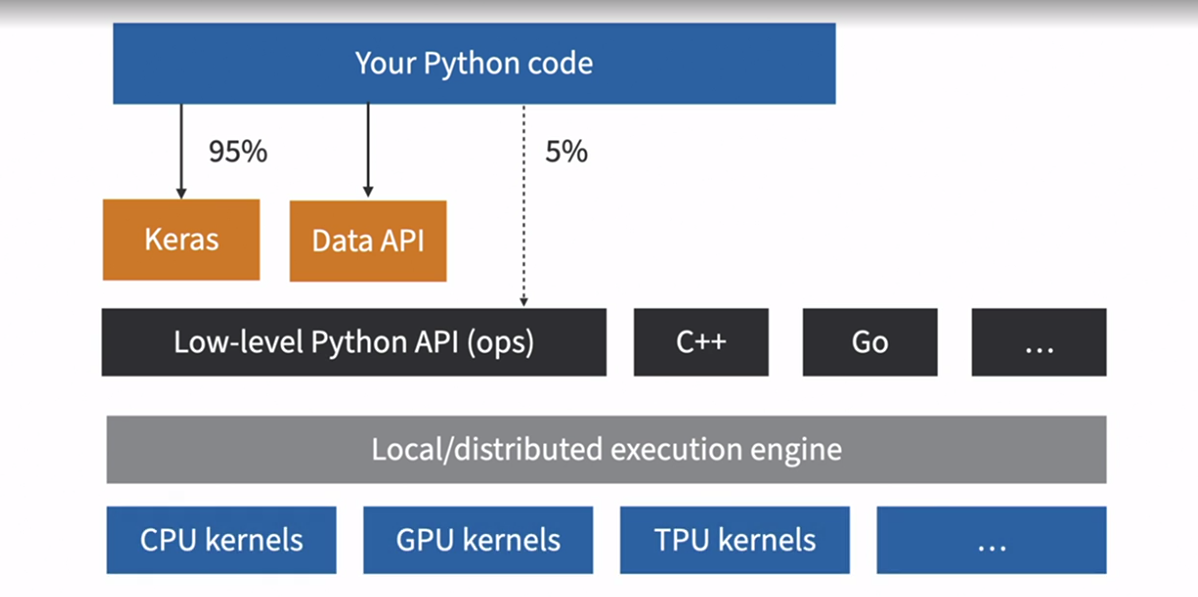
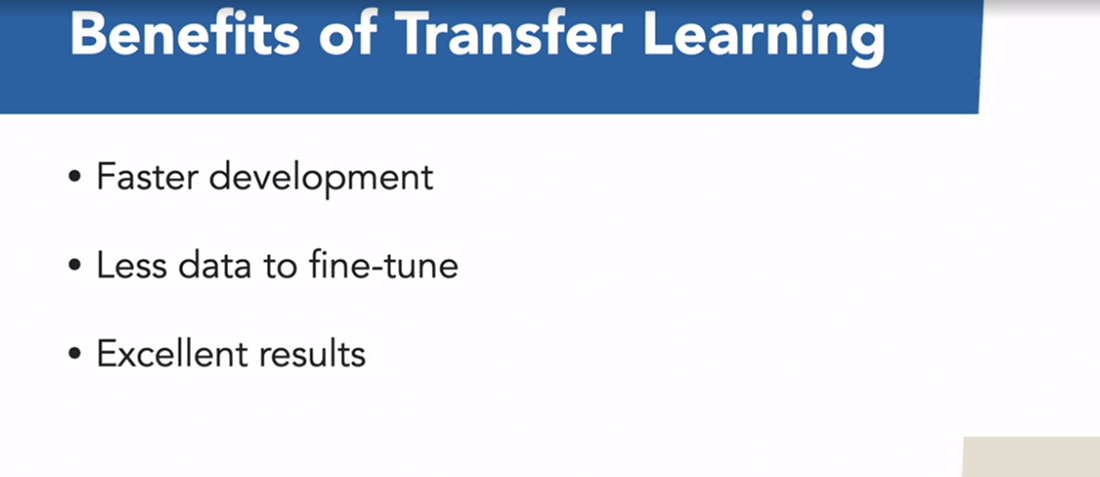
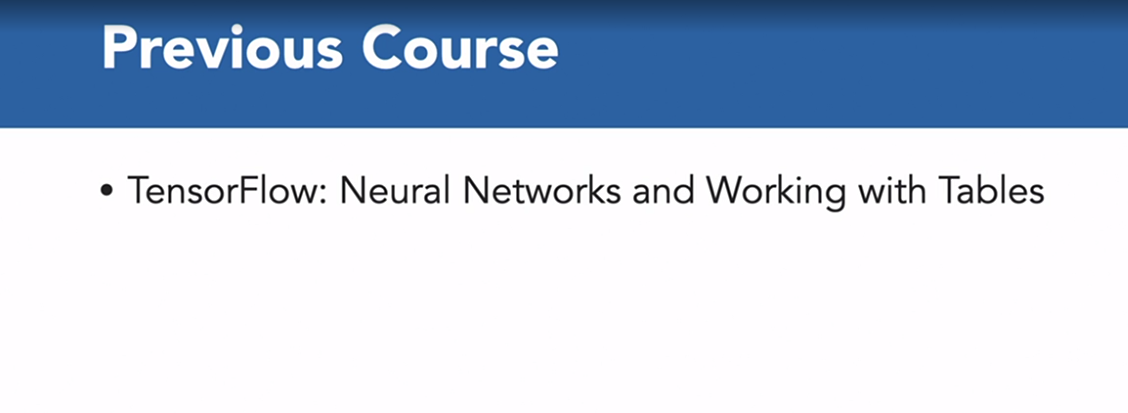
# **TensorFlow: Working with Images**

<https://github.com/LinkedInLearning/tensorFlow-working-with-images-3021324>





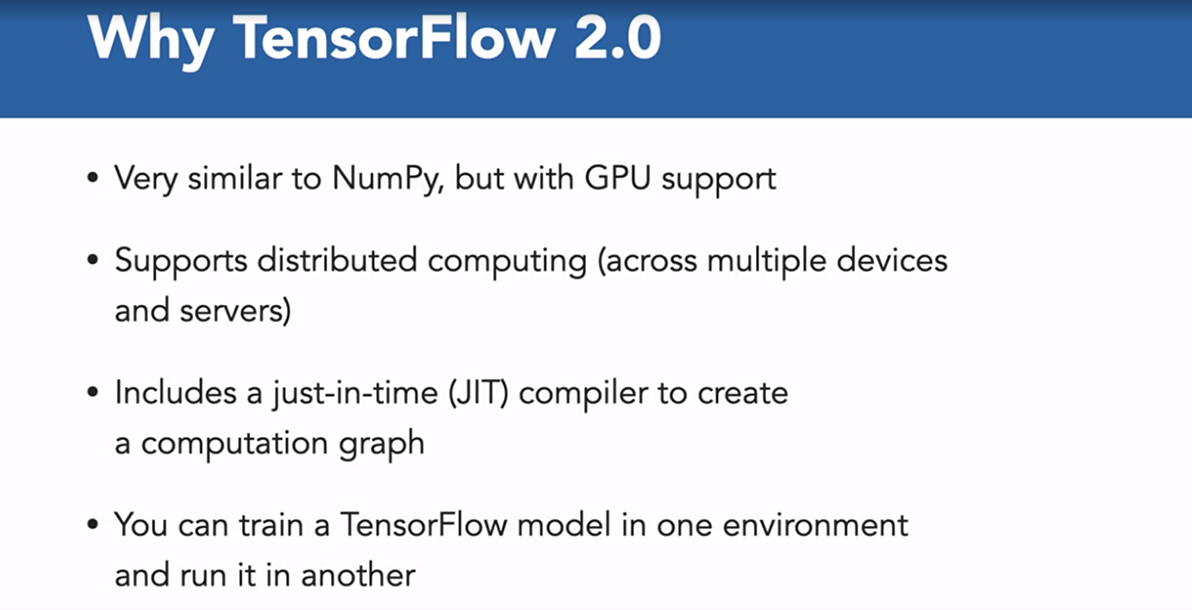


This course follows on from another course I put together called TensorFlow: Neural Networks and Working with Tabular Data, which is also available in the LinkedIn Library, so if you haven't worked your way through that course, you probably want to do that first. Now, if you know the basics of Python and have some understanding of machine learning and have worked with TensorFlow before, you should be just fine. I'll be using Google Colab, so if you haven't used this before, you'll need a Gmail account, which you can sign up for in a couple of minutes. Alternatively, you can just follow along.

### **What is TensorFlow**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] TensorFlow is a powerful library that we can use for machine learning, and was developed by the Google Brain Team, and it powers many of Google services such as Google Search, and it was open sourced to November, 2015. Now, you can use it for all sorts of machine learning tasks, such as image classification, all the way to natural language processing. At its core, it's very similar to NumPy, but with GPU support. So, it's support to distributed computing, that means it works across multiple devices and servers. And it also includes a just-in-time compiler, that allows it to optimize computations. It does this, by extracting the computation graph from a Python function, and then running independent operations in parallel. These computation drafts can be exported to a portable format. This means, that we can train a TensorFlow model in one environment like I'm doing now using Python on my Windows machine, and then run it on my Android phone using Java. Most of the time, we'll use high-level APIs such as Keras, but when you need more flexibility, you'll use the Lower-level Python API handling Tensors directly. TensorFlow runs not only on Windows, Linux, and the Mac OS, but also on mobile devices using TensorFlow Lite, including both the iOS and Android. Now, if you don't want to use the Python API, then the C++, Java, Go, and Swift APIs. Now, one of the biggest attractions to TensorFlow, is the ecosystem that's available. We've already seen that you can build and deploy models with Python, but there's even a JavaScript implementation called tensorflow.js, which means you can run your models directly in your browser. You then also have TensorFlow Lite, which is for using an iOS or Android, but there's more. There's TensorFlow Extended, which is a set of libraries built by Google to productionize TensorFlow projects. So, this includes tools for data validation, pre-processing, model analysis, and you can save these modules with the REST API using TensorFlow Serving. Now, TensorFlow Serving is really important because it's easy to create ML solutions that have one or two users. But what happens when you have to scale that out to hundreds of thousands of users? Then there's TensorBoard, which is great for visualization because it helps you when training your models. And finally, TensorFlow Hub provides a way to easily download and reuse pre-trained models. So, these are models that have been known to perform well on certain tasks. This means you aren't starting from scratch, but using the work of other experts.



<https://www.tensorflow.org/learn>

### **Review of neural networks**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Man] As I've mentioned TensorFlow neural networks and working with Tableau data is the first course in this series. Here I covered the Fashion-MNIST Dataset. Now, if you haven't seen that course, this video will provide a quick overview of the Fashion-MNIST Dataset. the Fashion-MNIST Dataset is made up of images from the fashion chains, Zalando. It contains a training set of 60,000 images and a test set of 10,000 images. Now each image is a 28 by 28 grayscale image, associated with the labels from the 10 classes. Each training and test example is assigned to one of the following labels. So for example label zero refers to a T-shirt and label one is a trouser and so on. Now this is what the neural network that we use to help us classify the 10 categories of the Fashion-MNIST Dataset looks like. We had 784 input nodes. And each of these nodes corresponds to a pixel of that image. We then have two hidden layers with 128 nodes and 64 nodes. The final output layer has 10 nodes. Now, Zalando's Fashion-MNIST Dataset has 10 classes. So if you wanted to know the probability that the neural network predicts it's one of the different classes you just need to find out which class has the highest number. So the higher the probability, the more likely the neural network predicts that that image belongs to that class of objects. The input layer needs to be 784 nodes, because these are the number of pixels in the input image. Similarly, the output layer has to have 10 nodes because there are 10 classes in the Fashion-MNIST Dataset. Now let's head over to the notebook to get a quick overview of this dataset. Now the purpose of this notebook is to familiarize yourself with the Fashion-MNIST Dataset. So just so you know, the code below downloads the Fashion-MNIST Dataset and then trains a neural network model, so that it can distinguish between the different classes of objects. Now, if you haven't used Jupiter notebooks before in co-lab, they allow you to combine both commentary and code in a single place. I'm going to head over to the run time and select run all. And this is going to run all of the cells in this notebook. So in the overview of the images, we get a look at 16 random images from the Fashion-MNIST Dataset and you can see that these are 16 different objects of clothing or shoes. So you've got T-shirts, bags, ankle boots and so on. And then going down a little bit further in the build, compile and train the model. What we're doing here is to do exactly that. Build, compile, and train our neural network model. You can see that the neural network model that was used has 128 nodes, and then 64 nodes and then 10 nodes. And these nodes correspond to the neural network that I showed you a couple of minutes ago. Now, if we look at the accuracy of this model, you can see that we have an overall accuracy of around 88%. Now let's go ahead and make some predictions and see how good the model is. So if we use ID one as an example, right? We're going to be using an image from the test dataset. So we can pick an ID between zero and 9,999. So image ID one, is a pullover and this is exactly what our model predicts. Our model predicts that with the highest probability that this is going to be a pullover, and then you can see that there are secondary predictions for a coat and a shirt. Now, let me go ahead and pick another item of clothing. So if I go ahead and do 1024 as an example, and then I go ahead and rerun the cell, so I can select the play icon here and that's going to rerun the cells. So this is going to suggest that this image is one of a sneaker, and that's exactly what our model predicts. Now once you spent a little bit of time on the Fashion-MNIST Dataset, you can go ahead and change some of the image IDs between the number zero and 9, 9, 9, 9. And you're happy with how this works. Let's head over and look at another image dataset.

### **Working with color images and neural networks**

The CIFAR-10 Dataset has the classes, airplane, automobile, bird, cat, deer, and so on. And the images in the CIFAR-10 are of size 32 by 32 by 3. The 32 by 32 corresponds to the height and the width of the images and pixels, and the three is the number of channels, so that's red, green, and blue. Let's head over to the Colab Notebook to see how well our neural network performs on the CIFAR-10 dataset. So, I'm going to head over to Runtime as before and run all of the cells. And what's going to happen, is we're going to load the datasets. We're going to train our neural network and so on. So, let's take a look at a couple of examples from the CIFAR-10 dataset. So, if we look at the first row, you can see that we've got examples from the dataset of a dog, automobile, another dog, a ship, and so on. And we've got 25 such examples of examples from the CIFAR-10 dataset. Now, let's take a look at a random image. So, that's image ID number four. And you can see that this is an image of an automobile. Now, what we see here is the red, green, and blue view, or the color view. But what we can also do, is to split this into three different channels, so that's the red, green, and blue channel. Now, the darker the image, the more pronounced that component of the channel. So, for example, in the original image, the sky is blue, and the front of the car has a blue reflection. When we look at the corresponding blue channel, you can see that these sections are darker for the blue channel, compared to the red and green channel. In the next section, we go ahead and use the same neural network that we used for the Fashion-MNIST dataset. (143) We need to make a change though, because the input shape for the Fashion-MNIST dataset was a 28 by 28 pixel. In the CIFAR dataset, we're using 32 by 32 by 3 instead. And then everything else can stay the same. So, we go ahead and train our model. And let's look at the accuracy that we're able to achieve. And you can see that our accuracy has dropped quite considerably. With the Fashion-MNIST dataset, we had an accuracy of around 88%. With the CIFAR-10 dataset, we've only got an accuracy of around 46%. So, let's look at a couple of examples to confirm this is the case. So, if we look at image ID 7260 as an example, you can see that this is definitely an image of an automobile, but our model has incorrectly classified this as a truck. And we can see some further examples here. So, if I take the last row as an example, we know the model accuracy is only around 45%, so we can expect there to be problems with classification in this last row. But you can see, for example, that a frog has been misclassified as a deer. A bird has been misclassified as a frog. The boat or ship has been misclassified as an automobile, and so on. So, we can conclude that this neural network that worked very well for the Fashion-MNIST dataset, giving us an accuracy of around 88%, isn't working well here, and giving us almost half that accuracy for the CIFAR-10 dataset.

### **Challenge: Experiment with hyperparameters**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] We haven't had good results with the CIFAR-10 dataset using a neural network, so we're going to try two things. We're going to add two more layers to the neural network. At the moment, the neural network accepts the input image, which is of size 32 by 32 by 3. So if we flatten that, that's an input size of 3072 pixels in total, and then this goes on to a next layer, which has 128 nodes. Now, perhaps that's too big a jump and it isn't able to capture some of the changes in the image so effectively. So what if we gradually reduce that from 3072 pixels to 1024 and then 512 nodes? This might help us get better accuracy. In the second part of the challenge, we found that neural networks were great on the fashion eminence dataset, which is a dataset of grayscale images. Now the CIFAR-10 dataset has three channels, red, green, and blue. What if we just extract the red channel as input to the network instead of using all three channels? With a single channel, we're closer to the single grayscale image, the fashion eminence dataset had, and perhaps this might help. So go ahead and open the CIFAR-10 challenge notebook. Now because we are training a network a couple of times, you'll find that it's faster to use a GPU instead. So select run time, select change run time type, and select GPU. And then you want to go ahead and select run time, and run all the cells in the notebook. So what you would want to do is to find the sections corresponding to each of the challenges. So this is the section for challenge one followed by the section four challenge two, go ahead and make the changes for that section. This challenge should take you between five and 10 minutes and I'll go through my solution in the next video.

### **Solution: Experiment with hyperparameters**

Selecting transcript lines in this section will navigate to timestamp in the video

- So let's head over to our first challenge. So we want to be able to include input nodes of 1024 and 512 nodes. So I'm going to reuse these two lines where we have 128 nodes and 64 nodes and let's replace these with 1024 nodes here, and 512 nodes here. Now, just so happens that with this iteration of my model, I had an accuracy of around 45%. So let's see how that compares with the changes that we're going to make here. So let me go ahead and rerun the cell. So our model has now finished training. You can see that our model now has an accuracy of around 49%, which is better than it had before. So it looks like including these internal nodes of 1024 and 512 helps to improve the accuracy of the model. Now let's head over to the second part of the challenge. And what I'm going to do, is I'm going to take the first section and I'm going to include this in a new cell. So let's take a look at the dimensions of train dataset. If we stopped on the train images, now each of the train images comes in batches. So the first section will correspond to the batch. That's the 32 pixels followed by another 32 pixels and then followed by the channel. So if I want to extract just the first channel, so that's the zero channel. You can see that now my images are of size 32 by 32. And I can do exactly the same thing for my validation dataset. So let me split this out into a new cell, and I'm just going to copy across the section so that I can just pull out the red channel, which is the first channel. And you can see that I've now got an image of size 32 by 32. But just for comparison, if I just use the test dataset without any changes, I've got a size of 32 by 32 by three. But if I then want to just extract the red channel, I can do so in this way. And now my test data set has a size of 32 by 32. Now, if we take a look at the input to our model, it's not going to be an image of size 32 by 32 by three anymore. We're going to be able to remove that third channel. And let's go ahead and run our neural network model here. So a model has now finished training. And if you remember, our accuracy was around 45% before we made any changes. Now it looks like the accuracy has in fact dropped to around 36%. And so by trying to extract only a single channel, we were not able to make any improvements to our neural network model. So you can see that although the first technique helped to improve our accuracy, it didn't change it significantly. So in the next section, we'll look at why this might be the case.

### **Why the poor performance with neural networks?**

Selecting transcript lines in this section will navigate to timestamp in the video

- So why did our neural network perform so poorly on the CIFAR-10 dataset? Well, a neuro network doesn't take into account the spatial structure of the image. What makes a face, a face is that typically we have a pair of eyes, a nose, and mouth all close together. If we just flatten an image at the start, these details aren't captured. There are a couple of differences between the complexity of the images between the fashion-MNIST dataset and the CIFAR-10 dataset. And so let's take a look at them. With the fashion-MNIST dataset, we only have a single object in the image and all of the objects are in the center of the image. The CIFAR-10 dataset is far more realistic. There are color images with other items also part of the image and not all the images have the main object in the center. Our neural network worked well for gray scale images that have only a single channel. Now with color images we have three channels, and so we have three times the number of input nodes. This means if your image is 32 by 32 pixels and has three channels, it has 3072 input notes. This is significantly higher than the input nodes for the fashion-MNIST dataset. So how do we find a solution to working with color images? To help us, we'll turn to the TensorFlow hub.

### **TensorFlow Hub**

Selecting transcript lines in this section will navigate to timestamp in the video

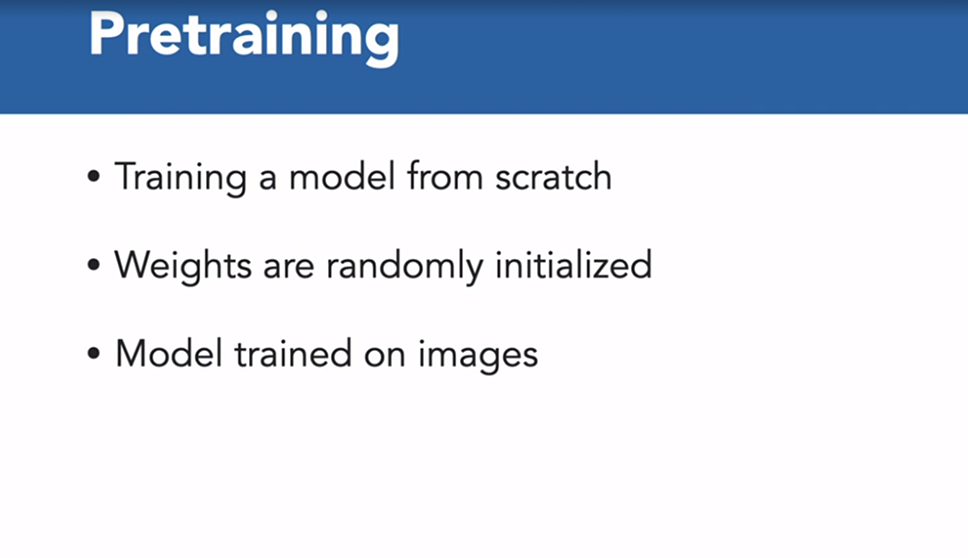
- [Announcer] Historically, deep learning engineers worked with complex models and trained them on large amounts of data, meaning that there are often significant costs involved in compute and in training of these models. So how can we use their work? Some of which might be TensorFlow without having to train these models from scratch. TensorFlow Hub is designed to solve this problem. It's enables transfer learning by making a variety of ML models, freely available as libraries or web API calls. Anyone can write just a single line of code to load the model and all models can be invoked via a simple web call and then the entire model is downloaded to your source codes runtime. And you don't even need to build a model yourself. This definitely saves development and training time. It also allows users to try out different models and build their own applications more quickly. Another benefit of transfer learning is that since you're not training the whole model from scratch, you may be able to get away with fewer and smaller GPUs created by InVideo or Tensor Processing Units or TPU developed by Google. So TensorFlow hub is a repository of machine learning models. And here you can find models that have been trained on specific datasets, or you can even contribute models that have been created for your use case. And the problem domains are broken down into text, images, video and audio. The text problems include language modelings, texts retrieval, question answering, text generation and summarization. The image domain includes classification, object detection, style transfer among several others. The video problem domain includes classification, generation, audio and text. And finally, the audio problem domain includes speech to text embeddings and speech synthesis amongst others. Now, if you head over to all models, you can access different kinds of architectures and models. You might've heard of some of them. Such as BERT, CNNs, and so on. Now you might also want to use models that have been created by well-known names, such as Google or DeepMind, and you can select these models from the TensorFlow hub. You then have the option of also getting access to a whole host of different datasets. You can see that this is an amazing resource and a great starting point for any TensorFlow machine learning problem that you have. Whether this is an image problem, or text, video, or audio.

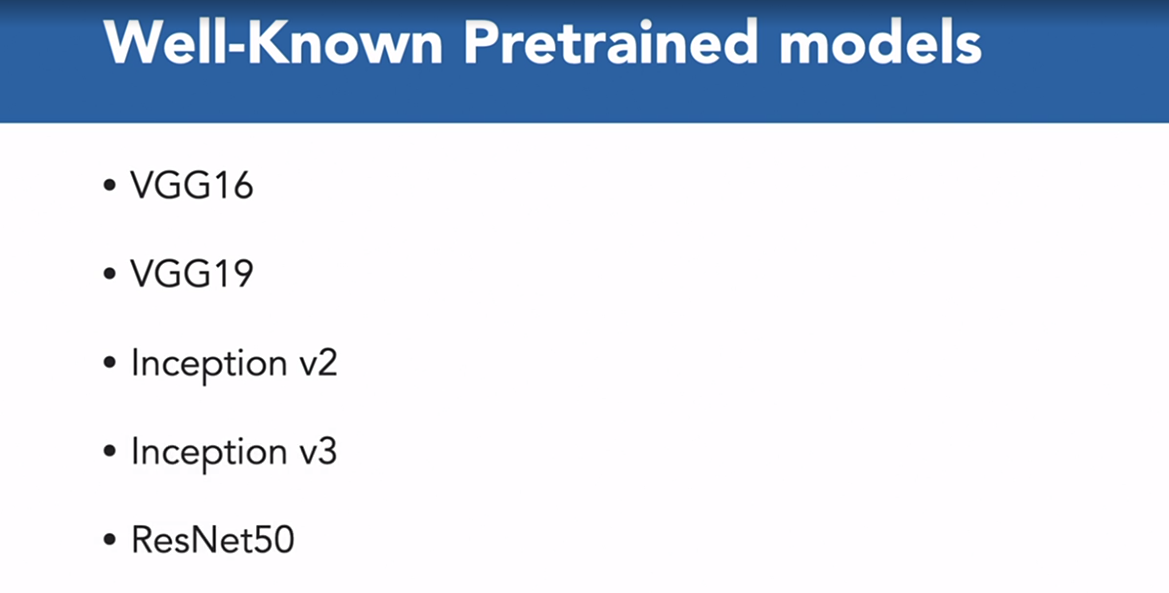
<https://tfhub.dev>

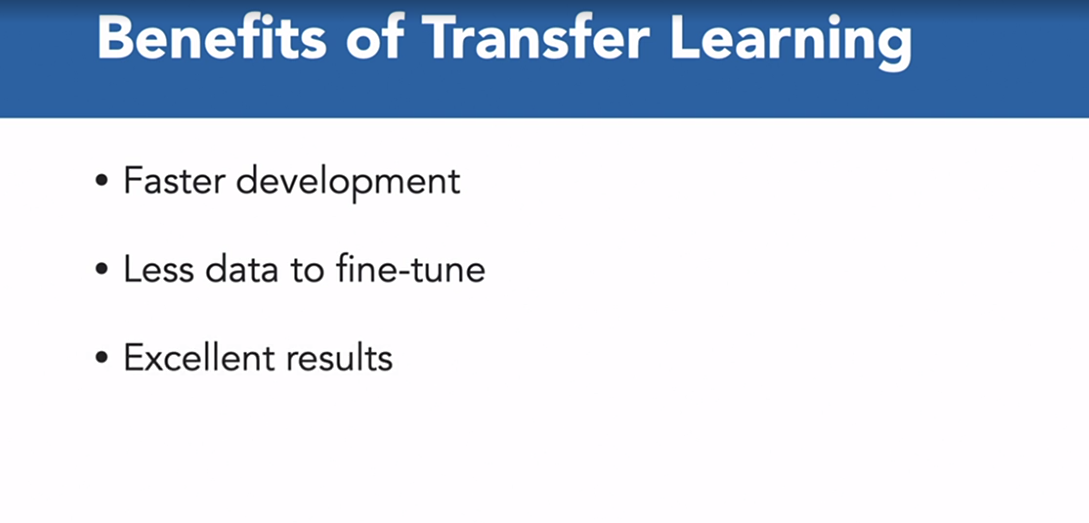
### **What is transfer learning?**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Transfer learning is made up of two components, pretraining and fine tuning. Pretraining involves training a model from scratch. So this means the model weights are randomly initialized. The model is of no use at this point. The model is then trained on thousands of images and becomes useful for computer vision tasks such as image classification, but you can use it for a wide variety of domain. So you could use it for text or video or audio. And as you can imagine, we need both a lot of data and a lot of compute power. For the ImageNet Challenge, computer vision models had to distinguish between 1000 different categories of images. This means that these deep learning networks learn a whole load of features such as edges and corners and textures of images in the process. There are many models that performed very well on this dataset, including VGG16, VGG19, Inception version two and three, and ResNet50 amongst others. The researchers that created these models then made them available for download along with their weights. So what are the benefits of transfer learning? It takes much less time to train a fine tuned model, usually minutes. You might only need to run somewhere between five and 15 epochs through your entire dataset. This is in contrast to a couple of hours required for pretraining from scratch, using several powerful GPU's. We don't need as many images when fine tuning the model. You can use as many as 20, 30 or 50, depending on the accuracy you're looking for. This is in contrast to when training on ImageNet, where there are more than 1000 images available per category. And quite remarkably, with this combination of pretraining and fine tuning, you're able to achieve excellent results.







### **Transfer learning with TensorFlow Hub**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Let's head over to the Colab Notebook and look at another dataset and an example of image classification. For this example, we will need to use the GPU, so I'm going to head over to Runtime, change the runtime type. I'm already using a GPU. And I'm going to go ahead and run all the cells in the notebook. Now we're going to be using the flower's dataset in this example. And the flowers dataset has five classes. And our job is to look at an image of a flower and predict which of these five types of flowers it is. So we store all of our data in the data\_dir directory. And you can see that we've got five different directories here, and each of these directories corresponds to a flower. So you've got tulips, sunflowers, roses, dandelions and daisies. Now each of the images is of size 224 by 224. And our batch size is going to be 128. The TensorFlow dataset API offers the dataset abstraction, so we will be using the image dataset from directory method that will take the images that we have just downloaded and generate a dataset object. And you can see at the end of this process, we've got five different classes corresponding to the five different types of flowers. Now, because for most deep learning models, we expect the inputs to be between zero and one. We can create a normalization layer for both the training and validation TensorFlow datasets, and we do that in this section here. As part of the training process, we'll need to read data from our directories, but we don't want to make our model wait for this data, so autotune allows us to fetch the data dynamically. We're going to be using Inception v3 from TensorFlow Hub, and this is the path to the model. So this has been pre-trained on ImageNet. Now, just you know, the TensorFlow Hub also distributes models without the top classification layer. So let's see how this model does without any fine-tuning. And not surprising, it struggles, because it hasn't been fine tuned on the flower dataset. Now it so happens that daisies are one of the classes in the ImageNet dataset. So if any image has a daisy, it should get it right, but there will be other images that don't have a daisy in them that will be flagged as a daisy, only because this is the closest looking thing to a daisy compared to the 999 other classes in the ImageNet dataset. Now, because the Inception v3 model hasn't been trained on ImageNet, this means that understands the basics of images, so things like the shape and the contour and the textures of images. So we don't want to lose this, and this is why we set the trainable equals false when we want to use this model. This will freeze the weights and parameters in our Inception v3 model. Now, the next thing we want to do is to take that Inception v3 model and add a fully connected layer with five outputs, because these are going correspond to the five classes of the flowers in our dataset. And this is the new model that we've created. And you can see that this is the first part of the model that we've created, and it has over 23 million parameters from the Inception v3 model. And the second half of the model is the five fully-connected layer that corresponds to the five outputs of five classes of the dataset. So now let's go ahead and compile and train this model, and we can train this model by calling our fit method. And you can see that over the last couple of minutes, it's trained over the 10 epochs of the network, and you can see that we are getting a validation accuracy of around 83%. Now, if we plot the training and validation loss over the 10 epochs, we can see that the validation and trading loss drops as the model learns. And similarly, when we plot the training and the validation accuracy, these increase over the 10 epochs. Now let's take a batch of images and see how well this does on our fine-tuned model. So you can see that this is going to be a prediction of a batch of 128 images. And the first column corresponds to a daisy. The second column corresponds to a dandelion. The third to a rose, and so on. And to determine which of the classes it is, we need to find out which has the largest value across that axis. That is the class of flower that's our model is predicting. And to do that, we use the argmax method. And finally, let's view 25 images and see what the model is predicting as the associated flower type. And you can see that by fine tuning our model, it has improved considerably. And that is the power of transfer learning. For image classification, you can take a model that's been pre-trained on the ImageNet dataset. There are some well-known models like VGG-16, ResNet-15 or Inception v3 like we used here. Your starting point is the work of deep learning and computer vision experts. You can then fine-tune these models on the dataset that we have, so that's the five different types of flowers, with this simple technique, and you can get excellent results. And that's the power of transfer learning.

### **TensorFlow Hub for CIFAR-10**

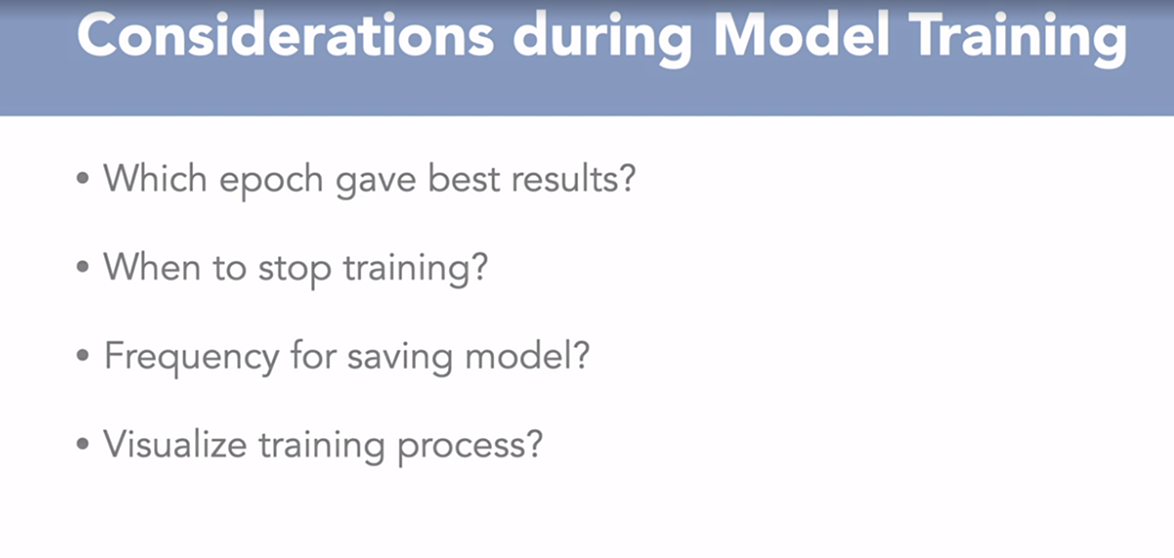
Selecting transcript lines in this section will navigate to timestamp in the video

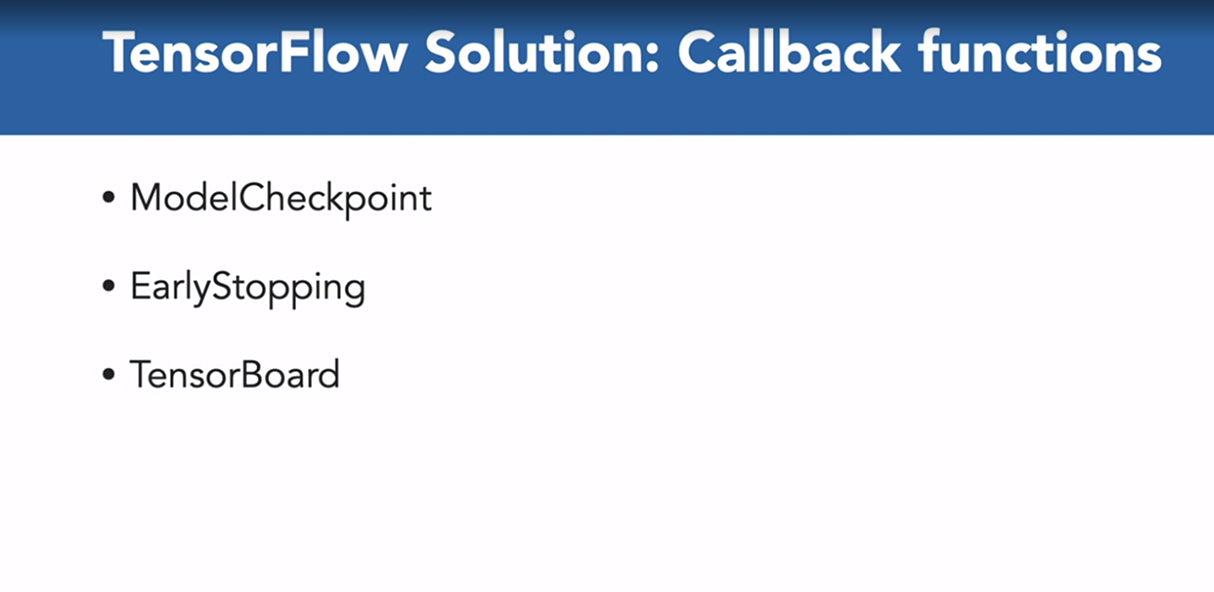
- [Instructor] So can we use the TensorFlow Hub to help us with the CIFAR-10 dataset? Let's take a look. So I'm now at the TensorFlow Hub, if I head over to all models and I look under the dataset, let's see if we can find the CIFAR-10 dataset here. So I'm going to select CIFAR-10 and this gives me a couple of the models that are available here. Now, I can see that there are a couple of GAN-based models, which are not going to be helpful. But this model, which is a deep VGG style network trained on CIFAR-10, looks very promising. And additionally, the fact that it's been downloaded 2,000 times means that it's probably pretty good and the publisher is DeepMind. So this is looking very promising. So let's head over to our notebook and let's use this model. So I'm going to just click on this, and this is the URL that I'm going to need to be able to access this model. So I'm going to head over to runtime and run all of the cells in my notebook. So this is just copying across all of the CIFAR-10 dataset. Now let's look at the build, compile, and train section. Now because the model has been trained on CIFAR-10, I'm not going to bother to train this model. All I need to do is to point it to the TensorFlow Hub model, which is what I've done here. I entered trainable as false. And then I went to go ahead and compile this model. I don't need to train this model because it's already been trained on the CIFAR-10 dataset. And so let's go ahead and see the accuracy that we're getting for this model. We've got a model accuracy of about 95% and just you know, because I had everything I needed, I didn't have to bother with transfer learning. Often when you're doing transfer learning, the models have been trained on images of size 224 by 224 by 3. This means that I would have to resize my images from the CIFAR dataset from being 32 by 32 to 224 by 224 and then I could use one of these models in the hub and then fine-tune it on the CIFAR-10 dataset. Let's see how good the model is on images that our previous model struggled on. And previously, our model predicted that this was a truck. This accurately predicts this is an automobile. And indeed, the 95 accuracy seems to hold. That final row is correctly classified as a frog, a bird, a ship, a horse, and an automobile. Now, this shows you how easy it can be to use TensorFlow Hub for image classification.

### **Monitoring the training process**

Selecting transcript lines in this section will navigate to timestamp in the video

- So far, things have worked relatively smoothly. Our notebooks have worked and we haven't had to do too much work. We've been able to use transfer learning and the TensorFlow hub. Now on the job, things can often be more complicated. Let's put ourselves in the shoes of a deep learning engineer, creating a model architecture and training a model from scratch on the CFR 10 dataset. Now, when I'm working on training models, here are some of the questions that I have. How can I determine the epoch which gives me the best model performance before over-fitting occurs? And how can I stop training if the model is not improving or is over-fitting? Other questions I have is how often should I save the model during the training process? And finally, is there a way to visualize the model training process? TensorFlow's solutions to these questions is Callback functions. We'll be looking at three of the most used classes, which are ModelCheckpoint, EarlyStopping, and TensorBoard.

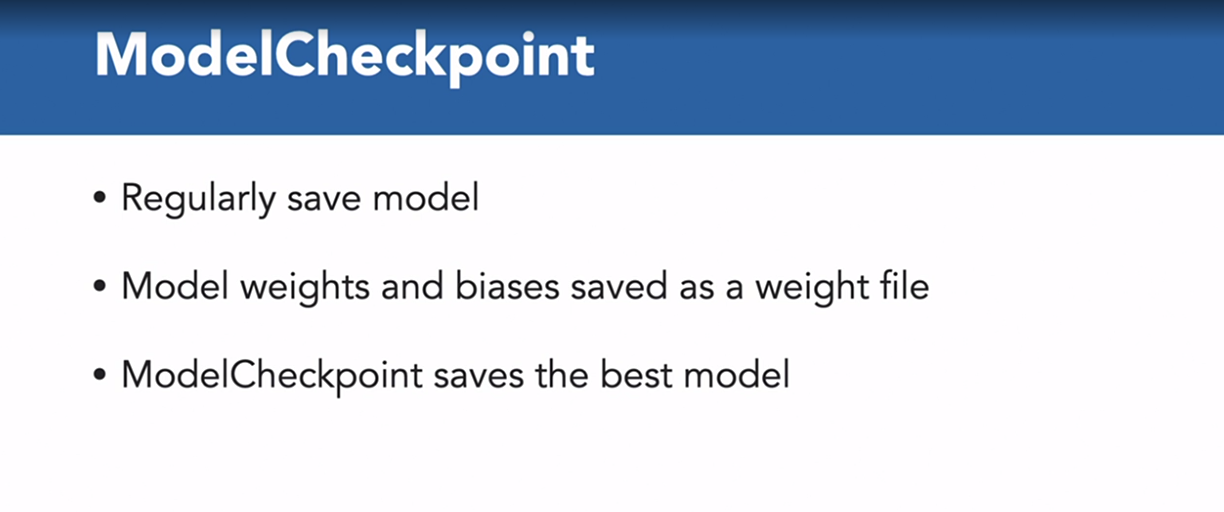




### **Using ModelCheckpoint**

Selecting transcript lines in this section will navigate to timestamp in the video

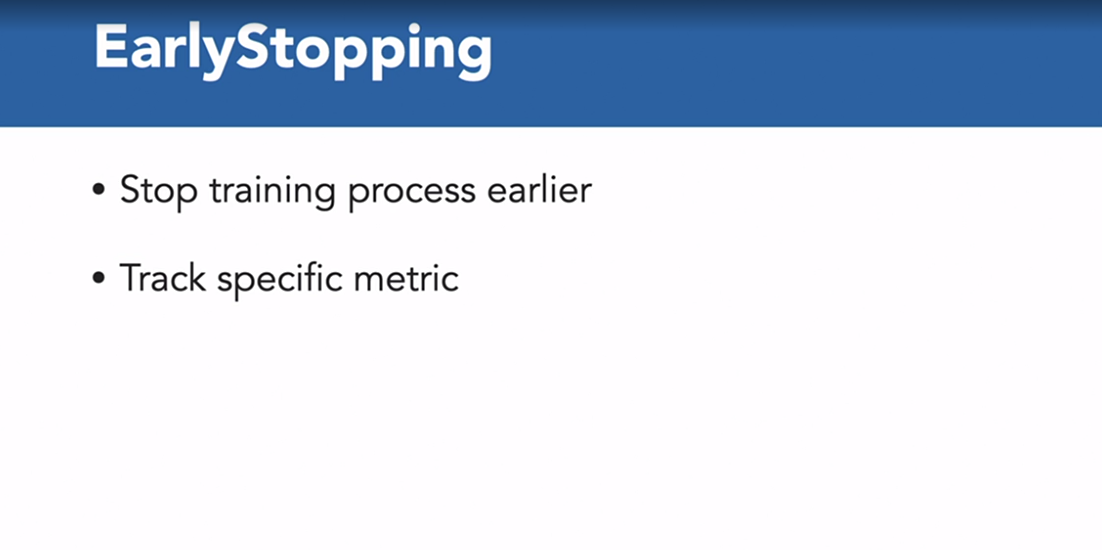
- [Instructor] ModelCheckpoints allow you to save your model regularly throughout the training process. So this is usually at the end of each training epoch. The model weights and biases are finalized and saved as a weight file. Use ModelCheckpoint to save the model only if it is improved from the previous epochs. So the last saved model is going to be the best. Let's head over to our Colab notebook to see how ModelCheckpoints work. So I'm going to head over to runtime. I want to make sure that I'm still at GPU as my hardware accelerator. And then I'm going to go ahead and run all the cells in this notebook. So what we're going to do is to download our CIFAR-10 dataset. We have the images here. Each batch will contain 128 images and each training batch is fed into the model during the training process to enable incremental updates to weights and biases. So to help with the understanding of how we might go about monitoring the training of a model, I've put together a simple model that takes as input and image of size 32 by 32 by three. This model architecture runs through two pairs of convolutions and max pooling layers. This is then flattened and passed to a neural network with 256 nodes and finally, the 10 nodes of our classes. Our architecture then has two pairs of convolution and max pooling lists, and so that's this section here. We then flatten it, passed through a layer with 256 nodes. And finally, we have a layer with 10 nodes. Now as part of ModelCheckpoints, we want to set the directory path for the checkpoints. Now let's specify the file path where TensorFlow will save the model after each epoch. And we're going to be monitoring the validation accuracy. So I go ahead and specify that I'm going to monitor the validation accuracy, and I've specified the checkpoint directory here. Now when you launch the training process with a callback, the callback will expect a Python list. So let's put the checkpoint object into a Python list. Now we need to go ahead and launch the training process. This command assigns the entire model training history to the object hist, which is a Python dictionary. We can now view the validation accuracy from the first epoch of training to the last. The best model is the one with the highest validation accuracy. So if I go ahead and look at all of the validation accuracies, you can see that I'm going to get the best validation accuracy over here where I have 0.59 as my validation accuracy. And this is confirmed here by the maximum epoch, which is eight. And so the best model has a validation accuracy of 0.59 on epoch eight. Now let's take a look at the checkpoint to directories. And you can see that each directory has subdirectories of variables and assets and the saved\_model and the keras\_metadata. Up to this point, checkpoint has saved the model at each epoch. If we only want to save the best model, we need to give the save\_best\_only flag a value of True. And so I can do that as part of my ModelCheckpoint definition. And you can see that for my first epoch, I have a validation accuracy of 0.57. And this seems to have the highest validation out of all of the epochs. So the model from the first epoch is always saved in general. And as you can see, since there isn't an improvement in the validation accuracy, we only have the first checkpoint. The model from the first epoch is always saved. The training continues all the way to the 10th epoch, but as there isn't any further improvement on what's achieved at the first epoch, none are captured. The first epoch directory contains the best model.



### **Working with EarlyStopping**

Selecting transcript lines in this section will navigate to timestamp in the video

- The early stopping callback object enables you to stop the training process before it reaches the final epoch. Usually you do this to save training time if the model isn't improving. So we track a certain metric, say the validation accuracy, and if this doesn't improve after a certain number of epochs the training will stop. Let's head over to our notebook. What's new here is that we set the patient's parameter as part of the early stopping callback. This means that if the validation accuracy does not improve within three epochs, the training stops. Now this time round, we're training with 15 epochs and you can see that since there wasn't an improvement in the validation accuracy over five epochs, our training has stopped at this point.



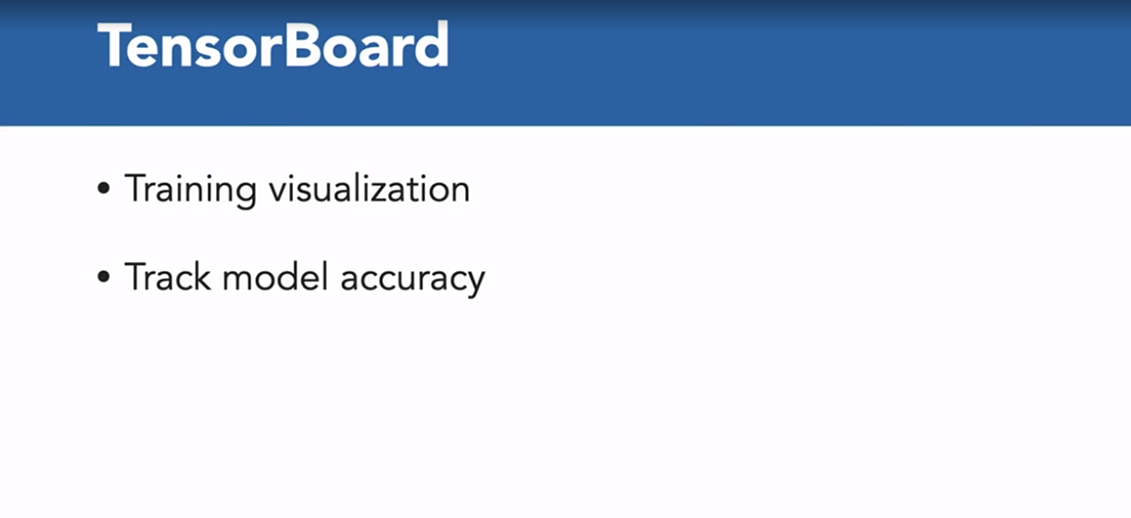
### 

### 

### **Using TensorBoard**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] TensorBoard provides a visual representation of how model parameters evolve throughout the training cycle. It's often used for training model accuracy over training epochs. TensorFlow can also let you see how weights and biases evolve in each model layer. And just like ModelCheckpoint and Early Stopping, TensorBoard is applied to the training process via the callbacks module. Just so that we have a clean starting point, I'm going to reset our notebook and start again for TensorBoard. I'm going to head over to Runtime. I'm going to select Factory reset runtime. And I'm going to reset that notebook. I'm going to select Reconnect, and then go ahead and run the next couple of cells. And you can see that we've got the same model. But what's different here is that we've got a name for each of the layers. So for our first convolution layer, we've got it as conv 1. Similarly, we've got a conv 2 for our second convolution layer. And you can see that when we use the plot model method, we've got our names that we've assigned to each of the layers. Now within the arguments for TensorBoard, log directory is the path to save the training logs. So as training progresses, your logs will be generated and stored here so that TensorBoard can pass them for visualization. Write graph is set to true so that the model graph will be visualized. And write images is also set to true. This ensures the model weights will be written in the log so that you can visualize how they change throughout training. And finally, histogram frequency is set to one, and this tells TensorBoard when to create a visualization by epoch. One means the visualization is created for each epoch. So now let's head over to our TensorBoard. So you can see that our model has completed training. And we can access TensorBoard by using the reload extension magic commands as well as TensorBoard. So let's head over to these SCALARS tab first. Now, as you can see the validation accuracy, which is in blue, is lower than the training accuracy. And similarly, the validation loss is higher than the training loss. Now, this makes sense, because the model performs better with training data than when tested with validation data. The darker lines indicate the validation metrics, and while the lighter lines indicate the training metrics. Now, if we head back to the validation accuracy, it looks like this peaked at around epoch number eight with a value of around 0.6. And after that, the validation accuracy started to decrease while the loss started to increase. This is a clear sign of model overfitting. And what these graphs are telling you is that after epoch eight, your model started to memorize training data patterns. So why does this help? You now know which epoch delivered the best model of the training process. You can see when the model starts to overfit and memorize its training data. And if your model still has room for improvement, you may want to increase your training epoch and keep looking for the best model before the overfitting pattern starts to appear. Let's now head over to the HISTOGRAMS tab. This shows the distribution for the weights and the biases. So by visualizing how these parameters are distributed and how the distribution changes over time, you can gain some insight into the impact of the training process. Now, each section is divided by the names that you gave your model. So the first layer is conv 1. And you can see that each slice in the diagram corresponds to an epoch. The first epoch is in the background, and the last epoch is in the foreground. For the first convolution, it looks like the variance starts to increase over that first convolution. By the end of that second convolution, you can see that the variance is decreasing. Now, these graphs are helpful, because when training data, I've often seen examples where I've had multiple peaks in my normal distribution. And often by adding another convolution pair, I'm able to normalize this data and ensure that I have a single normal distribution. So this HISTOGRAMS tab can be very useful in trying to understand why your model might not be performing as well as it could be. So you can see clearly here that the mean seems to be a couple of values initially for convolution one. And you can see by the end of that second convolution layer, the mean value seems to have adjusted itself. And this covers our section on the TensorBoard.



Which callback allows you to save your model throughout the training process?

ModelCheckpoint

Which callback object allows you to stop the training process before it reaches the final epoch?

EarlyStopping

Which of these allow you to visualize your model and training process?

Tensorboard