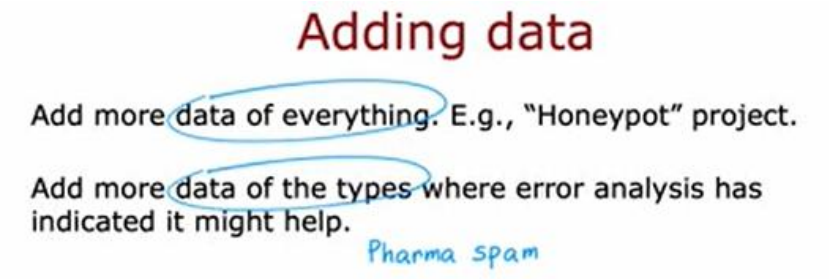
Adding data

In this video, I'd like to share with you some tips for adding data or collecting more data or sometimes even creating more data for your machine learning application. Just a heads up that this in the next few videos will seem a little bit like a grab bag of different techniques. And I apologize if it seems a little bit grab baggy and that's because machine learning applications are different.

Machine learning is applied to so many different problems and for some humans are great at creating labels. And for some you can get more data and for some you can't. And that's why different applications actually sometimes call for slightly different techniques. But I hope in this in the next few videos to share with you some of the techniques that are found to be most useful for different applications, although not every one of them will apply for every single application

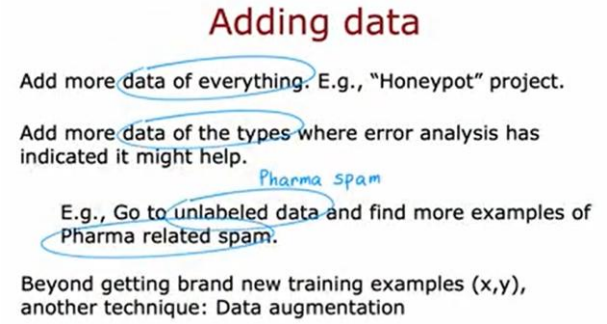
But I hope many of them would be useful for many of the applications that you'll be working on as well. But let's take a look at some tips for how to add data for your application. When training machine learning algorithms, it feels like we always wish we had even more data almost all the time. And so sometimes it's tempting to let's just get more data of everything. But, trying to get more data of all types can be slow and expensive.

Instead, an alternative way of adding data might be to focus on adding more data of the types where analysis has indicated it might help. In the previous slide we saw if error analysis reviewed that pharma spam was a large problem, then you may decide to have a more targeted effort not to get more data everything under the sun but to stay focused on getting more examples of pharma spam And with a more modest cost this could let you add just the emails you need to hope you're learning and get smarter on recognizing pharma spam.



And so one example of how you might do this is, if you have a lot of unlabeled email data, say emails sitting around and no one has bothered to label yet as spam or non-spam You may be able to ask your labors to quickly skim through the unlabeled data and find more examples specifically a pharma related spam. And this could boost your learning algorithm performance much more than just trying to add more data of all sorts of emails.

But the more general pattern I hope you take away from this is, if you have some ways to add more data of everything that's okay. Nothing wrong with that. But if error analysis has indicated that there are certain subsets of the data that the algorithm is doing particularly poorly on. And that you want to improve performance on, then getting more data of just the types where you wanted to do better. Be it more examples of pharmaceutical spam or more examples of phishing spam or something else.

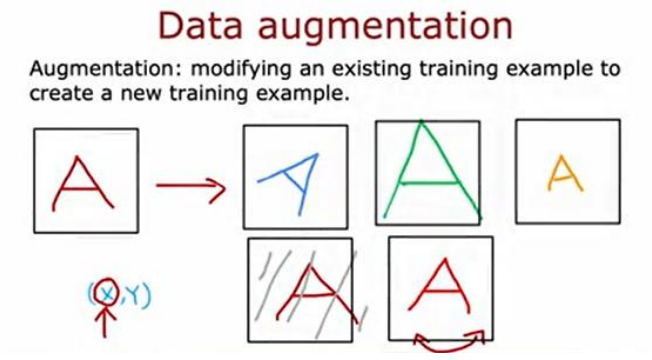


That could be a more efficient way to add just a little bit of data but boost your algorithms performance by quite a lot. Beyond getting your hands on brand new training examples xy. There's another technique that's widely used especially for images and audio data that can increase your training set size significantly.

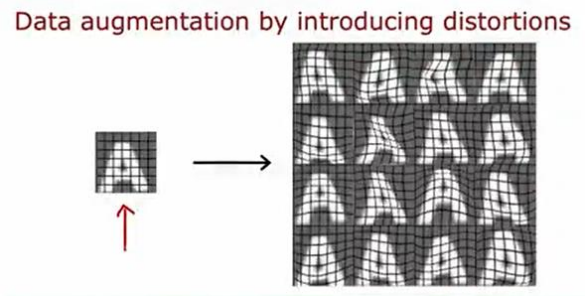
This technique is called data augmentation. And what we're going to do is take an existing training example to create a new training example. For example if you're trying to recognize the letters from A to Z for an OCR optical character recognition problem. So not just the digits 0-9 but also the letters from A to Z.

Given an image like this, you might decide to create a new training example by rotating the image a bit. Or by enlarging the image a bit or by shrinking a little bit or by changing the contrast of the image. And these are examples of distortions to the image that don't change the fact that this is still the letter A And for some letters but not others you can also take the mirror image of the letter and it still looks like the letter A.

But this only applies to some letters but these would be ways of taking a training example X, Y. And applying a distortion or transformation to the input And by doing this you're telling the algorithm that the letter A rotated a bit or enlarged a bit or shrunk a little bit it is still the letter A. And creating additional examples like this holds the learning algorithm, do a better job learning how to recognize the letter A.



For a more advanced example of data augmentation. You can also take the letter A and place a grid on top of it. And by introducing random warping of this grid, you can take the letter A. And introduce warpings of the leather A to create a much richer library of examples of the letter A. And this process of distorting these examples then has turned one image of one example into here training examples that you can feed to the learning algorithm to hope it learn more robustly. What is the letter A. This idea of data augmentation also works for speech recognition.

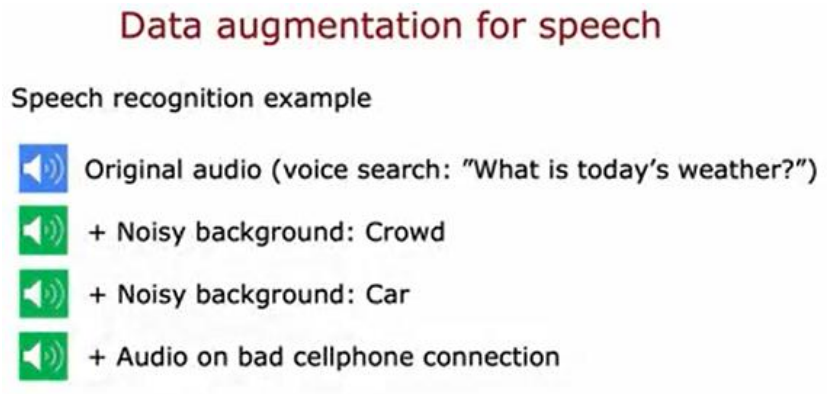


Let's say for a voice search application, you have an original audio clip that sounds like this. >> What is today's weather. >> One way you can apply data augmentation to speech data would be to take noisy background audio like this. For example, this is what the sound of a crowd sounds like.

And it turns out that if you take these two audio clips, the first one and the crowd noise and you add them together, then you end up with an audio clip that sounds like this. >> What is today's weather. >> And you just created an audio clip that sounds like someone saying what's the weather today

But they're saying it around the noisy crowd in the background. Or in fact if you were to take a different background noise, say someone in the car, this is what background noise of a car sounds like. And you want to add the original audio clip to the car noise, then you get this. >> What is today's weather. >> And it sounds like the original audio clip, but as if the speaker was saying it from a car.

And the more advanced data augmentation step would be if you make the original audio sound like you're recording it on a bad cell phone connection like this. And so we've seen how you can take one audio clip and turn it into three training examples here, one with crowd background noise, one with car background noise and one as if it was recorded on a bad cell phone connection.



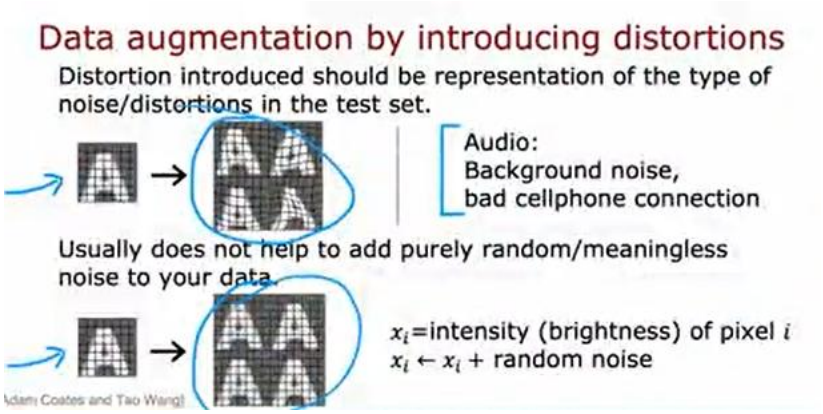
And the times I worked on speech recognition systems, this was actually a really critical technique for artificially increasing the size of the training data I had to build a more accurate speech recognizer. One tip for data augmentation is that the changes or the distortions you make to the data, should be representative of the types of noise or distortions in the test set.

So for example, if you take the letter a and warp it like this, this still looks like examples of letters you might see out there that you would like to recognize. Or for audio adding background noise or bad cellphone connection if that's representative of what you expect to hear in the test set, then this will be helpful ways to carry out data augmentation on your audio data.

In contrast is usually not that helpful at purely random meaningless noise to data. For example, you have taken the letter A and I've added per pixel noise where if Xi is the intensity or the brightness of pixel i, if I were to just add noise to each pixel, they end up with images that look like this. But if to the extent that this isn't that representative of what you see in the test set because you don't often get images like this in the test set is actually going to be less helpful.

So one way to think about data augmentation is how can you modify or warp or distort or make more noise in your data. But in a way so that what you get is still quite similar to what you have in your test set, because that's what the learning algorithm will ultimately end up doing well on. Now, whereas data augmentation takes an existing training example and modifies it to create another training example.

There's one of the techniques which is data synthesis in which you make up brand new examples from scratch. Not by modifying an existing example but by creating brand new examples. So take the example of photo OCR. Photo OCR or photo optical character recognition refers to the problem of looking at an image like this and automatically having a computer read the text that appears in this image.

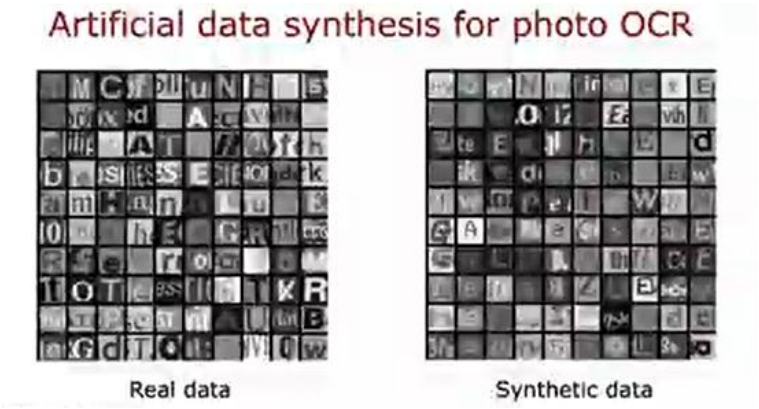


So there's a lot of text in this image. How can you train an OCR algorithm to read text from an image like this? Well, when you look closely at what the letters in this image looks like they actually look like this. So this is real data from a photo OCR task. And one key step with the photo OCR task is to be able to look at the little image like this, and recognize the letter at the middle.

So this has T in the middle, this has the letter L in the middle, this has the letter C in the middle and so on. So one way to create artificial data for this task is if you go to your computer's text editor, you find that it has a lot of different fonts and what you can do is take these fonts and basically type of random text in your text editor.

And screenshotted using different colors and different contrasts and very different fonts and you get synthetic data like that on the right. The images on the left were real data from real pictures taken out in the world. The images on the right are synthetically generated using fonts on the computer, and it actually looks pretty realistic. So with synthetic data generation like this you can generate a very large number of images or examples for your photo OCR task.

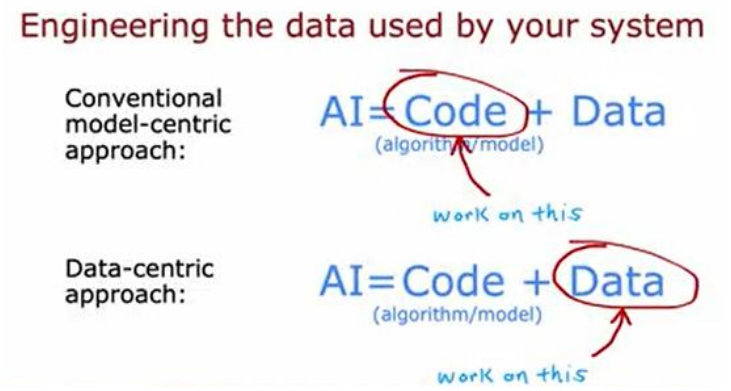
It can be a lot of work to write the code to generate realistic looking synthetic data for a given application. But when you spend the time to do so, it can sometimes help you generate a very large amount of data for your application and give you a huge boost to your algorithm's performance. Synthetic data generation has been used most probably for computer vision tasks and less for other applications.



Not that much for audio tasks as well. All the techniques you've seen in this video related to finding ways to engineer the data used by your system. In the way that machine learning has developed over the last several decades, many decades. Most machine learning researchers attention was on the conventional model centric approach and here's what I mean.

A machine learning system or an AI system includes both code to implement your album or your model, as well as the data that you train the algorithm model. and over the last few decades, most researchers doing machine learning research would download the data set and hold the data fixed while they focus on improving the code or the algorithm or the model.

Thanks to that paradigm of machine learning research. I find that today the algorithm we have access to such as linear regression, logistic regression, neural networks, also decision trees we should see next week.



There are algorithms that already very good and will work well for many applications. And so sometimes it can be more fruitful to spend more of your time taking a data centric approach in which

you focus on engineering the data used by your algorithm. And this can be anything from collecting more data just on pharmaceutical spam. If that's what error analysis told you to do.

To using data augmentation to generate more images or more audio or using data synthesis to just create more training examples. And sometimes that focus on the data can be an efficient way to help your learning algorithm improve its performance. So I hope that this video gives you a set of tools to be efficient and effective in how you add more data to get your learning algorithm to work better.

Now there are also some applications where you just don't have that much data and it's really hard to get more data. It turns out that there's a technique called transfer learning which could apply in that setting to give your learning algorithm performance a huge boost. And the key idea is to take data from a totally different barely related tasks. But using a neural network there's sometimes ways to use that data from a very different tasks to get your algorithm to do better on your application. Doesn't apply to everything, but when it does it can be very powerful.