EPISODE 505

[INTRODUCTION]

[0:00:00.3] JM: A modern farm has hundreds of sensors to monitor the soil health and robotic machinery to reap the vegetables. A modern shipping yard has hundreds of computers that are working to orchestrate and analyze the freight that is coming in from overseas, and a modern factory has temperature gauges and smart security cameras to ensure workplace safety. All of these devices should be considered edge devices. Over the last decade, these edge devices have mostly been used to gather data and serve it to an on- premise server or to the cloud. The edge devices have mostly been dumb computers.

Today, as the required volumes of data and compute continue to scale, we look for ways to better utilize our resources including those resources that are sitting at the edge, and we can start to deploy more application logic to these edge devices. We can start to build a more sophisticated relationship between our powerful cloud servers and the less powerful edge devices.

The soil sensors at the farm are recording long series of chemical levels, this time series data. The pressure sensors in a centrifuge at a factory are recording months and years of data. The cameras are recording terabytes of video from self-driving cars, or also at the factories or the farms, I'm sure there're cameras there, and these huge datasets are correlated with labeled events. Like if you're on a farm, you have crop yields and you can start to correlate those crop yields with certain soil sensor data and start to build a model of what kinds of soil conditions yield the best crops.

With these large volumes of data and the labeled outcomes that were seeking, we can construct models for responding to future events. We can start to build learning systems that adjust the conditions of our edge environments to better suit the goals that we have in mind such as the crop yield example, and deeper learning can be used to improve these models. The models can be trained in the cloud and deployed to devices at the edge, and we are in the very early days of building this relationship between the edge and the cloud that will make sense for the next — Who knows? 5 years? 10 years? We're in the very early stages.

Aran Khanna is an AI engineer with Amazon Web Services, and he joins the show to discuss workloads at the cloud and at the edge and how work can be distributed between the two places and the tool that can be used to build edge deep learning systems more easily.

To find all of our shows about machine learning and edge computing as well as links to learn more about the topics described in the show in this episode and other episodes, you can download the Software Engineering Daily app for iOS or android. These apps have all 650+ of our episodes in a searchable format. We have recommendations and categories and related links and discussions around the episodes. It's all free and also open-source.

If you're interested in getting involved in our open-source community, we have lots of people working on the project and we do our best to be friendly and inviting to new people coming in looking for their first open-source project, or if they are familiar with open-source and they are looking to get even more experience, we've got all kinds of projects around recommendations and the iPhone app, the android app, our web front-end, which is softwaredaily.com.

We're trying to build a lot of different things, and if you're interested in web development or mobile development, you might like to check it out. You can find all the projects that github.com/softwareengineeringdaily. You can join our Slack or send me an email about how to get involved. We're really building a nice community, and it's great to see. So I hope you enjoy this episode. I hope you check out the apps at the open-source project, and thanks to Aran for being a guest on this episode.

[SPONSOR MESSAGE]

[0:04:33.5] JM: The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like .NET apps, Java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 p.m. on a Friday and then crossing your fingers hoping for the best. We've all been there. We've all done that, and that's where Octopus Deploy comes into the picture.

Octopus Deploy is a friendly deployment automation tool taking over where your build or CI server ends. Use Octopus to promote releases on-prem or to the cloud. Octopus integrates with your existing build pipeline, TFS and VSTS, Bamboo, Team City and Jenkins. It integrates with AWS, Azure and on-prem environments. You can reliably and repeatedly deploy your .NET and Java apps and more. If you can package it, Octopus can deploy it.

It's quick and easy to install and you can just go to octopus.com to trial Octopus free for 45 days. That's octopus.com, O-C-T-O-P-U-S.com.

[INTERVIEW]

[0:06:03.8] JM: Aran Khanna is an AI engineer at Amazon Web Services. Aran, welcome to Software Engineering Daily.

[0:06:09.8] AK: Thank you so much for having me, Jeff.

[0:06:11.3] JM: Today we're talking about deep learning at the edge, and many people probably don't really know much about this space, so let's start with an explanation of edge devices and some explanation of deep learning and then we'll get into more technical concepts. What is an edge device?

[0:06:30.6] AK: A super high level, and an edge device is essentially any device that sits kind of outside of the data center. So anything from really low powered microcontroller unit, MCU devices, all the way up to maybe desktop computers or even whole boxes that sit on-promises but aren't necessarily connected to a larger data center or a data warehousing system.

[0:06:52.4] JM: Okay. A phone might be an edge device?

[0:06:55.2] AK: Correct. The phone, an IoT devise, things like even self-driving cars now are considered edge devices. So really, anything that sits outside of the data center is an edge device.

[0:07:09.0] JM: We've been building applications with smartphones and cloud computing for many years at this point. What are the characteristics of the workloads that we put on phones, and what do we put in the cloud? How do we divide our work? Because I think this is the most analogous pattern. If we're talking about edge devices interacting with a big cloud, is the smartphone interacting with the big cloud. So in that interaction between the smartphone and the cloud, what are the characteristics of workloads that we put on phones and what do we put in the cloud?

[0:07:41.3] AK: Yeah. That's an interesting question, because I think a lot of that has been shifting recently. I remember when I got my first iPhone, kind of the original iPhone that came out, and the processing power on that, while impressive for the time, was really not that great. So most of the workload for kind of early apps was going to be done in the cloud, a lot of the state, a lot of processing was done in the cloud.

But in the kind of recent 5 to 10 years, we've seen this enormous explosion in the performance of the chips that we're putting into our phones. So a lot of the processing actually is now starting to happen on the device. So to characterizing some of the workloads, I think the biggest workload really on device right now or up until the kind of this deep learning on device has been gaming. So gaming on devise is a real-time use case. Large amounts of compute cycles are needed to render things like 3D games on your device, and that is something that through smartphone manufacturers and chips supporting graphics workloads, like gaming, they've kind of enabled us to actually push workloads like deep learning now down onto the device.

[0:08:52.9] JM: Let's go there. So when we start to have these machine learning workloads, these deep learning workloads, what changes? How does that change the relationship between the edge device, like a phone, and the giant compute capability of the cloud?

[0:09:11.8] AK: Yeah. So when we have these deep learning models, it is important to note that there isn't just kind of one class of deep learning models. They come in all shapes and sizes. They have a whole bunch of different tasks that they can essentially perform, and the key thing to note is that they all kind of boil down to a large number of multiply accumulate operations for inference, for actually predicting with these models. While originally the training kind of has to be in the cloud because of the scale of data and the number of compute cycles, you need to

actually train the models. You need to pass through a large data set for sometimes hundreds of iterations to actually get the model trained. But for prediction, for inference, while you still need a lot of compute power, especially for the large models, you can do 10 million+ multiply accumulates for a single kind of pass-through of a modern vision model. That kind of scale of computation is now actually possible on devices like your smartphone if you have modern smartphone in your real-time.

So the workloads now we're seeing are shifting a little bit from just in the data center where the training and inference is happening to the training happening in the data center where all your data's warehouse and all that compute power can be brought to bear on it for actually training up the model, and the inference happening now at the edge.

[0:10:38.7] JM: So the inference happens at the edge, meaning we are deploying models to our phones or our other edge devices, and that's because these devices are the places where the information is being consumed. Like you can have a camera that's trying to detect known bad actors, for example, and you want the model deployed to the camera, so the camera can quickly identify if a person is a bad actor or not.

Now, in that interaction, o like let's I've got a camera on-premise that is deep learning enabled, does that identification of like, let's say, a person walks by the camera. So that identification takes place entirely on the camera and then maybe does that data also get pushed up to the cloud for further processing?

[0:11:36.7] AK: Again, it depends on the parameters that you need to construct your system within. To zoom out a little, the four real reasons that people will move deep learning workloads from the cloud down on to the device, and one is privacy and security. So if your data can't leave the premises where it's captured. Two is latency. So, essentially, if you need to have a real-time response, so in the case of a robotics workload or a self-driving car, for example. Three is reliabilities. So your network up to the cloud might not always be reliable. So you need to robust to that and, still, you need to run these deep learning workloads, so you have to run them at the edge. The last one is cost. So if that channel is actually costly to use to send the data up to the cloud, you would to put your workload down at the edge.

Essentially, if you're in one of these regimes, if these are the parameters that constrain the system that you're building, you might want to think about putting your deep learning workload on the edge. So in the case of this image classification workload, so with the smart camera detecting bad actors, the idea would be if you want to warehouse your data in the cloud and have reliable connectivity, yes, you can do the real-time inference on the device, make those snap decisions locally, but then warehouse it up into the cloud either in real-time or in a batch kind of system. So you can even stream it to something like a Snowball Edge, which is essentially a big rack of encrypted discs that will sit on-premises that you can then ship once they're full up to AWS for ingestion. There are a lot of systems you can kind of build and it really depends the parameters that are constraining you.

[0:13:14.3] JM: I've done enough shows about machine learning and deep learning to know that this is one of these topics that is hard to do over a podcast, but at least like if we're talking about the highly technical aspects of deep learning. I'm trying to get myself comfortable with just talking in the — I'm trying to find the right balance of talking — Because this show is typically about technical topics, but deep learning gets very technical and very mathematical and it's also somewhat domain-specific. So I'm trying to find the right balance of describing things in terms that the generalist engineer would want to know whether or not they are planning to have their career be saturated with deep learning. With that preface, how would you describe a deep learning model?

[0:14:04.3] AK: Yes. So kind of the highest level, and I'm assuming that folks would be familiar with things like linear regression. I think of deep learning in a large way of essentially abstraction over models like linear regression. It's saying, "What happens if we can take one regression model and string it together with a log linear model and sting it together with X model or Y model," and kind of what you get out of the other end of that sort of logical progression is something that looks a lot like a deep learning model where you have these individual, what we call layers, which are essentially just functions that you can train, that you can learn, much like you learn to linear regressor over data all strung together. So you stack these layers up in a deep learning framework like Keras of MXNet TensorFlow or PyTorch, and that defines what we call a compute graph, and that compute graph is something that's actually differentiable, which means you can take the derivative, and why that's important is because that's fundamentally how we train these system.

If you kind of remember your basic calculus, if you take a derivative of a function, it spits out, especially a multivariable function, it spits out essentially a vector, which points in the slope of greatest dissent of that function. So which way is it going downhill the fastest at the point where you took the derivative?

Fundamentally, what we're doing is we're stringing together all of these simple functions, simple layers, and then throwing a lot of data at it and saying, "Hey, how will I set the parameters?" Essentially, let's look at a linear aggressor, it's Y equals MX plus B. How do I set that M, that slope, such that with this data, it fits the date of the best? But instead of doing it for a single linear regressor, we do it across all of these layers strung together, and that has some really nice properties, because it allows us to reason about data that's not just simple preprocessed feature points. It allows us to throw kind of anything, including maps of pixels or random sounds at the model, and because of how deep it is, because it learns a hierarchical structure when we do this kind of naturally, it allows us to actually determine some of the features without having to hand select them. It'll actually learn that," Oh, this kind of set of pixels looks like a curve, and if I put three of these curves together, it looks like an ear," for example.

[0:16:40.9] JM: If we boiled down deep learning to the simplest explanation, and we just described it as a regression, and you're trying to draw a line between — Let's say, you've got a bunch of points on a two dimensional Cartesian plane and you're trying to draw a line between the most number of points, draw a function that hits the most number of points, and you see this function, it gets easier to hit more and more points, the more and more dots you have on that graph, and that is why we want more training data, because the more training data you have, the more points you get on this graph and the easier it gets to draw a function that goes through those different points. I know that's a reductive explanation, but do you think that's a useful way of thinking about deep learning for a —

[0:17:35.2] AK: Yeah, I think that's super useful and very spot on. So much like with regression, the more data points you have, the more accurate your line kind of threw them that you draw can be. So kind of going without analogy, data becomes really the fuel of deep learning because, again, unlike a simple Y equals MX plus B line that you're trying to draw through the

data points, this is much, much higher dimensional. You're often putting in an image which is a map of, let's say 24 by 24 pixels. It's a gigantic vector in super-high dimensional space.

As the dimension of the space goes up, the amount of data you need to actually draw kind of an accurate high dimensional, let's say, line or hyper plane in this case through the space, that goes up. So data really becomes the fuel that's needed to train these deep learning models.

[SPONSOR MESSAGE]

[0:18:34.8] JM: DigitalOcean is a reliable, easy-to-use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user-interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU-optimized droplets, perfect for highly active front-end servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance, and the prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make $100 go pretty far on DigitalOcean. You can use the credit for hosting or infrastructure and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage and, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed and his interview is really inspirational for me, so I've always thought of DigitalOcean is a pretty inspirational company. So, thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[0:20:42.3] JM: If I understand correctly, the typical way to do this is, let's say, you want to build a deep learning model to deploy to a camera. Okay, you've got all the data in the cloud already, so you do the training of the initial model in the cloud where you have all that data stored already and you get a good model that's trained, and that model is essentially a compacted series of lessons that were learned from all that data, compacted series of functions, and then you've got that model and that takes up a lot less space than all of that data that you have in the cloud, and you can just send that model down to the devices that need to do classification, for example. Correct me if from wrong, but I think that's how you do it.

But then you have additional examples that are going to be seen by those devices that are sitting at the edge. So what do we do about that? Do we process the data at the edge and then continually update the models at the edge or do we have some system where we ship the — I think you mentioned this edge snowball. You could just collect all these data in a different device and then physically ship it to a data center and then maybe you can have this process where you update the model back again on the data center, and then you have this continuous deployment. Well, I guess it would be like a batch deployment of the models back to the edge.

Give me a sense of the learning and deployment system that people are typically using.

[0:22:20.3] AK: Sure. I think there — Again, it depends on the parameters of the system you want to build, but if you want one of these continual learning systems, you can definitely create one. The caveat being that you need to make sure that the data coming in, the new data coming in actually gets labeled. So again, one of the important things, especially in the vision scenario is that it's all supervised learning.

You actually have the points that are labeled that are actually what we call a ground truth sample. So those have to be there. So if you're continually classifying data and then ingesting it up into the cloud, at some point there you need to generate a label. You can you either do that automatically if that's the kind of system that can get feedback immediately from the decisions it makes at the edge, or you might have to use a service like Mechanical Turk or kind of another human-based labeling service to actually generate those labels of the new data coming in.

So that said, we have seen a lot of these systems in the wild, where people are going up into the cloud, training the model on an initial set of data, deploying it down to the edge, running their classifications, collecting the data, taking the data back up into the cloud and taking the same model they trained and throwing that new data at that model in a process we call fine tuning or it's often also referred to as transfer learning, where you can actually take a pre-trained model and take a whole set of new data that we assume is kind of an augmentation to your original data set and without having to go through the whole original data set again, you can actually o just take this new data and use it to train the model much more cheaply. Then it becomes kind of augmented in the way that it now knows about this new data that it's taken in and it can learn to make classifications over that as well.

[0:24:06.8] JM: Okay. So I guess the point that I had missed earlier was that you need your data to be labeled at some point, because, let's say, you've got your model that was trained originally in the cloud, you deploy it to your camera, your camera is detecting, let's just say, it's detecting red cars, and you have a bunch of cars that drive by the camera and those cars get classified as red car or not red car, and they get classified on the fly and maybe the camera can take some action based on whether it's a red — Maybe it takes a picture of all the red cars, takes an action, or all the things it thinks are red cars, but you can't use that data to train the model, because it's not actually been labeled yet. So you actually have to take that data. You can't just you use that data to update the model somehow. You actually have to find some way to label it. Is that right?

[0:25:02.5] AK: Right, or verify the labels that the model's placed on it. That is kind of a necessary part of the loop. You want to make sure that each data point you're feeding in is actually a pure kind of ground truth data point.

[0:25:14.2] JM: As I understand, this is actually a tremendous problem and it's not really been — There's no, like label. I mean, maybe there's some kind of labeling as a service company, but I think this is actually just a tremendous problem and you have people at these — Even at giant companies, they have a really complex system where they have to get Mechanical Turks, they have to get them to vote on these — It would send each picture that was a red — That the model thought was a red car. You would need to ship it to three Mechanical Turks and have

them vote on it and have — You take whatever two out of three people voted on it. The labeling process is not straightforward at all. Is that right? It hasn't been standardized, right?

[0:26:03.3] AK: Yeah. I mean, every single shop will maybe have their own labeling procedure. Again, it's also domain dependent. Depending on what you're labeling, because deep learning doesn't just work on images. It also works on, say, voice data or text to data. So you might have different pipelines that you set up if you're, say, trying to build a natural language understanding model where you have them annotating text versus an automatic speech recognition model where you have them transcribing audio files. Again, it's also very domain-dependent, but there's not really a standard procedure as of yet because of how heterogeneous this sort of process can be.

[0:26:47.7] JM: How much of a bottleneck is there on these people who are trying to build deep learning systems that are updating really quickly?

[0:26:56.8] AK: It depends, right? The systems that we have right now, largely, are trained off these large common data corpuses. So a lot of the image models end up being trained off of ImageNet, and because of how large these labeled data sets are, it's often actually quite easy to get started by taking this data, running it through a model, training that model, and then maybe taking a much smaller data sample, let's say, a thousand data points from the specific thing that you want to recognize and using that fine tuning or transfer learning procedure on top of the model train with this gigantic labeled corpus. There kind of ways to accelerate the process, but, again, we have seen it be a bottleneck, but it's not something that's insurmountable. You can get around it by using things like transfer learning.

[0:27:50.4] JM: Okay. Well, let's get into more of the mechanics. Amazon has adopted a deep learning framework called Apache MXNet as the framework of choice. Explain what Apache MXNet is and how it compares to the other popular machine learning frameworks?

[0:28:07.7] AK: Yeah. Apache MXNet is a fully open- source community project. Again, it's an Apache project, so a whole bunch of different institutions, from academic institutions like CMU and UDub, to to businesses, like Microsoft and Amazon actually contribute to it, and it's a deep learning framework. So what that means is it allows you to build these computer graphs, which

are essentially these layers of the deep learning model and it allows you to build these modular computer graphs and throw data at it for training and train these things across a whole number of different machines, often GPU machines, that will allow you to really accelerate the training process.

Apache MXNet is fundamentally a way of abstracting yourself away from the hardware and the low-level mathematical mechanics of training the deep learning model and allowing you to, at a high level, define the modular components that comprise it, define the data you want to train it with and kind of click a button and just watch it go.

[0:29:14.6] JM: And is there any comparison between other machine learning frameworks that's worth making here?

[0:29:20.7] AK: So the other machine learning frameworks, again, it's different tools for different tasks fundamentally. At AWS, we support all the major deep learning frameworks and machine learning frameworks, and Apache MXNet we found is the most open and one of the most performant. In fact, it actually scales linearly up to — I believe, our last test was 200+ GPU's in terms of training performance. That means is, essentially, while you're training, let's say, a big image net model in MXNet, you can throw up to 200 GPU's with it and get essentially a linear speed up with each GPU you throw at it. There's no real degradation of performance, and that something that's extremely exciting because of the data scales that are needed and the training time that is incurred in training these models. It often takes weeks or sometimes months for large image models to train. So this is something that is really exciting for scientists working with large models and large data sets.

Beyond that, we have kind of a broad breadth of support for actually interfacing with these compute graphs that you build. So we have an imperative interface which allows you actually act on the compute graphs as if it was a set of NumPy arrays, and we have a symbolic interface that allows you to build things layer-wise, kind of that nice high-level modular fashion similar to Keras or TensorFlow and a huge breadth of language support, so Julia, R, Python, C, Java, Scala kind of all support it.

[0:30:51.1] JM: So if I train a model in the cloud on a ton of data and then I deploy it to a device, what are the resource constraints? How much memory does it take? What are the other resource constraints that I need to keep in mind when I am deciding whether to deploy a model to — For example, like a Raspberry Pi that's running on maybe in an agricultural farm somewhere?

[0:31:18.3] AK: Right. So the constraints are maybe twofold. One is the memory constraint, and one is the processing time, the speed constraint. So if you need a model running in real time and it's a very large model, the Raspberry Pi might not be your best option, but let's say you can have something that will maybe take two, three seconds for inference, it doesn't have to run in real-time, then the bottleneck becomes the memory. Again, these models aren't one-size-fits-all. They're really slim models, like SqueezeNet, which actually comes in around 4.8 MB. Really tiny, really lightweight, and it gets pretty good accuracy on classifying images.

Then there's AlexNet, which is kind of the model that sparked this whole deep learning for vision revolution back in 2012. That's around 233 MB. Then there are models like VGG that can be up to 548 MB, and even models can go up to a gigabyte. Again, there's a huge range here. Again, all these models I just mentioned are for the same task, for the same image classification task on ImageNet and, really, what you're trading off there is accuracy than versus the number of parameters and the speed of inference that you want on your device.

[0:32:36.9] JM: So how do those — If I'm throwing away certain parameters and I'm losing some accuracy in order to make my model smaller so that it fits on a device, how am I testing that judgment or how do I decide, "Oh! My devices is small. Like it's not going to be able to perform with this model." Maybe you could give some description for how people — Because like — Okay. I think about Netflix, for example, and Netflix — Or I'm sure Amazon Prime Video, if we're talking about Amazon. But when they're thinking about which — They transcode of a movie into a bunch of different transcoding, like smaller ones, more resource-intensive transcodings, because if you're watching on your big screen TV, they want to give you one transcoding, and if you're watching on your mobile device, they want to give you another transcoding, and it's similar with these models. Like maybe if you're deploying to somebody's smartphone, you would prune the model and make it less accurate by sacrificing one set of things, and if you are deploying it to maybe a set of cameras that are more performant than your

smartphone, you would do something else. How do people decide how to reduce or play with their model's accuracy?

[0:34:03.6] AK: Yeah. So there are a couple things you can do. So one thing, as you mentioned, is you can just choose a different architecture for your smartphone. So let's say in the cloud, I want the same model that's running on my smartphone, but in the cloud I have so much more processing power. So I can go with the 1 GB VGG GG model, for example, which is a very large architecture. A whole bunch of parameters, and I know it's really accurate, but then down on the device, I might want to go with a 4.8 MB SqueezeNet architecture trained on the same data. That's one thing that people can do.

But on top of that, there are little tricks that you can do to actually deploy the same model without having to go through and train two separate models, actually train the same model, and then slim it down for deployment on to, let's say, a mobile device. So, for example, there are things like quantization where you can actually take a model that was trained with 32-bit floating point weights, and actually without throwing new data at it, without retraining it, bring it down to half precision, to FB16 or even into 8 precision, so that actually reduces the bit width of the model, reduces the size of the model, but because these deep learning models are so robust to noise. It's one of the really nice properties when you stack all these layers up and train them, they become really robust to noise. They don't fail very frequently when you inject noise into your training data, or even throughout the network, you can inject noise, and that's fundamentally what's happening when you're quantizing. The accuracy is actually not hurt that much by just naïvely quantizing the weights. That's one trick you can use to actually slim down the models for devices.

Then on top of that, there are tricks that you can use during training, like parameter sharing. So you can actually, instead of saying, "Layer one has one distinct set of parameters, and layer two or three has another distinct set of parameters." You can actually force the layers during training to share a set of parameters, and that actually will allow them to learn something the kind of works for both layers. It might reduce accuracy a little bit, but that actually will reduce the number parameters you're then shipping down to your end device. So there are a lot of small tricks you can do to actually reduce the number of parameters.

Another big one is weight pruning, which we have in MXNet as a training operator, which means that while you're training, you can actually start snipping away weights that are really small and you can actually get up to 80% sparsity we've shown most cases and not lose a whole bunch of accuracies. So 1% or 2% accuracy drop by pruning away 80% of the weights in the network while you're training.

[0:36:33.5] JM: I think one other place for savings that I saw you discuss in a talk you gave was, I guess, you can do something to reduce the cost of convolutions. Can you remind us what a convolution is and why it's expensive and how you can reduce the costs there?

[0:36:50.3] AK: Sure. In many image classification, image recognition tasks using deep learning, we use this thing called a convolution, which is essentially a sliding window over the image that we call a set of filters, and the sliding window of filters over the image is essentially applied to each patch and it gives this really nice property in the image recognition model of translation invariance, which essentially means that if your dog is in the bottom right of the image or the top left of the image, it doesn't really matter, because no matter where it's translated to in the image, this convolution window will pick it up and be able to actually recognize it.

This operation is actually quite expensive, because for each patch you slide over in the image, you actually have to multiply all three of the color channels, the RGB color channels, by each one of the filters, and there can be sometimes 3 to 10 filters. It's a really expensive set of multiplication operations. While it might not reduce the parameters like we were talking about earlier, it actually helps the speed of the model lot if you can either take this convolution and run it on the GPU where you can do a lot of these multiplications in parallel, or if you're in a constrained device, like a smartphone that only has a CPU, you can actually factor this convolution. You can do something like a depth-wise susceptible convolution, where you actually convolve, i.e., you apply the filters to each one of the color channels individually instead of to all of them and then apply another convolution to merge all the color channels.

Essentially, what this is doing is you don't really have to understand the mechanics of it. What it's essentially doing is reducing the number of multiplications that you really need to do to kind of approximate a whole convolution by factoring it out into these two separate steps.

[SPONSOR MESSAGE]

[0:38:48.8] JM: Your company needs to build a new app, but you don't have the spare engineering resources. There are some technical people in your company who have time to build apps, but they're not engineers. They don't know JavaScript or iOS or android, that's where OutSystems comes in. OutSystems is a platform for building low code apps. As an enterprise grows, it needs more and more apps to support different types of customers and internal employee use cases.

Do you need to build an app for inventory management? Does your bank need a simple mobile app for mobile banking transactions? Do you need an app for visualizing your customer data? OutSystems has everything that you need to build, release and update your apps without needing an expert engineer. If you are an engineer, you will be massively productive with OutSystems.

Find out how to get started with low code apps today at outsystems.com/sedaily. There are videos showing how to use the OutSystems development platform and testimonials from enterprises like FICO, Mercedes Benz and Safeway.

I love to see new people exposed to software engineering. That's exactly what OutSystems does. OutSystems enables you to quickly build web and mobile applications whether you are an engineer or not.

Check out how to build low code apps by going to outsystems.com/sedaily. Thank you to OutSystems for being a new sponsor of Software Engineering Daily, and you're building something that's really cool and very much needed in the world. Thank you, OutSystems.

[INTERVIEW CONTINUED]

[0:40:42.1] JM: Okay. We've outlined the relationship between the cloud and the edge. We've talked about some savings that you can use to reduce the size of your model if you're deploying it to an edge device that is less powerful. Let's talk about some actual deployments and what

they look like. If I am deploying this camera that's going to identify red cars versus not read cars, or maybe I am deploying a Raspberry Pi based system that detects the soil quality of a farm, and agricultural farm, describe some deployment that you've seen in more detail. What kind of cloud instances do we need? What kinds of devices do we need? What kinds of services do we need? Just give a description for maybe some prototypical example.

[0:41:37.4] AK: Sure. So a really nice prototypical example, and I think we talked a little bit about this, is essentially the vision use case where you have a bunch of cameras with enough compute power to run these models in real-time sitting down at the edge. A perfect example of this is the new Deep Lens device that we actually released at Reinvent about two weeks ago.

So the idea of the kind of system that you can set up is that you want to be able to run some sort of model. Let's say, in one case, a security camera that detects people and actually start streams up to the cloud and triggers alerts when people are detected in the shot. So you want to stream like that running at the edge, so in real-time you're getting that feedback, but you also want to start keep ingesting that data into something local that is able them to ship the data up to the cloud to re-label it and fine tune your model that you're deploying down to the edge. So kind of the prototypical system I've seen is something like AWS Greengrass running on the devices, which is essentially a way that we allow users from the AWS cloud console to manage fleets of devices at the edge over these lightweight MQTT channels that we set up securely back to the AWS cloud, and over these channels they can push things like lambda functions, which are just little Python scripts that run at the edge as well as serialized model files.

So, essentially, what a user will do, will take a serialized pre-trained model file, like a person detector that they've trained on Pascal VOC. They'll bundle that with some Python code. They'll hit a button in their AWS cloud console and it will be deployed over Greengrass down on to something like their Deep Lens, which then runs that model in real-time on the device, and using the outputs from that model in the Python code that users define will send alerts back to the cloud over that MQTT channel and also warehouse that data locally to eventually send in batch backup in the cloud. That sort of system kind of is set up entirely from your cloud console with devices running Greengrass such as AWS Deep Lens.

[0:43:47.8] JM: Let's say I'm going to deploy this deep — I'm going to put this Deep Lens camera throughout a shipyard, and I want to put some of these cameras in places where the shipyard does not have a Wi-Fi connection. What can I do to get the information between the Deep Lens camera and my cloud if I don't have a Wi-Fi connection in these places?

[0:44:13.1] AK: So there are a couple things. The first is you can store the information locally and go and manually extract it from the cameras. That's one kind of simple step that you can take. We have an SD card slot on the camera and you can just pop out the SD card, pop in a new one and take the data.

Another thing that's really exciting about Greengrass is it allows your local devices to coordinate with each other So you can actually start to form a mesh network to get your data off these cameras if on one camera there isn't an immediate Wi-Fi connection. So things like that become possible. Beyond that, there are steps to actually go and wire the cameras together, but that can often be expensive, and the mesh networking scenario is actually much, much easier to set up.

[0:45:00.6] JM: What if you had the people that work in the shipyard, you force them to put an app on their phone that fits into the mesh network as well so they can just have like — Their phones can be the like gossip nodes between the different disconnected devices in the shipyard. Does that pattern happen at all?

[0:45:20.6] AK: We haven't seen, or at least I haven't seen it, but that's totally feasible. Any sort of mesh network, you can have them set up their phones to be repeaters or carrier around little repeaters with them, but that's all kind of possible and enabled with Greengrass.

[0:45:35.5] JM: All right. That's pretty cool. Yeah, let's talk about that Greengrass, AWS Greengrass. This is a tool that lets you run local groups of devices. Can you talk about like what that actually means?
What does Greengrass give you?

[0:45:50.7] AK: Yeah. Greengrass kidn of sits on top of our AWS IoT service, which is essentially a set of secure MQTT channels that we can set up from devices backup to the cloud

and among devices at a local network. What Greengrass gives you is essentially a really nice suite of management tools on top of that and a way to locally coordinate devices AWS kind of IoT things in a local network, even if they're disconnected from the cloud.

So one big thing that Greengrass lets you do is actually it lets you write lambda functions, pieces of Python code up in the cloud and then deploy them down on your devices to run kind of either as a pined lambda function, which means it's running in a continuous loop if the device shuts off and comes back up. This lambda function will still be running, or run kind of one-off actions on the device. You can employ down the lambda function to make a quick action on the device and then sit dormant on there.

So it allows you to essentially really easily, with kind of Python code, manage logic on these IoT devices down the edge, and with the newly announced Greengrass ML inference service, that is — I think it was announced last week at Reinvent. The Greengrass suite of services also now allows you to deploy model files down to the device [inaudible 0:47:07.1] MXNet models to the device, and that actually allows you to then manage your entire inference stack on the device, which is a pretty difficult thing to do if you don't have kind of an easy management service like this to handle the large model files that could potentially be pushing down as well as setting up all of the interfaces to the low-level device primitives that frameworks like MXNet TensorFlow or Café need to actually run these models for inference on the device.

[0:47:42.9] JM: Sorry. Can you explain that in more detail? You've got like a large model the you want to deploy and — I'm sorry. How does Greengrass help you in that situation?

[0:47:52.9] AK: Yeah. So with Greengrass ML inference, you can actually add this model asset as part of your deployment alongside your lambda function, and the lambda function can also be provisioned with things like MXNet. So instead of having to go down into your device and manually you'll, say, pip install MXNet, and the right version of MXNet with the right configuration for that device, the Greengrass service will actually come with a kind of prebuilt instance of MXNet that can deploy down on to the device and run that serialized model that you're pushing down for inference.

[0:48:28.2] JM: I see. We've done a bunch of shows on these AWS lambda functions and other serverless functions. How does the AWS lambda tool — These are functions as a service, basically, these one-off functions that you can deploy and they'll be run on demand. How do lambda functions fit into this environment of Greengrass IoT machine learning stuff?

[0:48:53.9] AK: Right. The lambda functions that we're talking about in the Greengrass setting are obviously a little bit different from the ones that run in the cloud, but the general concept and the semantics are the same. The lambda functions are fundamentally a piece of code, a Python code, a function as a service, but instead of being run on our AWS device fleet, it's actually being run on one of your devices down at the edge, and there are a couple of other things because this is an edge device that you can do with these lambda functions that you can't really do with the lambda functions that you're familiar with from the AWS cloud. So you can actually access things like local devices. So you can access your local microphone or your camera on this device. Say, if It's like a Raspberry Pi, you can access the pi camera.

Then on top of that, you get things like pinned lambdas that can actually gushy run forever on the device and be robust to device restarts and device failures and things like that. So when the device comes back up, this lambda functions has now come up with Greengrass and is running again. The semantics kind of look the same for the cloud, but obviously it's a little bit of a different paradigm.

[0:49:56.7] JM: We explored this deep Lends video camera a little bit, this video camera with deep learning functionality built-in. I'd love for you to go into a little more, like maybe some speculation about what other hardware devices would make sense to custom build with deep learning in mind.

[0:50:17.4] AK: Yeah. So, I mean, maybe I can speculate too much, but I was actually one of the early folks on the Deep Lends project.

[0:50:24.9] JM: Oh, congratulations.

[0:50:26.2] AK: Thank you. I can speak a lot to that. The idea of a vision at the edge is something that's really natural with kind of the state of deep learning right now. Most of the

vision use cases we've heard from customers have been, “Hey, I have all these vision data collecting at the edge. How do I play with it up in the cloud,” or being collected at the edge, but I'm just having to throw it away because I don't even have the compute power or the bandwidth or I don't want to set up a whole snowball edge to bring it back up.

Kind of a natural solution is to actually bring a lot of those deep learning compute cycles down to the edge. Beyond that, what’s really nice about a device that can run deep learning kind of locally is that we can really use it as a teaching tool to help customers play around with deep learning and prototype with deep learning down at the edge, because while you could build a full production system with these things, there is definitely — With any piece of hardware, it was really targeted to come out as a developer platform, as a prototyping platform, and if you want to go to full production, there might be different parameters that you have to build your hardware with it.

That said, the idea of vision being kind of the initial use case is something that's really natural given the impact that deep learning and, more broadly, machine learning has had to kind of in the vision space in the last 5 to 6 years.

Other devices too, there’s — I know Google has an AIY audio kit. Deep learning has also started to make impacts in audio and ASR and NLP as well. These are all domains where there is a lot of value to be added, but vision is definitely the big one.

[0:52:04.9] JM: Yeah. I mean, kind of imagine little bits of deep learning being useful for everything. You can imagine deep learning being useful for your TV. Like what should the acoustic settings on your TV be? Well, they should probably be based on the room that your TV is situated in. You could say the same for any speaker system. How should your refrigerator manage power? Well, probably depends on the temperature of the house that you live in. We think of these as kind of like little things that are on the edge of usefulness, but — I don’t know, it seems practical to me.

There's this other service, AWS SageMaker. This was announced fairly recently. Can you explain what SageMaker does and how that fits into the machine learning workflows that people have been implementing for a while in AWS?

[0:52:59.5] AK: Yes. The idea behind SageMaker is it's essentially a managed service that helps you with kind of all the steps of the lifecycle of your model. So from the prototyping phase where you have usually data scientists working with an IDE, like IPython notebooks, which is something that you SageMaker offers Jupiter notebooks as a service, which is, essentially, this IDE that you can use to slice and dice your data, test out different models and come up with a good model for your problem. So from that kind of workflow to the actual training workflow and retraining workflow where we have these training containers that can actually automate a lot of the steps to ingest that new, let's say, label data, kickoff a training job, run it in the cloud and spit out a new model, and then all the way down to the deployment step. At lease the deployment step in the cloud where you can actually create these inference containers that then you can take that serialized model and then put it behind something we call an endpoint for serving, and that endpoint also allows you nice things, like kind of A/B test different models and run these little simulations over the models that you can put out as well as auto scale the models so you don't ever run out of compute power to run your workload.

[0:54:16.7] JM: To wrap up, I'd love to get your perspective on how fast this stuff is coming to market. We're seeing lots of great tools. We've been talking about some great tools that people could use to build IoT systems into their shipyard or their agricultural facility or their windmill or something, but are these kinds of — Are shipyards and other types of places, are they actually adopting this kind of technology, or do you have a vision for how our old world industry will start to adopt this kind of stuff?

[0:54:56.8] AK: . Yes. So a lot of our old industries are kind of seeing the light that, look, deep learning is making your labor and capital more efficient. So that's fundamentally what the role is playing in your business, and adoption is definitely starting. We've seen a lot of big players start to move in this space. There are some recent acquisitions that can speak to that such as the Blue River acquisition by John Deere, the acquisitions of a lot of the companies in the self-driving space by large automotive manufacturers. We've seen the adoption start to pick up. We've seen the trend kind of starting, and I think obviously we're going to see a lot more in the years to come, but it's there, it's happening and it's going to be something that really is shaping industry and, more broadly, our world in the next 2 to 4 years.

[0:55:46.8] JM: All right. Aran Khanna, thank you for coming on Software Engineering Daily. It's been fascinating talking to you.

[0:55:51.7] AK: Thanks so much for having me, Jeff.

[END OF INTERVIEW]

[0:55:56.9] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers because I talked to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and prove themselves, and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineering.com.

If you're listening to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. So send me an email at jeff@softwarengineeringdaily.com.

Thank you.

[END]