In this set of videos, we'll discuss Transfer Learning, which allow us to leverage already trained networks to make predictions

for new datasets. In this section, we're going to cover an overview of

transfer learning, starting off with the motivation behind transfer learning, as well as understanding

some guiding principles in regards to fine tuning our

transfer learning models. Now, generally

speaking, the earlier we are within our neural network, those earlier layers are going to be the slowest to train. This is generally going

to be in large part due to the way that our

weights are being optimized. So if we recall that

vanishing gradient problem, we recall that because

of backpropagation. By the time we get to the partial derivative in

regards to our earlier layers, it's very possible that we're not making any major

updates to our weights. But if we think about how our convolutional

networks actually work and we think back

to past lectures, those earlier layers are meant to represent only the most

primitive features, such as an edge. Now, our later layers,

on the other hand, are going to be capturing

those features that are particular to these specific

images in our dataset. Those features in

the later layers will build off of those

earlier primitive layers that we just discussed. These later layers

will also be much easier and quicker to train as it doesn't suffer from the

problems just mentioned from those earlier layers

in regards to how fast they can train

and we'll have a more immediate impact

on that final result. Just to motivate

this a bit further, any one of our Famous

Competition-Winning Models are going to be incredibly difficult to train from scratch. This due to the fact

that they're going to be trained on huge datasets. Huge datasets will obviously take much longer

time to train on. They're going to go

through a very long number of iterations to get to

that optimal answer. We saw just in that

last notebook, how long did it take for

a very simple model to learn the optimal

weights and that was, of course though on our

own personal machines. With that in mind, when we build these

award winning models, we also will need some very heavy

computing power to learn these patterns in a

reasonable amount of time. That's all assuming

that you've got your framework weight

on that first time, we will also have to spend time experimenting to get

those hyper-parameters, number of layers, what kind of strides we want to flatten, etc. Now, what we often see though, is that the basic

features such as those edges and simple

shapes learned in earlier layers of

the network will generalize fairly well

to any similar problems. If you just want to

store the results, It's just a matter of storing

those learned weights, not the actual lift

that was needed to learn those weights

in the first place. Our new idea will be to save those early layers of

a pre-trained network and then just re-train

the later layers for a specific application for

whatever our dataset is. This concept is going to be what we call Transfer

Learning of taking those earlier layers of a preaching network and then just retraining

those later layers. Now, let's walk through

a visualization of this concept of

Transfer Learning. What we have here in the

image in front of you represents our first trained

convolutional neural network with a number of

convolutional layers. Then that fully connected

layer ultimately leading to that final softmax classifier as very similar to what we

did in our last notebook. The idea will be to remove

that final output layer. Then we can use

what we learned so far or even go back further. For example, back one of the

fully connected layers or even further removing one of those convolutional

layers and so on. Then we can use that

pre-trained network and train only on that last

layer or last few layers, using those learned

earlier layers from the prior problem in order to make a

prediction on new data. Now, this idea is going to be more of an art than a science, and figuring out how long

to train that last layer, whether or not to

go back further and retrain more layers, and so on. In the next video, we're going to discuss some of the

options available to you, as well as some basic guiding

principles to keep in mind. All right, I'll see you there.

Let's start off with some different transfer learning options that are available to us. The additional

training that we do of a pre-trained network on

a specific new datasets. So those extra steps on top of that pre-trained

network is going to be referred to as the

step of fine-tuning. As mentioned earlier,

understanding exactly how to fine-tune

in regards to how much or how far back is going to require you to think through

a lot of different options. Should you just train

the very last layer? Should you go back a few layers, or even retrain

the entire network using that pre-trained network to just initialize

the weights for your new data and for

your new framework? Now, while there are

no hard and fast rules for fine-tuning your

transfer learning model, there are going to be

some guiding principles that you're going to

want to keep in mind. First off, the more

similar your data and problem are to the source data of your pre-trained network, the less fine-tuning

you'll have to do. For example, if you're using a pre-trained network

that was pre-trained on ImageNet to distinguish

between dogs and cats, you should need relatively

little fine-tuning. You don't need to go as far back, say in your model and use a lot of those pre-trained weights. That's due to the fact that

ImageNet was already used to distinguish between

different breeds of dogs and different

breeds of cats. Likely already has learned all of those features that

you're going to need. Also, the more data you have available about

your specific problem, the more the network

will benefit from longer and deeper

fine-tuning of your model. For example, if you only had 100 dogs and 100 cats in

your new training set, you probably want to do

very little fine-tuning. Maybe just remove

that final layer or two again, for example, and use a lot of those

pre-trained attributes that you learn,

from say ImageNet. On the other hand,

if you have 100,000 dogs and 100,000 cats, you may get more value from longer and deeper fine-tuning. Going back further or even

retraining the full network using that pass network to

initialize your weights. Also, if you're data is

substantially different in nature than the data the

source model was trained on, transfer learning may

actually be of little value. An obvious example is if a

network that was trained on recognizing type Latin

alphabet characters, it probably won't

do a good job in regards to helping you distinguish

between cats and dogs. But likely would be useful as a starting point for

recognizing, say, Cyrillic alphabet characters as they're both some

type of alphabet. To recap this idea of

transfer learning, we had an overview of

transfer learning, as well as providing that

motivation and understanding that it takes a while to

learn those smaller pieces, those lower level features such as an edge and you may want to actually take pre-trained

networks if you want to do something like image

classification and you don't have that

large of a data set. With that in mind, we discuss a few guiding principles in regards to fine-tuning

that model. You want similar datasets. You want to ensure that if you

have only a small dataset, not to do too much fine-tuning. If you have a larger dataset, perhaps you'd benefit from

doing even more fine-tuning. Now that closes out our lecture here on

transfer learning. In the next video, we'll show you how to actually

conduct transfer learning using Python. I'll see you there.

Welcome to our demo notebook

here on Transfer Learning. In this exercise, we're going to be using

the well known MNIST digit data set, which is just going to be a bunch of

handwritten digits between zero and nine along with their labels. So if there's a five written down

then that's labeled as a five, a zero written down

that's labeled as a zero. And we're going to use this data

set to illustrate the power and the concepts behind transfer learning. So we're going to train a convolutional

neural net on just the digits between five and nine. And after that we're going

to train just the last layer of the network on

the digits zero through four, and see how well the features

learn from five through nine. Those earlier features before that final

layer are going to be able to help classifying zero through four. So we're going to import

the necessary libraries, you should be familiar

with all these before. The only ones that are new is we're

importing the MNIST data set which is available in tensorflow.keras.datasets and

then we'll see how this is used later on. We're also importing the back end from

Keras and are importing that as K. We're then going to pull out this

now function and the reason for that is just we want to get the actual

timing and we can use a magic function within these Jupiter notebooks to get the

timing as well and we've done that before. But generally that tries to

compute a confidence interval and asked to do more than one loop through

all the data and it may take some time. So we're just going to lose the aspect

of having a confidence interval, but be able to do it a bit more quickly. We're then going to set

some of the parameters, so we're going to have the same batch size,

same number of classes, the same number of epochs each time, and

those are 128, five and five respectively. We're also going to set the pixel numbers

for the number of rows and the number of columns in regards to the pixels of our

image, and that's going to be 28 by 28. We're going to set our filters which is

going to be the depth of each one of our next layers using our

convolutional neural net. We're then also going to set the pool

size and that will just be the square. So it will be two-by-two in regards to our

maxpooling as well as the kernel size and that will be three by three as

we create those kernels as well. Now we're bringing out that

K that we mentioned earlier, which is just the back end. And we're saying four images depending

on your back end, when you pulled in this data set, it will either have

the number of channels of your image or that depth of your image first or last. So if you think about this as RGB,

then there would be a depth of three. So if the depth was first or

the channels were first, then using RGB it would

be three by 28 by 28. This is just the gray scale, so

there's only a depth of one, so it's one by 28 by 28, and

that's the dimensions of your image. Now if it's not channels first but

channels last, then it'll be 28 by 28 by one and this

is just ensure no matter your back ends, that you're going to be producing

the same results as we are here. Now we're going to create a function

in order to actually run our model in the same aspect. So we're actually going to be

pulling in a model that will set up the framework before actually passing it

through this function that we have here. We'll have our train set, which will be

our both our x train and our y train. And then so it will be a tuple and then

we'll have our test set which will be x test and y test so also a tuple

as well as the number of classes. So those are going to be the parameters

that we pass through this train model function. Now, recalling that this train that we

pass in is going to be both x train and y train, to define x train, we say we

want the first value from that tuple. So that's going to be x train and

not y train, and we're going to reshape that so

that it has the same number of rows. So if you think about pulling out

the x train and calling .shape zero, that's just going to give you how many

examples you have plus the input shape. And this is the input shape that we

defined up here which will either be the channels and image rows and

image columns or the image rows, image columns, and then the channels and this just ensures that we have

those in the right ordering. So I'm going to actually run the next

cell to make this a bit clear. So this is going to initiate all of our

values and we think about our x train, this is going to be our

first value in our tuple and then if I call shape That first value

is going to be the number of examples. So we say we just want 60,000

plus that input shape. So rather than being 60,000 by 28 by 28,

it's going to be 60,000 by 28 by 28 by 1. Or 60,000 by 1 by 28 by 28 depending

which one of these is true. Well then going to do the same for X test. We're then going to ensure that we're

working only with float values and then we will make sure all those

values are between 0 and 1. And that's again, just by dividing by 255. These are pixels will all be between 0 and

255. Well then print out the shape so that will be able to confirm that

the shapes are as we expected. And then we can see how many train samples

we have, how many test samples we have. Or then because we are using

classification of values between either 0 and 4 or 5 through 9. We're actually going to have to create

those categorical variables as we've done before. Create those different classes doing

something along the lines of 1 high decoding. So I do that for

a train set as well as our test set and then we call model.com pile. Whatever model we pass in,

we will compile it using this law. So we're using a different optimizer. I wouldn't worry too much about this being

different than what you've seen before. It's fairly similar to the math for

RMS prop that portion of your Atom, if you recall as well. But I wouldn't worry too much about it. We use this specifically so that it

wouldn't train quite as fast as something like RMS prop or Adam actually work. After you run this, you can try switching

this for Adam or RMS prompt and see that it actually gets to those

optimal values much quicker. We're then going to also track the metrics

of accuracy we call T equals now to get the timing. We then going to fit our model and fitting our model is going to be what

takes the most time on RX train RY train. That batch size that we specified earlier, the number of epochs we specified

earlier verbose equals 1. That's the defaults if you set

verbose equal to 0 rather than showing the steps throughout

each epoch that we've seen, every time we've run

those deep neural Nets. Those just would not show up, so that would keep all those

extra lines from showing up. I generally find those useful, but

if you see that they're taking up a lot of room on your screen,

feel free to set verbose equal to 0. And then we have that validation set to

see how we're doing on our holdout set throughout. And then to figure out how long it take,

we call now again, and we subtract that T that we initialize

here before fitting our model. We can then call model that

evaluate on x test and y test in order to get the scoring and

that will give us both our error and our accuracy for that model that we ran. So here we have initialized

our trained model. We're then going to get

the data that we need. So we're loading all of our data

calling M NIST.load data which will give us the extra in y trained to pool as

well as the X tests and y test to pool. We're then going to separate out

less than 5 and greater than 5. So that we have x train such that our

outcome variables are all less than 5 or are extra in such that they're

all greater than or equal to 5? We're then going to define the feature

layers and these are going to be those earlier layers that we hope

to transfer onto our new problem. And we're going to freeze these layers

during the fine tuning process. So these are going to be those layers

that we freeze if we think back to what transfer learning is and how it works, and

we're going to set these all to a list. So features layers are equal to this

list and it's going to have this convolutional layer with the filters and

colonels that we specified earlier. That activation than

another convolutional layer with another activation that

Max pooling some drop out. And then it's going to flatten

our convolutional layer until 1 dimensional array. And we pass this into list and we'll

see later on that sequential model can actually take in a list of those features,

so it has that add functionality. But also if needed, you can pass in all those layers as

a list interior sequential function. And then we're going to have the layers

that we're actually going to be fine tuning. And that's going to be this dense layer,

another activation layer, some dropout, and

then another dense layer to get it down to the number of classes plus

that softmax function. So I'll run each of these and

then we're setting our model equal to sequential again as mentioned, we can just

pass in that list of the different layers. Now we have our model. We can look at the summary. And we see here in the summary that we

have each one of those layers that we specified before in regards to our

different feature layers as well as those classification layers leading all the way

to the ends of our softmax function. And then we see that we have

a total number of parameters of 600,165 that we're going to have to train. Now that we have our model, we can call

that function that we created earlier, using that model. Then our training set,

then our test set and then our number of classes

which is still equal to 5. We're going to start running this and

this will take maybe 2 minutes, something along those lines. So I'm going to pause here and

once it's done running will come back and look at these results and

discuss these results. So we see here that took about 3 minutes

to train, 3 minutes and 12 seconds here. We can also look at the improvement

in the accuracy step by step going from 0.22 to 0.3,

0.38 and so on. In regards to our training set,

you look at validation set as well. But you see that it's slowly getting

that accuracy up at each step. An probably can continue to improve. Now our goal here with

transfer learning is going to be to freeze certain layers and

only train on those later layers. So Keras allows layers to be frozen

during the training process. In order to do what we just said, that is, some layers would have their weights

updated during the training process, while others will remain frozen and

they won't be updated. And this is going to be that

core part of transfer learning. You also want to note that

a lot of the training time and we mentioned this in lecture

is going to be spent back propagating the gradients

back to that first layer. Therefore, if we only need

to compute the gradients for a small number of layers the training time

should speed up hopefully quite a bit. So in order to freeze the layers, we just set each one of those layers which

are going to be for each one layers in our feature layers that we defined earlier

as those that we will ultimately freeze. For each one of those we just

said I.trainable = False. And that will freeze the training where it

is and won't allow for further training. Now when we look at

the model.summary we see that our total number of

parameters is 609,000. But the trainable parameters is

going to be less at 600,165. And that's going to be due to the fact

that we are freezing these upper layers in place. Now we do still have to train a lot but

again those being later layers, they'll be able to update a bit

quicker than those earlier layers. So now we can call train\_model. This time on the values less than 5, so if we recall going back up we originally

trained on the values greater than 5. We froze those layers and now we want to see our model that we again

just froze those first few layers on and are only allowing for

training on those final layers. Four values less than 5. So I run this and

again this will take some time, but hopefully faster than the last one did. And we will come back as soon as it's

done training in touch on our results. Now looking at the results that we have

here, we see that the total training time came down by 4 minute when

were only training 5 epochs. That's quite a bit. Also, we're seeing that towards

the ends we were getting higher overall accuracies both in our training set

as well as in our validation set. So we see how this power of transfer

learning allowed us to save time while gaining higher accuracy in

that short amount of time. Now just to closeout,

you want to flip these two steps. So rather than doing the ordering of first

on training on our greater than 5 and then freezing layers were going

to train on are less than 5, then freeze our layers and then run

our final model on greater than 5. So in order to do that We're going

to reset our feature layers that are going to be the same values, but

now we want to retrain them, set the right values trainable after training them

on a different data set, and then doing the same thing for our classification

layers, leaving those as trainable. And then we set that up equal to model 2,

so we have sequential with our new layers which are going to be the same

layers as before, just not trained yet. We look at the summary and it should

be the same steps, except now our total parameters and our trainable

parameters should remain the same. We can then call our train model function, this time starting off with the less

than five values, and we run this. And this will take some time to run, so we'll see you as soon as

this is done running. Now that it's done running we can

see the different accuracy numbers, so as the validation accuracy,

we can go through the same steps of freezing those trainable layers,

we look at the summary again. We see that same number

of total parameters versus the number of trainable parameters. And then we can again set our trainer

model called that train model function, and then pass in our new model model

2 with those layers frozen, and try to get that greater than 5 accuracy. We run this, and again this will take

a bit to run shorter than the last one. I'm going to pause it here and then we will look at

the results once it's complete. So as we see here now that the results

are completed, we are able to reduce the training time, but once we flipped

which one we were performing first. We didn't quite get the same

accuracy results that we had before, with a little bit less accuracy, a little

bit less axiom that validation set, and that can happen,

transfer learning is a bit of an art, takes a bit more of understanding where we

can fine tune, how deep we can fine tune. Another thing that we can keep in mind, is

the fact that each epoch is moving a lot faster, and we're getting

continuous improvement on accuracy. So we could even add on an extra epoch or

two, get that improved accuracy while doing it in less time than just

running it from scratch as well. Feel free to play around with training

different layers, going deeper back, seeing how we were able to work with

holding certain parts constant and so on. That closes out our notebook

here on transfer learn, and I look forward to seeing

you back in lecture. All right, I'll see you there.