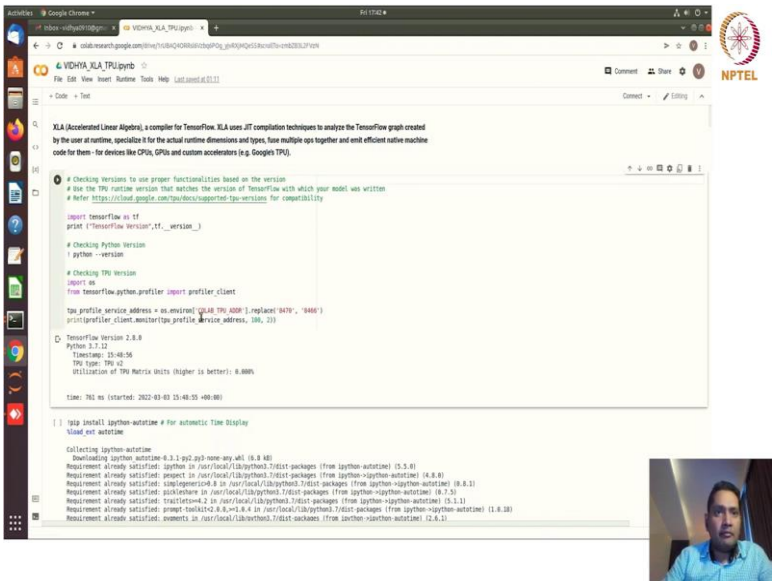
Applied Accelerated Artificial Intelligence
Dr. Satyajit Das
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

Lecture - 29
Accelerated TensorFlow - XLA Approach Part - 2

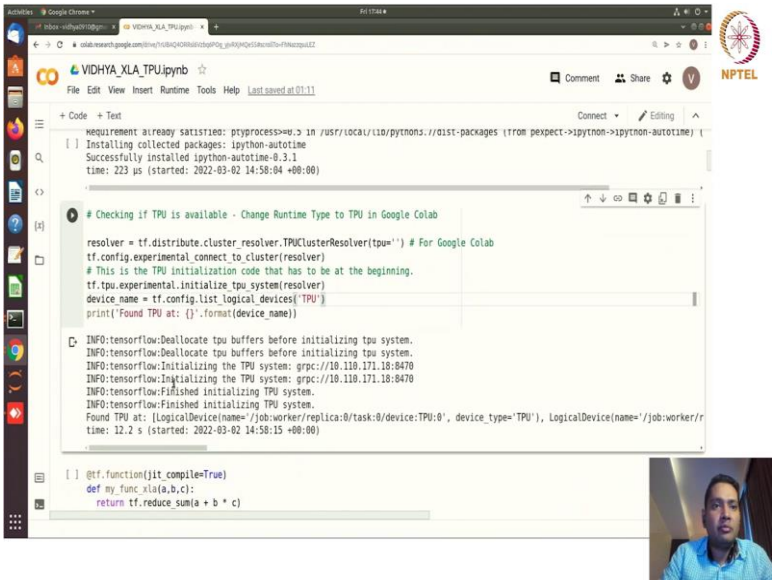
(Refer Slide Time: 00:15)



So, now first we will talk about how to use TensorFlow for your TPU environment ok. So, this is the very very important for your TPU cluster if you want to use TPU clusters. So, basically for your colab you can go to your runtime and change runtime type to your as simple as that. So, basically in by default TPUs will be available for you and you can use that.

Now, how we can use that and we will use that with different strategies ok. So, basically XLA mixed precision tf.function run time and so on distribution strategy right. So, let us see all of them in one place ok. So, importing the things here, so all the things are imported and then you just see which profiler which TPU version is being used we are just printing here and using auto time for time displays just few libraries.

(Refer Slide Time: 01:31)



Now, checking if TPU is available and changing the runtime to your TPU. So, basically distribute.

Student: Sir.

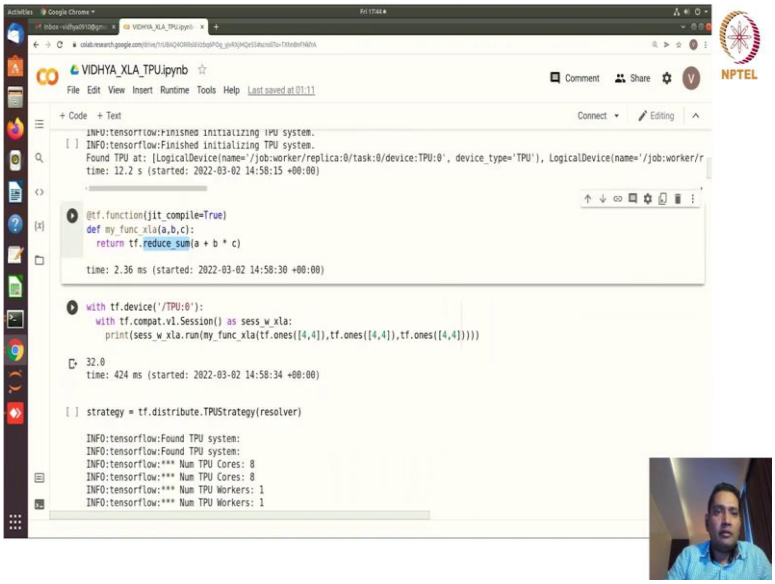
TPU cluster resolver, yes.

Student: Sir we cannot view the text properly. So, requesting to increase the size.

Yes. So, I hope this is visible now. So, till now whatever we have talked about is basically the imports and setup that is very simple, but now what we are going to see is that how to define the how to use the TPU for your devices and distribution strategies and so on.

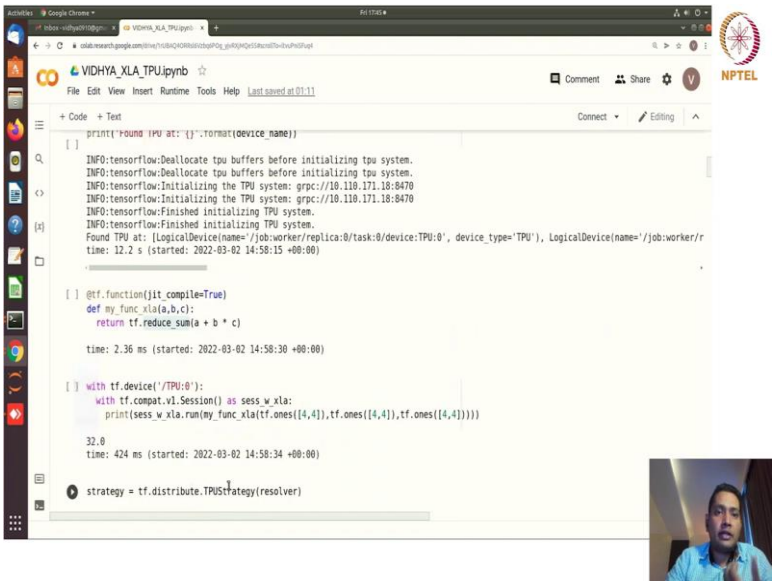
So, we are defining the resolver how we are defining here TPU cluster resolver that we have seen in the previous class experimental connect to cluster we are connecting it and then TPU initialization code right. So, to just at the beginning you need to initialize your TPUs. Now by the way XLA is very good for your TPUs ok because many fused operations that you are you the XLA will generate is directly supported onto TPU because TPU is basically from Google TensorFlow is from Google XLA is from Google. Now, you connect everything ok.

(Refer Slide Time: 03:17)



We will see what kind of performance we are getting here tf.function JIT compiler equal to True.

(Refer Slide Time: 03:45)

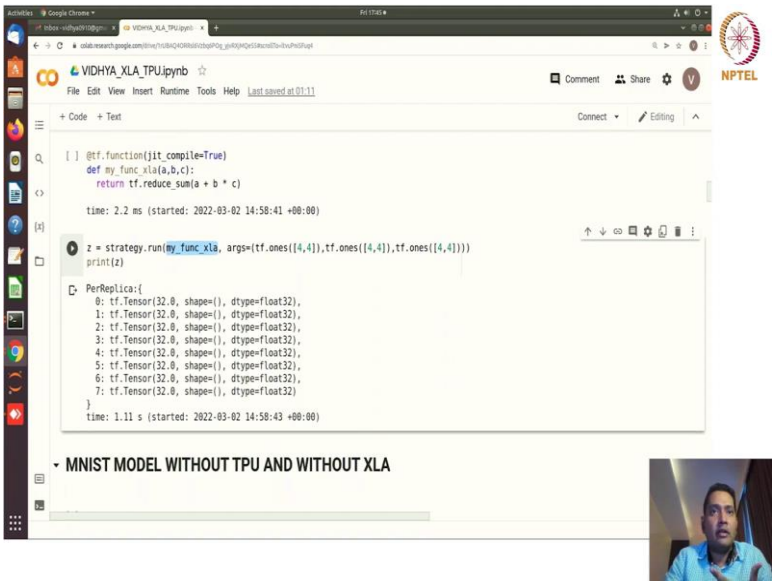


So, this is how we will enable here now first we are just wrapping up with some function. So, this is one function that we are going to reduce. So, basically this is the graph computational graph and we are running it with the TPU that is it right.

So, all the TPU information you are getting and then you can see the runtime here and the strategy that we have to defined before right. So, the strategy we have defined here

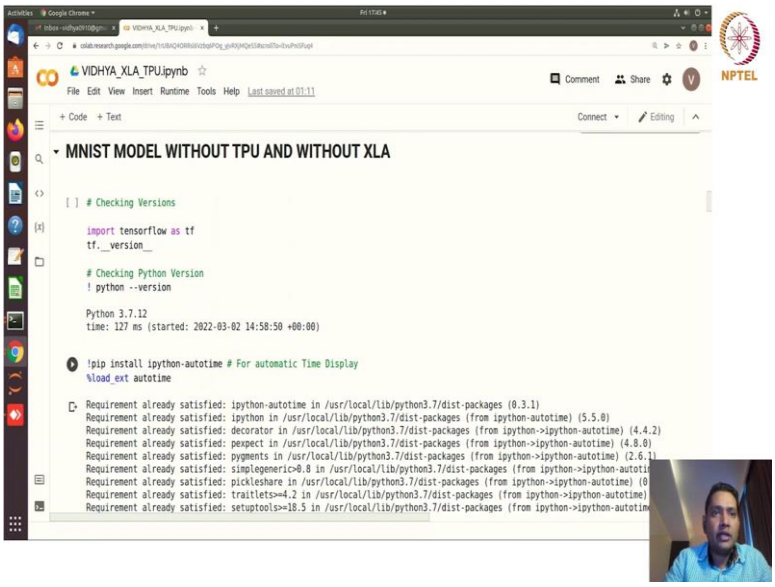
distribute strategy for your TPU and we are using that strategy for running this particular function that we have created.

(Refer Slide Time: 04:06)



So, this is a very simple example how you will actually create the strategy enable the XLA compiler and run ok.

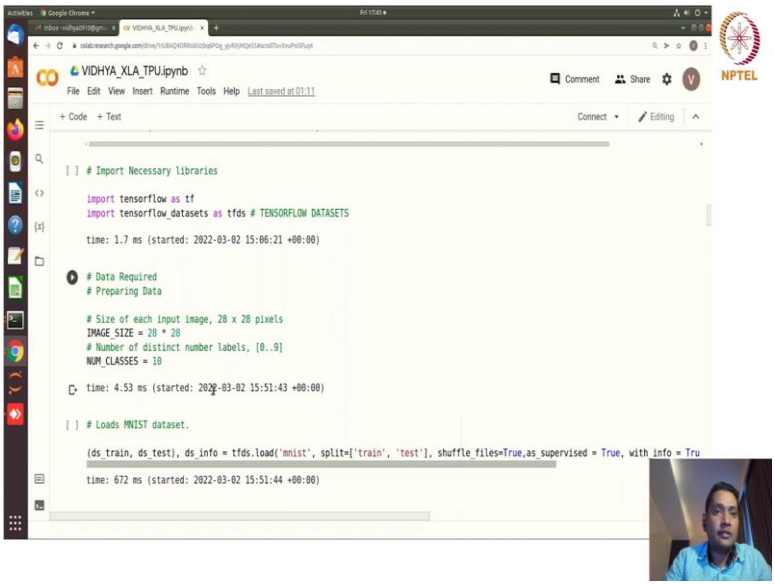
(Refer Slide Time: 04:19)



But let us take a concrete example for model training where you will apply all this. So, basically here what you are seeing that all the necessary imports and version checking

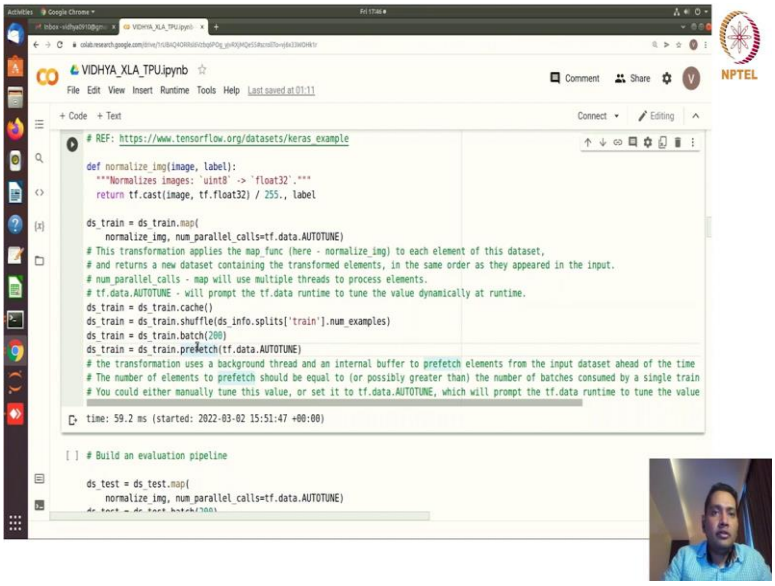
and that is fine. Again, TensorFlow input for the data set because we will use tfds for importing the data sets.

(Refer Slide Time: 04:34)



Whatever the data set that we want to load here we are loading mnist datasets splitting train test and shuffle True supervised True all these things we are setting up good.

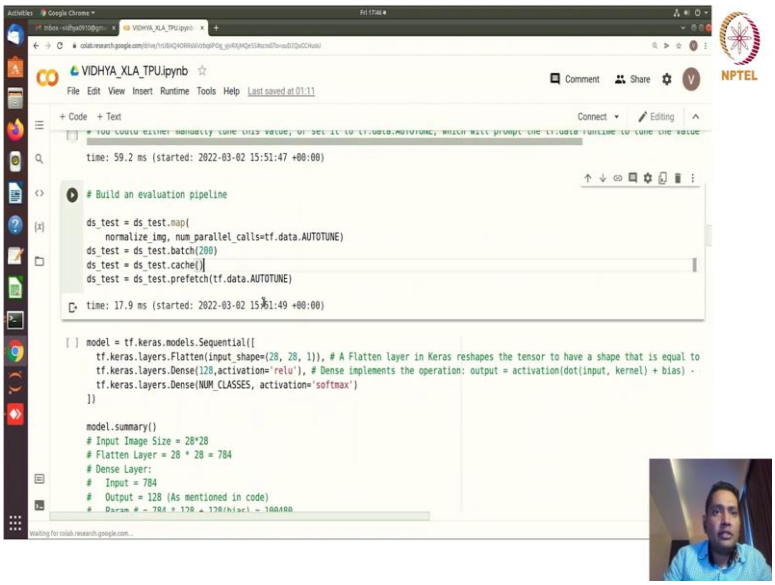
(Refer Slide Time: 04:54)



Then we are normalizing the data and setting up one input pipeline. So, that we talked about previous section also that map is basically transformation with this normalized

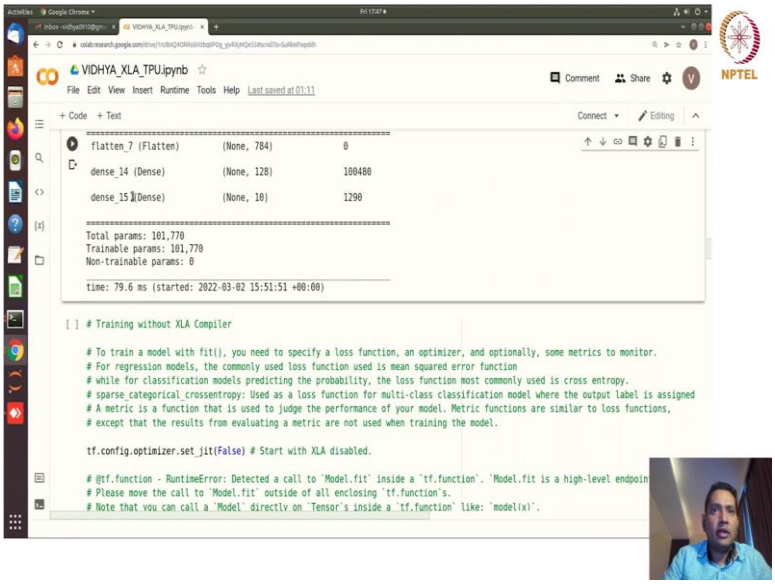
function right. And the number of parallel calls we are letting how many number of parallel calls that will be generated.

(Refer Slide Time: 05:32)



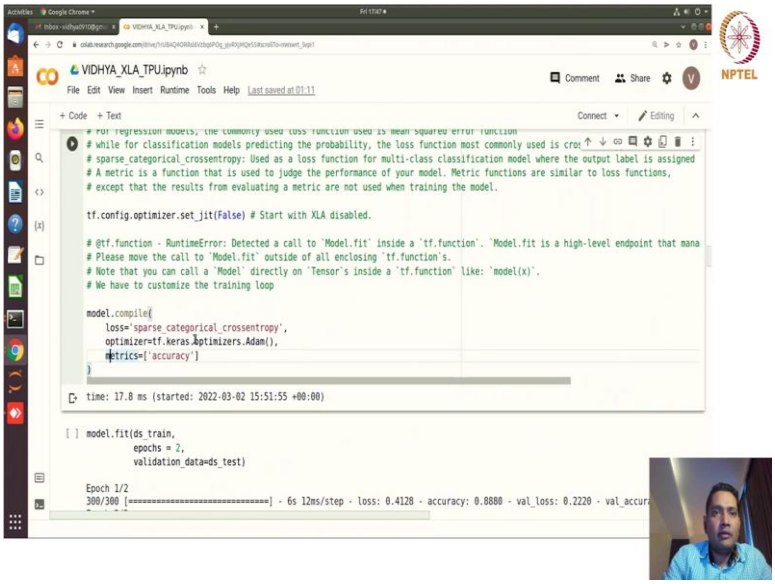
ds_train we are getting the cache enable, shuffle batching prefetching everything is there. So, you now know how to do that now build an evaluation pipeline. So, basically just checking here and creating the model here for the sequential model simply we are creating three layers of model one flattening layer that is just taking the input and flattening it two dense layers are there and this is the output, output of that is soft max function ok.

(Refer Slide Time: 05:58)



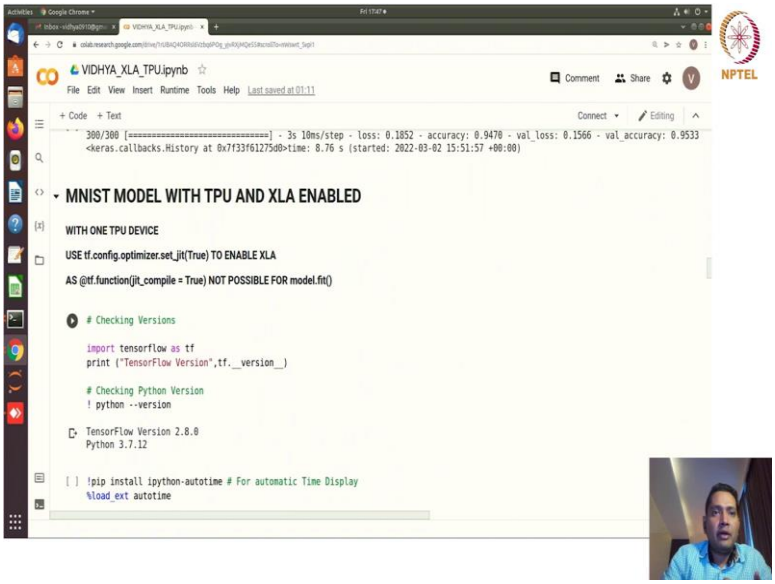
So, model.summary () as you can see here we have three layers here and the summary.

(Refer Slide Time: 06:07)



Now, at first you just to see how it is performing we are setting the tf.config.optimizer.set _ jit (False), this is another way to set JIT compiler equal to True or set JIT compiler equal to False. So, we are disabling XLA because we have enabled previously for that toy example learning then model equal to compile. So, basically, we are compiling the model.

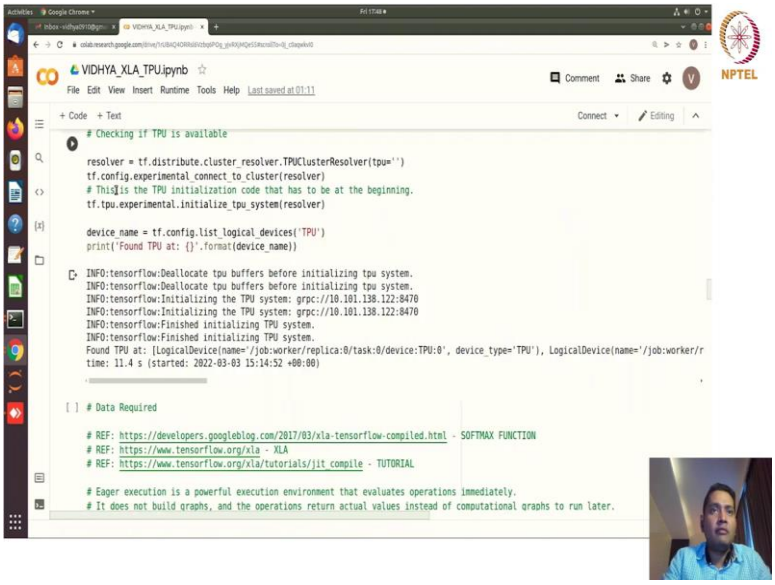
(Refer Slide Time: 06:34)



And now we are creating the training pipeline for that and we are using we will be using since you can see that here we have generated the model using the sequential api. So, it is I mean highly likely that you will use model.fit () ok.

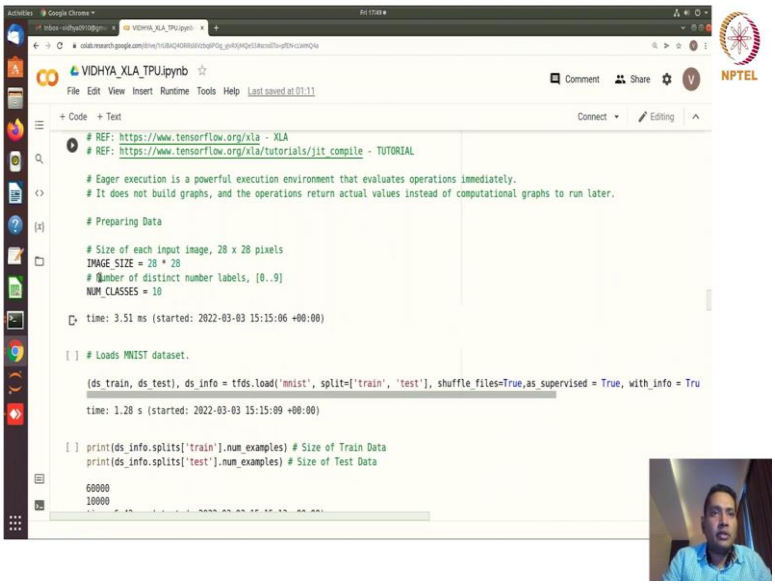
So, but as you can see here as tf.function (jit_compile = True) is essentially trying to make the computational graph and model.fit() is actually again you are doing that, but; that means, tf.function runtime is not supported for your model of fit.

(Refer Slide Time: 07:32)

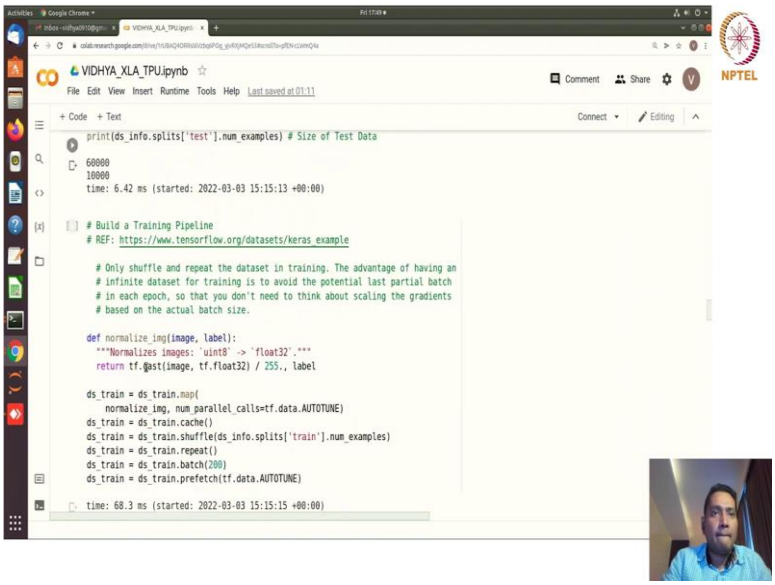


So, if you are using model of fit you will not be able to enable the XLA. So, just importing the tf and data set and running the resolver initiating the devices right.

(Refer Slide Time: 07:41)

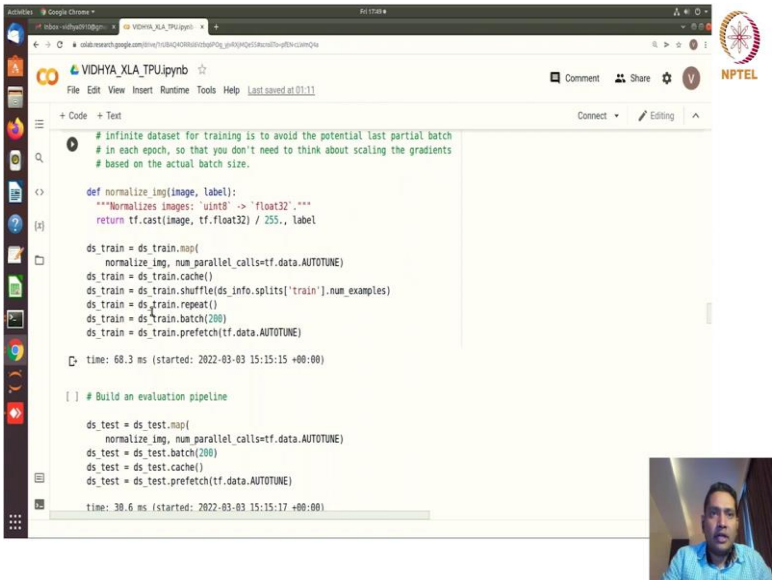


(Refer Slide Time: 07:47)



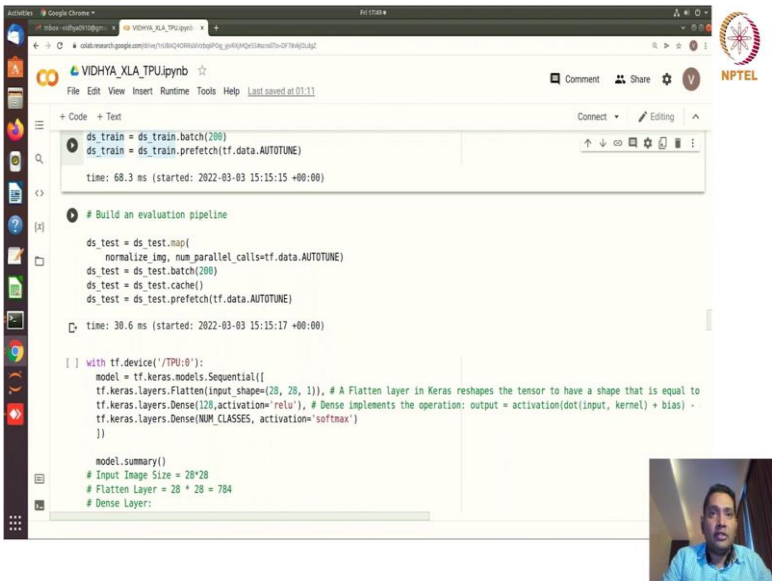
So, all these for TPU that you have seen so far. Just getting some data about the image size and image classes I can see 60000 images up there for your train and test data is 10000 and so forth.

(Refer Slide Time: 07:55)



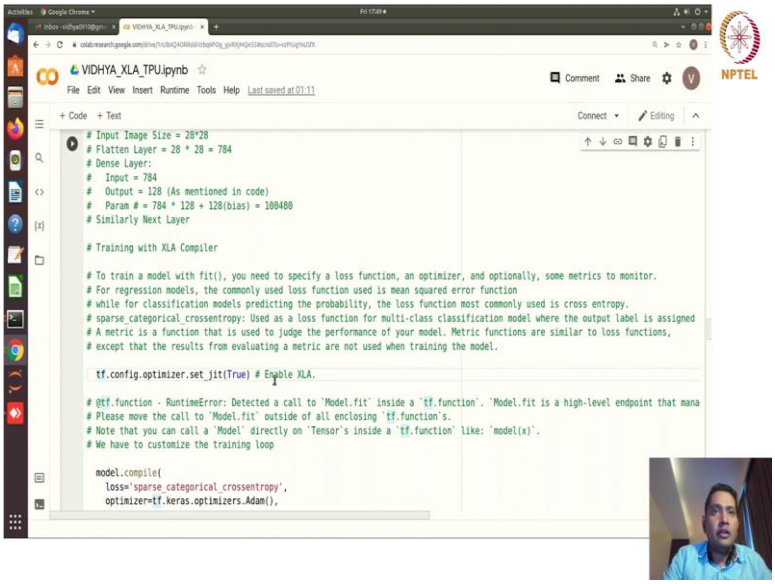
And then this data input pipeline normalize which is used in map in cache shuffle repeat batch prefetch.

(Refer Slide Time: 08:06)



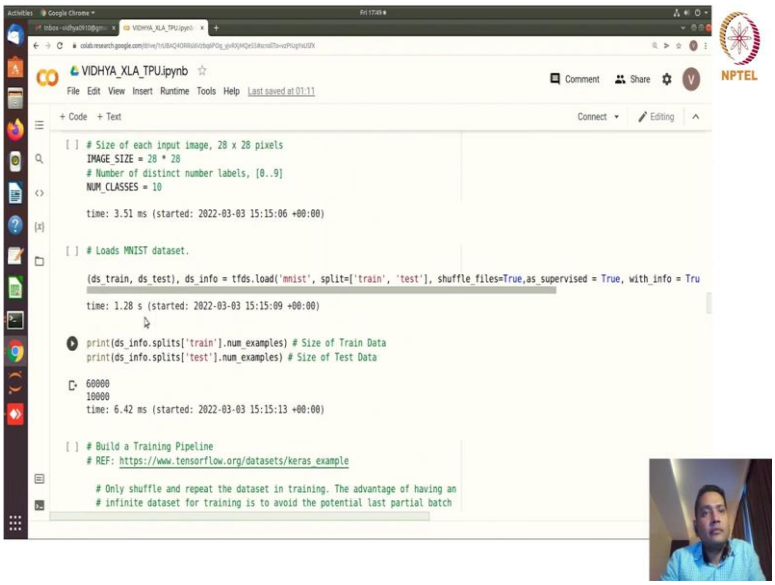
And then we are building the evaluation pipeline now. So, basically calling the functions here again using the tf.device ().

(Refer Slide Time: 08:16)



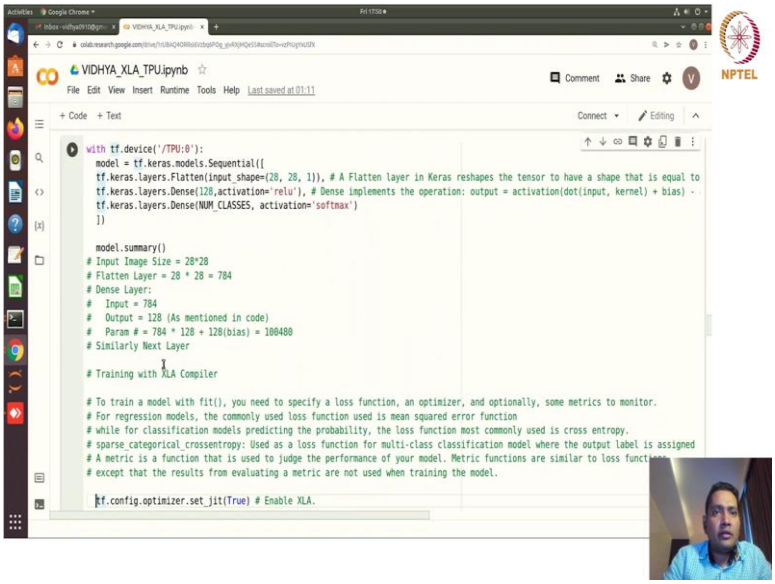
But here now we are enabling the JIT set JIT equal to True. So, if you see the earlier training for that matter.

(Refer Slide Time: 08:29)

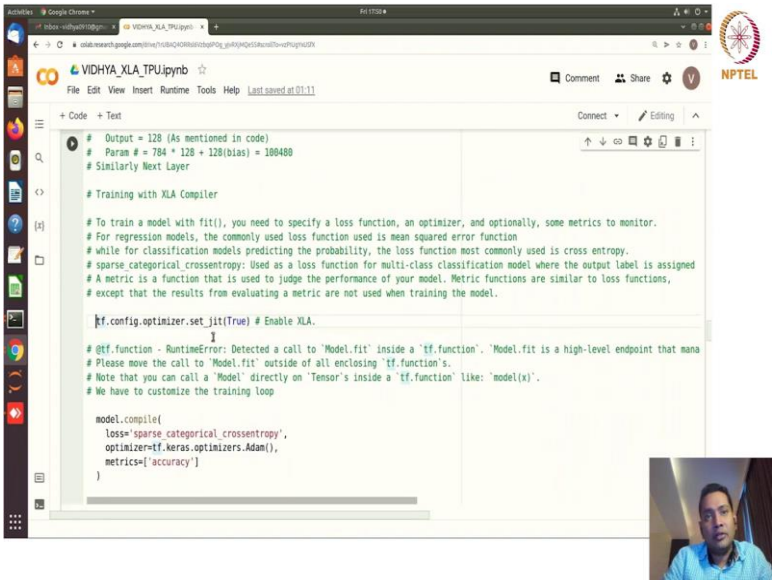


So, if you see here where we are actually using the modeling ok. So, where we have repeated yeah ok. So, ok this is the model summary and yeah.

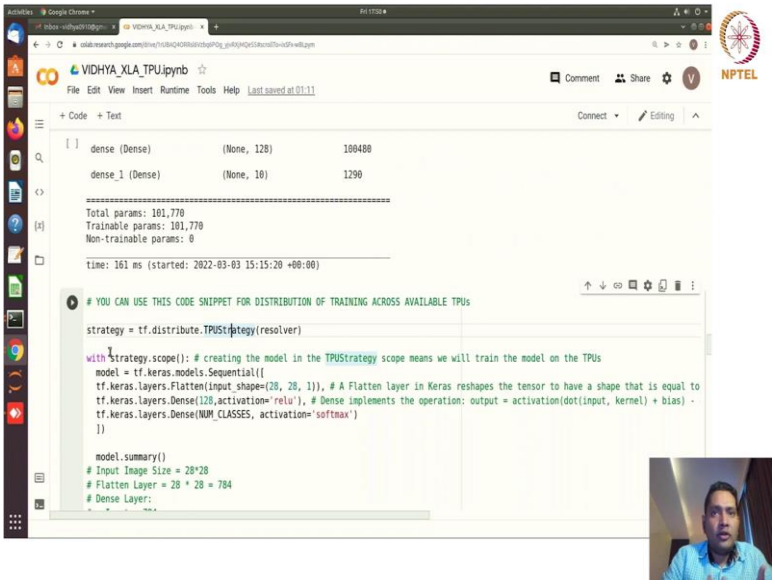
(Refer Slide Time: 08:53)



(Refer Slide Time: 08:56)

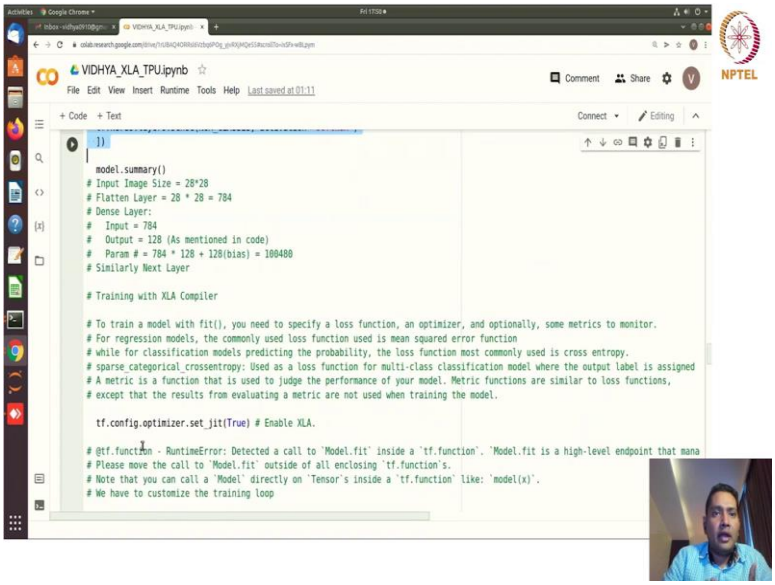


(Refer Slide Time: 09:02)

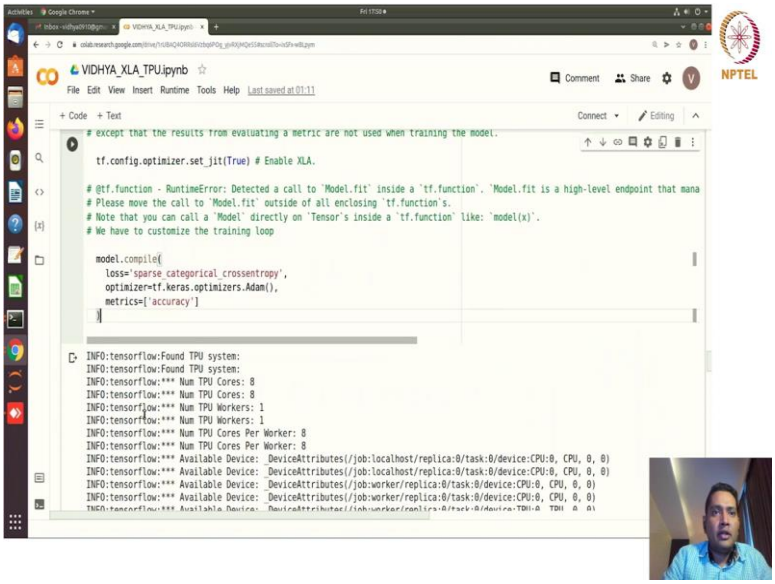


So, set JIT True and compile. And then we are using the distributed strategy ok TPU strategy, because we want to distribute all the TPUs that is available here. And thus inside this strategy scope we are defining the model sequential.

(Refer Slide Time: 09:16)

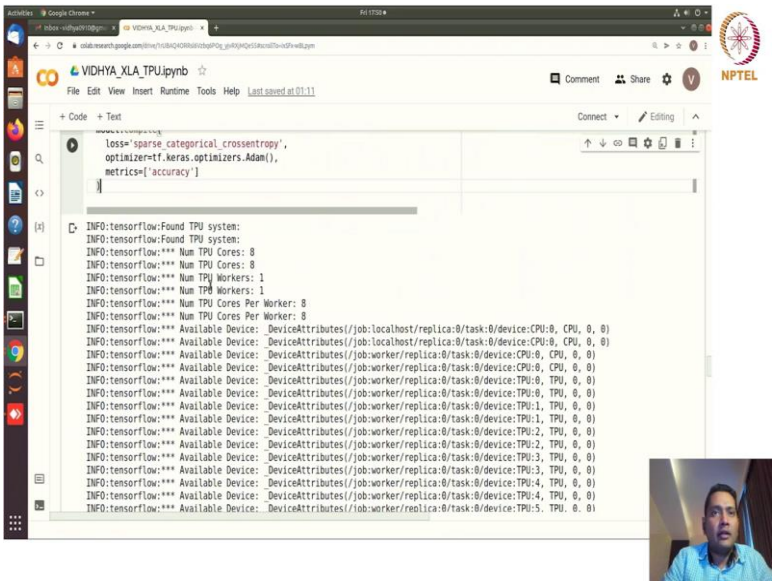


(Refer Slide Time: 09:24)



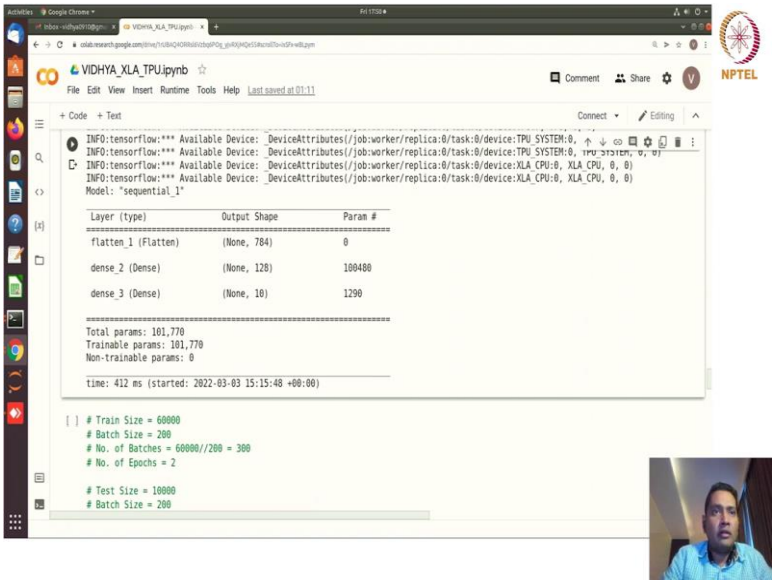
And after that we are generating the model summary and setting up the compiler here ok XLA True.

(Refer Slide Time: 09:26)

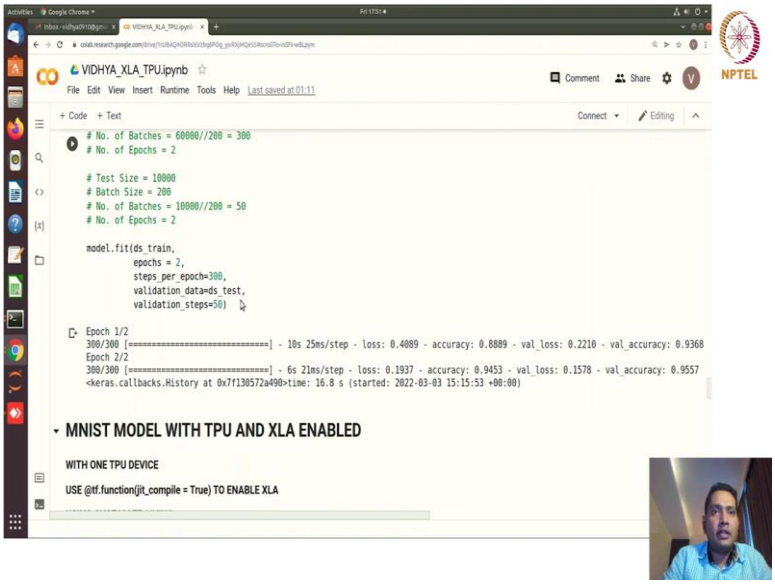


Now, we can see total TPU cores are 8 and TPU workers = 1 and you can set it like how many workers you want you have seen in the previously how you can configure those things.

(Refer Slide Time: 09:39)

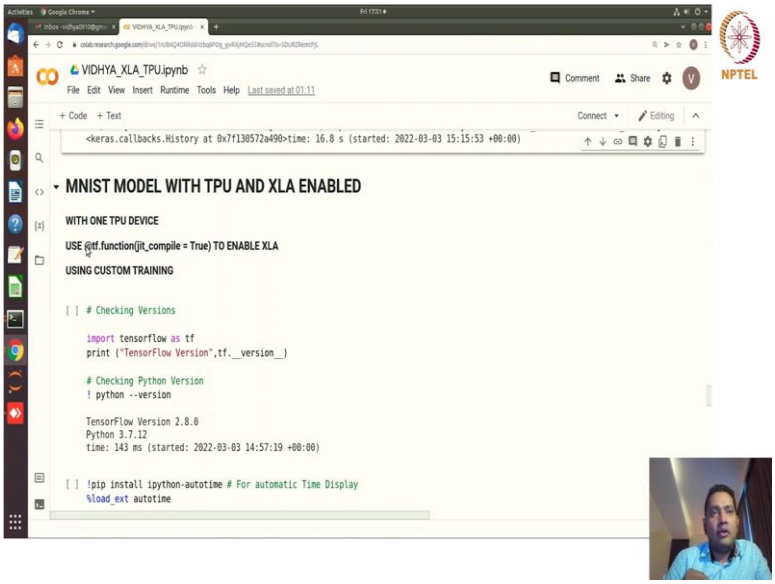


(Refer Slide Time: 09:43)



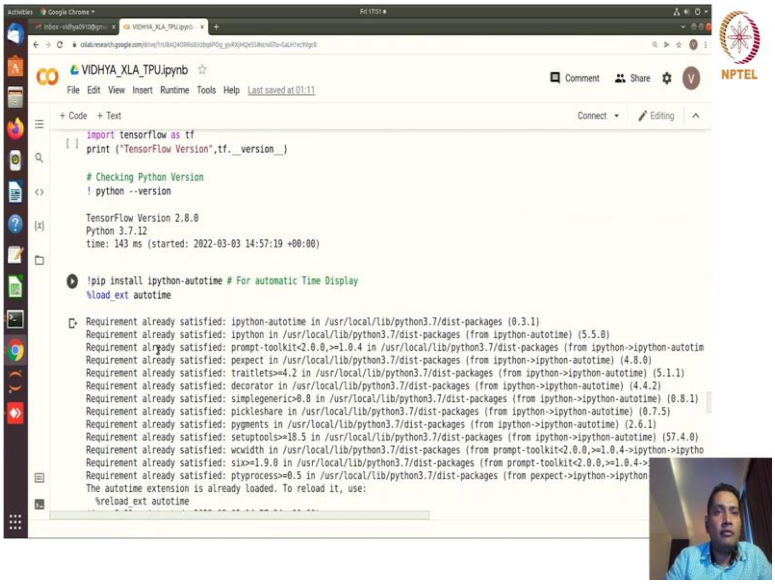
And then model summary you have seen and then we are fitting the model to the model fitting is essentially running the model for you generated graph for that and you are using that and total of 16.8 seconds you are spending for your training of 2 epochs.

(Refer Slide Time: 10:05)

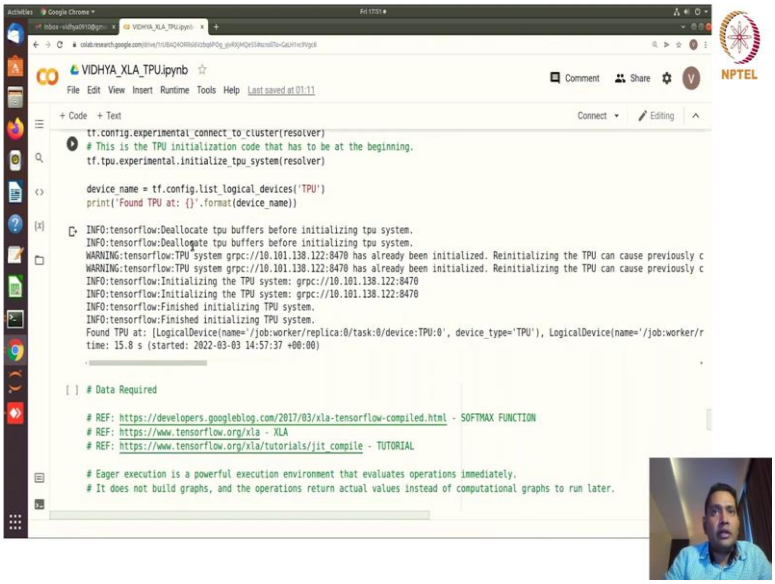


Now, we are moving towards the example where we want to enable the XLA, but for with the TPU and also with tf.function runtime because the tf.function runtime has its own optimizations that you have seen. Because it is serializing the computational graph, that is not possible for your model.fit() and that is why the performance was not that good simple.

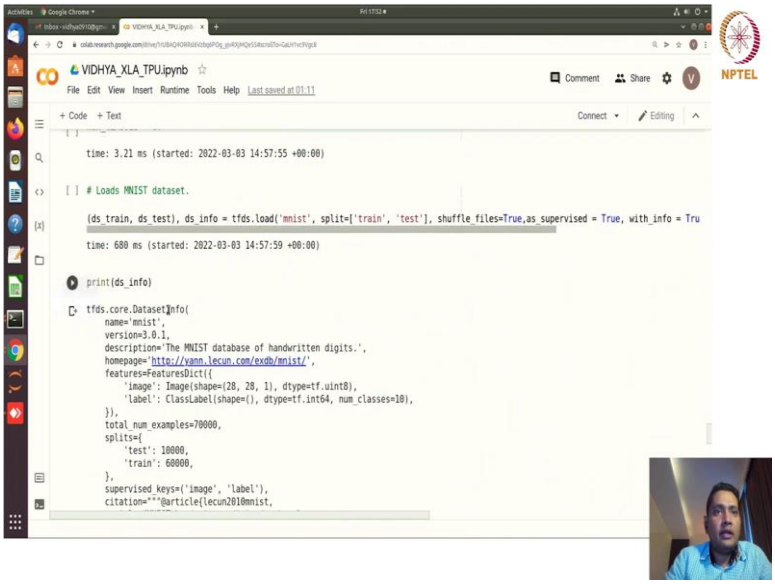
(Refer Slide Time: 10:32)



(Refer Slide Time: 10:38)

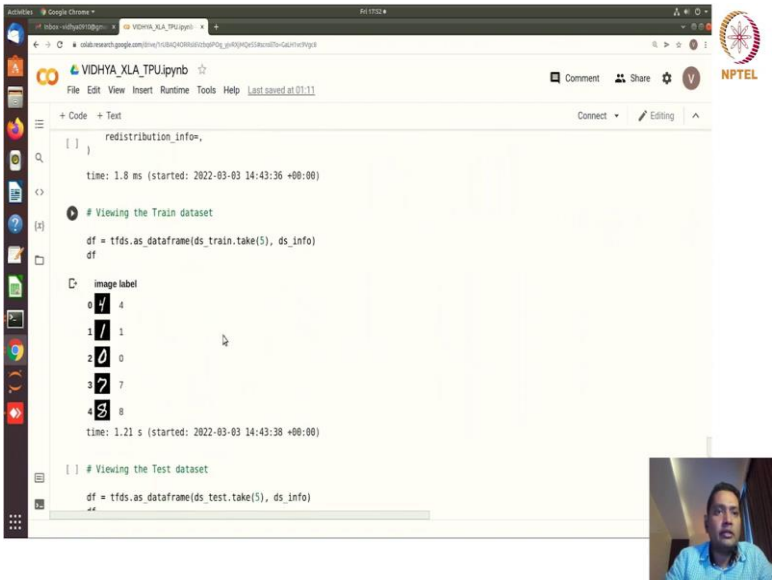


(Refer Slide Time: 10:44)

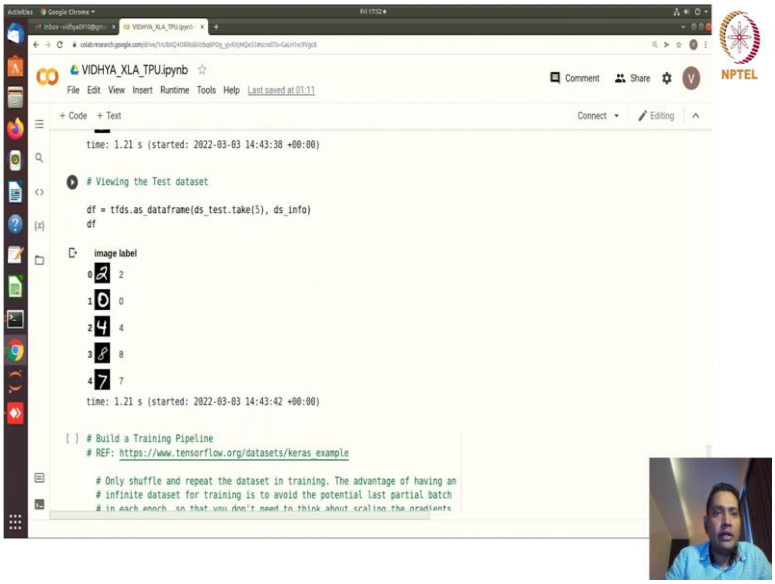


Now, the same setup importing the TensorFlow all the data set libraries then initializing the resolvers right.

(Refer Slide Time: 10:48)

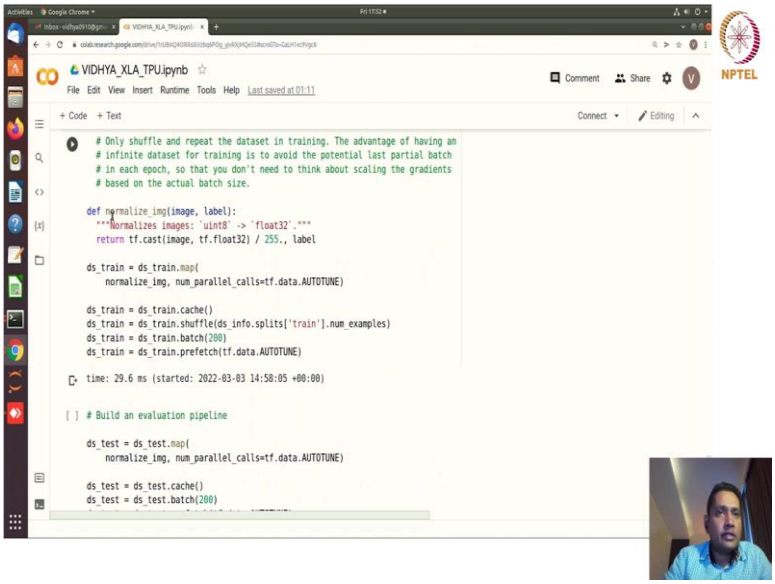


(Refer Slide Time: 10:53)



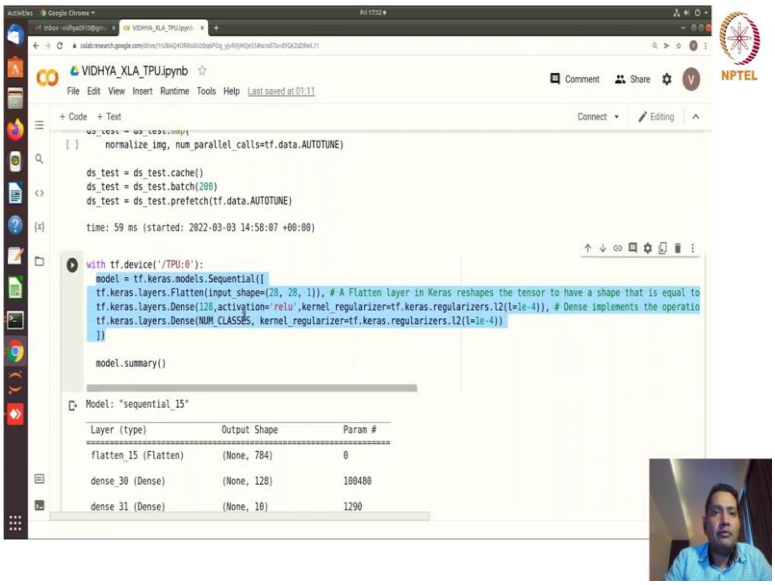
And something regarding the data and setting up the just viewing the data frame also you can see these are the handwritten dates images from the data set itself.

(Refer Slide Time: 10:56)



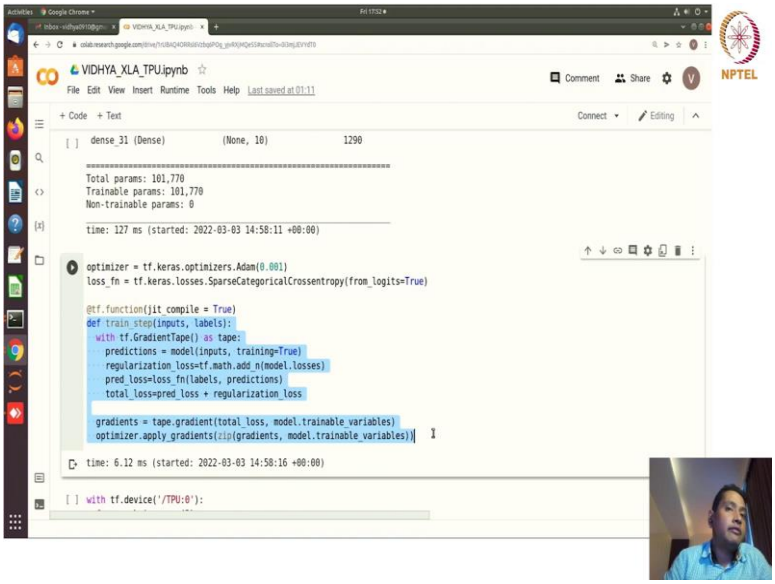
And then setting up the input pipeline data input or data loading pipeline for normalizing mapping caching shuffling batching and prefetching.

(Refer Slide Time: 11:08)



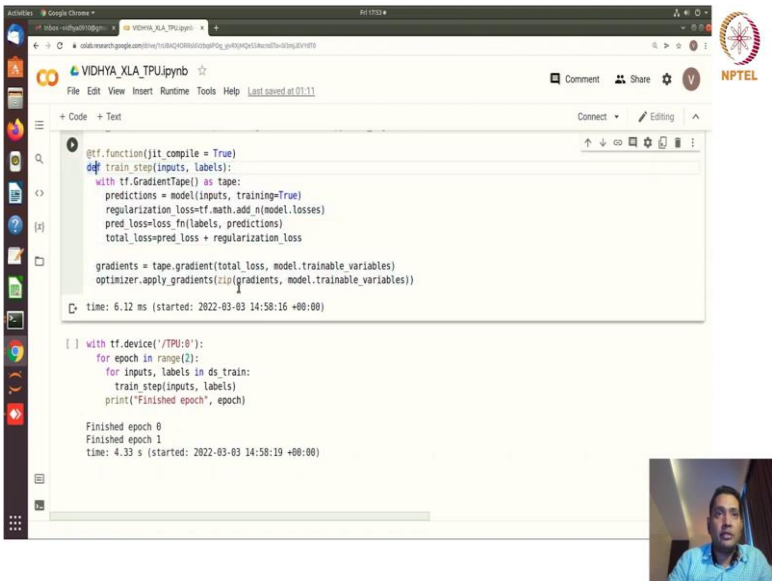
And then with this TPU device. So, one TPU device we are using again you can also use any other strategy also. So, sequential model that we have defined for that device.

(Refer Slide Time: 11:22)



And then defining the optimizer and loss function. But now we are actually defining the training step, but not using the model of fit.

(Refer Slide Time: 11:41)



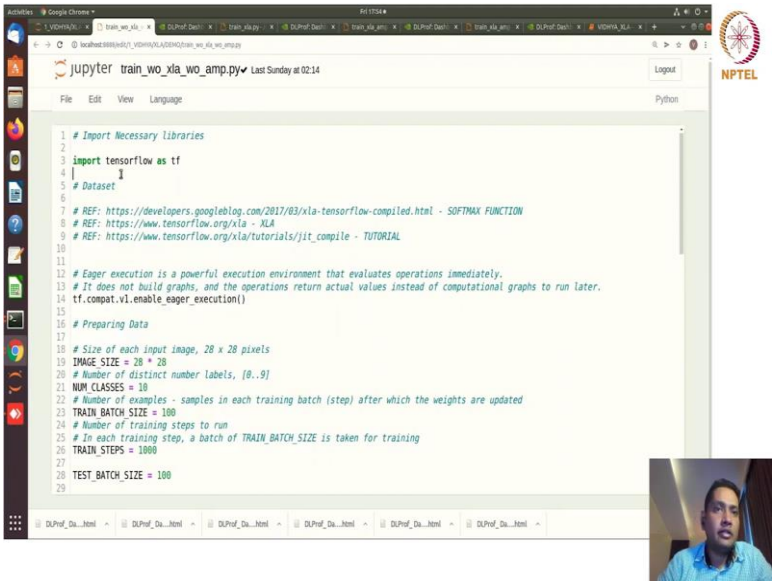
And we are wrapping it with tf dot function runtime with JIT compiler equal to True and after doing that we are actually calling the train step for these number same number of epochs. Because we have run for two epochs previously and 4 times speedup that we talked about in the session in the slides you can see 4 times speedup you are getting right.

So, this is a very simple example where you can set up your own pipeline for your training and you can define like, which exactly steps which are the steps that will be wrapped with your with your trained tf dot function runtime which JIT compiler. And this is one example where we have done in TPU ok.

Now, we can extend this for running it several TPUs and see what are the strategies supported for a XLA compile and play with it ok. So, this is where your exploration for different strategies and you know whatever methods that you have learned from this course we will apply right. So, you take this code and play with it right.

You can also use keras data set and see what happens for the particular example that we have seen. Because we have we have used tfds the TensorFlow data set and try to use keras data set and see what happens ok.

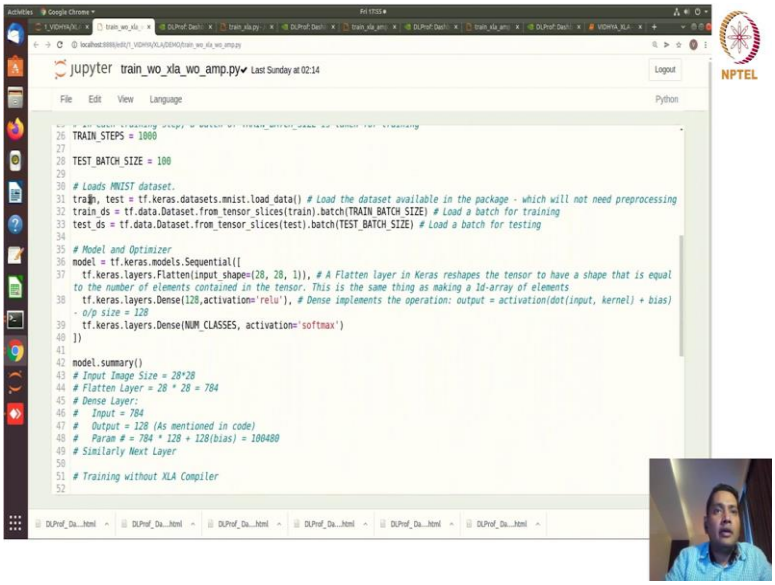
(Refer Slide Time: 13:28)



So, explore all these things you can note it down and you can explore this, because a lot more interesting things that you can discover I am telling you ok. So, let us go into the GPU now right. So, we have seen the TPU part where we have enabled the TPU execution setup environment and wrapped the function with the tf.function(jit_compile = True).

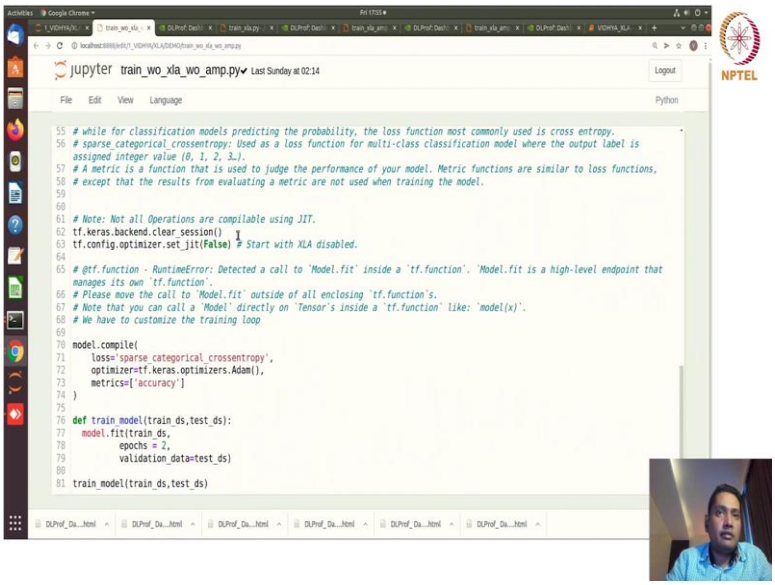
Now we will train first one naive way because we want to see right what is happening here with the GPU and of course, without amp the automatic mixed precision without XLA without anything its vanilla training model right.

(Refer Slide Time: 14:09)



So, where you have the strengths of imported the image the same thing that you have seen in the TensorFlow model.

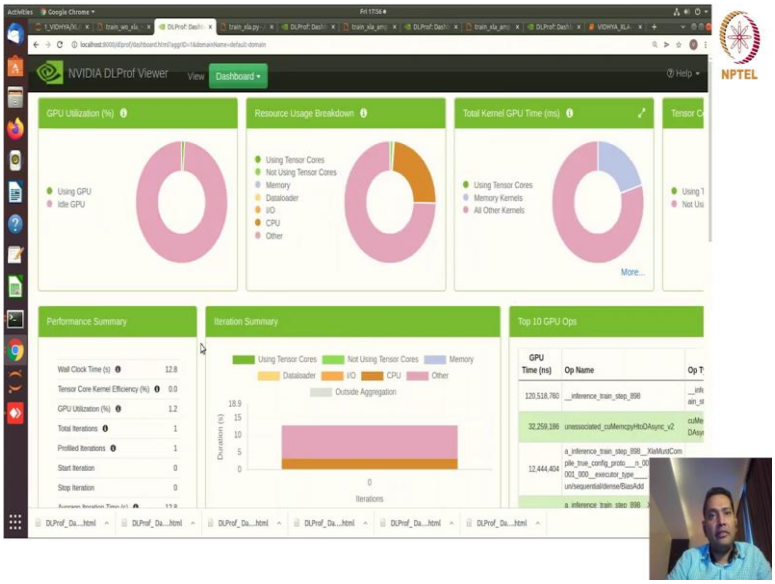
(Refer Slide Time: 14:28)



So, same training module here train set test data set we are unpacking it model defining with the sequential api model summary we are checking. And then clearing the session if

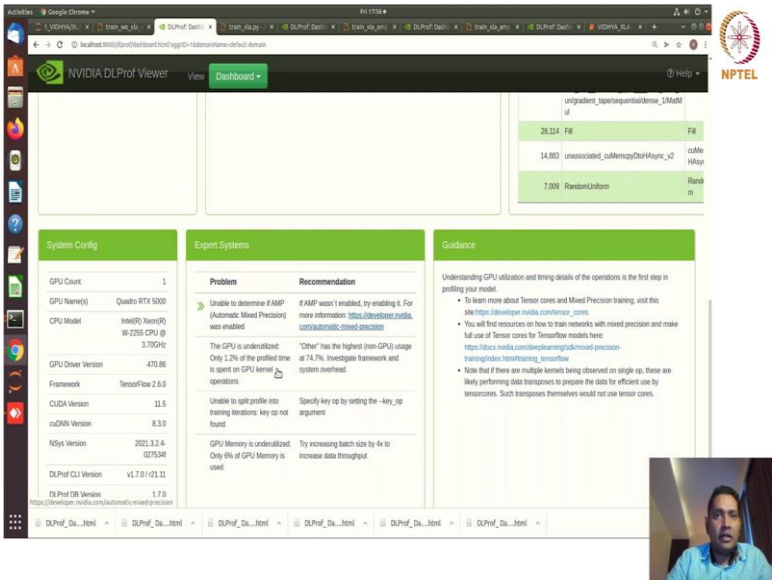
we have, let us say running previous JIT compilation session is already on we are just doing it just for safety. We are clearing the session and setting the XLA disable model dot compile will compile the model train model with this fit function right. So, that is it that is the vanilla training for the TensorFlow particular pattern.

(Refer Slide Time: 14:57)



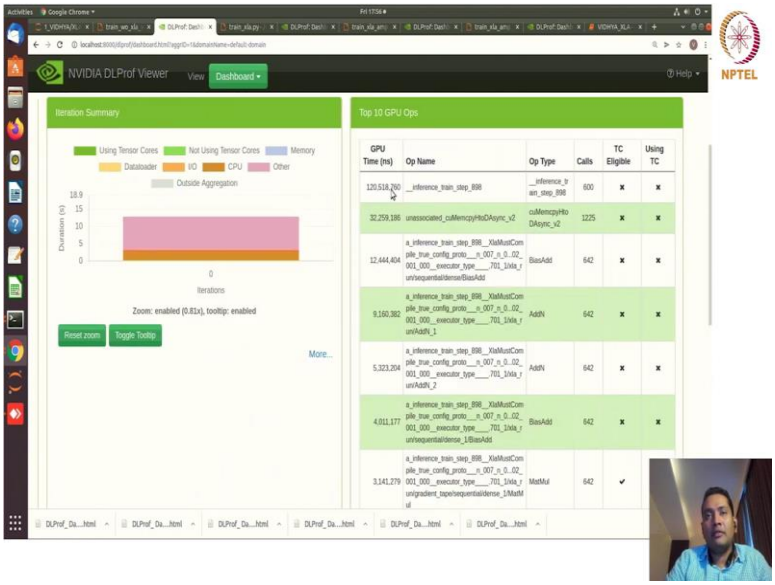
So, interestingly your DLProf will figure it out that you are actually not using XLA think I do not know. So, we will see I hope this is the profile for after training with this.

(Refer Slide Time: 15:07)



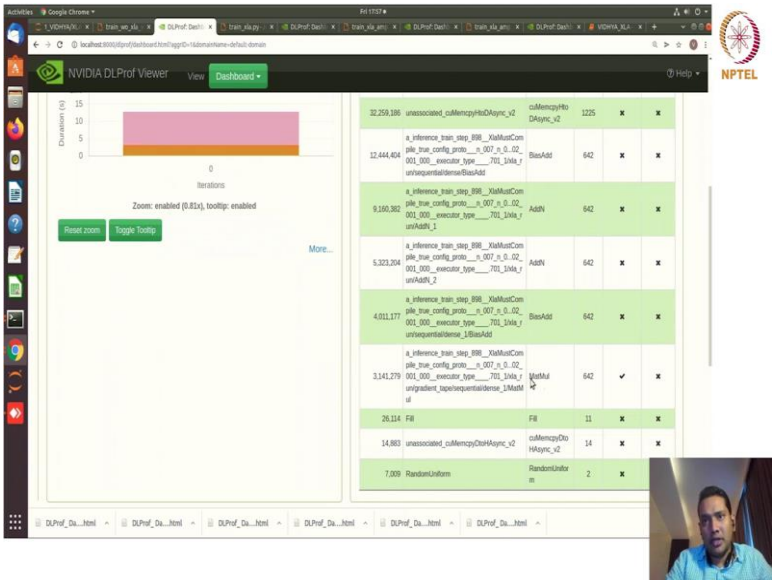
So, basically you can see if amp was not a enabled GPU was not greatly used and so on and so forth. Unable to split profile into training iterations, so ok that is ok. GPU memory is underutilized that is fine because of the very small model right.

(Refer Slide Time: 15:34)



And all the operations that are being actually called and executed how many times of operations execution call and how much time you have spent for those operation call that you can see profile it here.

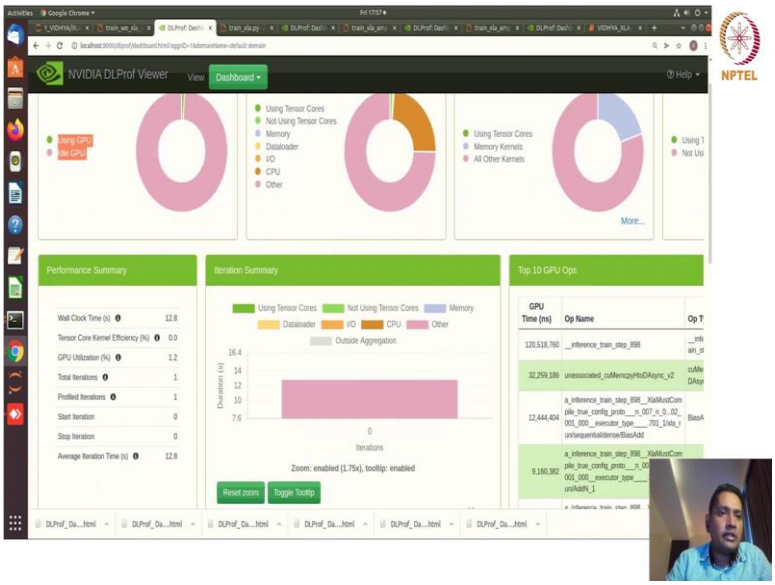
(Refer Slide Time: 15:51)



And you can see few of the operations that are matrix multiplication operations which are actually tensor core eligible or not eligible. So, whether using tensor core now we are not using because we are mixed precision.

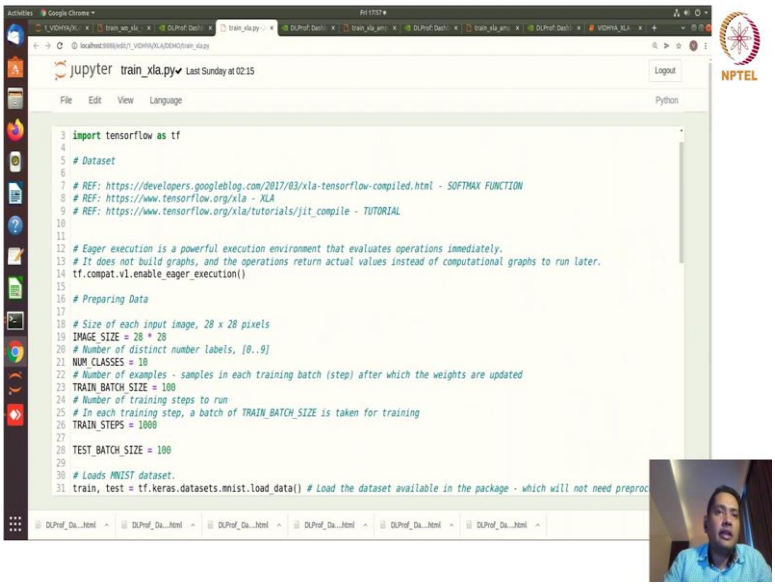
And all these profiles you can get it from right. So, ok so just to see like this is very my way of setting things up right.

(Refer Slide Time: 16:24)

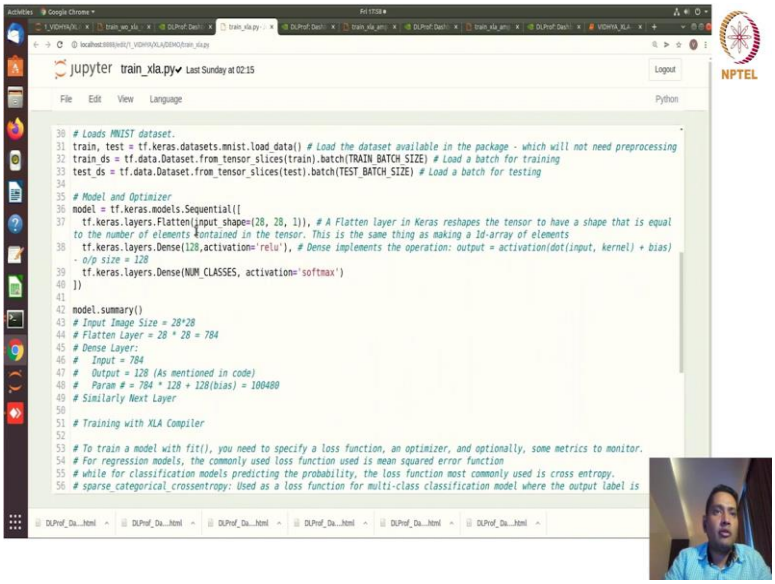


Now, of course, average time you can see 12.8 now of course, we want it accelerated right. So, let us see what happens if we enable the accelerate right.

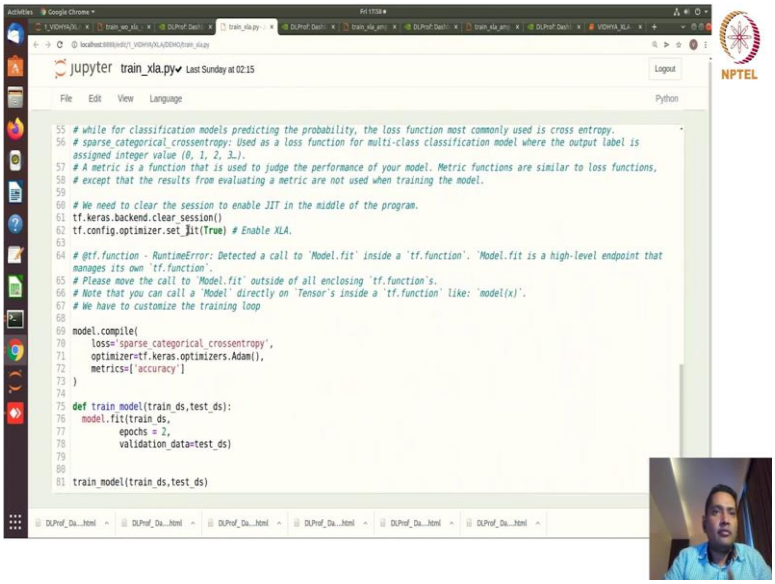
(Refer Slide Time: 16:34)



(Refer Slide Time: 16:41)

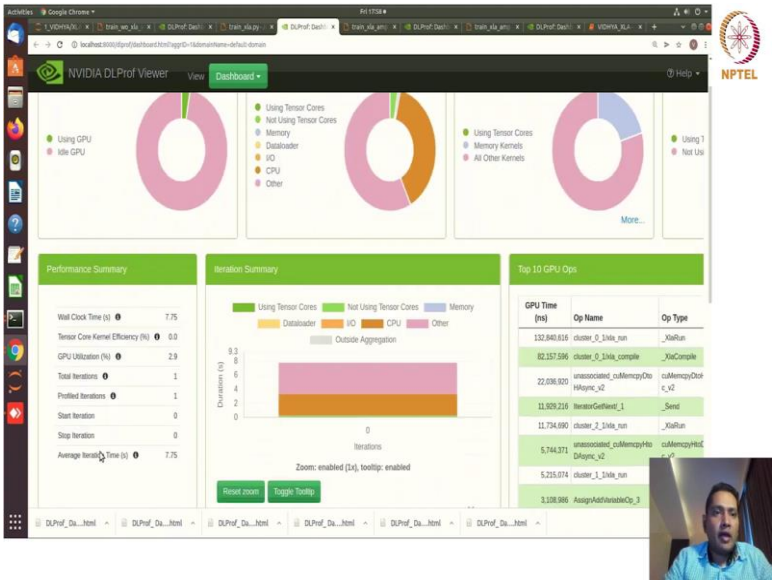


(Refer Slide Time: 16:46)

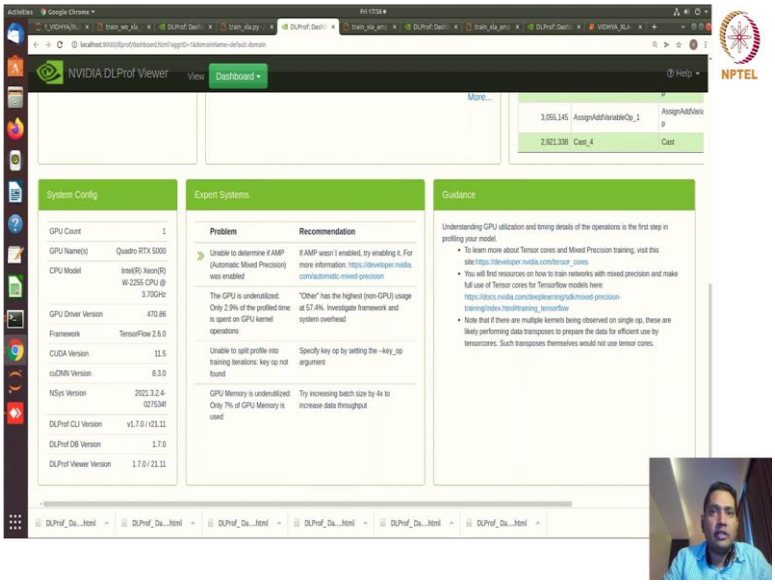


So, the same thing same train the data set unpacking model definitions are clearing in the background and here we are actually enabling the XLA and then model compile model train model with fit function and train model. So, this is a simple same just with two lines of code we are actually enabling the XLA.

(Refer Slide Time: 17:05)

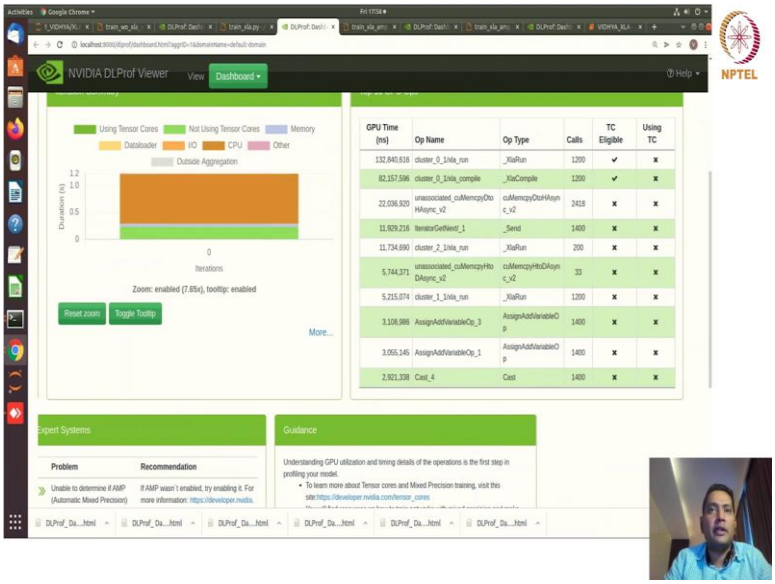


(Refer Slide Time: 17:16)



So, let us see what happened here average time got reduced to 7.75 that is good and, but still. So, if you see of course, amp was not used and GPU is underutilized that is understandable.

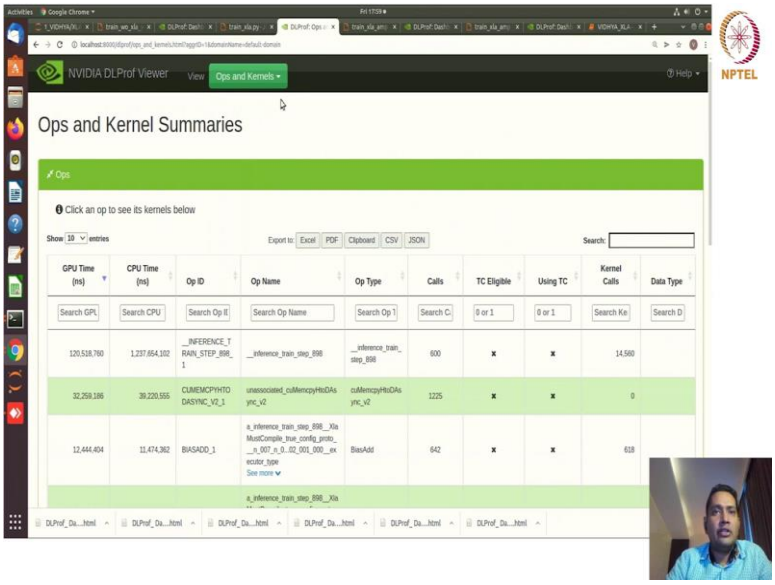
(Refer Slide Time: 17:25)



And of course, again you can see the functions, that is and also you can see which is the operation that is being actually run by your optimized code ok. So, for the from the XLA part.

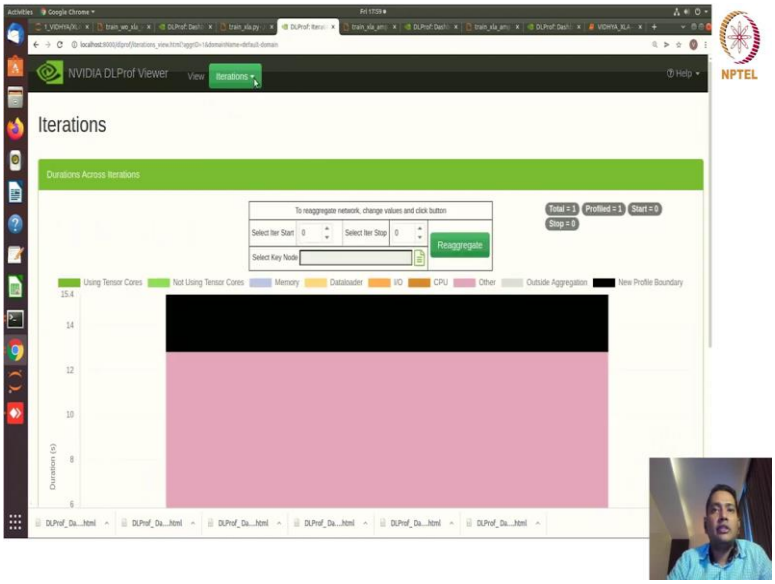
I have not shown other views you can exploit those. So, as I was mentioning that it is not possible every time to go to every tab and explain you everything, but you can explore the other views as well.

(Refer Slide Time: 17:59)



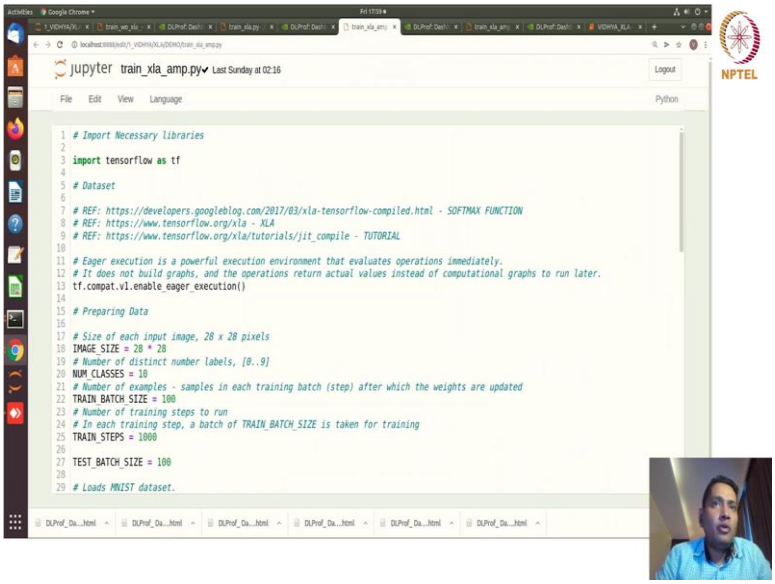
Let us say you want to see the operations and kernels that are being executed you can see the list of the operations and all the statistics behind those you can export them in PDF, CSV, JSON format or excel format whatever you want. Kernel's by operations iteration kernels by operation iterations.

(Refer Slide Time: 18:15)



So, all the things you can explore from this profiler right. But of course, this is where you will explore more ok.

(Refer Slide Time: 18:26)

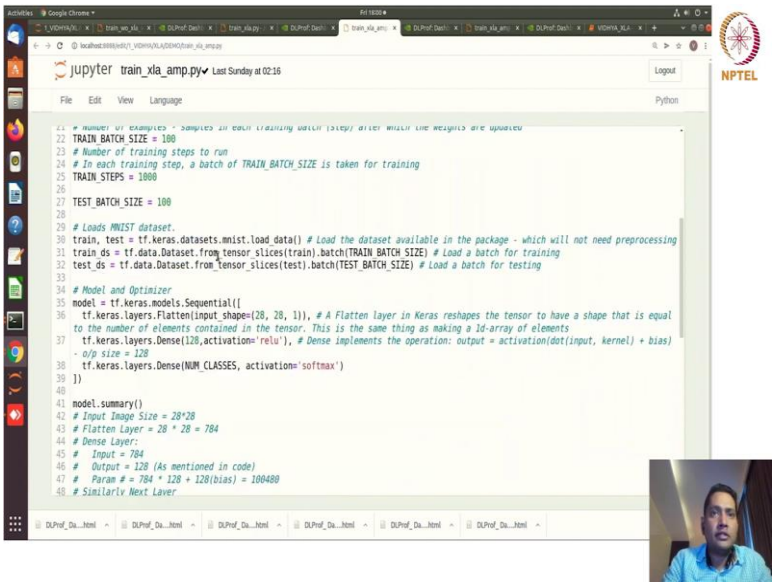


Now we want to introduce amp also and enable XLA at the same time right.

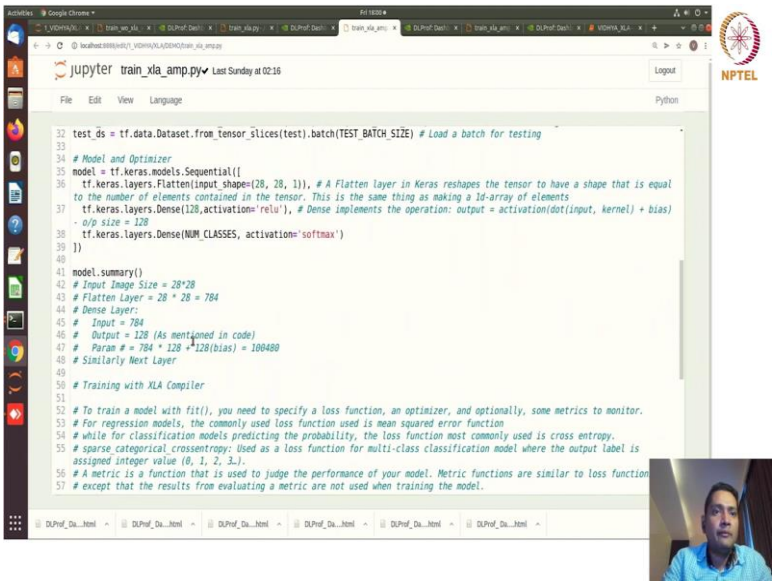
Now, we want more performance we are just we do not know what is what will happen right, but we are just we know that if we enable amp and use mixed precision then we can you know use the tensor cores more ok we can increase the utilization, so let us do that.

And also let us keep the XLA on right. So, I do not know what will happen. So, let us see eager execution. So, enabling the eager execution here the image size all the images train test data set unpacking and definition of the model, model work summary.

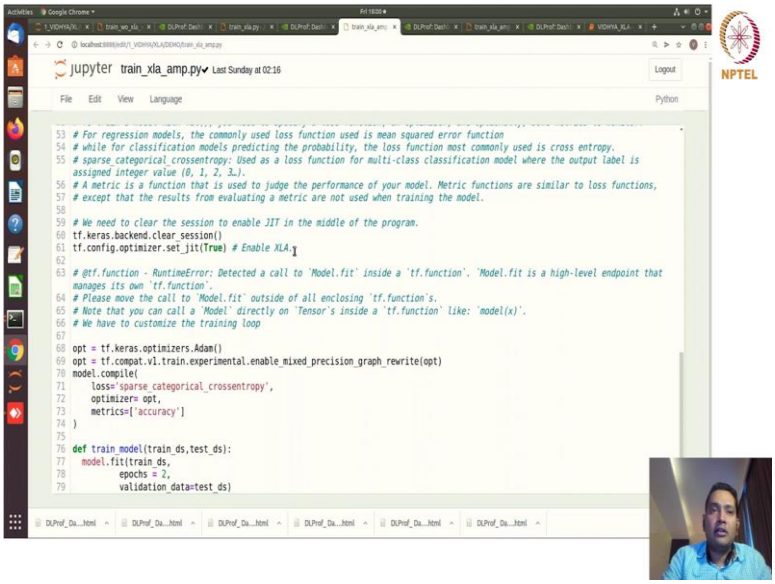
(Refer Slide Time: 19:11)



(Refer Slide Time: 19:16)



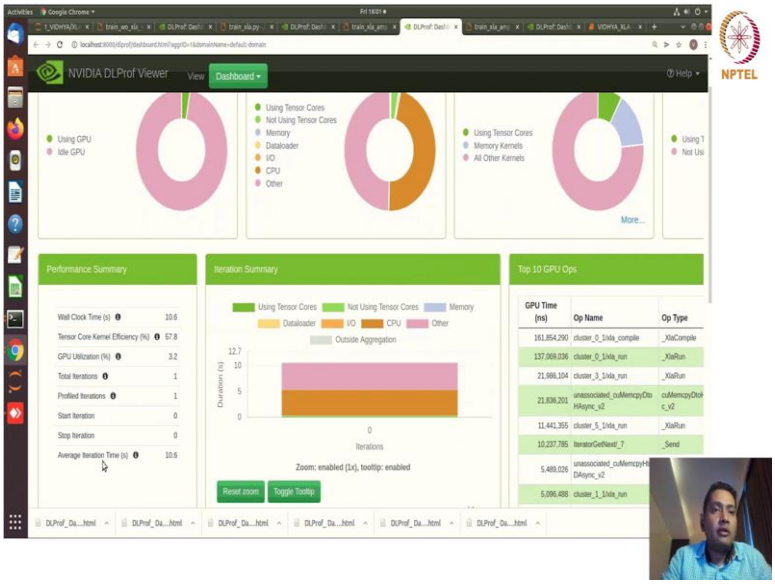
(Refer Slide Time: 19:18)



And here we are enabling XLA also at the same time. So, in the previous example in the prior class that you have seen, how we are getting more utilized tensor cores by using mixed precision in our pytorch class.

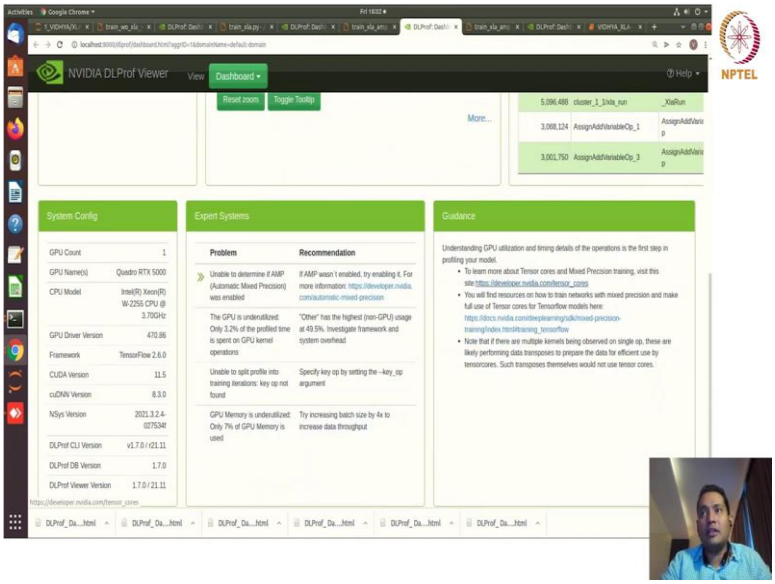
So, extensively we have not used mixed precision in TensorFlow, but you can use that how you can use that here with optimizer basically you can scale it right. So, experimental dot enable mixed precision graph rewrite you can enable that for this optimizer and you can use that optimizer inside your model and train that model as simple as that use mixed precision.

(Refer Slide Time: 20:07)

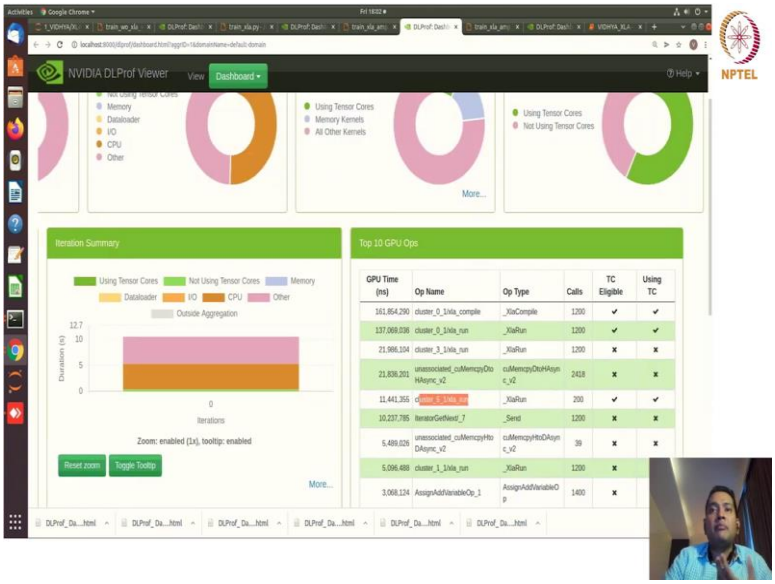


And let us see what happened here and oops our average iteration time got increased so; that means, we have degraded performance. So that means, the, but introducing everything XLA mixed precision everything in the tensor processing unit which is the GPU we were getting much more performance right and that is because the that that is because Google ok.

(Refer Slide Time: 20:47)



(Refer Slide Time: 20:50)

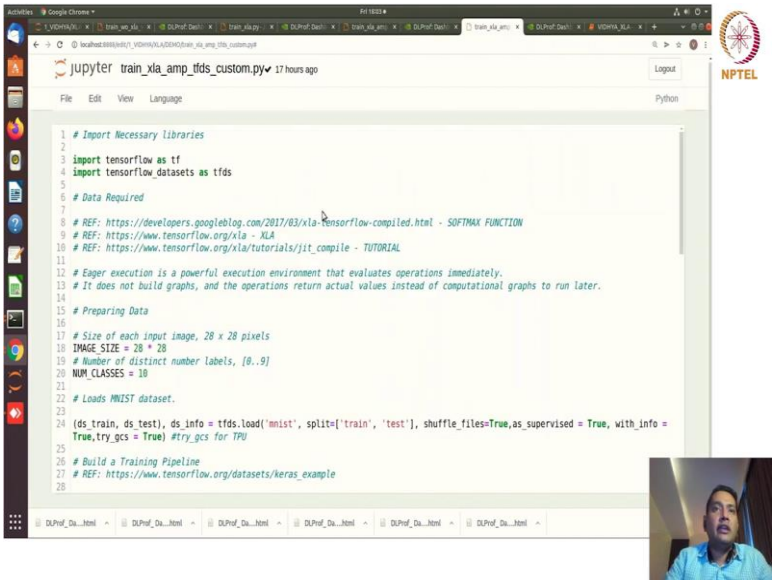


So, basically google supports the mixed precision binary fusion which is supported in GPU, but not supported inside your GPU and you will not keep tensor course utilized. So, still your GPU is underutilized and whatever XLA run you have defined it is not using the tensor core.

But some of the in the cluster of 5 a xla run this 200 is using the tensor core ok. So, some parts will not be actually compatible for your target gpu. So, you just need to use some libraries.

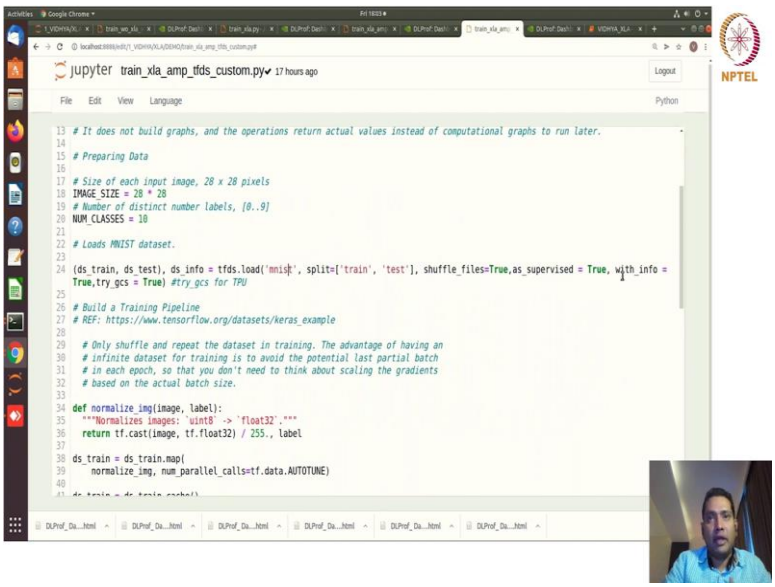
So, now here we are using the amp library from Google itself again you know there are two libraries for that one is NVIDIA apex and the native library from PyTorch both from PyTorch and the tensor core. Now you can know what is what we use for getting enhanced performance right.

(Refer Slide Time: 21:43)

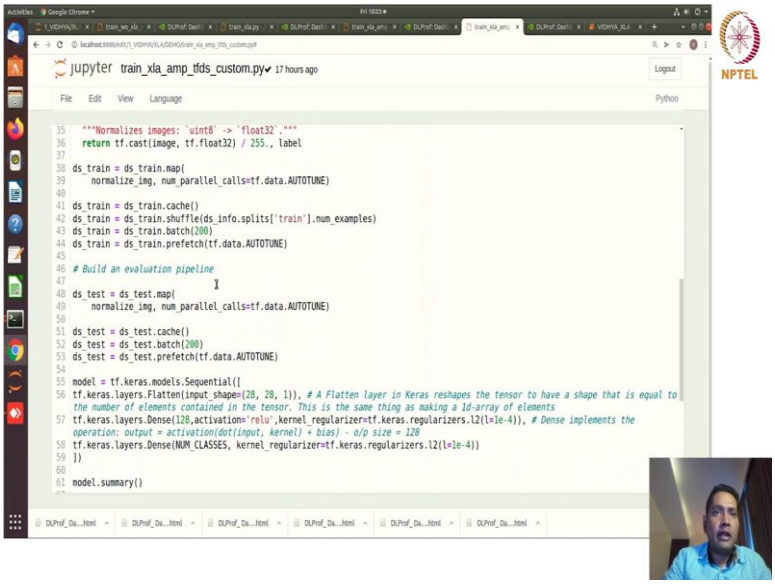


Now in the next setting, we will see how you can actually also enhance the training pipeline with tfds right. So, we will now formulate the pipeline with tfds right as you have seen in the TPUs right.

(Refer Slide Time: 22:04)

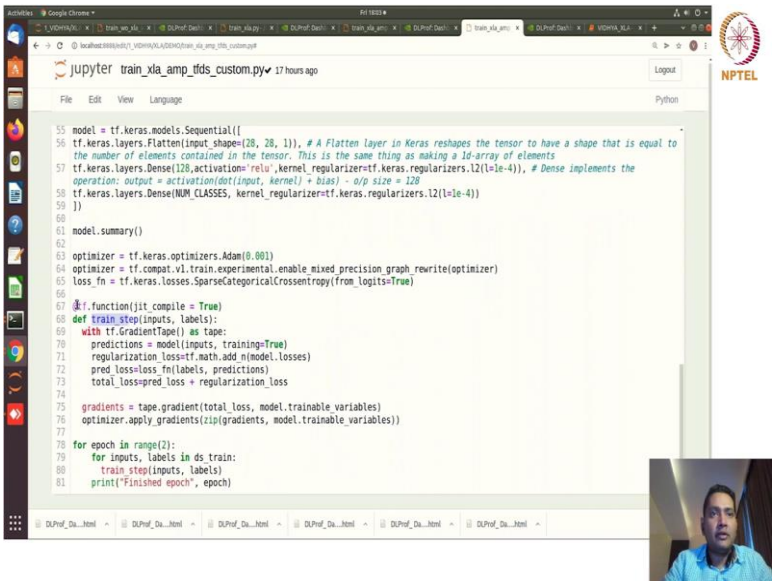


(Refer Slide Time: 22:14)



So, training data with tfds dot load from mnist splitting training test and comparing and formulating the pipeline here.

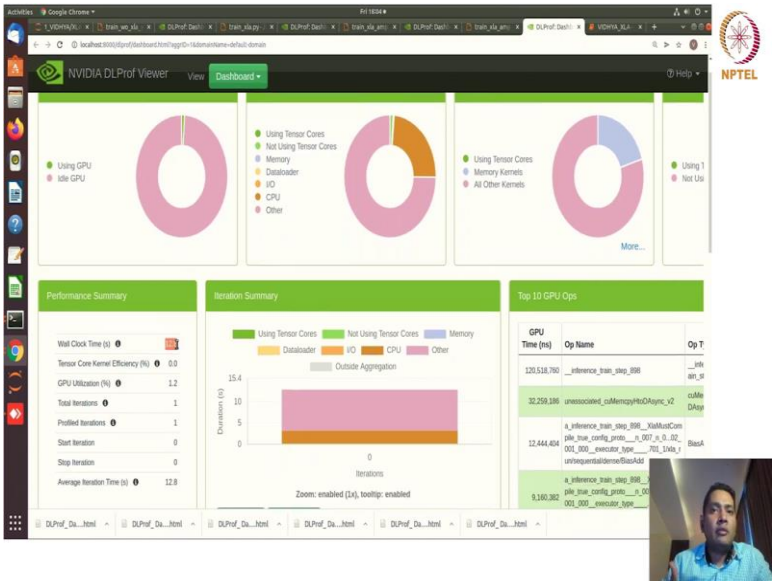
(Refer Slide Time: 22:24)



And then map cache batch prefetch sequential model definition model summary optimizer and mixed precision enabled sparse category loss we are using here from the logics, that we are getting from the last layer and defining the train step with runtime function tf.function(jit_compile = True) with this we have wrapped this function.

And. So, I do not know everything we are using here. So, the tfds data pipeline the JIT compiler for the function training step and mixed precision caching of data load I mean I do not know what else we can have here right.

(Refer Slide Time: 23:05)



So, let us see the dlprof ok. So, with enabling XLA we will not give you always the best performance that I have told you in the slides right. So, that is exactly what we are doing here 12.8 it is not exactly what we want. So, if you blindly just enable everything that does not mean that you will get the better performance.

Conclusion is you see analyze what is best for you and actually what you will get from the exploration and profiling. So, use the profiler always like how you can actually see where is the performance you are going to increase or decrease depending on which strategy you are using. So, all these are basically exploration based ok.

And for your target GPUs for your target TPUs what will be the best for you explore that and see what actually fits for you ok. What are the optimizations that are good for your target architecture you explore. You explore this the Kernel's here that is that that are being generated and see what kind of visions you have after the training.

You can inline the you can inline to see the intermediate results. So, I have shown you in the class slide that part of the techniques for enabling the optimizer graph for generated.

So, you can use this and see what kind of optimized graph you see. So, with this we will conclude the session and.

Thank you for that.