadata about it, like read the dimensions or the resolution can help you [INAUDIBLE].

RAMA RAMAKRISHNAN: Oh, I see. So if you actually had structured data about the image, such as various characters about, that might be helpful. True. But let's just focus on the case where we only have the raw image and nothing else. And under that constraint, what else might go wrong or what else might be suboptimal?

Well, the first thing that might happen is that we may have too many parameters. So these numbers are from my older iPhone. I noticed that when I take a color picture with my phone, it's a 3,000-times-3,000, roughly, grid. So the picture is actually 3,024 pixels on this axis, 3,024 on that axis. So that gets us to roughly 9 million pixels.

But remember, there's a color picture, which means there are three channels, which means there are 27 million numbers, each of which is between 0 and 255 from that little picture. And now let's say we connect it to a single 100-neuron dense layer, a single 100-neuron dense layer. How many parameters are we going to have just in that one little part of the network? Could the mumbling be louder?

Yes, roughly 2.7 billion. Because 27 million parameters times 100, roughly, of course. Forget about the biases for a moment. It's 2.7 billion. 2.7 billion parameters. Do you think we can actually get 2.7 billion images to train any of these things so that we don't overfit? Too many parameters. We have to be smarter about this. It's not going to work. That's the first problem.

So this clearly is computationally demanding, very data hungry, and increase the risk of overfitting. Next, we lose spatial adjacency. We literally are ignoring what's nearby. So that's a huge, huge factor. There is a third factor that we have to worry about, which is that, let's say that the picture has a vertical line on the top left side. And it has some other vertical line on the bottom right side.

What this sort of dumb approach is going to do, it's going to learn to detect that vertical line on the top left. And independent of that, it's going to learn to detect the vertical line on the bottom right, which doesn't make any sense. A vertical line is a vertical line. So you want to be able to detect it wherever it happens. Detect once, reuse everywhere. That's what you need to do.

So this, by the way, is called translation invariance. Translation is math speak for move stuff around. You take a line and it moves around. It doesn't matter. It's still a line. Let's figure it out. So these are the three things we need to worry about. So we want to learn once and use all over the place. We want to take spatial adjacency into account, number two. And number three, let's just find a way to make sure that we don't have billions of parameters for simple toy problems. Any questions? Yeah?

AUDIENCE: Is this the problem of just because we are compressing the image, or that would've happened nonetheless?

RAMA RAMAKRISHNAN: It would have happened. So the question was, is it a problem because we're compressing the image or would it have happened anyway? The answer is it would've happened anyway. You can take any picture, that's going to happen. Because I'm not making any assumptions about how the image is coming in to me, whether it's compressed or not, and so on, and so forth.

All right. So convolutional layers were developed to precisely address these shortcomings, and they're an amazing solution, as you will see. Very elegant. All right. So the next, I don't know, half an hour is going to be me defining a whole bunch of stuff before we actually get to the fun Colabs, and so on, and so forth.

So just to put it in perspective, I have a PowerPoint, two Colabs, and an Excel spreadsheet, and maybe even a Notability file to cover today. But hang on for the next 30 minutes because it's going to be a little concept heavy before we get to the fun stuff. Stop me. Ask me questions because we do have time.

All right. A convolutional layer is made up of something called a convolutional filter. That's the atomic building block. A convolutional filter is nothing but a small matrix of numbers, like this. It's just a small square matrix of numbers. That's the convolutional filter. Now a layer is just composed of one or more of these filters. All right, filters and layers.

Now the thing about the convolutional filter that makes it really magical is that if you choose the numbers in a filter carefully and then you apply the filter to an image, and I'll get to what I mean by applying the filter. If you choose the numbers carefully and you apply it to that image, this little, humble thing has the ability to detect features in your image. It can detect lines, curves, gradations in color, circles, things like that. It's pretty cool.

And so I'm going to claim and I'm going to prove shortly that this little humble filter with the 1's and 0's, it can detect horizontal lines in any picture you give it. This thing here has the ability to detect vertical lines. So I will demonstrate how this thing actually detects all these things. And then we will ask the big question that's probably in your minds already. Where are we going to get these numbers from?

It all sounds great, Rama. Where are we going to get the numbers from? And we have a beautiful answer to that question. All right, so let's go. Now I'm going to first explain to you what I mean by applying a filter to an image. And then I'm going to give you examples of how the filter works for detecting vertical and horizontal lines.

So let's say that this is the image we have. Again, an image, I assume it's a grayscale image. So you just have a bunch of numbers between 0 and 255. So this is the image we have. It's a little, tiny image. And this is the filter that's been magically given to us by somebody. And what we're trying to do now is to apply it.

So what we do is that we literally take this filter, the little one, and then we superimpose it on the top left part of the image. So you have the image here. You take this little filter and then you move it to the top left so that they're right on top of each other. Once you have it right on top of each other, you have these matching numbers.

You have three numbers in the image. There are three numbers in the filter and they're all matching each other right on top of each other. So you have nine pairs of numbers. And then what we do once we overlay it, we literally just multiply all the matching numbers and add them up. You just multiply all the numbers and match them up. And you can confirm later on that the arithmetic I'm doing is actually accurate.

And once you do that, you'll get some number. And once you get that number, what we do is we go to our good, old friend, the ReLU, and then we just run it through ReLU. Now, in this case, all that effort came to nothing because it's 0. That's OK. So 0, and this number becomes the top left cell of your output.

So this is called the convolution operation. And we won't get into why it's called that, and so on, and so forth. There's a long, and rich, and storied history of these things. But this is the convolution operation. And once we do that, you can now predict what's going to happen. We take the same exact operation and we just move it to the right. We move this little 3-by-3 thing to the right and repeat the exact same process.

Matching numbers, multiply all the matching numbers together. Add them up, run them through a ReLU. And then boom, you get the second number here. And you keep doing that until you reach the very end. You fill up all these numbers, then you come to the top of the second row. And you keep on doing that till you reach the very bottom. So this is what I mean when I say apply a filter to an image. Any questions? Microphone, please. Microphone.

AUDIENCE: What happens when they aren't particularly [INAUDIBLE] the remaining but the filter doesn't perfectly match?

RAMA RAMAKRISHNAN: Yeah. So you start from the left and then you keep on going. At some point, the right edge of the filter is going to match the right edge of the image, and then you stop. Now there are some nuances here. So for example, you can actually pad the whole image on its borders so that you can actually go outside the image and it'll still work. Number one.

Number two, nuance. Instead of just moving one step to the right every time you finish, you can move two steps to the right. And that's something called a stride. So there are a bunch of pesky details here, but I'm just ignoring them because this basic default approach works well, amazingly well, almost all the time. So that's the mechanics of how this operation works.

Now I'm going to switch to a spreadsheet which shows this really beautifully, courtesy of the fast.ai people. So what I'm going to do here-- this is a big spreadsheet. I'll upload the spreadsheet after class so you can see it. So all I have done here, or rather, all they have done here, thanks to them, is that they have essentially created a table of numbers in Excel, as you can tell. And they have just put some numbers.

Most of the numbers are 0, but some of these numbers are all more than 0. They are 0.8, 0.9, and so on. Basically all they have done is, instead of working with numbers between 0 and 255, they're just dividing all the numbers by 255 so you get fractions. And they just put the fractions in the table.

And then they have used Excel's very cool conditional formatting to essentially mark in red all the values that are high. If the number is closer to 1, the more reddish it gets. And when you do that, the 3 obviously pops out. So there is a 3 in the image. Yes? OK, good.

So now what we're going to do is we're going to move to our little filter here. You can see the filter. And I'm claiming this detects horizontal lines. Sorry. This table here is the result of applying that filter to that tree And you can see here, I'm looking at the top left cell here. Look at this top left cell.

The formula is nothing more than multiply all those things and add them up. And then once you add it up, run it through a max of 0 comma that, which is just the ReLU. Basic arithmetic. So we do that. And this is the output. And the output is also conditionally formatted to show you where things are lighting up.

And you can see, only the horizontal lines of the 3 are lighting up. Everyone see that? So now you understand. The filter, in fact, is living up to the claim I made for it. Similarly, if you look at what's going on here, this is a vertical filter, the same thing. You apply it, only the vertical line is lighting up.

Now what you can do is, I would encourage you to do this after class, is you can look at all these numbers here, for example. And then ask yourself, OK, why is that lighting up? And you will discover that what's actually going on is that it's looking for edges. It's looking for rows in the table where there is some non-zero thing in the first row and zeros in the second row.

And by choosing the numbers carefully, you multiply the ones with positive numbers and you multiply the zeros with zeros. And then you'll come up with a positive number, and thereby you detect an edge. So what I would encourage you to do is use this Excel thing here. So here is a cell we have. So let's trace its incidence.

So you can see here, these numbers, this is what it's processing. That is a grid that's being processing to come up with that big number. And you can see here in this grid, these numbers are here. And then these numbers are a lot lower than these numbers because there is an edge. The numbers are a lot lower. That's why you can see the horizontal part of the 3.

And so what this filter is doing, it's basically saying, well, the row that I'm catching here has the 1's. The middle has 0's. The rest are all minus 1's. So the small values are going to get very small. The big values are going to get very big, and the overall thing is going to be emphasized. So that's the basic idea of edge detection. Spend some time with it, with Excel, and it will become clear to you what I'm talking about here. All right, cool. So that's that.

By the way, I also have-- there is a very cool site here in which you can actually go in, and punch in your own numbers, and see what it detects. A lot of edges, and curves, and this, and that. It's very cool so I encourage you to try it out. So the key thing here I want to say is by choosing the numbers in a filter carefully and applying this operation, different features can be detected.

Now I mentioned earlier that a convolution layer is composed of one or more of these filters, so one or more of these filters. And so you can think of each filter as a specialist for a particular feature. So it's a specialist. Maybe it specializes in detecting vertical lines, horizontal lines, semicircles, quarter circles. You don't know. You can imagine each of them as being specialists.

And given that modern images could be very complicated, they may have lots of interesting features going on, you probably want to have lots of these filters. But the key is that you don't have to decide upfront. Hey, you, filter, you better specialize in detecting vertical lines.

And you, on the other hand, do not. Stay in your lane. Do vertical lines. You're not going to do that. You will let the system figure out what it wants to figure out. So there is no human bottleneck in doing this. And I mention this because there used to be a human bottleneck before deep learning happened.

Now let's just make sure we understand the mechanics of what happens when you have two of these filters, not one. So this is the input image as before. This is the filter we saw earlier. And this is another filter we have. The thing is, we just run them in parallel. We take each filter, do the operation, come up with an output. Take the other filter, do the operation come up with its output.

And then when you do that, the first one gives you that, the second one gives you that. And this output is a table of some-- it's actually not a table. What is it? Louder, please. It's a tensor. Thank you. It's a tensor. And so these two 5-by-5 matrices can be represented as a tensor of what shape? And there are two right answers.

AUDIENCE: 5 by 5.

RAMA RAMAKRISHNAN: 5 by 5?

AUDIENCE: Into 2.

RAMA RAMAKRISHNAN: Into 2. Correct. So you can either think of it as 5 by 5 times 2, or 2 times 5 by 5. They're both fine. Which one you go with actually ends up being a matter of convention. So now you begin to see why we care about tensors. Imagine if, instead of having 2 filters, we have 103 filters. The resulting tensor is going to be 5 by 5 by 103. OK, good.

Now let's look at the slightly more complex situation, where you have not a black and white image, a grayscale image with just a little table, but an actual color image. So we know how to apply a filter to a 2D tensor, like this, and to get that. But let's say we have something like this, where it has three. It's got three channels, red, blue, green, RGB. It's got three tables of numbers.

So this is a tensor of shape 6 times 6 times 3, let's say. And you want to apply this 3-by-3 filter, just like before, to this thing. You want to apply the convolution operation. How is that going to work? Do we just apply this to each? We first apply it to the red, then we apply it to the green, then we apply it to the blue. Should we do that, or is there a problem with that approach? Yeah?

AUDIENCE: The problem--

RAMA RAMAKRISHNAN: Could you use the microphone, please?

AUDIENCE: The problem with the approach would be the same as what you said earlier, that it wouldn't learn. The lines are probably the same in each channel. The location of the lines are probably the same each channel.

RAMA RAMAKRISHNAN: Yes. The location of the line is going to be the same thing because that line, if you will, is the aggregation of information from the three different channels. Right. But the problem here is slightly different, which is that if you do them independently, the network has not been informed that these things are all part of the same underlying concept. As far as concerned, it's just three things. It's just going to process them independently.

So we need to somehow change the filter so that it understands, what is at this pixel location, the three numbers under it? RGB. They are actually part of the same thing, underlying thing. So what we do is actually very simple. We just take this filter and make it 3D. So we take this filter. Instead of having just one of them, we just make it a cube, like that, three times.

And once we do that, you can imagine taking this thing here and essentially doing that. Now, instead of having 9 numbers in the image and 9 numbers in the filter, you have 27 numbers in the image, 27 numbers in the filter But you still match them up, multiply them, add them up, run them through a ReLU. By the way, I tried to get ChatGPT to give me a picture like that. It just completely bombed. I tried three, four, five different variants. It just gave up. Then I found this nice picture in DeepLearning.ai and I used it.

AUDIENCE: So then if you put different numbers in each of the layers of that color processing, would you get a different thing to green and blue?

RAMA RAMAKRISHNAN: I'm sorry, say that again.

AUDIENCE: If you put different numbers in each of the layers of your-- not layers, in each of the different depth dimensions of your convolution filter, would that be like color processing?

RAMA RAMAKRISHNAN: Yeah.

AUDIENCE: [INAUDIBLE].

RAMA RAMAKRISHNAN: Yeah. You will put different numbers. In fact, you have 27 numbers now. But we haven't gotten to the question of where these numbers are coming from. So just hold the thought till we get there. So any questions on this? You literally take the 2D thing and make it 3D. You basically give it depth, and the depth just matches the depth of the input. So if the input is 10 deep, your filter is going to get 10 deep. OK? Yes?

AUDIENCE: Rather than increasing the rank order of the tensor by one, is there any instance where you would create an abstraction layer where you would run an operation across the different layers to come up with a intermediary layer that you would run a lower rank tensor or filter upper?

RAMA RAMAKRISHNAN: Yeah. So there is a lot of stuff in the research literature which tries to do things like that. I'm just describing the most basic approach to doing this. And as it turns out, this basic approach is extremely powerful. And of course, researchers try to go from the 95th percent thing to the 95.1%. So they invent all sorts of crazy, complicated stuff, which is all good for us, humanity. But for practical use, this is good enough.

AUDIENCE: How do you convert the 3-by-3 layer into a single 4-by-4 matrix? [INAUDIBLE], what with the three layers part of it?

RAMA RAMAKRISHNAN: Yeah. So we are coming to that. I think we have a slide here. Actually, we don't. Never mind. I'll answer that. So here you have one filter. You have one 3-by-3-by-3 filter, which plugs into this thing here. And then it gives you the 4 by 4 at the end. So for one filter, we know that by doing this operation, we get this 4 by 4.

Let's say that you have another filter which is also 3D. You do that thing, you'll get another 4 by 4. And if you have 10 filters, you'll get 10 of these 4 by 4s, which then gets packaged up into a 4-by-4-by-10 tensor. Remember, whether it's 2D, 3D, 10D, what is coming out is always 2D.

Because ultimately when you apply all this operation at each position, you just have one number. And then ultimately you just do all those things, you just come up with a table of numbers always. So what's coming out is always a 2D number table like that. But when you have lots of filters, you have lots of these 2D tables, one after the other, and therefore, they get packaged up into a tensor.

So textbook Chapter 8.1 has a lot of detail on intuition, which I think is really good. So please try it out. And folks, by the way, this convolutional stuff, it grows in the telling. So I would encourage you to revisit it, revisit it a few times, and then it slowly becomes part of your muscle memory. So don't expect to just understand all the nuances like one shot. Do it a few times. It'll become wired into your head.

The big question-- these seem excellent, but how are we supposed to come up with these numbers? In fact, traditionally these filters actually used to be designed by hand. Computer vision researchers would invest prodigious amounts of time, and effort, and talent to figure out the right kinds of filters to use for various specific applications.

So if you wanted to build an application which would look at, say, MRI images and figure out, OK, what kind of features should I extract from this MRI thing to be able to predict the evidence for a stroke? They would actually hand design the filter, try lots of different values, and then come up with, I got the perfect filter for this thing here. So that's the way it used to be done.

But as we figured out how to train deep networks with lots of parameters, we figured out things like ReLU activation, stochastic gradient descent, GPUs, backprop, things like that, this big idea emerged. Why don't we think of the numbers in the filter as just weights? And why don't we just simply learn them from the data using backprop, just like we learn all the other weights? What's the big deal?

This simple idea, and it feels a bit, I don't know, blindingly obvious in hindsight. I'm sure it was not obvious in foresight. This was the breakthrough. This was the key breakthrough. And now it's actually possible to do this because a convolutional filter that we've seen is actually just a neuron. The underlying arithmetic of it is just a neuronal arithmetic. And so it just happens to be a slightly special one.

It's actually even simpler than a regular neuron. And in the interest of time, I have one or two slides in the appendix which tells you exactly why it's a neuron, so check it out. But just take my word for it. It's just a particular kind of neuron. And because it's a particular kind of neuron, and we know how to work with neurons, we know how to work with neurons, which means that our entire machinery, layers, loss functions, gradient descent, SGD, blah, blah, everything is immediately applicable. We don't have to invent any new stuff to make it work.

AUDIENCE: Do you initialize the data basically between applications or just because the network has different sizes? Does computer vision versus [INAUDIBLE] do it differently, or it's just because the network has different numbers of neurons?

RAMA RAMAKRISHNAN: Yeah. So the initialization, it's a good question. Let's come back to it when we get to something called transfer learning, which I'm going to get to by about 9:30. So that's it. So this turned out to be a huge turning point in the computer vision field. And this was the massive unlock in the year 2012.

This computer vision system that used this technology called AlexNet burst out onto the world stage because it crushed the competition in a competition called ImageNet. And the previous best score was 26% error rate. And this thing came in and had 16% error rate. It's the kind of thing where if you see it, you'll be like, oh, this must be a typo. Because every year the improvements are very little, 0.5%, 1%. And then this year was 10%, and that was because of this approach.

Now one other thing I want to talk about is that with every succeeding convolutional layer, this particular convolution, any particular convolutional filter is basically implicitly seeing much more of the input image as we go along, which means that if, in the very beginning, if this is the input, this little convolutional filter, this number here in the first layer, let's say only sees the top of the chimney or whatever of this house. But then the next layer, remember, the next layer's input is this particular layer. And so this particular little thing here is getting information from this whole square here.

And every one of the points in that square is actually something big in the original picture. So with every additional layer, you're seeing more, and more, and more of the image. And this is a key part of why these things work, because you're essentially hierarchically building a better and better understanding of the image. It is the hierarchical understanding, the hierarchical learning that's a very key part of the unlock.

And so if you look at networks and what they're visualizing, this actually is a face detection deep network visualized as to what it's learning. You'll see that the first layer is just learning lines, and edges, and so on, lines. And the second layer is actually learning edges. Look at this thing. It's learning to put these lines together to get some sort of an edge here, another edge here. This looks like 3/4 of somebody's ears.

And then these things are now being assembled to get whole faces out. Can you imagine the researchers who did this work? They built the network. It's doing really well on detecting faces. And they're saying, OK, let's see what it's actually doing. And then this picture pops up. I mean, the goosebumps.

So pooling layers, the next one. So far you talked about convolutional layers. This is the second thing, second building block, and then we'll go to the Colabs. So pooling layers are also called subsampling or down-sampling layers. And so the idea is that every time a tensor is coming out of these convolutional layers, we'll try to make it slightly smaller. Because the act of making it smaller will force the network to try to summarize and learn what's going on in this complicated thing that's coming into it.

So I will describe the mechanics first. So let's say that this is the output of a convolutional layer. This is four of them, a 4 by 4. So what we do is that there are two kinds of pooling, max pooling and average pooling. This is called max pooling. And the idea is really simple. In this max pooling layer there are no weights parameters to be learned. It's just a simple arithmetic operation.

We basically take this. We basically superimpose a 2-by-2 empty grid on the top left. And then we say, hey, what's the biggest number among these four numbers? Well, the biggest number is 43. Boom. OK, I'm going to stick a 43 here. Then, I move my 2 by 2 to the right so that it overlaps with these numbers in blue. And I say, hey, what's the biggest number here? OK, that's 109, and I move it down. What's the biggest number here? 105, stick it in here. Biggest number here? 35, and I stick it in there. That's it. This is max pooling.

Similarly, there's this thing called average pooling, where instead of taking the maximum of these four numbers, we just average the four numbers. The average of these four things in yellow. Am I done? The average of these four numbers is 22.2. The average of these blue numbers is 45.5. You get the idea. That's it, max pooling and average pooling.

Now as you can see, when you apply pooling, the number of entries drops significantly. The number of entries drops significantly. And the output from this layer is just fit to the next layer as usual. There's nothing crazy going on. So it's a way to shrink the output from one convolutional layer. Before it passes on to the next convolutional layer, you interject with a pooling layer.

Now I have actually, even if I may say so myself, a very nice handwritten explanation of what pooling does, the effect of pooling. And unfortunately, I can't get my iPad to actually show up on my laptop so I'm not going to be able to do it. But I will record a walkthrough and I'll post it. Check it out.

But the intuition that I try to convey with that thing is that-- oh, sorry. I'll come back to this. So max pooling acts like an OR condition. It basically says, I have this big picture. So in the four things that I'm looking at, if there is any number which is really high, that means that some feature is being detected. The number is really high coming out of a convolutional layer. That means that something somewhere fired up, lit up.

And so I'm just looking to see if anything lit up in that part. If it did, I'm going to say, yeah, something lit up. If nothing lit up, then I'm going to say, oh, nothing lit up. So in that sense, you can imagine it's acting like an OR condition. Anything fired up? Anything fired up? Anything fired up? Anything? Yes? OK. Otherwise, no.

Sadly, I can't switch to Notability. So it acts like a feature detector. So if you have lots of things going on in a particular picture, you want to be able to summarize and aggregate all the things that are going on so that you can say, you may have a big picture with lots of things lighting up here and there. But you want to step back and say, you know what? In this picture, the top left, nothing lit up. The top right, something lit up. Bottom left, something lit up. And the bottom right, nothing lit up. So you're operating at a higher level of abstraction. That's the effect of pooling.

AUDIENCE: But once you use spatial elements, [INAUDIBLE]?

RAMA RAMAKRISHNAN: You don't, because what you're actually saying is, the top left has this thing. You already know it is in the top left. And you've already moved up to that level of abstraction. So the fact, for example, in the top left there is a human eye and there is a circle detector.

It's going to fire up and saying, hey, in the top left, there is an eye lit up. So you're not looking at the pixels anymore. You're already operating at a higher level of abstraction, and that's how we get around it. But this proceeds slowly and incrementally, which is why you have these big networks.

So now, as we saw, some successive convolutional layers can see more and more of the original image. The max pooling layers that follow them can detect if a feature exists in more and more of the original input as well. So by the time you get to the seventh, and eighth, ninth layers, and so on, this thing is actually really smart. It's operating at a very high level of abstraction. You can think of it as it has basically tagged all the features of that image at various resolutions, and it can work with it.

AUDIENCE: Is there a trade off between doing the pre-processing as opposed to adding additional convolution layers? I'm thinking if you had a video, turning it into black-and-white static images in a sequence, as opposed to shoving in a color video with a time, the parameters have to expand. Is there a trade-off element?

RAMA RAMAKRISHNAN: There is a trade off. If your particular data set and input has some-- there is some very important domain knowledge that you want to encode when you set up the network, so that the network doesn't waste its capacity learning things that have to be true, then, yeah, modify the input. But if you're not sure, then you want just the network, learn whatever it can, as long as it's focused on predicting accuracy as well as possible, then just let it be.

So that's the basic idea. And again, I'm sorry this is Notability thing is not working. But take a look to really understand how this max pooling thing business works. Oh, I think I skipped over this. So when you have something like this, so this, let's say, is a tensor coming out of some convolutional layer and its size is 224 by 224 by 64. Then you apply something like a pooling.

The thing I want to point out is that the pooling will work with every slice of the tensor. So if the tensor is 224 by 224 by 64, it has a depth of 64, which is basically like saying it's got 64 tables of 224 by 224. And the pooling will work on every one of those tables, which means that you'll still have 64 things at the very end. It's just that every one of the things of the 64, the 224 by 224 will shrink to 112 by 112. So each table shrinks due to pooling, but the number of tables does not change.

By the way, this link here is a beautiful explanation of all these things with a little bit more complexity as well from a course taught at Stanford in 2018, or 2019, or something, I forget. So just check it out if you're curious about this stuff. It's really good. So that brings us to the architecture of a basic CNN.

And so what we do is we have an input. We take that input. We run it through a bunch of convolutional and pooling layers. So this is a convolutional layer. And then we pool it, which is why it has shrunk in size. And then it goes through another convolutional layer. Then, we pool it, which is shrunk again, and then it keeps on doing it.

So we have a series of these. These are called convolutional blocks. So a convolutional block is typically one to two convolutional layers followed by a pooling layer. So you have a series of convolutional blocks. And the thing to notice is that, as you go further and further in the network, the blocks will actually get smaller and smaller because of max pooling. It gets smaller and smaller, but they'll get deeper and deeper.

And we have empirically figured out that actually that model of reducing the size, the height and the width, but then making it deeper, tends to work really well in practice. And so, in fact, apologies to the live stream that I can't use iPad. I'm going to do it on the board. So let's say that you have a picture, which is coming in as 224, 224. And then you have, say, three of them because it's a color picture.

So you have three of them. Can you folks see this OK? All right. So let's say this is the input coming in. And ResNet, which is a very famous network that we're actually going to work with in a few minutes, when it actually gets done with all this convolution pooling business, the final tensor that it has is actually of shape 7 by 7, but it is 2,048 long.

It has processed something which is 224, 224 times 3, to a much smaller height and width, just 7 by 7. But it has gotten much deeper 2,048 layers. This is a numerical example of what I am talking about there in terms of, as you go along, things get smaller but deeper. Yes?

AUDIENCE: Is the reason that it gets deeper because each-- it gets deeper because each layer has a single feature that is picked up, and then it gets stacked on top of each other?

RAMA RAMAKRISHNAN: It's not so much that each layer has picking up a single feature. It's more that, basically, the way I think about it is that the number of atomic features that you may want to detect are probably not that many. Lines, curves, gradations, and color, and things like that. But the way in which you can combine these atomic features to depict real world things is combinatorial.

It's sort of like, I have 10 kinds of atoms. How many molecules can I make from it? You can make a lot of molecules from those 10 atoms, which means that you better give the network the ability to capture more and more of these possible things that the real world can come up with. And so as the depth increases, you have more filters. And every filter now has the ability to pick up some combinatorial combination of what's coming in.

AUDIENCE: Sorry, a quick question related to this. So right now our model is being trained to detect specific features, like a line, eye color, or something of this sort. But still, it doesn't have meaning to this. Still, they don't know if that arc is a sun or is an eye.

RAMA RAMAKRISHNAN: Yeah. So we don't tell it what to learn. It just learns. All we tell it is, make sure that you minimize the loss function. Now, once it has finished learning, if it's a good network, it has good accuracy. Then, we can introspect. We can peek into the internals and try to understand, what is it learning? And sometimes, like you saw in the face detection example, it's actually learning interesting things, like basic lines, and edges, and then slowly, more complicated shapes. And then finally, entire human faces. Sometimes it may not be understandable.

AUDIENCE: And the way it's doing this is by constructing features, not by learning.

RAMA RAMAKRISHNAN: How do you figure out what it's learning?

AUDIENCE: Yes.

RAMA RAMAKRISHNAN: Oh, I see. So I'm going to give a reference in just a few minutes. Read the paper. That was one of the first ones to actually visualize what these things are learning. And that will give you an idea of how it actually works. And I'm also happy to talk about it offline. It's a bit of a tangent, but it's a really rich tangent. So if I keep talking about it, I'll end up spending 10 minutes on it. So I'm going to back off.

So now, once we do that, now we are back in familiar territory, where we take whatever tensor is coming out from these convolutional operations and pooling operations, and then we just flatten them, only now into a long vector. And once we flatten them, we can connect them to some good, old dense layers, like we know how to do.

And then we finally connect them with whatever output layer you want. In this case, this example is using some multi-class classification of classifying images to what kind of automobile or whatever it is. So it's like a softmax. So this is the general framework. Any questions? Yeah?

AUDIENCE: Can you explain again how the depth increases exactly?

RAMA RAMAKRISHNAN: Oh, the depth increases because you decide what the depth is. So when you add a convolutional layer, you decide how many filters it has. So you just keep adding more and more filters the later on you go in the network. So it's in your hands. So remember, the number of neurons in a hidden layer is in your control. Similarly, the number of filters is in your control. It's a design choice. And we design it so that the later we go, the more depth we have.

AUDIENCE: So you stack layers with-- each of those layers has a different filter applied to the end output?

RAMA RAMAKRISHNAN: Yeah. A layer is made up of filters. And so the depth just comes from having lots, and lots, and lots of filters. And you get to choose what they are. So now let's go to the Fashion MNIST Colab that I did the video walkthrough on, and then actually solve it using a convolutional network.

All right. Cool. So at this point I'm going to zip through some of this stuff because the preliminaries have to be done. Import all these packages. Set the random seed here. Great. And then we will load the MNIST data set, just like I did in the Colab yesterday. We create these little labels, and then we just have these standard functions to plot accuracy and loss that we've been using so far.

All right, now we come to the convolutional thing. And so as before, we're going to divide it by 255 to normalize everything to a 0 to 1 range. Let's confirm to make sure that the data, nothing has gotten tampered with. Yes, we have 60,000 images. Each one is 28 by 28 in the training set.

Now convolutional networks, they expect the input to have three channels, or it expects to have an additional thing, which is like a channel. The color images have three channels. But black and white images have only one channel, one table of numbers. So instead of saying 28 by 28, we tell the convolutional layer, expect 28 by 28 by 1. It's the same thing conceptually, but that's the format that it expects.

And so we go here. And then we say, all right, there's a thing called expand_dimension. I'm just telling it to expand its dimension. And once I do that, you can see here, it's still 60,000. But instead of 28 by 28, it has become 28 by 28 by 1. Same thing. Now, let's define our very first CNN.

As before, the input is just keras.Input, as before. No difference here. And we tell it the shape. And the shape is, of course, just 28 by 28 by 1. That's what I have here. And then we come to the first convolutional block. And this is the key thing. If you want to tell Keras to use a convolutional layer, you use this keyword, layers.Conv2D. And from this you can probably also figure out that there's a Conv1D, and there's a Conv3D, and so on, and so forth, which, explore. It's really good stuff. But for image processing, Conv2D is all you need.

And now we tell it how many filters you want. So we decide on the number of filters. So I've decided to have 32 filters. And then we also have to decide the size of the filter. The simplest size is 2 by 2. So I'm just going to go with that, kernel_size is 2 by 2. And then the activation is, of course, ReLU. I give it a name, convolution 1, and then I feed it the input.

And then once I do that, I followed up with a pooling layer where I use MaxPool2D. And MaxPool2D, you just literally pass the input. You get the output back, it just shrinks everything using pooling. So that is the first convolutional block. And you know what? I know how to cut and paste. Cut and paste, I get the second convolutional block. Here is the second convolutional block.

And I know in just the lecture I mentioned that, as you go deeper, you get more depth to it. But this is just a starting point. I'm just going to use the same depth. It's not a big deal. It's a simple problem, so which is why in the second convolutional block, I'm still using only 32. But you can totally go to 64, for instance, to make it much deeper.

And once I do that, I finally come to the point where I flatten everything into a long vector. Then, I connect it to one dense layer of 256 neurons. And then finally, I come to the softmax where I have 10 outputs, 10 categories of clothing, softmax.

And then I tell Keras, OK, take this input and the output. String them up together. Define a model for me. That's it. That's the convolutional network. The new concepts we are seeing here are Conv2D for the convolutional layer, and then MaxPool2D for the max pooling layer. That's it. Coming. So let me just run this thing, make sure it runs. OK. Good. Yeah?

AUDIENCE: How do you decide when to flatten? And would there ever be a situation in which we just use the method that we used before and not using CNN?

RAMA RAMAKRISHNAN: Well, we already tried it with MNIST. We didn't use the CNN. We just flattened it right away.

AUDIENCE: Yeah, it did work.

RAMA RAMAKRISHNAN: It's not bad. But we are like, can we do better than 85, or 88, or whatever the percent was? But we are working with images. It's typically a good idea to just start with the CNN straight off the bat because you're not losing any. You're not giving up anything. So in terms of how many layers you should have, my philosophy is start simple. And if it works, stop working on it. If it doesn't, add more layers. Yeah?

AUDIENCE: Yeah, just to build on that, is it, the architecture design, so the number of filters, the kernel size, the number of layers, convolution, pooling, is that just all based on trial and error to see what works best?

RAMA RAMAKRISHNAN: Yeah. So typically it's based on trial and error, to answer your question. But as you will see in the transfer learning discussion we're going to have soon, you can actually, instead of doing anything from scratch, it's much better to just download a pre-trained model and just adapt it for your particular problem. That is actually the norm by which people do these things. The reason I'm doing it from scratch is because you should know how it was done. It should not be a black box to you. That's my goal. Yeah?

AUDIENCE: Just from a notation perspective, I noticed that you made all of these layers x. Is that a habit we should get into, making them all the same, or is that just--

RAMA RAMAKRISHNAN: Actually, I'm not naming the layers as x. What's going on here is I'm feeding it x. And whatever is coming out of it, I'm just calling it x. That's all. It's just a notational convenience for me. I'm just calling the input and the output, and Keras, under the hood, will track everything and make sure the right thing happens. Otherwise, I'd have to be, x1, x2, x3, x4. And then if I want to add a new layer somewhere in the middle between x3 and x4, I'll have to call that x4. And then I'll change everything to 5, 6, 7. Complete pain in the neck. That's why I do this.

So model.summary, it has got 302,000 parameters. I'll just plot it. Great. And I encourage you to hand calculate it later on and make sure the numbers tally. For now, let's just go. So as before, we'll just use the same compilation. We'll use adam, and then we'll train it for just 10 epochs. We'll use a validation split, again, as usual of 20%. So let's just run it. It's actually going to run. And as you will see, convolution networks, there's a lot more going on so it's going to be a bit slower to run. Hopefully not too much slower. While it's doing, other questions?

AUDIENCE: So if we have a task other than image classification, do we still flatten the model like for semantic segmentation?

RAMA RAMAKRISHNAN: Yeah. So this is for image classification. For other kinds of applications, typically you run it through a bunch of convolutional layers, and so on, and so forth. But the output side of the equation gets much more complicated. Because instead of classifying just the whole picture into dog or cat, if you have to take every pixel and classify it, then, well, you better have an output shape that is the same dimensions as the input shape.

So for that we use a different architecture. It's called U-Net, and so on, which unfortunately I won't be able to get into. But I am planning to post another video walkthrough where I show you how to use the Hugging Face Hub to very quickly build models for the other applications, like segmentation, and so on. I'm hoping to post that tomorrow. It's an optional viewing thing. That might help with that.

So is it done? OK, good. It's done. All right, let's plot the thing here. All right, so it seems like this training is going down nice and nicely. Validation is flattening out somewhere here around the 8th epoch. Let's look at the accuracy. Same situation here. The accuracy is in the 90s. Of course, the final question, of course, is how well it does on the thing. Whoa, 90.5%. It's pretty good.

By the way, if you're not impressed that we went from 88 to 90, these applications are the proverbial diminishing returns problems. So what you should always think of is, look at the amount of error that's left, and ask yourself, how much of that error am I able to reduce? We had 12%, roughly, of error left when we did the simple Colab yesterday.

From that 12%, we have knocked off 2% of the 12% to get to over 90%, which is amazing. And in fact, I think the state of the art on this is 97%. So I invite you to take this thing, and try different filters, and so on, and so forth to see if you can get to the mid 90s. It's not easy, but try it. Yeah?

AUDIENCE: Is the number of epochs has to do with it's related to the number of batches? Because you put 64 batches and then epoch.

RAMA RAMAKRISHNAN: No. The epochs is just the number of passes through the whole data. But within each pass, within each epoch, the batch size tells you how many batches you're going to process. So it is basically the number of examples you have in your training data divided by the batch size that you have chosen. That number rounded up is the number of batches within each epoch. And here I'm just choosing 10 because--

SIRI: I found this on the web.

RAMA RAMAKRISHNAN: Siri found something on the web. I chose 10 because it's going to be fast for me to do it in class. And 10 is actually more than enough because you can see it's already beginning to overfit. Yeah?

AUDIENCE: This is more of a conceptual question, but is it always the case that a neural network will have better accuracy than its machine learning algorithm? And I'm asking more on the case of the heart disease problem.

RAMA RAMAKRISHNAN: Oh, yeah. Great question. So neural networks are really good for unstructured data, like the ones we are having here. But if you have structured data like the heart disease problem, sometimes it actually works really well. Sometimes things like gradient boosting, XGBoost, work really well. So if I am actually working on a structured data problem, I'll try both. I'm not going to axiomatically assume that the DNN is going to be the best thing. But if you have structured data, it's the best game in town.

By the way, I have a whole section here on once you build a model, how do you actually improve it? Check it out. It's an optional thing. All right, I'm going to stop this here. So the next thing I want to do is, so we went from 88% to 90-plus percent using convolutional networks.

Now let's work with color images. Let's kick it up a notch. So I actually web scraped all these pictures for you folks for your enjoyment. I web scraped about 100 color images of handbags and shoes, roughly 100 handbags and shoes. So the question is, with these essentially 200 images, can we build a really good neural network to classify handbags and shoes?

It seems kind of absurd because 200 examples, it's not that much. It doesn't feel like a lot. The MNIST data, Fashion MNIST, has 60,000 images. Even with that, we are overfitting in 5, 6, 7, 8 epochs. With 200 images, maybe, is there any hope? Obviously, there is hope. Otherwise it wouldn't be in the lecture.

So we're going to take this data set and let's see what we can do with it. So we will first actually build a convolutional network from scratch to solve this problem. I'm actually going to run through the code because at the end of it, we'll have a live demo. So I would like one volunteer to give me a handbag and one volunteer to give me their footwear [INAUDIBLE] in class.

Unlike the previous data set, this one actually, I just web scraped it. I've stuck it in this Dropbox folder. Let's just download it and unzip it. And once we do that, we have to now organize it with these 200 images. So I have to do some boring-ish Python stuff here.

So here what we're doing is that we have 100 handbags, roughly 100 shoes. And what this code is doing is it's actually creating a directory of saying, it's splitting stuff into train, and validation, and test. And then for each of the splits, it's doing the handbags and the shoes folder. So once we do that, basically this directory structure is created.

Training, validation folder, test folder, handbags, and shoes. In fact, actually, I think you can see it here, see here, handbags and shoes. And within that, there is train, test, validation. And within each of these there's handbags and shoes. So the idea is that when you're working with images, what you can do is you can just create folders for each kind of image.

Let's say dogs, cats, two folders with cat images and dog images. And then just point Keras at it. It'll automatically figure out those are the labels that it needs to use. So it's very convenient when you're working with images. And the book explains this thing in great detail.

So when working with these color images we'll follow this process. We'll read in the JPEGs. We'll convert them to tensors. And then since I'm web scraping it, they all come in different shapes and sizes. So I need to bring it all to the same size. I resize it, and then I'm going to batch it into whatever. I'm going to batch it using a batch size of 32 here. And this utility from Keras will do all that for you very quickly.

So basically what it says is that I found 98 images in the training data belonging to two classes, 49 in the validation, and 38 in the test. So less than 100 examples in the training set. That's what we have here. What's the time? 9:38. Now let's just check the dimensions to make sure. Good. So 224, 224 by 3. And the reason why did I pick 224 to 24? As you will see later, we're going to use something called ResNet. And the ResNet expects it to be 224 by 224 by 3. That's why I resize it to 224, 224.

Let's look at a few examples of my wonderful web scraping in action. Oh. It's pretty wild, right? Now let's do a simple convolutional network. And before, we would take all the x values in Fashion MNIST and divide them manually by 255 to normalize it to 0, 1. Well, you know what? We are actually graduating to the higher levels of Keras now so let's not do that. Manual stuff is bad.

So we'll do it within Keras by using something called a Rescaling layer, where we just tell it how much to rescale, and boom, it will do it for you. The first convolutional block, just like the Fashion MNIST, 32. Second block, again, 32. Maxpool, flatten. And then here we only have handbags versus shoes. Just a sigmoid is enough. It's just a binary classification problem.

So I'm just using one output layer with a sigmoid and that's our model. So let's do the model. model.summary. 101,000 parameters in this little model. OK, let's compile it and run it. And note here, because it's a binary classification problem I'm using binary cross entropy. And this is the same Adam and accuracy compile.

And then, boom, let's run it. We'll do it for 20 epochs. Hopefully it won't take too long. While it's doing this business, I'm going to shift to the PowerPoint. So we'll go back to see how well it did. But the question is, whatever it did, we built it from scratch. So the question is, can we do better than that? Because we only have 100 examples of each class, and which brings us to something very cool and very powerful called transfer learning.

So the key thing is there are two research trends that are going on that we take advantage of. The first one is that researchers have designed architectures which exploit the kind of input you have. So Olivia asked the question, if you have a particular kind of input images, do you actually change the input or do you actually change the network? As it turns out here, for example, if it's images, we know that we should use convolutional layers because convolutional layers were designed to exploit the image nest of the input.

Similarly, if you have sequences of information, obviously natural language, audio, video, gene sequence, and so on, and so forth, these things called transformers were invented to exploit them. And we'll spend a lot of time on transformers starting next week. So that's the first trend.

The second trend is that researchers have used these innovations to actually create and train models on vast data sets. And thankfully, they have made them publicly available for us to use. So transfer learning is the idea that if you have a particular problem, let's just take a pre-trained network somebody may have already created. And then let's just customize it to our problem rather than actually build anything from scratch. That's the basic idea.

So here, we have this. Basically we have to build a classifier which takes in an arbitrary image and figures out if it's a handbag or a shoe. That's our goal. Now handbags and shoes are everyday objects. And so what you can do is you can look around and see if there are any networks that have been trained by other people which actually have been trained on everyday images, as opposed to MRI, or x-rays, specialized images, everyday images.

Of course, the first thing you should probably do is to see if anybody has built the specific thing you want, handbag, shoes classifier on GitHub. Assuming it's not, then you do transfer learning. So now it turns out that there is this thing called ImageNet which is a database of millions of images of everyday objects in 1,000 different categories, furniture, animals, automobiles. You get the idea. And so we can look for networks that have been trained on ImageNet.

Let me just go back to the Colab just to make sure it doesn't time out. So it has finished doing it. Let's just plot these things. So there is some overfitting that happens around here on the training and the 10th epoch. Let's look at the history. OK, interesting. So the training accuracy is actually getting almost to 100%.

But we're not impressed by training accuracy. We care about validation and test accuracy. And that seems to be hovering around in the 80s. So let's just evaluate it anyway to see what happens. So it gets to 87% accuracy on this data set. It's actually pretty good, given that we only have 100 examples. So 87% accuracy, but we pre-trained the whole thing. Sorry, we did everything from scratch.

There's this whole section about data augmentation, which, you know what? Do we have time? Yeah. So the idea of augmentation is that when you have an image, let's say you take this image and you just rotate it slightly by 10 degrees. If it's a handbag before you rotated it, it sure as hell is a handbag after you rotated it.

It doesn't change. The meaning of the image doesn't change just because you rotated it slightly, or maybe you zoom in slightly, you zoom out slightly, you crop it slightly. Nothing happens. So what you can do is you can take any image you have and you just perturb it slightly, like that. And then add it as a new example to your training data. It's an unbelievable free lunch, frankly. And the same thing actually, the same kinds of techniques actually work for text also, which we'll cover later on.

This broad area is called data augmentation. And it's a great way, when you don't have a lot of data, to artificially bolster the amount of data you have. Of course, Keras makes it very easy for you to do all these things. It has already predefined a whole bunch of data augmentation layers for you.

So here's a little example where I basically take a picture and then I randomly flip it. So if it looks like this, I flip it this way horizontal. And then I randomly rotate it by 0.1. I forget if it's 0.1 degrees or radians. You can look up the documentation. And then random zoom, zoom in and out a little bit.

But it won't do this for every picture. You only do it randomly. So there are only some pictures will get perturbed in some ways. And that's how you make sure there's enough diversity of pictures that you have. So once you do that, you can actually take a picture and see what it does.

I just randomly grab a picture so it keeps changing every time. Yay. Look at this handbag. Handbag slightly rotated this way, rotated that way, some more, maybe a little bit of zooming going on, and so on. You get the idea. And there's a whole list of these things you can do. But when you do those things, make sure that what you're doing doesn't actually change the underlying meaning of the picture. It's really important.

So for example, if you're working with satellite data, you must be very careful not to do flips of crazy flips, or even if you're working with everyday images, horizontal flips are OK. Don't do vertical flips. How many times will you have an upside down dog picture that you need to classify? Make sure your augmentation doesn't go nuts.

Once you do that, you can actually just insert the data augmentation layers in your model right there, right after the input. The rest of it can stay unchanged. So this is a great way to increase the size of your training data. And here is a model. And then I invite you to actually just play with it and train it. In the interest of time, you won't actually train this model. But it's in the Colab. You can just try it. It also figures prominently in homework 1, by the way, data augmentation. So you'll get more experience with this.

So back to the PPT. So this is what we have. And so any network that has been trained on this ImageNet thing, turns out, learns all kinds of interesting features in every one of its layers. So here this is the first layer. And you can see it's picking up gradations of color, line-ish kind of behavior. Layer 2 is actually picking up, hey, look, it's picking up an edge. Can you see that edge, like that?

And then layer 3 is picking up these interesting honeycomb shapes, and so on. Oh, it's actually, this thing is already picking up the shape of a human torso. Yeah, this here is actually picking up what looks like a Labrador retriever. Isn't that cute? Come on, even if you're not a dog person.

So this visualization I was referring to earlier, to figure out what are these networks actually learning, this paper was one of the first ones to actually visualize what's going on inside. So if you folks are curious how these pictures are actually produced, I would encourage you to check this out. Yeah?

AUDIENCE: Sorry, we spoke about images and you referred to classes on text next week on transformers. But what about, say, an email which has both text and image, and there'll be whitespace depending on who has written it? How does that get put in as an input for an image?

RAMA RAMAKRISHNAN: So we'll revisit this great question a bit later on in the course. So the answer is a bit complicated so I want to do it justice. So we'll come back to it. So it turns out this thing called ResNet is a family of networks which are trained on this ImageNet data set. And they did really well in this competition that's associated with dimension data set called ImageNet.

And so this is an example of such a network. So you would expect the weights and the parameters of ResNet, given that it's been trained on ImageNet, to have some knowledge about lines, and shapes, and curves, and things like that. So maybe we can just use that.

But the thing is, we can't use ResNet as is because remember, it was trained to classify an incoming image in 1,000 possibilities. Here, we only have two possibilities, handbags and shoes. So what we do is very simple and elegant. We do just a little bit of surgery. We take ResNet and stop just before the final layer.

So take my word for it. This thing here, what it says is, fully connected 1,000 because it's got 1,000 way, 1,000 objects. So what we do is we just take everything except, and we stop just before that last layer. And then what comes out of that layer hopefully will be a very smart representation of the images that has been trained on.

And so what we do is we can think of headless ResNet as our model. We can take all our data and run it through ResNet, up to but not including the last layer. You'll get some tensor. And that tensor probably has a very rich understanding of what's going on in that image, all the objects, and features, and things like that. And then we can just simply connect that.

We can think of it as a smart representation of our input. We can connect it to just a little hidden layer. And then we have a little sigmoid which tells you handbag or shoe. We can just run this network. And so since the outputs to the hidden layer now are not raw images anymore, but this much higher level of abstraction that ResNet has learned, hopefully it can get the job done with hardly any examples.

And now you can get fancier. That's the basic idea, but you can get much fancier. You can connect up headless ResNet directly with our little network with a hidden layer and the final thing. And the whole thing can be trained end to end. But when you do that, you must start the training with the weights that you downloaded with ResNet because that is the crown jewel that's been learned. So you'll want to start from there. And you will do this in homework 1.

By the way, these pre-trained models are available all over the internet. There is the TensorFlow Hub, the PyTorch Hub, and then there is the Hugging Face Hub. When I checked it on the 13th, yesterday, it had over half a million models available for download, half a million. I think last year it was 50,000 when I taught the course. Yes?

AUDIENCE: I was just wondering, doesn't this make you prone to adversarial attacks because the weights are known by everybody?

RAMA RAMAKRISHNAN: Yes. There is some adversarial risk. I'm happy to talk about it offline. All right. So that's what we have, so back to Colab. So that's what we have. This is ResNet. And ResNet is all packaged up. It's available for download, so we download it here.

And you see here that I'm saying, include_top=False. So basically, you are telling Keras, the top, the very final layer of the thing, don't give it to me. Just give me everything up to but not including that. And of course, I think of it as left to right. People think of it as bottom to top. So the very, very top layer, don't give it to me. You're telling it so that you don't have to manually go and remove it.

And then I'm not going to summarize. Well, I'll just summarize the model to show you how big it is. 23 million parameters, ResNet. And I won't plot it because then I'll be scrolling for five minutes. So let's just do this now. So what we're now going to do is we're going to run all the data through this thing. And whatever comes out in that penultimate thing, I'm going to just grab it and store it. So that's what this thing does.

And now we create-- this is a little, handy function to do all these things. And once I do that, every image has been sent through ResNet, up to but not the final layer. And then whatever comes to the final layer, we are storing it. And then we're going to create a network where we'll only feed that layer, that information to a simple network.

So what is coming out of ResNet? You can see here 98 examples in the training data. And each example is now a 7-by-7-by-2,048 tensor. That's what came out of ResNet. And you saw that's what I did there. So that's what it looks like. Now let's just create our actual model now. We have our input, which is just a 7 by 7 by 224, 2,048.

We flatten it immediately, then we run it through a dense layer with 256 ReLU neurons. And then we use Dropout, which I haven't talked about yet, which I will talk about early next week. And we'll come back to it. Don't worry about this detail for the moment. And then we just run through a sigmoid and that's our model.

Finished, plot the model. This is what we have. model.summary, so on, and so forth. All right, good. Now let's actually train this thing. I'm just going to run it for 10 epochs because I tried running it previously and it seems to do a fine job with just 10 epochs. OK, it's already done.

It's so fast because we ran everything through this monster ResNet thing, and basically took all the output values and used them as a starting point. We don't have to run it every single time. So you can see here, the accuracy is quite high. Let me just plot it. Interesting. So the 10th epoch, something bad happened. So maybe I should have stopped at the 9th epoch. I didn't see this yesterday when I was running. So much for random reproducibility. So let's just run this. Oh, wow. Look, on the test set, it's achieving 100% accuracy. It's unbelievable.

OK, folks, now for the moment of truth. All right. I have a little code snippet here to capture stuff from the webcam. Because that last epoch it went down, I'm a little worried that the demo is going to flunk. But you know what? We all have to live dangerously, so here's a little function to predict what's going to happen. OK, now I tried it at home yesterday, by the way, and it's like, yeah, it's a handbag. OK, now let's just do something else. OK, any volunteers? I want a piece of footwear or a handbag.

AUDIENCE: I'll give a handbag.

RAMA RAMAKRISHNAN: That's kind of like a backpack, right?

[LAUGHTER]

I don't know. This feels like an adversarial example, but yeah, let's just try it.

AUDIENCE: Someone else?

RAMA RAMAKRISHNAN: No disrespect. Let me go with the shoe first. I have a better chance of it working. So it's a pretty big shoe. If it can't get this shoe, I'm worried about this model. All right. OK. Hold on, hold on, hold on. All right. Please don't get distracted by my hand. Capture. It's a shoe! Look at that!

[APPLAUSE]

Phew. All right, thanks. OK, now let's try that. I'm feeling brave now.

AUDIENCE: OK, yeah.

RAMA RAMAKRISHNAN: Thank you. All right, let's do this. All right. Camera Capture. OK. I'll put its better side.

AUDIENCE: Yeah.

RAMA RAMAKRISHNAN: It's a handbag! Look at that!

[APPLAUSE]

Phew. I swear, every time I do the demo, I age a few years. All right, folks, I'm done. Thank you.