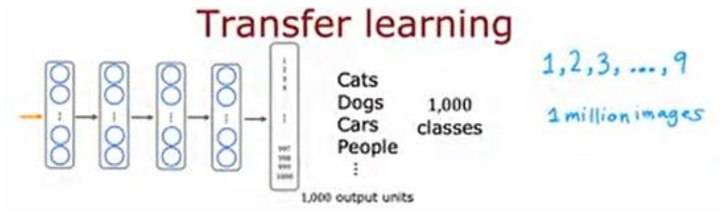
Transfer learning: using data from a different task

For an application where you don't have that much data, transfer learning is a wonderful technique that lets you use data from a different task to help on your application. This is one of those techniques that I use very frequently. Let's take a look at how transfer learning works. Here's how transfer learning works.

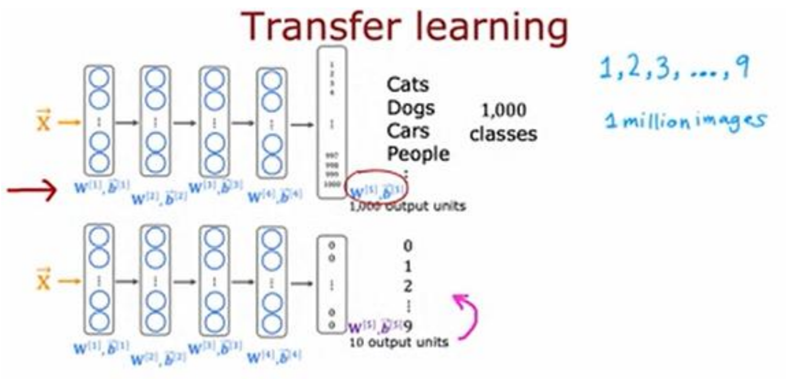
Let's say you want to recognize the handwritten digits from zero through nine but you don't have that much labeled data of these handwritten digits. Here's what you can do. Say you find a very large datasets of one million images of pictures of cats, dogs, cars, people, and so on, a thousand classes.



You can then start by training a neural network on this large dataset of a million images with a thousand different classes and train the algorithm to take as input an image X, and learn to recognize any of these 1,000 different classes. In this process, you end up learning parameters for the first layer of the neural network W^1, b^1, for the second layer W^2, b^2, and so on, W^3, b^3, W^4, b^4, and W^5, b^5 for the output layer.

To apply transfer learning, what you do is then make a copy of this neural network where you would keep the parameters W^1, b^1, W^2, b^2, W^3, b^3, and W^4, b^4. But for the last layer, you would eliminate the output layer and replace it with a much smaller output layer with just 10 rather than 1,000 output units. These 10 output units will correspond to the classes zero, one, through nine that you want your neural network to recognize.

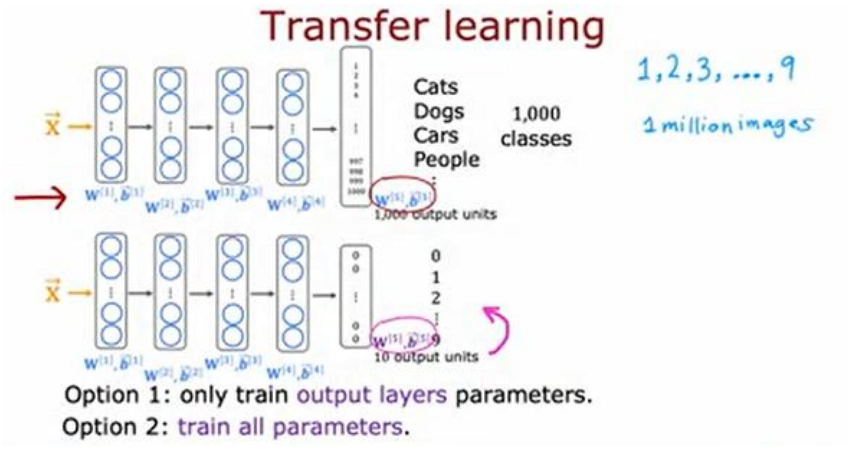
Notice that the parameters W^5, b^5 they can't be copied over because the dimension of this layer has changed, so you need to come up with new parameters W^5, b^5 that you need to train from scratch rather than just copy it from the previous neural network. In transfer learning, what you can do is use the parameters from the first four layers, really all the layers except the final output layer as a starting point for the parameters and then run an optimization algorithm such as gradient descent or the Adam optimization algorithm with the parameters initialized using the values from this neural network up on top.



In detail, there are two options for how you can train this neural networks parameters. Option 1 is you only train the output layers parameters. You would take the parameters W^1, b^1, W^2, b^2 through W^4, b^4 as the values from on top and just hold them fix and don't even bother to change them, and use an algorithm like Stochastic gradient descent or the Adam optimization algorithm to only update W^5, b^5 to lower the usual cost function that you use for learning to recognize these digits zero to nine from a small training set of these digits zero to nine, so this is Option 1.

Option 2 would be to train all the parameters in the network including W^1, b^1, W^2, b^2 all the way through W^5, b^5 but the first four layers parameters would be initialized using the values that you had trained on top. If you have a very small training set then Option 1 might work a little bit better, but if you have a training set that's a little bit larger then Option 2 might work a little bit better.

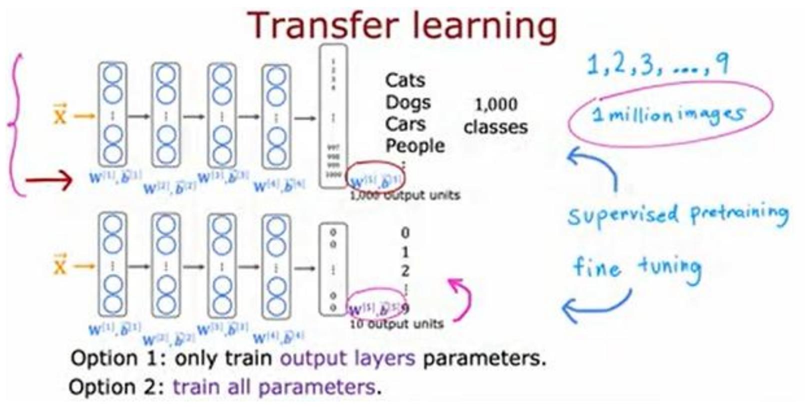
This algorithm is called transfer learning because the intuition is by learning to recognize cats, dogs, cows, people, and so on. It will hopefully, have learned some plausible sets of parameters for the earlier layers for processing image inputs.



Then by transferring these parameters to the new neural network, the new neural network starts off with the parameters in a much better place so that we have just a little bit of further learning. Hopefully, it can end up at a pretty good model. These two steps of first training on a large dataset and then tuning the parameters further on a smaller dataset go by the name of supervised pre-training for this step on top.

That's when you train the neural network on a very large dataset of say a million images of not quite the related task. Then the second step is called fine tuning where you take the parameters that you had initialized or gotten from supervised pre-training and then run gradient descent further to fine tune the weights to follow the specific application of handwritten digit recognition that you may have.

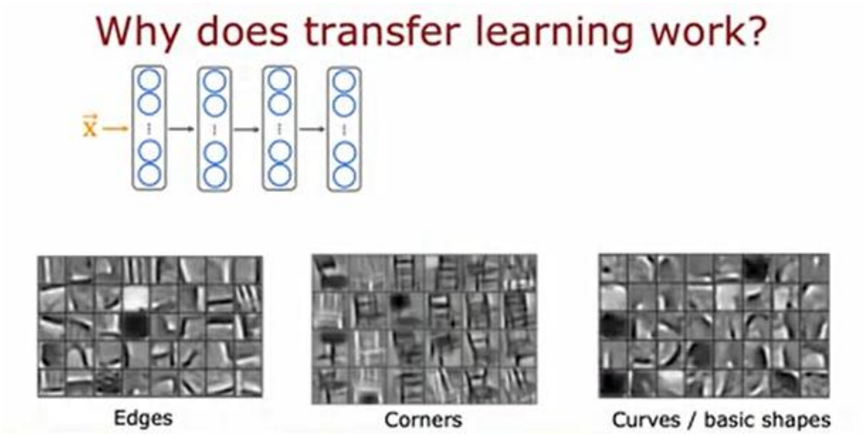
If you have a small dataset, even tens or hundreds or thousands or just tens of thousands of images of the handwritten digits, being able to learn from these million images of a not quite related task can actually help your learning algorithm's performance a lot. One nice thing about transfer learning as well is maybe you don't need to be the one to carry out supervised pre-training.



For a lot of neural networks, there will already be researchers they have already trained a neural network on a large image and will have posted a trained neural networks on the Internet, freely licensed for anyone to download and use.

What that means is rather than carrying out the first step yourself, you can just download the neural network that someone else may have spent weeks training and then replace the output layer with your own output layer and carry out either Option 1 or Option 2 to fine tune a neural network that someone else has already carried out supervised pre-training on, and just do a little bit of fine tuning to quickly be able to get a neural network that performs well on your task. Downloading a pre-trained model that someone else has trained and provided for free is one of those techniques where by building on each other's work on machine learning community we can all get much better results.

By the generosity of other researchers who have pre-trained and posted their neural networks online. But why does transfer learning even work? How can you possibly take parameters obtained by recognizing cats, dogs, cars, and people and use that to help you recognize something as different as handwritten digits? Here's some intuition behind it.



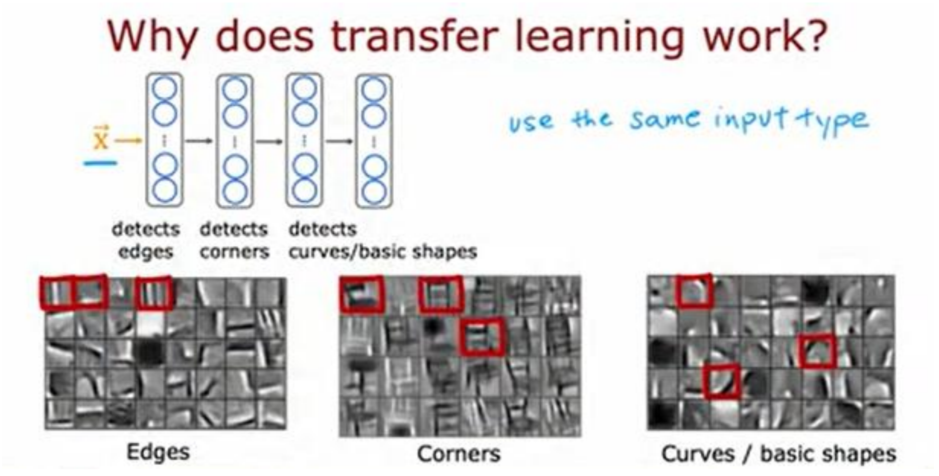
If you are training a neural network to detect, say, different objects from images, then the first layer of a neural network can learn to detect edges in the image. We think of these as somewhat low-level features in the image which is to detect edges. Each of these squares is a visualization of what a single neuron has learned to detect as learn to group together pixels to find edges in an image.

The next layer of the neural network then learns to group together edges to detect corners. Each of these is a visualization of what one neuron may have learned to detect, must learn to technical, simple shapes like corner like shapes like this. The next layer of the neural network may have learned to detect some are more complex, but still generic shapes like basic curves or smaller shapes like these. That's why by learning on detecting lots of different images, you're teaching the neural network to detect edges, corners, and basic shapes.

That's why by training a neural network to detect things as diverse as cats, dogs, cars and people, you're helping it to learn to detect these pretty generic features of images and finding edges, corners, curves, basic shapes. This is useful for many other computer vision tasks, such as recognizing handwritten digits. One restriction of pre-training though, is that the image type x has to be the same for the pre-training and fine-tuning steps.

If the final task you want to solve is a computer vision tasks, then the pre-training step also has been a neural network trained on the same type of input, namely an image of the desired dimensions. Conversely, if your goal is to build a speech recognition system to process audio, then a neural network pre-trained on images probably won't do much good on audio.

Instead, you want a neural network pre-trained on audio data, there you then fine tune on your own audio dataset and the same for other types of applications. You can pre-train a neural network on text data and If your application has a save feature input x of text data, then you can fine tune that neural network on your own data.



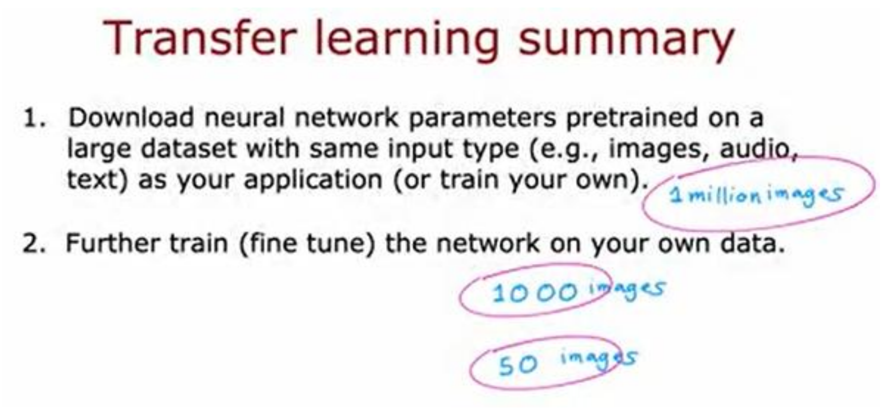
To summarize, these are the two steps for transfer learning. Step 1 is download neural network with parameters that have been pre-trained on a large dataset with the same input type as your application. That input type could be images, audio, texts, or something else, or if you don't want to download the neural network, maybe you can train your own.

But in practice, if you're using images, say, is much more common to download someone else's pre-trained neural network. Then further train or fine tune the network on your own data. I found that if you can get a neural network pre-trained on large dataset, say a million images, then sometimes you can use a much smaller dataset, maybe a thousand images, maybe even smaller, to fine tune the neural network on your own data and get pretty good results. I'd sometimes train neural networks on as few as 50 images that were quite well using this technique, when it has already been pre-trained on a much larger dataset.

This technique isn't panacea., you can't get every application to work just on 50 images, but it does help a lot when the dataset you have for your application isn't that large. By the way, if you've heard of advanced techniques in the news like GPT-3 or BERTs or neural networks pre-trained on ImageNet, those are actually examples of neural networks that they have someone else's pre-trained on a very large image datasets or text dataset, they can then be fine tuned on other applications.

If you haven't heard of GPT-3, or BERTs, or ImageNet, don't worry about it, whether you have. Those have been successful applications of pre-training in the machine learning literature. One of the things I like about transfer learning is just that one of the ways that the machine learning community has shared ideas, and code, and even parameters, with each other because thanks to the researchers that have pre-trained large neural networks and posted the parameters on the internet freely for anyone else to download and use.

This empowers anyone to take models, their pre-trained, to fine tune on potentially much smaller dataset. In machine learning, all of us end up often building on the work of each other and that open sharing of ideas, of codes, of trained parameters is one of the ways that the machine learning community, all of us collectively manage to do much better work than any single person by themselves can.



Transfer learning summary

1. Download neural network parameters pretrained on a large dataset with same input type (e.g., images, audio, text) as your application (or train your own). 1 million images
2. Further train (fine tune) the network on your own data.

1000 images

50 images

I hope that you joining the machine learning community will someday maybe find a way to contribute back to this community as well. That's it for pre-training. I hope you find this technique useful. In the next video, I'd like to share with you some thoughts on the full cycle of a machine learning project. When building a machine learning system, whether all the steps that are worth thinking about.