RYAN ALEXANDER: All right, so as this XKCD comic points out, in CS, it can be very difficult to figure out when something is just really hard or something is virtually impossible. And until a couple years ago, people thought this idea of image classification would be something that was closer to the impossible side. But with the advent of deep learning typology, we've made significant strides in image classification, and now the problem's actually quite practical.

So today we'll be going through how the process of image classification with deep learning works. So we're first going to talk about what deep learning is, and then we'll move into some of the image processing techniques that researchers use, followed by the architecture of the convolutional neural networks, which will be the main focus in our presentation. We'll also talk about the training process, and then go through some results and limitations of CNNs and image classification.

So what is deep learning? Well, the term is particularly vague, and it's purposely so for a couple of reasons. The first is mystery is always good for marketing. But the second reason is that deep learning refers to a pretty wide range of machine learning algorithms. They do have some commonalities. They all seek to solve problems of a complexity that previously, people thought only people could solve. So these are more sophisticated classification problems than traditional conventional machine learning algorithms can do.

So how do they go about doing this? Well, all of these deep learning programs tend to take all the processes that need to happen, and split them up. They've got different parts of their program working on different things, all while performing calculations, and then at the end, it all comes together, and we get a result. Of course, this isn't unique to deep learning, and lots of distributed systems decentralize their calculations.

But the key thing about deep learning is that every part is performing these calculations. The calculations are not simple calculations. They're not we'll do this one simple operation over, and over again on a lot of data, and then we'll get a result at the end. Each part is performing

some particularly complicated process on all the little parts before they come together.

So why is this architecture a good idea? Why did engineers come up with this sort of decentralized, multi-layered complex process? Well, we take the example of image classification. It turns out that the human brain does a pretty similar process. So here's the human visual system, and it's pretty much a hierarchical process. So you begin by moving from the retina into the first areas of the brain, and as the information gets processed, it moves from one region of the brain to the other, and each spatial element of your brain is performing an entirely different calculation.

For example, the v1 area over here is picking out edges and corners, and then over here, a couple steps later in v4, you're starting to group those figures together. And so the brain kind of operates in a way that is very similar to the way these networks operate. So let's talk about to classify a face. If I asked you guys how would you classify a face, what is the first thing you might do? Well, as I mentioned before, the first thing out brain does is it finds these edges. The first thing to do is identify where the face is versus everything else. Now, does anyone have any idea as to what we could with the next step? Julian, you have an idea?

AUDIENCE: Maybe you could group these edges together.

RYAN ALEXANDER: Right. We could maybe identify some of these features that we're working with. So these are things like noses, and lips, and eyes. And then what do we do after we have these individual features? Steve.

AUDIENCE: Well, maybe we can group some of those together.

RYAN ALEXANDER: Exactly. Yeah, we can organize them into what we know the pattern to be. We know that a face has to have two eyes, above a nose, and then above the mouth. So that is precisely what a neural network actually ends up doing, and we'll walk through the process of how it does this later on in the talk. But as you can see, the intuitive way that we classify a face, and the way our brains are wired to do it, is pretty similar to the way got these neural networks to operate.

So like I said, we're talking a lot about these convolutional neural networks. There are other types of architectures involved. Like we mentioned before, deep learning is a pretty wide variety of algorithms, but we're going to focus on these CNNs. To give you a precursor of how good these CNN are, this results from ImageNet competition. So the ImageNet

competition is basically exactly what it sounds like, a bunch of computer scientists get together, and see how many images they can correctly classify.

And their error rate was pretty high. Almost a third error rate over here in 2010, 2011, and then in 2011, the CNNs were introduced to the topic, and the error rate plummeted. As you can see over here in 2015, we've got a significant improvement in these ImageNet competitions. So clearly, the CNNs have been very effective, and it's definitely been something that is exciting in the field and happening right now. All right, so now we're going to move into image processing.

ISHWARYA ANANTHABHOTLA: OK, so Ryan gave us a nice overview of where we get this concept of neural networks, but let's take a time travel, and go into a quick history lesson. So suppose I had a chair, and I wanted the computer to classify this chair. I have some a priori knowledge about what sort of things make up a chair. So I might be interested in looking at arms, and corners of the chair, legs, things like that. So I would go ahead, and feature engineer my discovery scheme to be looking for specific things. So I'm going to talk about some techniques that are traditionally used.

For example, chairs, doors, these things have corners. So I might use an image processing technique called a Harris Corner Detector, where we basically look at large changes in intensity as groups of pixels move from an image to an image that indicate the presence of corners, and you can use common corners to say that OK, all of these images are chairs, or doors, or whatever. Similarly, I want to say I have a bunch of pictures of chairs of different sizes, but that they all must have so many corners or something. So typically, we use a sift algorithm to scale invariant feature transforms. It basically says that across different sizes, I still should be able to extract information about the placement of corners.

Another common technique that's used in image processing is what we call HOG, Histogram of Gradients. So basically, for example, in this image, if I want to say I want to find all the images that have faces in them, or consist of faces, let's say, I might come up with a template of a face that basically assigns gradients to groups of pixels that form an outline of what looks like a face, and then scan it across my sample images, and say OK, a face is present in this image. Obviously, there are some errors. A mead cap, and a logo back here have become a face, but this is the traditional approach.

But here's the problem, what if I don't actually necessarily know what features are the most

critical depending on the dataset that I get? I want the system itself to figure out what techniques to apply without having any a priori knowledge about the dataset. So this is exactly the idea of CNNs, the convolutional neural networks. We want the techniques to be learned automatically by the process, by the system. So if I'm trying to classify faces, I want the system to figure out that eyes, and ears, and nose, these are the most important things. Or if I'm trying to classify elephants, the ears and trunks are the critical features, without me having to say OK, we're going to do corner detection, so on, and so forth.

So this is the idea. So to be able to understand this process in greater detail, I'm first going to go into a little bit of math, and the idea is to present the most fundamental operation here, which is the convolution. So this is the formal definition of the two-dimensional convolution, and since we're working with images, we're only considering the two-dimensional case. So in a more graphical presentation, which is a little bit easier to understand than just seeing the formula, the idea is that we have a kernel, or a convolutional filter that we seek to apply on another image, and that extracts some information about that image that we can use to help us classify the convolution.

So assume that this is our kernel, or this is our filter, and suppose this-- oh, there it is. So suppose we're applying the kernel [INAUDIBLE] here to the image that's in green. So the idea is we want to slide this filter across the image, and what we're basically doing this is a succession of dot products. So at each placement on the image, we multiply the overlayed numbers, and the sum becomes the output image on the convolt output. So this is basically the way the process works. You probably notice that there's a reduction in dimension, and Henry will talk a little bit more about why this is.

Let me get to it, and then [INAUDIBLE] So let's see some examples of what information we get by applying the convolution. So you see the image of a tiger on the top left. When we apply a filter that's a low pass filter, basically-- it's a Gaussian-- then we get low spatial frequency information about this image. So basically, we blurred it, and this tells us something specific that we might want to learn. So the kernel actually looks like a two-dimensional Gaussian function that's been distributed across this three-by-three kernel.

Similarly, we might be interested in high spatial frequency information. So in this case, we're looking at sharp features. So horizontal edges or vertical edges. So a question for you is if I have this kernel, which of these outputs when this kernel was applied to the original image, which of these outputs do you think it produced?

AUDIENCE: The third one on the right.

ISHWARYA ANANTHABHOTLA: Yeah, that's exactly right, and it's probably pretty easy to see why that's the case, given that the numbers are all horizontal bands here. Lastly, we also may be interested in extracting information at a particular frequency. So we can take the difference of a high pass filter and a low pass filter, and add it to your frequency you can extract information about that as well. OK, one last helpful piece of information is that there's another way that you can think of the information that's learned at each stage because a convolution can also be thought of as a Fourier transform in the frequency domain. You can think of the image transformation that way.

So from an image perspective, what a Fourier transform is is basically a sum of a set of sinusoidal gratings that differ, say, in frequency, in orientation, in amplitude, and in phase. So you can think about the zebra image here that's actually a composite of different gradients that might look like this, and the Fourier coefficients would be how much of each of these pieces come together to make that final image.

So just to get a sense of what kind of information this could convey, we typically take a Fourier transformation, and break it apart into the magnitude and phase representation. So you see magnitude, and you see phase. So those images weren't particularly clear, but this is a really good example for this. So if we take the Fourier transform of all the horizontal text here, you see how the magnitude reflects this, and you can go back to the math to understand why it's reflected in a vertical marking.

And similarly, if I were to take that same image, and rotate it, and then ask for the Fourier transform, you see how that information is contained very clearly in the magnitude spectrum. So these might be things that a network would learn at each stage to try to identify this as a text, or as as body of text that's tilted one way or the other, so on and so forth. So with that, we can now go into what the actual architecture of a convolutional neural is.

HENRY NASSIF: All right, so as it was said earlier, in order to classify or detect objects, you actually need certain features. You need to be able to identify these features. And the way you can identify these features is using certain convolutions or certain filters. In many cases, we don't know what these features are, and as a result of that, we don't actually know what the filters are to extract these features. And what convolution neural networks allow us to do is actually determine what these features are, and also determine what the filters are in order to extract

these features.

Now, the idea for convolutional neural networks, or the idea for replicating how the brain works started in about 1960s or 1950s after some experiments by Hubel and Wesel. And what happened in these experiments, as can be seen here, is a cat was actually shown a light band at different angles, and the neural activity of the cat was measured using an electrode. And the outcome from this experiment show that based on the angle at which the light was shown, the neural response of the cat was different.

As you can see here, the number of neurons, as well as the neurons that were firing were very different based on the angle. So what you can see also here is really a plot of the response versus the orientation of the light. And what this has led Hubel and Wesel to is the idea that neurons in the brain are organized in a certain topographical order, and at each filter, it fills a specific role, and the only fires when its specific input is shown, or when the angle is show, or the angle is specified.

Now, the first step to actually replicating how the brain works in code is really understanding how the building block, the neuron, works. That's a quick reminder of 7012 here. So a neuron is actually a cell with dendrites, nucleus, axon, and a terminal. And what the neuron actually does is aggregate the action potentials or the inputs it gets from all the neighboring neurons that are connected to it through the timelines, and it sums these action potentials, and then compares them to a certain threshold that it has internally, and that would determine whether or not this neuron would fire an action potential.

And that very simple idea can actually be replicated in code. An artificial neuron looks very much like a natural one. So what you would have is a set of inputs. Here we have three inputs that are summed inside of a cell, or a neuron. The sum here is not just a regular sum, it's a weighted sum. So the neuron specifies some weight, which you can think of as how much it values the input coming from a specific neuron, and then the input is multiplied by its weight, and then the total sum that the neuron computes is then fed into an activation function that produces the output that the neuron then basically produces.

Now, what we just saw here is really a simple neuron, a single neuron. You can't really do much with just one neuron, so what you would do is combined these neurons in a certain topography, or in that case, we have a network with seven neurons organized in three different layers. And what you can think of that is really as one big neuron with 12 inputs, and

one output.

So for example, in the case of the chair that was previously mentioned, if you're trying to identify whether a specific image has a chair in it or not, these 12 inputs here could be some sub images, or some small areas of the initial image that you feed into the network, and the output here could be a yes or no. Whether the image has a chair, or doesn't have a chair. And that is really the concept behind convolutional neural networks, which we'll go into details in a bit.

So what each neuron would be doing in that case, is really just performing a dot product, which if you aggregate that with the dot products computed by each of the other neurons, you would obtain a convolution. So what we have here is three inputs. If the input, in that case, is an image or a sub image, then the inputs would be pixels. The weights that you would be using here would be the filter weights, which is the filter that you use in the convolution.

And then the sum here would be the dot product of the weights and the inputs, and that sum would be computed by a specific neuron in your network. Then, that would be the convolution step, and then that convolution step would happen at the first layer in the network. So you would be applying this to the input, but you also would be applying this at the second layer, and third layer. In that case, we're only showing what happens in the first layer.

The next step after the convolution would be the activation step. So the dot product computed here would be there's a function that would be applied to the sum, and then that function would produce the output of the neuron. And this is where the activation layer is. You also have another activation layer here, and then a final one here. What we just went through now are convolutions and activations, but this is not the only thing that actually happens in a neural net.

What we also have is a step called subsampling, which we will be talking about next. For now, we will dig deeper into the activation, and specifically, what activation functions to use. In that case, we can see that that's a neuron, and what the neuron is doing here is the weighted sum that we talked about, or the dot product. And then the output from this would be fed into a certain activation function. Common activation functions are sigmoid, tanh, or rectify linear unit, and we will go through each one independently.

So here, we can see the sigmoid activation function. So what this function essentially does is

map any input to an output in the range of zero to one, and it's defined as one divided by one plus e to the minus x. The other common activation function is tanh, and that's any input to an output between minus one and one. And then finally, would be the rectified linear unit, which maps an input to itself if it's positive, or to zero if it's negative.

Now, in theory, you could use any function as an activation function in your network, but that's not what you want to do in practice. You want your activation functions to be non-linear for one main reason, that the goal of the activation function is actually to introduce non-linearity in your system. And if all your activation functions are linear, then you would essentially be having a linear system, which prevents you from achieving the level of complexity that you would ideally want to achieve with a neural network.

And there's a formal proof for as to why you need non-linear activation functions. They don't all need to be non-linear, but you need to have a least a few non-linear activation functions in your network. And the proof is available in the appendix, or the link to the paper that has the proof. So after we've discussed what happens at the activation layer, now we want to talk about convolution.

So as I said earlier, an image is obviously a two-dimensional image, but we're using RGB images. So actually need three channels. So what this means is that an image is actually three-dimensional, and each 2D matrix represents one channel. One of them corresponding to R, one of them corresponding to G, one of them corresponding to B. So a 32 by 32 image would essentially be represented by a 32 by 32 by three matrix, as can be seen here.

So what happens at the convolution layer? So here we have a nice animation that shows what is happening at each convolutional layer. So assume we have a five by five by three filter. So what this is, essentially, would be doing is covering a certain patch in the original image, which is 32 by 32 by three. So what can see here is that for that five by five by three patch in the original image, we have a neuron that is actually performing the dot product on all the pixels in that specific patch.

So what is happening here is the pixel values, which in that case, we have five by five by three pixels, are being multiplied by the filter values, and this operation is being performed here. Then, after that dot product is performed, it's fed into an activation function, as can be seen here, and this produces the output of this neuron.

Now, this is what this single neuron is actually doing. It's just covering that area of the original

image. What you would have in a neural net is many neurons, each covering a certain area of the original image. And if you aggregate the output of all of these neurons, what you would be doing or performing is, essentially, a convolution on the original image.

And to formalize what happens here, or what's the output that's being produced from that operation, we can look at that from a more mathematical perspective. So if you have an input of size H1, W1, D1, and you're performing a convolution with a filter, then the output, W2, would be related to W1 with the following formula.

So W2 plus W1 minus filter width plus one, and the same formula applies for the height, and the depth would actually be the same because in that case, we're using a filter that has the same depth, or three, as the original image. So what this would produce in aggregate is if you have 28 by 28 by one neurons, each one performing a dot product on some pixels in the original image, the output would be an activation map of size 28 by 28 by one, and the output of each neuron would be one pixel in the activation map.

Now, if we go back to the points that we made earlier, one thing we said was that the reason you use a neural network is because you don't know exactly what features you want to extract, and you don't actually have specific filters that you want to apply to the image. So ideally, what you want to do is have multiple filters being applied to the first image, and perform multiple convolutions, and this is what you can do with multiple neuron layers.

So what we described before was just for one neuron layer. In that case, we can assume we have five different neuron layers, each one performing a different convolution on the original image. So in that case, we would have 28 by 28 by one neuron per layer, and then if we aggregate all these neurons together, we need to multiply it by five, and that would be the total number of neurons we have in that specific number.

So this actually leaves us with a pretty complicated system. It would have many parameters. The neurons have weights, the number of neurons is also a parameter So how do we actually formalize that? If we have an input volume of 32 by 32 by three, which is our original image, and a filter size of five by five by three, then the size of the activation map that would be reduced would be 28 by 28 by one. Then in that case, we also said we have five different neuron layers that perform five different convolutions.

Then the total number of neurons would be 28 by 28 by five, and then the weights per

neuron are five by five by three, which is 75. In that case, we're assuming that the neurons independently keep track of their own weights. This could be simplified to each layer having their own weight, which would tremendously reduce the number of parameters. But in that case, just to get an upper bound, this leaves us with a total number of parameters of 294,000. And this is just using a 32 by 32 by three image. You can think of this as a pretty small image. So if you have a bigger image, you will have many more parameters.

Great. So what we just saw now, and described, were convolutions, activations, and these steps happen sequentially in a convolutional neural network, specifically as can be see here. One step that also happens occasionally is subsampling, and we'll discuss that step in detail here. So there are two main reasons why you would actually subsample your input. One is to obviously reduce the size of your input, and your feature space, but also because you want to keep track of the most important information, and get rid of everything else that you don't think is going to be relevant to your classification.

And the common methods used in subsampling are either max pooling or average pooling. We will describe max pooling here. So what happens in max pooling is, essentially, you are dividing the image into different sub images, non-overlapping sub images, and you perform an at max operation. So in that case, if we consider two by two filters, we would split the image, which in that case is four by four. We'd split it into four sub images, and for each two by two square, we would take the maximum. In that case, for the first square it would be six, then eight, then three, then four.

And the reason that actually works is because what you want to do is really keep track of the response of the neurons that-- or the highest response produced by your neurons. In that case, for example, the first highest response in the first square is six, and that means that if you get that high of a response, it means that something has been detected in the image, or has been detected. And this is something you want to keep track of as you move forward in your network.

And although this moves around the location of pixels, because you can think of that as subsampling an image, it does keep track of the information you care about because you only care about the fact that something has been detected in the image. At this point, you don't really care about where it's located in the image, and you want to keep track of all the features that your neurons have detected in order to be able to eventually classify the input correctly.

So if you have multiple feature maps-- so in that case, if you have 224 by 224 by 64, what your subsampling operation would be doing is reducing the height and the width so the depth would remain unchanged. So in that case, you would go from 224 by 224 by 64 to 112 by 112 by 64, and that would be reducing your output size by a factor of four. And formally, what this would look like is if you have an input of size H1, W1, D1, the size of your output would be related to your input in the following ways. W2 would be W1 minus the pool width plus one. The same applies for H2, and the depth would remain unchanged.

So these are, essentially-- these are the steps that happen in a convolutional neural network. What you could be doing is repeating these steps on a certain number of times in your network. But eventually, you have to make a classification, and decide in our case, whether our image has a chair or doesn't have a chair. So how does that happen? So after you perform all these steps, there's a step that happens here that would allow you to make that prediction, and that step is usually called a fully connected layer, or a multi-layer perception.

And what this essentially is is layers that are very similar, or exactly the same as what you had before, except that every neuron in the layer is connected to all the previous neurons. So what it's allowed you to do is really consider everything you currently have about your input, or everything that's left about your input, and compute a dot product on that, rather than focusing on a subset subsample of your input like previous layers do.

In that case, if you're actually trying to classify your input into four classes, you would ideally have four different neurons in your output layer, each one corresponding to one of the classes that you have, and then you would perform the same operation as you would in a previous layer, compute the dot product, and then once you obtain the values at every neuron, you would perform a normalization operation on all the output.

This organization operation is called softmax, or normalized exponential, and what it does is really, put more weight on the highest value. And by computing the softmax at the output, you're able to compute the posterior probabilities, and allows you to make a more informed-- or basically make a classification decision on your input. Great. So that's everything. And now, the next step will be talking about back propagation.

ISHWARYA ANANTHABHOTLA: OK. So now that Henry has given us an overview of the entire architecture of a CNN, I'm going to quickly spend some time, and talk about standard preprocessing tricks and tips that people might use on the image dataset before they actually feet it through a neural net to

classify the images. So let's suppose we have a dataset x, and there are n number of data points in the dataset, and each point has a dimension, D. So they have D features per point.

So in this example, we use these graphs as an example. Basically, our original data here has just two dimensions, and it spans this range of values. So for example, if we want to center this data, what we would do is a mean subtraction. So we basically subtract the mean across all the features of all the points, and we basically center it, and you can see that transformation here. And then we might, again, go for normalizing the dimension so that you have it. The data points span the same range of values in both dimensions. So you can see that transformation, and how it's taken place here. And we just divide by the standard deviation to do this.

Something that's very commonly done is called PCA, or Principal Component Analysis. And the idea here is sometimes we have a dataset that has a very, very high dimensionality, and we would like to reduce that dimensionality. So basically, what our goal is is to project the higher dimensional space onto a lower dimensionality space by taking the subset of those features. And if you've seen a little bit of 1806 from linear algebra, the way we do this is by generating a covariance matrix, then doing the single variable decomposition.

And I'll gloss over the math now, but that's the idea. And you can see here how the original data spanned two dimensions. I would decorrelate it so that it spans a single dimension. And even with this data, you might want to ensure that it's widened, which is the same deal. You want the values to span the same range in both dimensions. So then you would just divide by your Eigenvalues to get the widened data.

This last bit is something that's very commonly done as a preprocessing trick, though people aren't entirely sure why it works very well, or that it really does help, but it's something that people do, and it's called data augmentation. So basically, if I have a dataset that contains a bunch of images of chairs, a bunch of images of tables, and then a bunch of images of say, trees, I might want to intentionally augment that dataset further by introducing a few variations on these same images. So I might take the chair image, rotate some, reflect it a few more, scale, crop, remap the color space, or just kind of have a process that does this randomly to create more variation on the same dataset.

And this is a good illustration of why this makes a difference. I've taken an image here of what looks like a waterfall or some spot of nature, and simply just inverted the colors. And

what I see, if I were to just see this image alone, it maybe looks like a curtain, or a bit of texture, or something. And the idea is even to a human perception, these images have two very different meanings, and so it's interesting to see what effect they would have on a neural network. And with that, we'll go over to image classification results.

ALI SOYLEMEZOGLU: So, so far we've seen how convolutional neural networks are built, and certain image processing techniques we can use on the input images to get them into formats that are there for the classification process, but so far, it seems a bit abstract. It's good to know how CNNs work, why CNNs work, but why don't we take a look at some of the practical results from CNNs, and what they're used for so that when you're done watching this lecture, you can go home, and try classifying images on your own time?

With that, let's first revisit the ImageNet competition. I hope you remember the graph at the bottom from the beginning of the lecture, where we used this to motivate the use of CNNs. CNNs came onto the picture in 2012, but the winning CNN from 2012 was used on the 2010 ImageNet competition as well, and it managed to bring down the top five error rate to 0.17, which is pretty much on the same level as how performed in the 2012 competition when it was first used, which was at 0.16.

So this just goes to show that these convolutional neural networks are the state of the art when it comes to image classification, and that's why we're currently focusing on that. But you might be wondering what the ImageNet competition exactly looks like, what the images looks like. So why don't we take a look at that. As you can see here, these are images from the ImageNet competition.

Underneath each image is a bold caption, which is considered to be the ground truth, or what the competition believes to be the correct classification of the image. Underneath that ground truth, you see a list of five different labels. Now, these five labels are produced by a convolutional neural network, and the different bars-- the different lengthened bars, some pink and others blue, represent how confident the CNN is that what it sees in that image is that specific label.

As you can see in certain examples, the CNN is pretty confident in that it has a correct answer. For example, when we look at the container ship, it's pretty confident that what's in that image is exactly a container ship. There are certain cases when it doesn't get the correct label on its first try, but it does have in its top five labels. For example, you can see grill and

mushroom.

Now, the funny thing about the mushroom image is that what it thinks the image should be classified as is agaric. And if you don't know, agaric is actually a type of mushroom, and in fact, it's a mushroom that image. And it make sense that their confidence levels are pretty much the same. Agaric is slightly-- it's slightly more complex that what it sees in the image is agaric.

But there are certain cases when the CNN fails to classify the image correctly in its top five levels. This will be registered as a top five error, as you just saw in the previous slide about the top error rate. One example here on this slide is cherry. Now, the ImageNet competition believed that this should be classified correctly as cherry, even though there's also a Dalmatian in the background. The CNN, on the other hand, is pretty confident that what it sees in this image is the Dalmatian.

But if you look at some of the other results within the top five, although it doesn't guess cherry at all, it does guess certain fruits that it may think look sort of like cherries like grape or elderberry. So the CNN does actually pick up on two different distinct objects within the image, but as a result of how it's built, or its training set, it ends up classifying it as a Dalmatian. But it goes to show you that CNNs could also be used not just as an image classification, but also as object detection, which we do not touch up on in this lecture at all. So I'm not going to go further into that.

Now, this is all fun and all, but what about some real world applications? So this is a study that they did at Google with Google Street View house numbers, where they used the CNN to classify photographic images of house numbers, as you can see here, of certain examples of these house numbers-- what they look like. So what the CNN was tasked with doing was that it was supposed to recognize the individual digits within the image, and then understand that it's not just one digit that it's looking at, but it's actually a string of digits connected, and successfully classified as the correct house number.

This can be quite challenging, even for humans sometimes when the image is quite blurry. You might not exactly know what the house number exactly is, but they managed to get the convolutional neural network to operate around human operator levels. So that corresponds to around 96% to 97% accuracy, and what that enables Google to do is that they can deploy the CNN such that the CNN automatically extracts the house numbers from the images

online, and uses that to geocode these addresses. And it's gotten to a point were the CNN is successfully able to do this process in less than an hour for all of the street view house numbers in all of France.

Now, you might asking where this could be useful for. If you don't have access a lot of resources to actually do this geocoding process where you match latitude and longitude to street addresses, then your only resource might be actually photographic images. So you actually need something, hopefully not human, but some sort of software that can do this successfully. And so this is, for example, a place in South Africa, a bird's eye view. Not sure if you can exactly see, but there are these small numbers on top of each of the houses. All of these numbers were extracted and correctly classified using this previously seen CNN.

Another example from robotics is recognizing hand gestures. So obviously, robots come equipped with a lot of different hardware. They can sense sounds, they can also capture images of their surroundings. And if you're able to classify what you see-- if the robots is able to classify what it sees, then it can actually act upon it, and take certain actions. That's why it becomes really helpful to successfully classify the images.

So this is what they did using hand gestures, where there were five different classes. Each class corresponds to the number of extended fingers. So a, b, c, d, the top row, corresponds to the same class. They all have two fingers sticking out. The bottom row has three fingers sticking out. So that's another class. And they get the error rate down all the way to 3%. So 97% of the time, the convolutional neural net correctly classified the hand gesture. And you can use these hand gestures then to give certain commands to a robot, and it can train the CNN to act upon something else besides hand gestures.

For example, if it's in some sort of terranean and you train it on certain images that you might find in nature, then it can take those classifications, and act upon it once it sees, for example, a tree, or some sort of body of water. It's all thanks to image classification. Now, obviously, gestures are not necessarily static. You could be waving your hand, and so that would require a temporal component. So it's not just an image you're looking at, but a video.

And so if follows that we can probably extend image classification into video classification. After all, videos are just images with an added component, specifically time. Obviously, the added temporal component comes with a lot of additional complexity. So we're not going to dive into any of that, but in the end, it comes down to the same thing. You extract features

from the videos, and you attempt to classify them using convolutional neural nets.

So why don't we look at a study done, again, at Google, where they extracted one million videos from YouTube, sports videos, with somewhere between 400 to 500 different classes, and they used CNNs to attempt to classify these videos. Now, they used different approaches. They used different approaches, different tests, different types of CNNs that I'm not going to go into.

But as you can see here, these are certain stills from these videos where the caption highlighted in blue is what the correct answer should be, and underneath it, the top five labels that the convolutional neural network producers. The one highlighted in green is supposed to be the correct answer. So you can see on all of these, it gets it within the top five, and for the most part, within the top two, and it's pretty confident when it does get it correctly.

Now, when I said that they use different types of classifieds, some of them were more stacked classifieds, where they were just trained on stills within these images, while others were what they called fusion ones, where they sort of add the temporal component by fusing in different stills from these photo images together. Now, the current accuracy rate-- the best one they've achieved so far-- has been around 80% accuracy within the top five label.

Now, 80% accuracy is nowhere near what we saw with the ImageNet classification, where in 2015, they had managed to get it up to 98% or 99% accuracy. But obviously, there's way more complexity involved in this. So it makes sense that it's not quite there yet. But it does provide a good benchmark, and something to improve upon in the future as well. Now, that being said, convolutional neural networks do come with certain limitations. They're not perfect. And so Julian will now talk about the limitations.

JULIAN BROWN: Thanks, Ali. So Ali talked about the ImageNet competition, and talked about how the recent winners have been convolutional neural nets. So before, the best was about 26% top five error rate, but now they've actually gotten it down to a 4.9% top five error rate, and that was-- the winner of that competition, the 2015 one, was actually Microsoft. They've got the current state of the art implementation, and so because it's the ImageNet competition, that means they can identify exactly 1,000 different categories of images.

So there are few problems, actually, with the implementation, or just in general with convolutional neural nets. So one of them is that 1,000 categories, well, it may seem like a lot-- ImageNet is actually one of the largest competitions-- that's not actually that many

categories. So it doesn't contain things like hot air balloons, for instance. So these things that children would be able to classify, the neural nets actually aren't able to, even in the biggest competition.

And each of these categories also requires thousands of training images, whereas you could show a child a couple examples of a dog or a cat, and they'd be able to, generally, get a feel for what a dog or a cat looks like. It takes thousands of images per category for the neural nets to learn, which means that the total number of images you need to train for the ImageNet competition is over a million. And so this leads to very long training times, even with all of the heavy optimizations that Ishwari was telling us about like how efficient convolution is, it still takes weeks to train on multiple parallel GPUs working together to train the net.

There's actually a more fundamental problem with neural nets as well. So here on the left, we have a school bus, some kind of bird, and an Indian temple. And all of these images on the left side are actually correctly identified by convolutional neural nets. But when we add this small distortion here in the middle, that doesn't change any of the images perceptively to the human, this actually causes the neural network to misclassify these images, and now all three of them are ostriches.

So that's a little weird. How does this work? How did we find those distortions. So here on the left side, we see how a neural network typically works. You start with some images, you put it through the different layers of the neural network, and then it tells you a certain probability that it is a guitar, or a penguin. So it classifies it, and so we can use a modification of that method by applying an evolutionary algorithm, or a hill-climbing or gradient ascent algorithm. We take a couple of images, and we put them through, it classifies it, and we see what the classification is. And then we can do some crossover between the images.

So we take the ones that look a lot like what we're training for in the guitars or penguins, in this case, and we take the features of those that identify very strongly as a guitar, and we combine those together in the crossover phase. This is for the evolutionary algorithm. Then we mutate the images, which is making small changes to each one, and then we re-evaluate by plugging it back in through the neural network, and only the best images, the ones that looked the most like a guitar or penguin, are then selected for the next iteration. And this continues until you get to a very high identification rates, even higher than actual images of the objects.

So using gradient ascent, these are some of the images that you could produce if you start with just a flat grey image, and then you run it through this algorithm. So here on the side, we have a backpack, and we can actually see the outline of what looks like a backpack in there. And over here, we have what looks like a Windsor tie right here, but all of these objects-- and perhaps, there are things in these other images, but they seem to be lost in the LSD trip of colors here. So that's kind of strange. That's definitely not how humans do it.

So let's try a different method. What if instead of directly encoding, which is where we change individual pixels, what if we change patterns in the images, like different shapes? Then this is the kind of output that we get. So in the upper left, we have a starfish. So you can see that it has the orange and blue of the orange in the starfish, and also the blue of the ocean of the environment that typical images of starfish are taken in. And you can also see that it has the points, the jagged lines, the triangles that we associate with the arms of a starfish.

But the strange thing here is that they're not arranged in a circular pattern. They're not pointing outwards like this, like we would expect of an actual starfish. So clearly, it's not latching onto the same large scale features that humans do. It's actually just looking at the low down features. Even though it's a deep neural network, it doesn't grab onto these abstract concepts like a human would. So the reason for this problem, or at least why we think neural networks aren't as good as humans at things like this is the type of model.

So a human would have more of what's called a generative model, which means if we have examples here, these dark blue dots, say, of lines, a few examples of images of lines, then we could construct a probability distribution, and say that images that fall somewhere in this region are lines. And over here, we have a few examples of giraffes, say, and so anything that falls in this region would be a giraffe.

And so if you had a red triangle in here, that would be a lion. But if the red triangle is instead over here, it actually wouldn't classify at all. We wouldn't know what that is. We would say that's something other than a lion or a giraffe, but neural networks don't work the same way. They just draw a decision boundary. They just draw lines between the different categories. So they don't say that something really far away from the lion class is necessary not a lion. It just depends how far away it is from the decision boundary.

So if we have the red triangle way over there, it's very far away from giraffes, and it's just

generally closer lions, even though it isn't explicitly very close to it at all, and that will still be identified as a lion. So that's why we think we're able to fool these neural networks in such a simplistic way or in such a really abstract way.

So the main takeaways from our presentation, and the salient points are that deep learning is a very powerful tool for image classification, and it relies on multiple layers of a network. So multiple processing layers. Also CNNs outperform basically every other method for classifying images, and that's their primary use right now. We're currently exploring other uses, but that's generally where it's at, and this is because convolutional filters are just so incredibly powerful. They're very fast and very efficient. Also back propagation is the way that we train neural networks.

Normally, if you were to train a neural network that has a lot of layers, there's actually an exponential growth in the time it takes to train because of the branching when you go backwards because each neuron is connected to a large number of neurons in the previous layer. You get this exponential growth in the number of dependencies from a given neuron. By using back propagation, it actually reduces it to linear time to train the networks. So this allows for efficient training. And even with back propagation and convolution being so efficient, it still takes a very large number of images, and a long time with a lot of processing power to train neural networks.

Also, if you'd like to get started working with neural networks, there are a couple of really nice open source programming platforms for neural networks. So one of them that we used for our pset was actually TensorFlow, which is Google's open source neural network platform, and another one would be Cafe, which is Berkley's neural network platform. And they actually have an online demo where you can plug in images, and immediately get identifications. So you can get started very quickly with that one. Thank you.