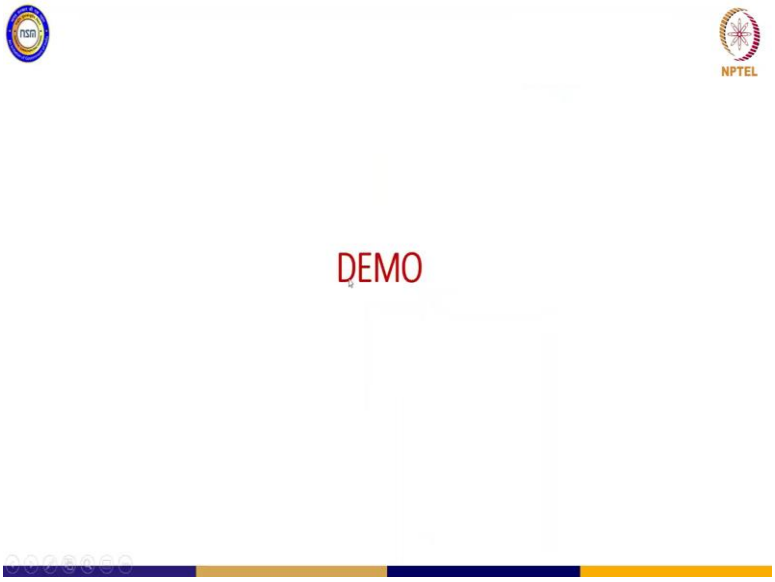
Applied Accelerated Artificial Intelligence
Dr. Tosin Adesuyi
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

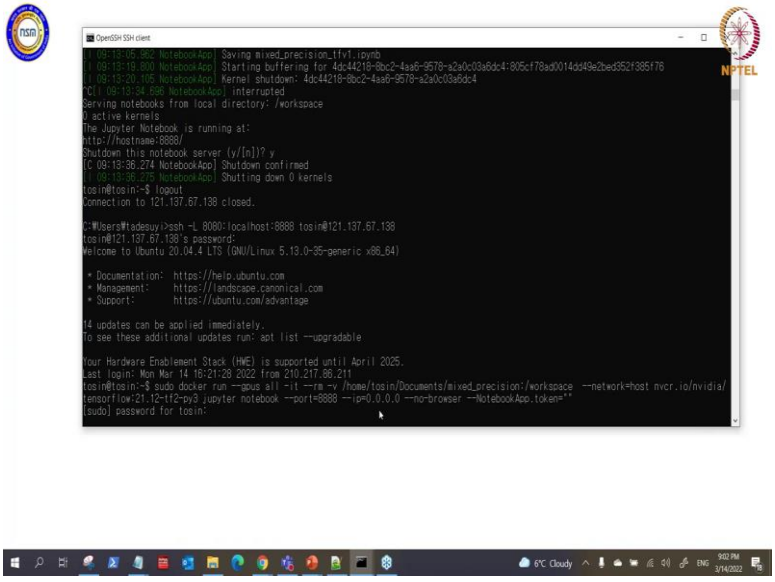
End to End Accelerated Date Learning
Lecture - 31
Optimizing Deep Learning Training: Automatic Mixed Precision Part - 2

(Refer Slide Time: 00:18)



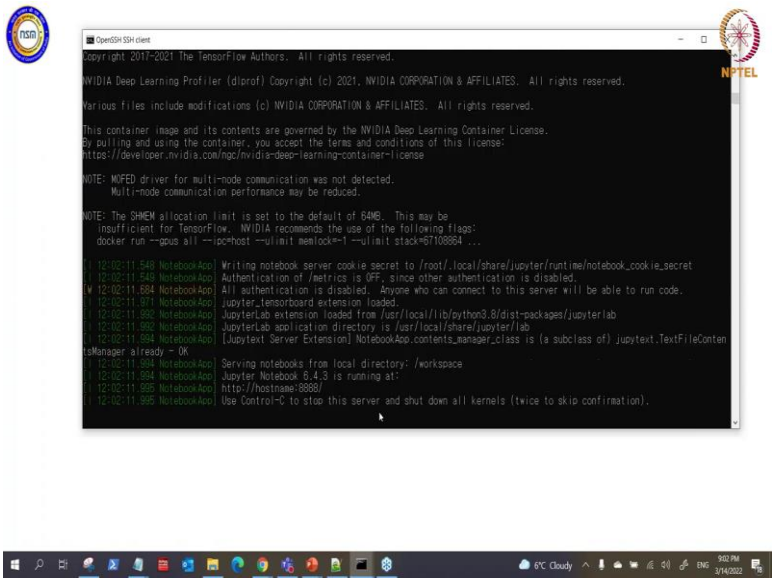
So, ok let us look at the DEMO here. So, I will go to demo here alright.

(Refer Slide Time: 00:25)



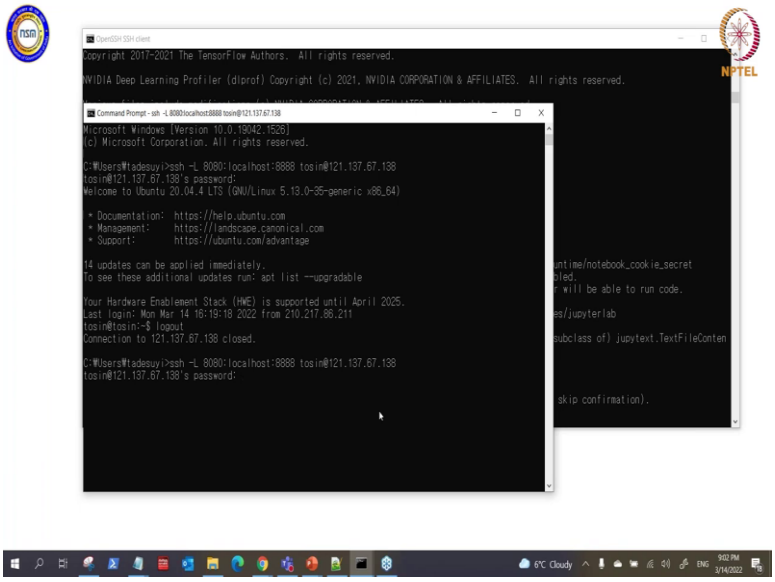
So, the first one I will be dealing with is the Mixed Precision. So, for mixed precision you have to use TensorFlow version, because I will be dealing with tensorflow first tensorflow version 2. So, I have pulled my so this is the tensorflow this is 21.12-tf2 this tf2 shows that this version 2. So, I will be running that because I am running my workstation.

(Refer Slide Time: 00:58)



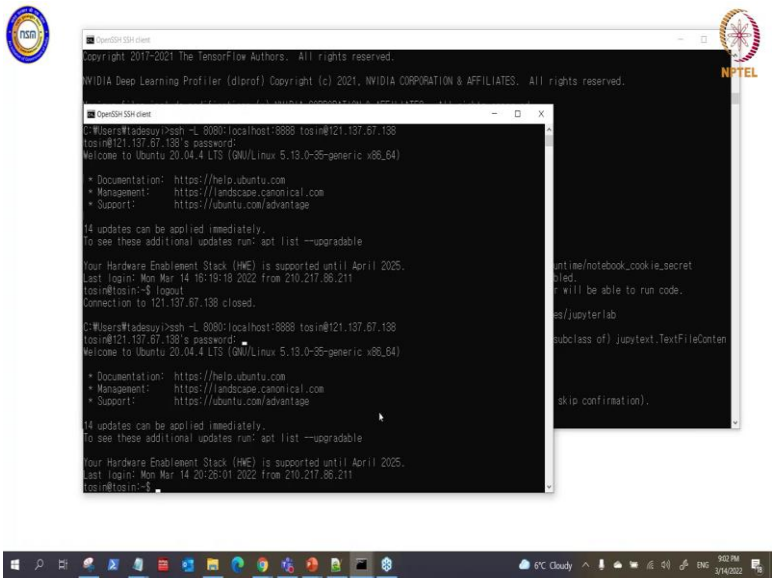
So, I am just using ok.

(Refer Slide Time: 01:09)



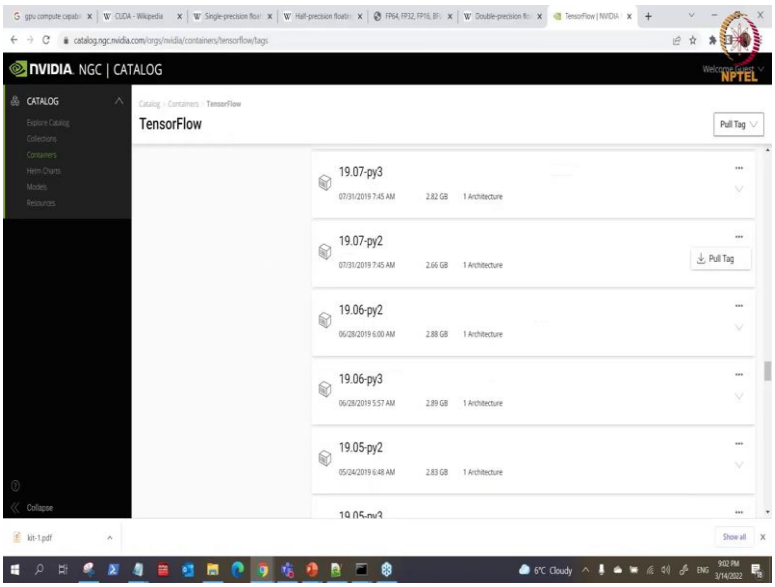
So, I am gonna connect here, you may not have to go through all this process.

(Refer Slide Time: 01:18)

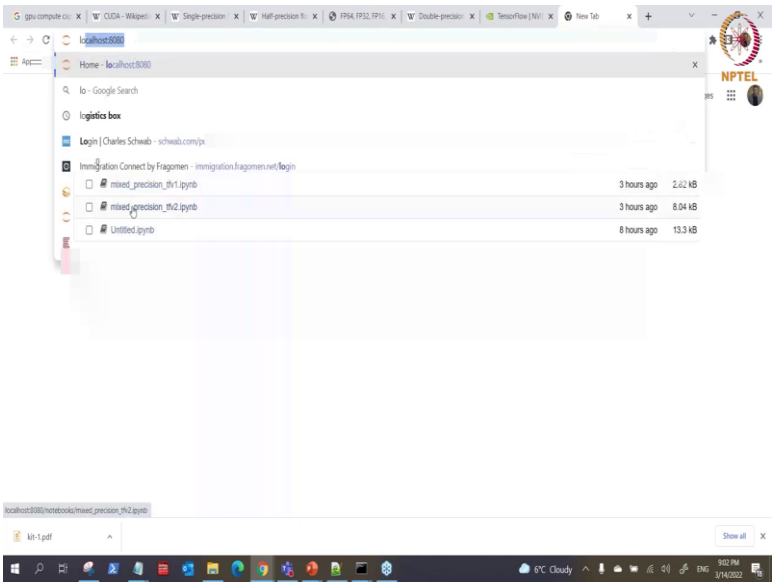


If you have your tensorflow on your workstation or on your laptop maybe via conda or virtual environment it is because I am using a container that is why I am going through this process ok.

(Refer Slide Time: 01:35)

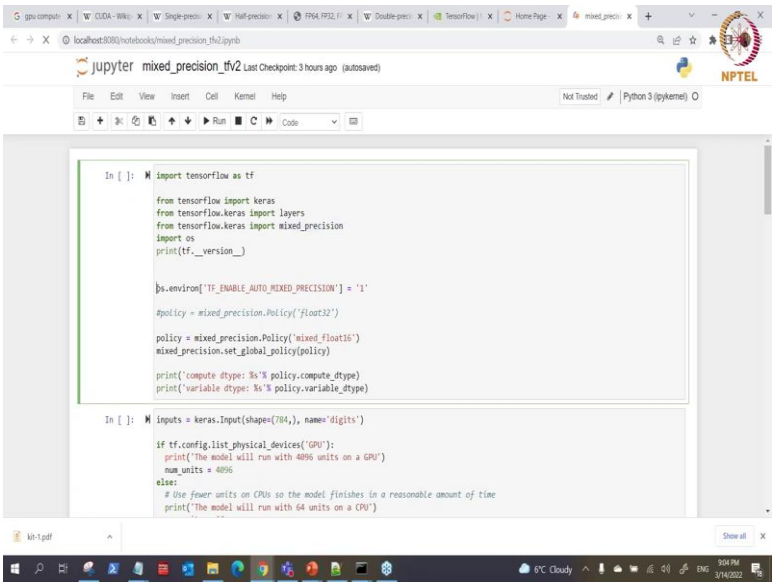


(Refer Slide Time: 01:41)



So, here I have local host ok. So, I will be dealing with version 2 first of all.

(Refer Slide Time: 01:48)



So, this is just I just put these together adapted from tensor flow website, I hope you can see my screen.

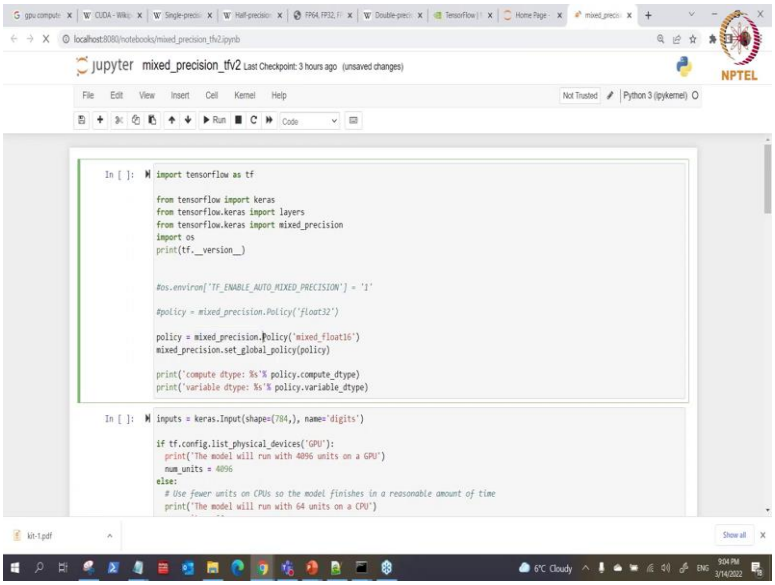
Student: Yes Tosin we can see the Jupyter Notebook.

Thank you very much. So, I just speak a simple example that so that we can actually understand very well, then on our own we can later on do complex example because of

our time. So, the first thing here is that we import our tensorflow and then we from tensorflow we get our keras and then we import the layers we wanted to use here, then from tensorflow keras we import the mixed precision.

Now, please note what we are trying to do here is actually we are using mixed precision, it is not the automatic mixed precision. Mixed precision and automatic mixed precision are the same thing. So, the difference is that automatic mixed precision you do not need to do some manual job you just with 1 line or 2 everything is sets its done for you. But with mixed precision you have to do some level of manual stuff yourself. So, we are dealing with mixed precision here because, mixed precision only is possible with tensorflow version 2. So, let me comment this one out first.

(Refer Slide Time: 03:16)

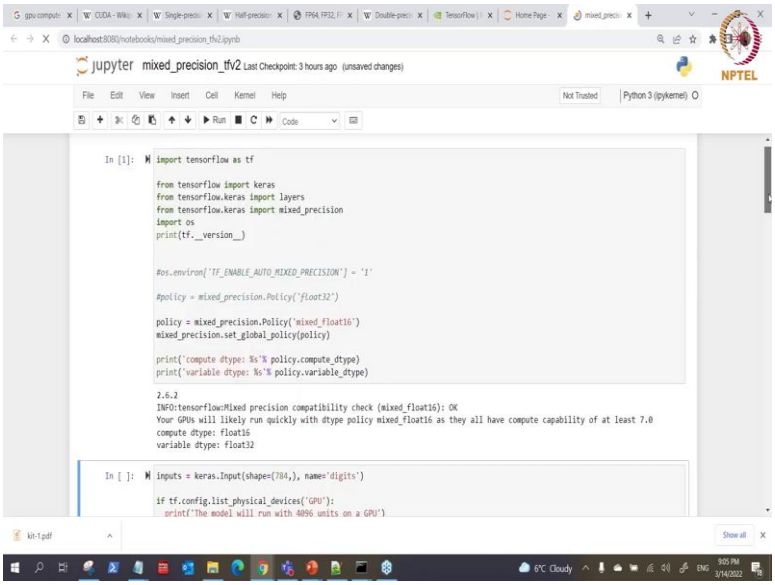


So, the first thing you do is that because we are dealing with mixed precision we need to work with a float16, fp 16. So, you have to what use your mixed precision library I have imported the class then you call this method Policy. So, we set policy for one mixed precision that is you use what mixed of what float 16. So, when you set that into the policy then you can take your word you make your mixed precision set global policy then you put the policy in there.

So, when you put the policy in there you need to confirm if it is actually sets. So, for us to confirm we need to get the word the policy.compute_dtype what is the type is going to

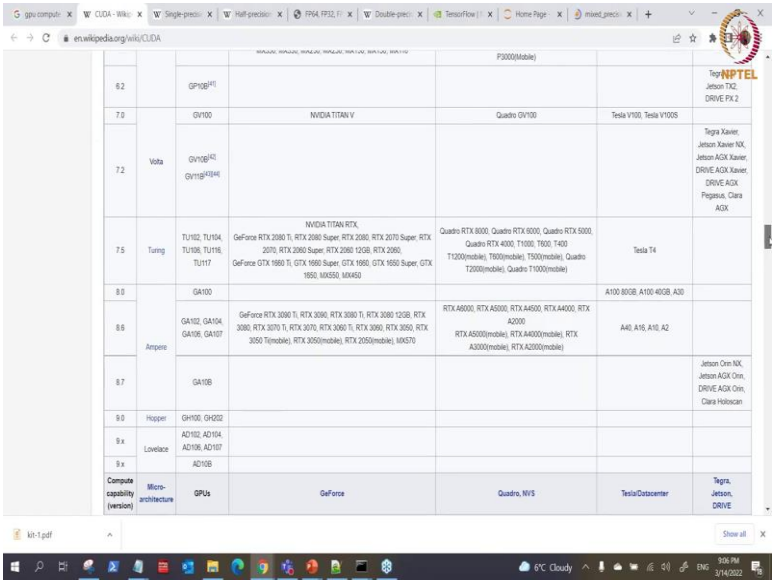
use for computation and also policy.variable_dtype that is what is the variable type. Remember mixed precision is the use of fp16 and fp32.

(Refer Slide Time: 04:18)



So, I will run this notes book here. So, let us see what happened oh good. So, it is showing us that you can see this is a version that is version is what tensorflow version 2, then we say tensorflow mixed precision compatibility check that is mixed precision that is it is ok it is compatible. We can use also your GPU will likely to run quickly with this policy of mixed float 16, also if the compatibility is at least that is your computes capacity must be at least 7.0 which I have told you before.

(Refer Slide Time: 04:54)

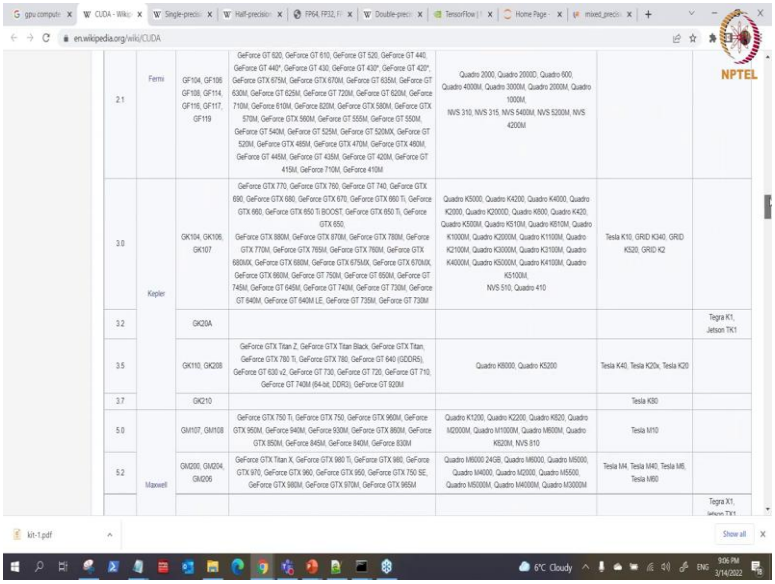


So, you have to you know if you go to Wikipedia you can see from here that from Volta.

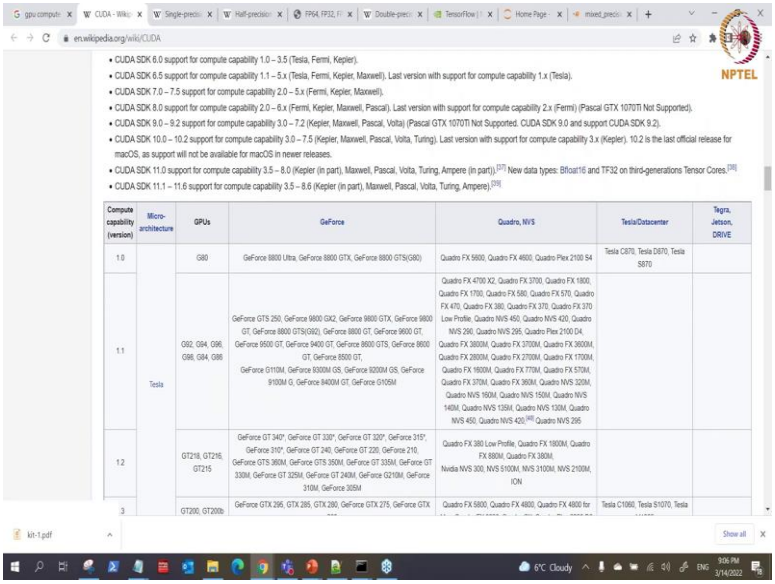
(Refer Slide Time: 04:56)

This is where we have 7.0 Pascal is lesser than the compute.

(Refer Slide Time: 05:04)

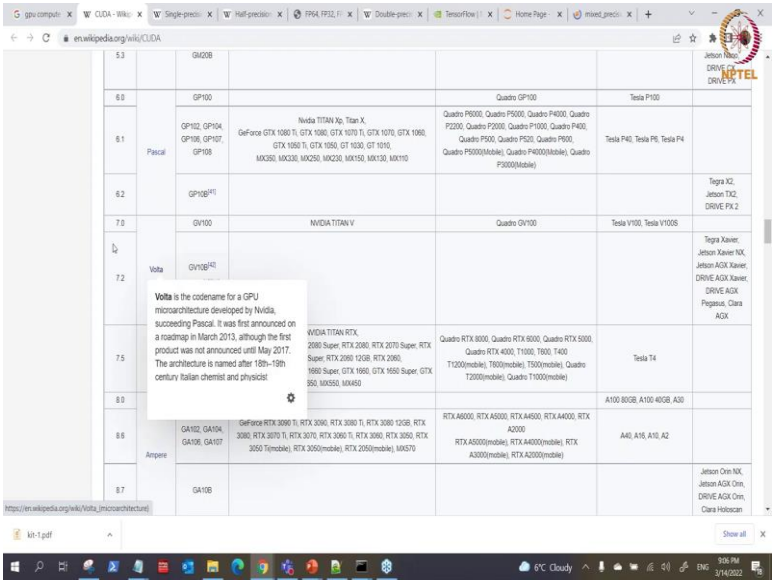


(Refer Slide Time: 05:06)



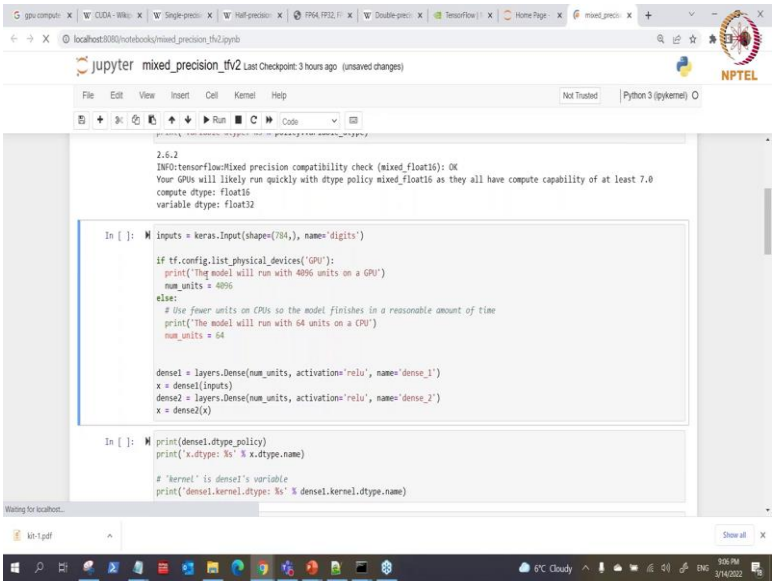
The computes are capacity capability this is compute capability here the first column. So, you can see for Pascal is less than that then from Volta you can access if your GPU is between these 7.5 this turing here. If you are using GeForce RTX 380 and all the lights.

(Refer Slide Time: 05:25)



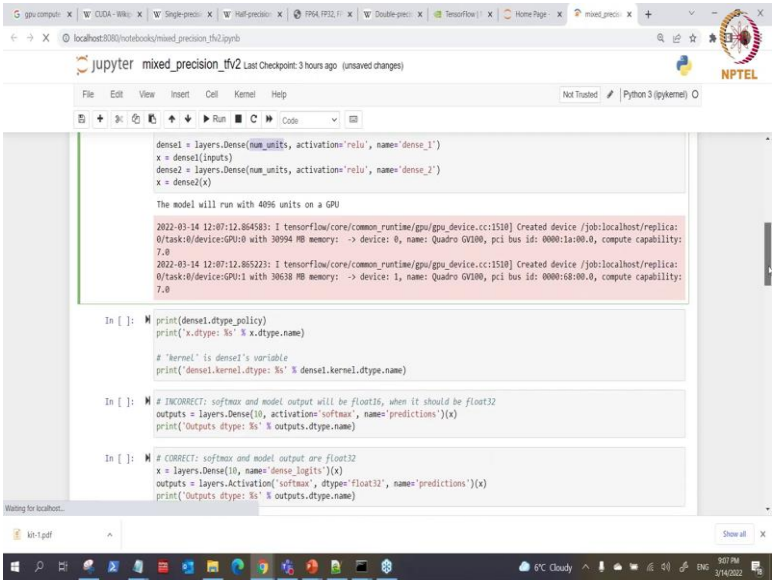
So, you can check the architecture of your GPU here. So, we can proceed from there. So, we know that so the next thing we do is that we want to work with mnist.

(Refer Slide Time: 05:35)



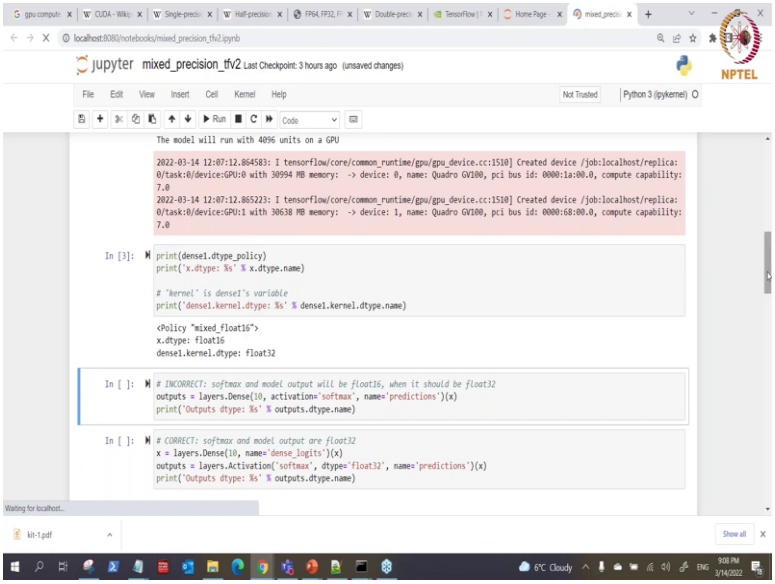
Which is like the hello world for training, which will make things simpler for us. So we create our inputs here, so the shape of the input is just 784 and then we check here if we have a GPU on this machine. So, if this GPU on this machine we want to sets our layers number of units to 4096 and if it is not we reduce it to the size this size of 64. So, these are the reflecting through dense layer here.

(Refer Slide Time: 06:07)



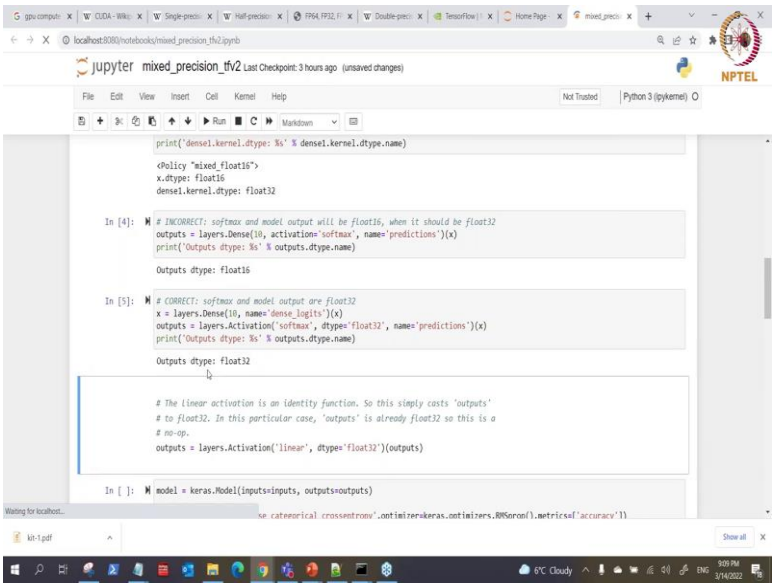
So, I can run this through dense layer and see that ok it has to detect that it has a GPU which is running and it is we are going to use this units here, this is the same as putting 4096 is the same as writing it to row directly here. And then we want to be show that ok each of the dense layer what policy are they using is the dense layer is it using the fp16 policy and is the kernel on fp32.

(Refer Slide Time: 06:37)



So, we need to check that because mixed precision is the use of fp16 and fp32. So, we will check that again. So, we can see saying yes the policies is mixed float 16 which was the type is float 16 and a dense kernel float 32. So, we are on the right track here.

(Refer Slide Time: 06:59)

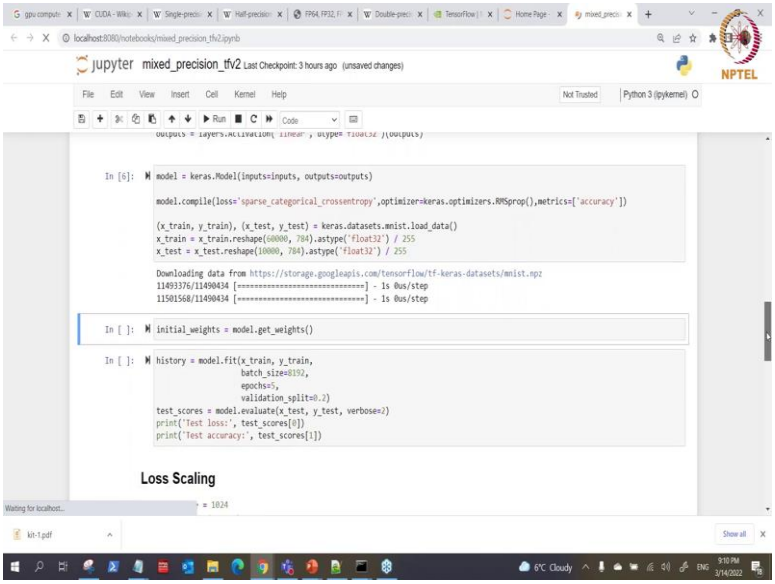


So, the next one here is what to do after the 2 layer then we get our last layer dense layer which is where we supply this softmax here. So, and we check that this is float 16. So, the rule here is that when you are dealing with what the softmax it has to be because you are about to move into doing your calculating your loss and gradient descent.

So, it has to be float 32 there in order to escape the dynamic rate limitation that fp 16 has. So, in order to do that is that we will separate what we have here we separate this layer in toward just the dense layer and then separate the activation layer from it. So, that is what we have done here, if you look at this line here we just have the dense layer here and the second one the output will be just the layer of activation function.

And how do we do that because we are using fp 16 a policy. So, we need to write this out to cast it back to fp 32 here. So, we are only casting back the softmax layer only. So now we are good to go. So, this is just the exception that we have there.

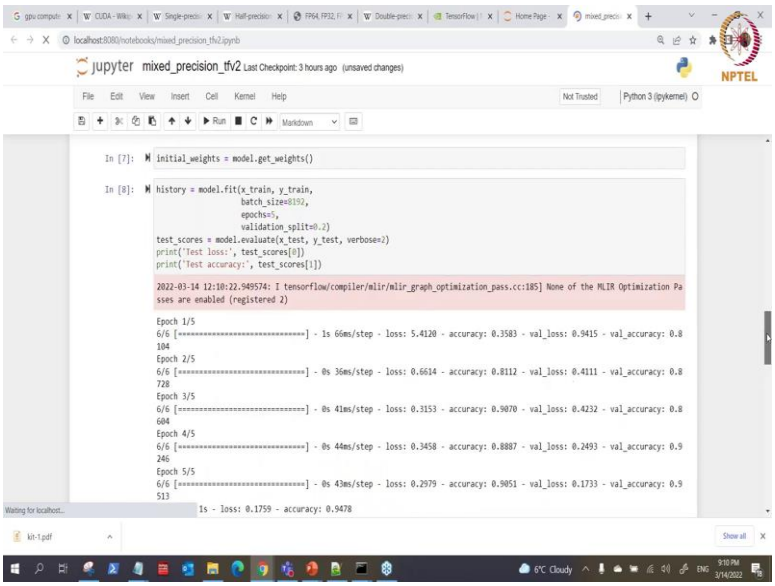
(Refer Slide Time: 08:17)



And then so you can also do that using this line of course, you have a linear here it can also cast it back with that. So, what we need to do next is what to instantiate our model we have the input into the model, we have the output into the model and then we can compile our model by setting the loss and also the optimizer set here and our metrics is what for calculate the accuracy.

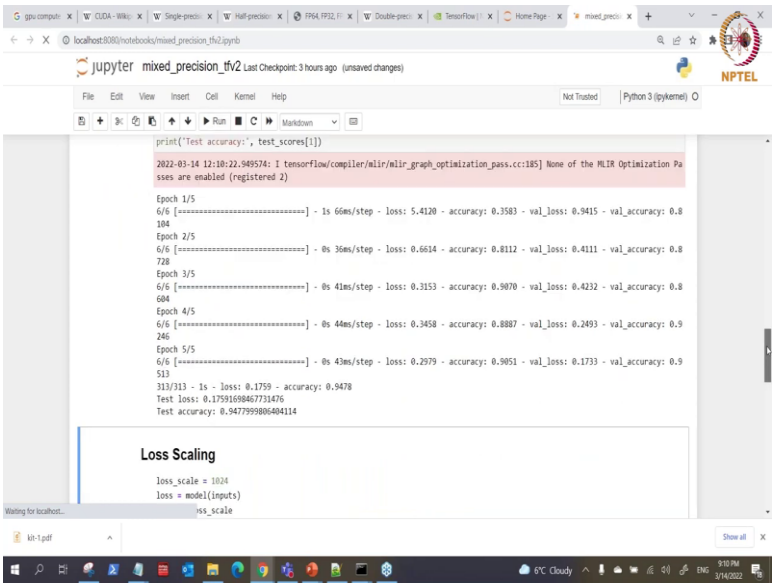
Then we get here what we do here is to get our data you know the mnist data, you get your mnist data the training will be 60000 and the test will be 10000 as well. So, it is downloading we download that, so after downloading that so the next thing to do is to what to initialize our weights.

(Refer Slide Time: 09:10)



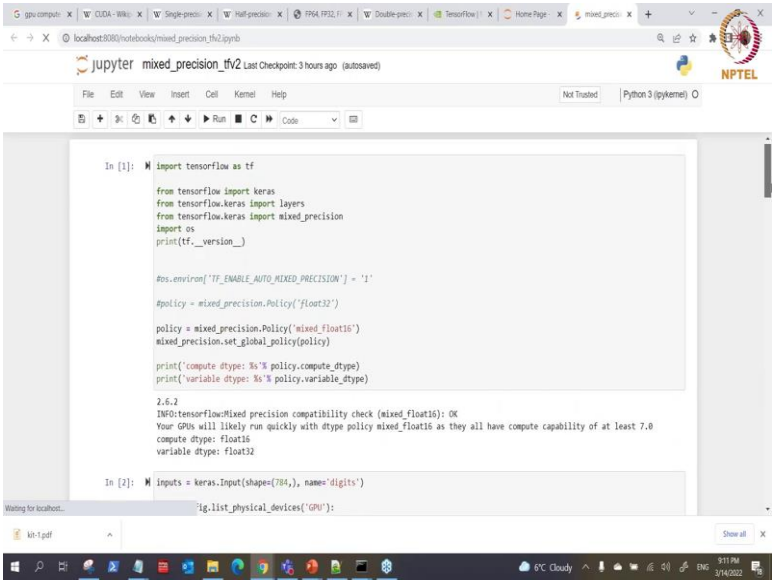
So, we initialize the weights here that we will initialize the weights, then we have to work do what and train the model here. So, we can see the model here. So, what I want you to notice here is that this is just put it on the 5 epoch that now we times.

(Refer Slide Time: 09:27)

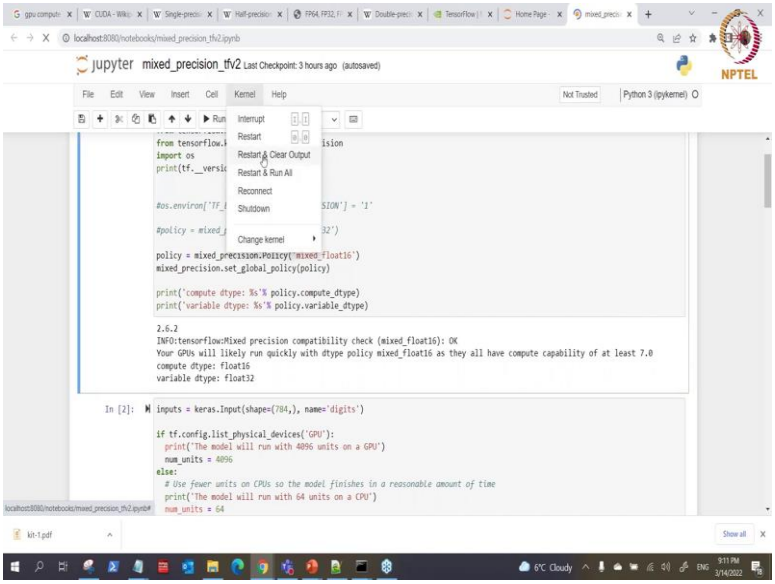


If you can see the number the time it takes for the for each steps, you can see the first one was the one the first step is where it takes longer which is 1 seconds 66 the others are just 0 seconds 0 seconds 0 second 1 second is that it is doing the casting here, this is the first epoch. So, at the subsequent epoch it becomes more faster there.

(Refer Slide Time: 09:53).

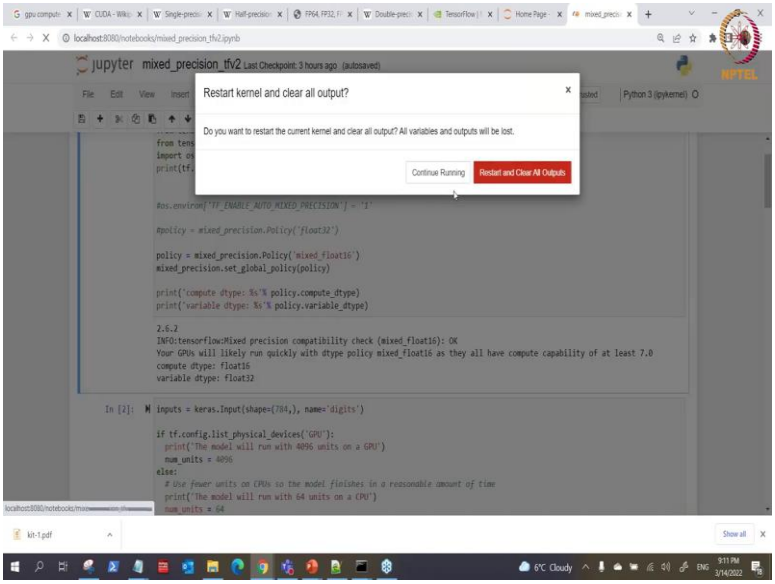


(Refer Slide Time: 09:54)



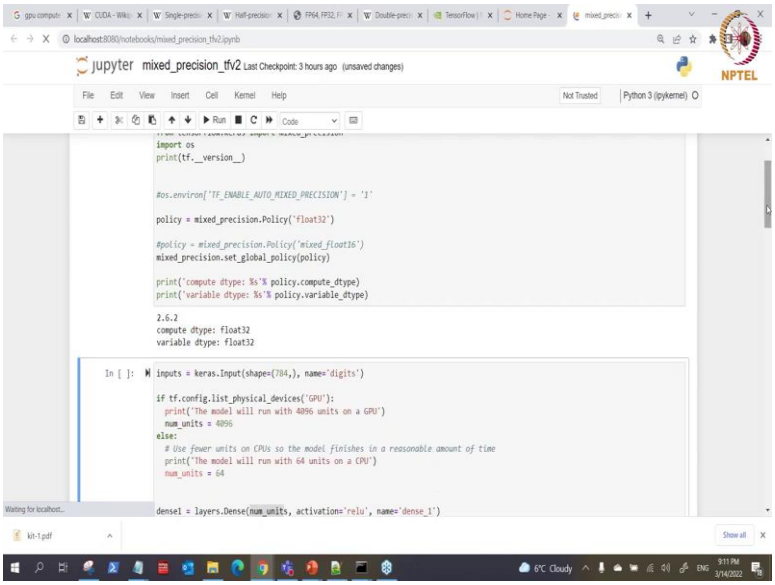
So, let us see if this were to be trained without using mixed precision. So, if we are not using mixed precision. So, what do we do? If we change from mixed precision to maybe we want to use fp 32.

(Refer Slide Time: 10:09)



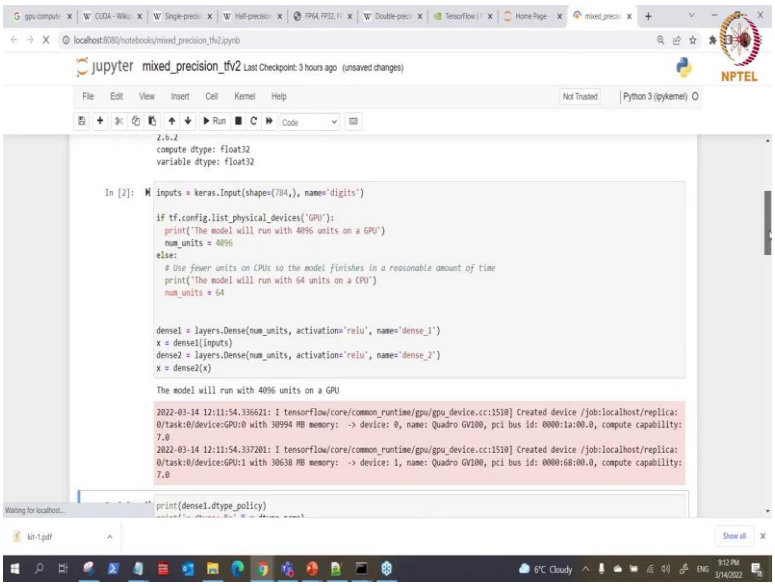
So, I can just come here and say like let me the connect first clear the output just to show us the difference because this is very important we have limited time ok.

(Refer Slide Time: 10:13)

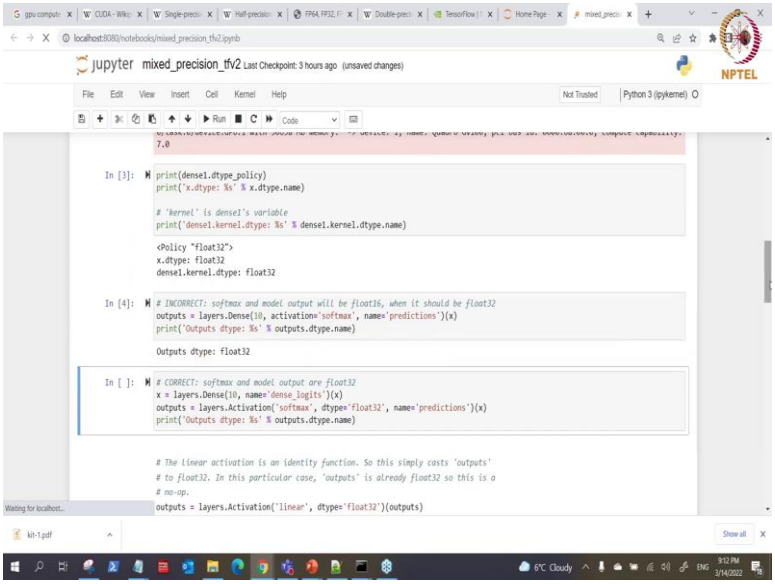


I have cleared that, so what I would do is what this mixed precision policy? I erase it then I just turn it back to floating point 32. So, this is without using mixed precision here without using mixed precision. So, if you compute here, so what we will see is that you see the compute type is no longer fp 16 it is fp 32 the variable type is fp 32. So, let me quickly run all of these.

(Refer Slide Time: 10:49)

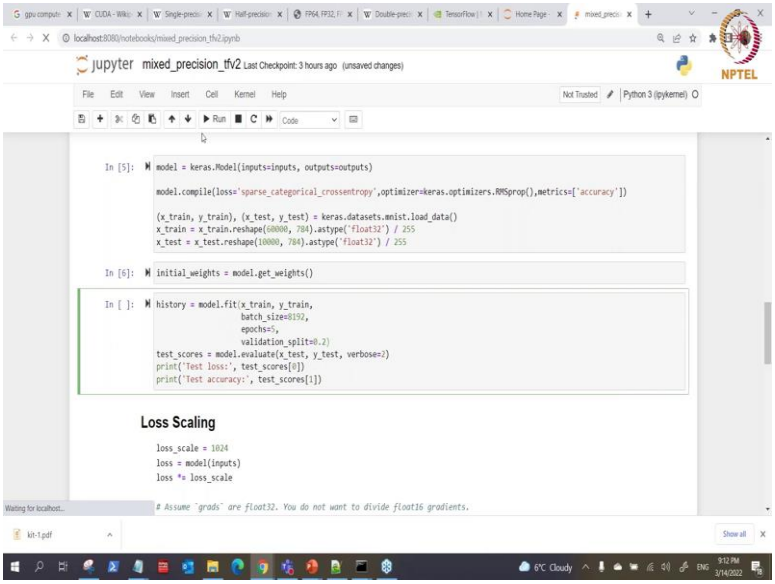


(Refer Slide Time: 10:51)



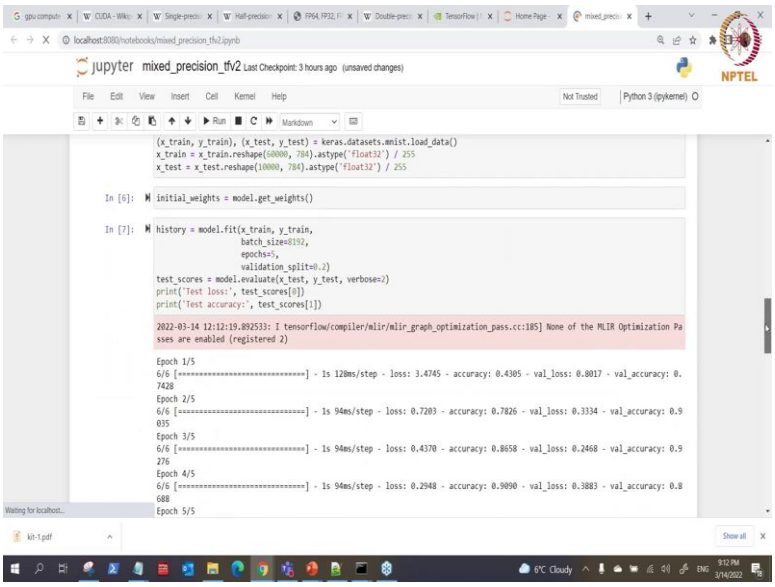
All of this which I have run before I am only interested in the results, so everything is fp 32.

(Refer Slide Time: 11:00)



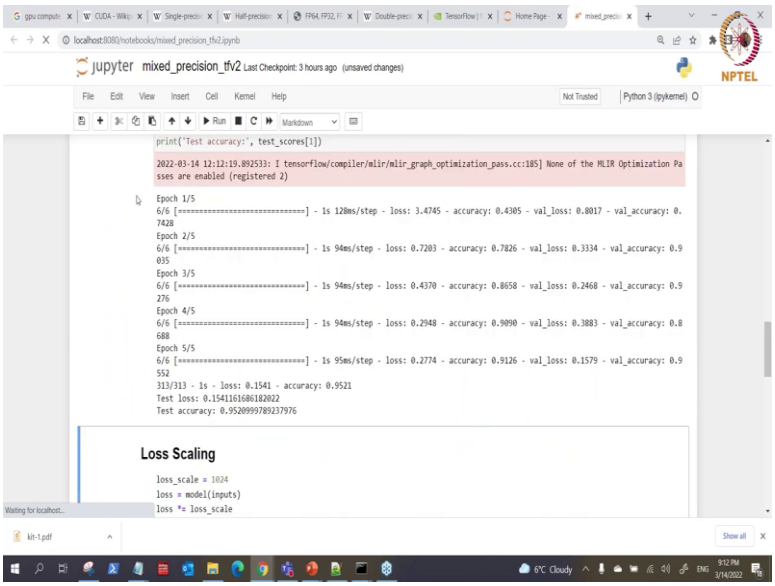
Since its fp 32 I do not need to run that one again to convert.

(Refer Slide Time: 11:14)



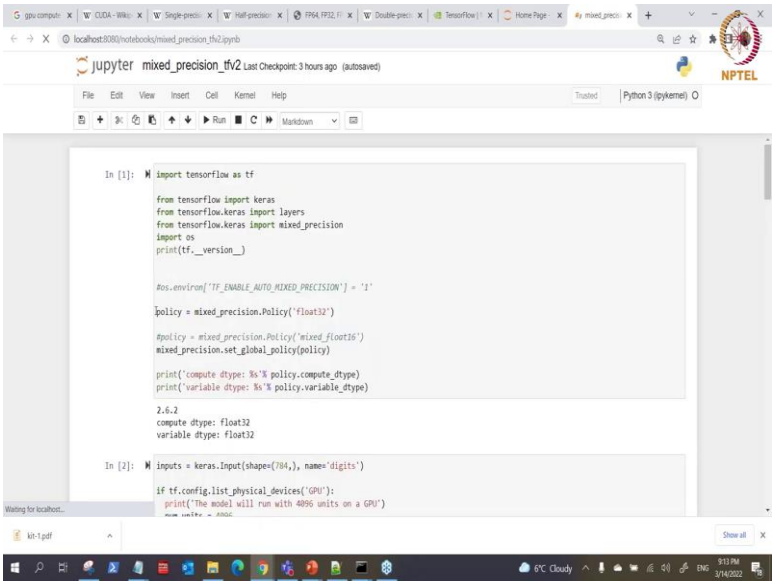
So, let us run this. So now, what I want us to see here is that.

(Refer Slide Time: 11:20)



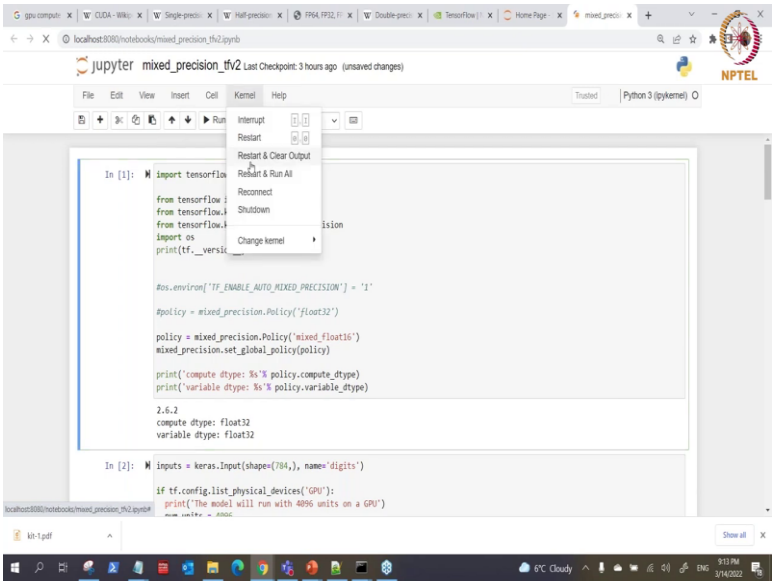
If you look at this we have at the initial is 1 seconds subsequent epoch is also 1 seconds 1 seconds 94 the 1 seconds 94 you can see that the time it takes when it is not mixed precision is higher than when it is mixed precision. It might looks small is because we are dealing with smaller data and smaller model size here that is why. So, it is just to show you the differences that are there, when it was mixed precision fp 16 you can see only have 1 here and all other subsequent epoch I just took only took 0 seconds.

(Refer Slide Time: 11:57)



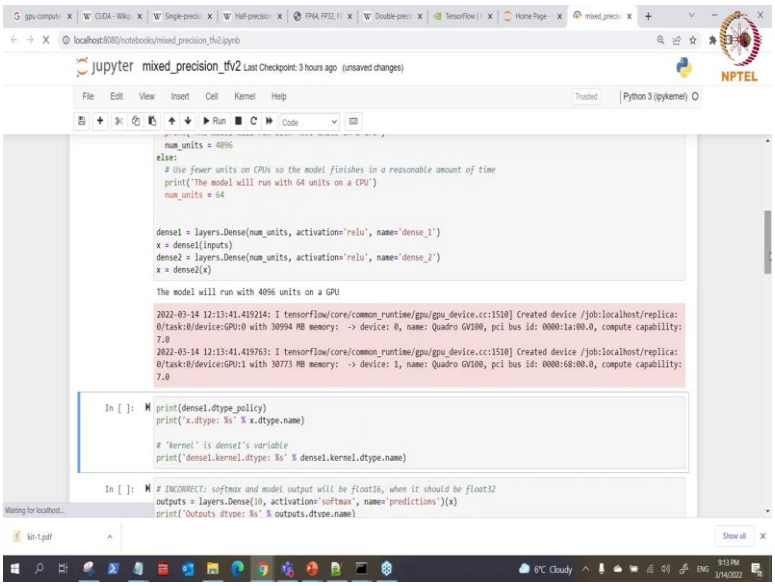
So, I will change it back because I need to show us in case you are now you are the way you write your code you are using custom loop within your code.

(Refer Slide Time: 12:04)



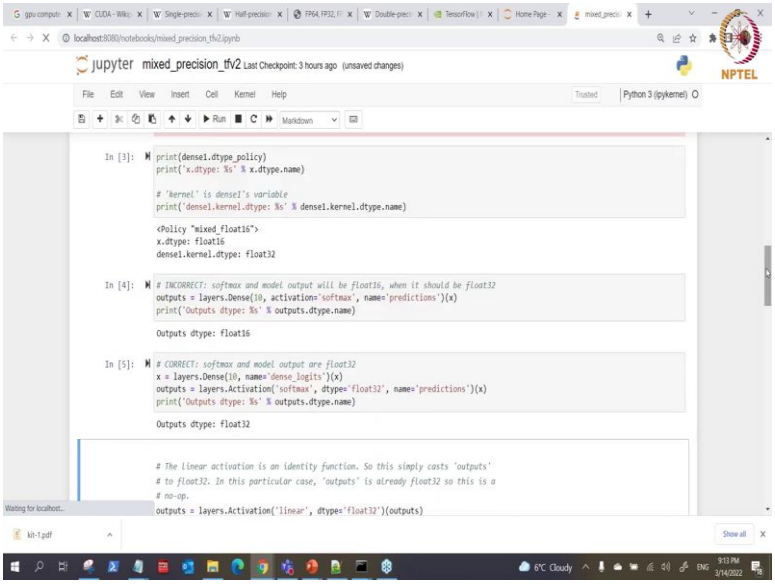
So, if you use custom loop within your code, then how do you use mixed precision when you use custom loop? So, to use when you use custom loop, then I will show you that 1 in a GV sorry I do not want to mess things up that is why I keep now. So, we have set back we set back to our mixed precision here.

(Refer Slide Time: 12:34)

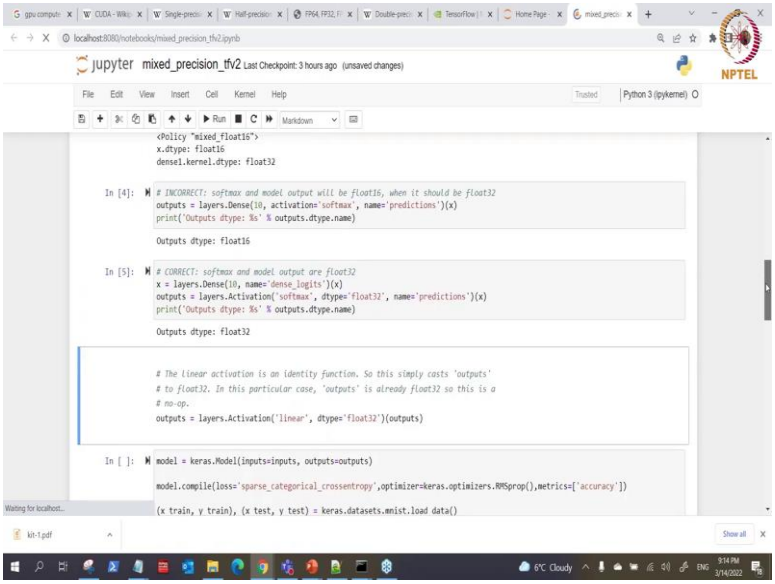


So, I can run this now back run this.

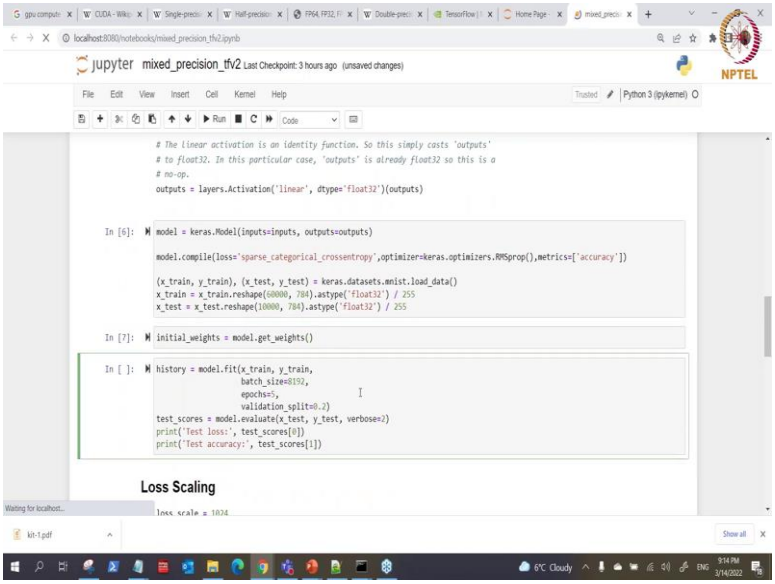
(Refer Slide Time: 12:41)



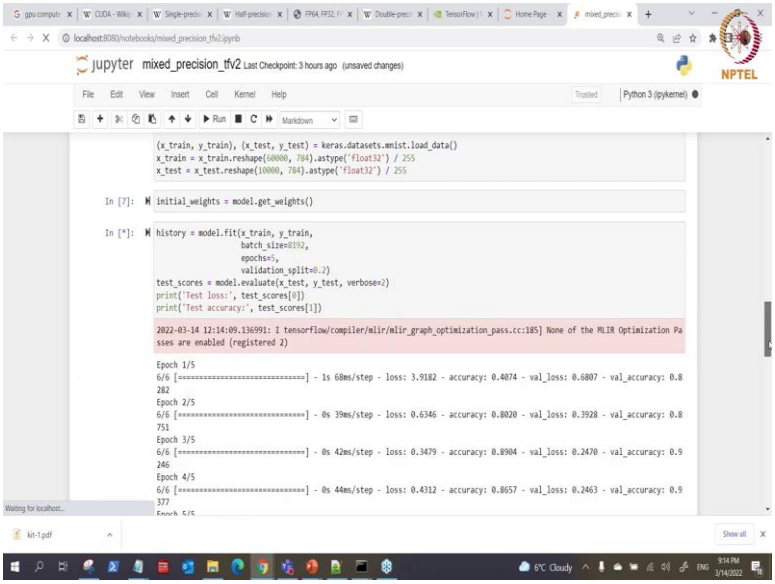
(Refer Slide Time: 12:49)



(Refer Slide Time: 12:52)

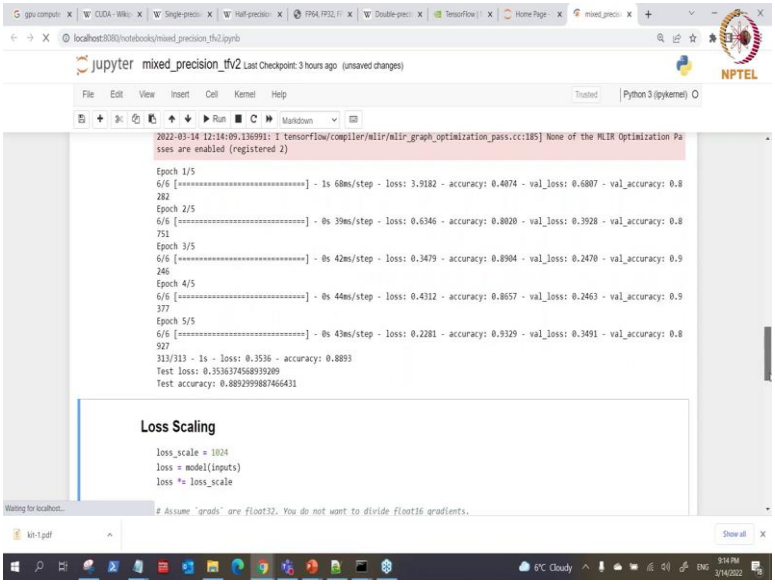


(Refer Slide Time: 13:03)



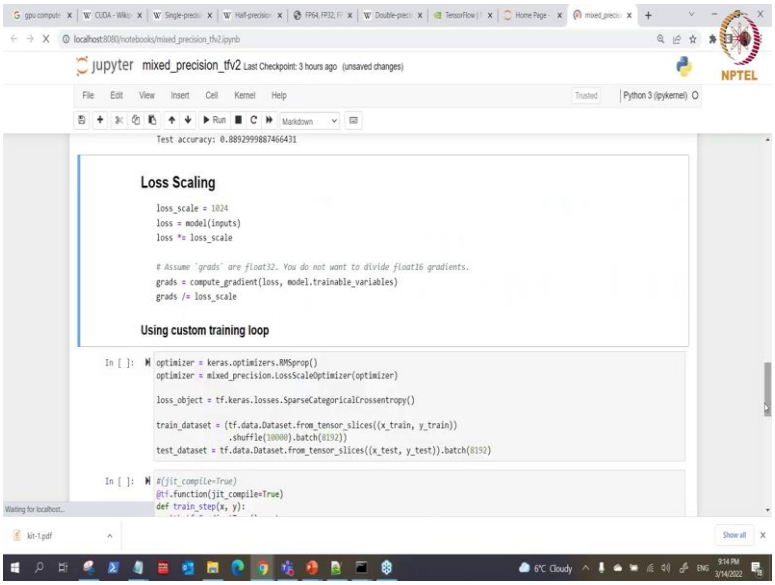
Run that is then?

(Refer Slide Time: 13:10)



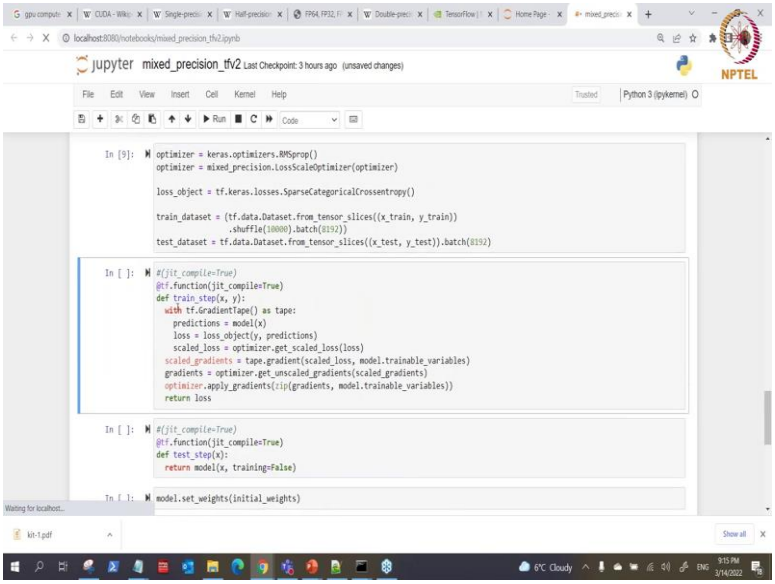
So, look at it when we are back it is only the first step that we have 1 seconds and the others just 0 0 0 there.

(Refer Slide Time: 13:17)



So, when now we want to do with what loss scaling here. So, loss scaling you need to compute the loss scaling when you are using custom training loops, here we do not need to what compute that. But when we are using custom loop it will be done automatically by itself.

(Refer Slide Time: 13:34)



But in this case when you use custom loop here you need to specify a mixed precision loss scaling optimize this is the scaling we are talking about. So, when you try to use custom loop you must specify the scaling optimizer. So, you get your optimizer here this is our optimizer here is the optimizer root mean square props. So, we use the mixed precision loss scale optimizer to wrap it, after we wrap it through that we get our loss objects here using the cross entropy then we here we get our data again here.

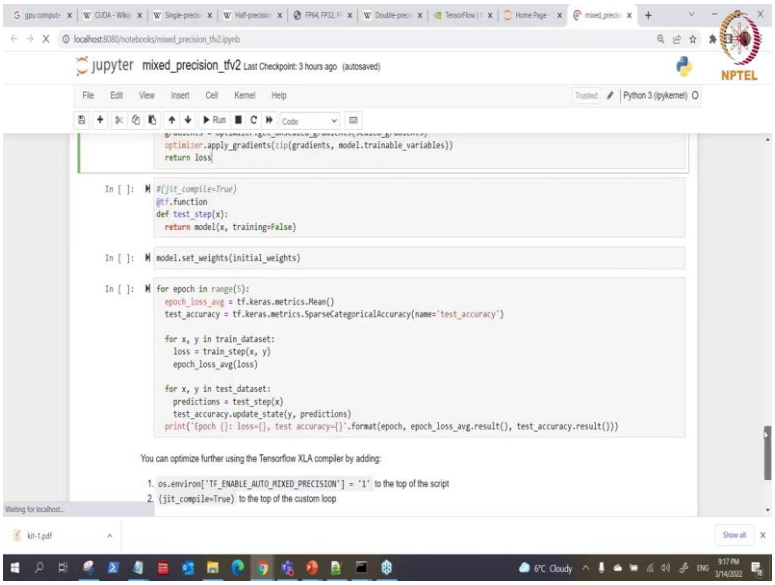
So, let me run this get our data then. So, we want to compute the training step. So, in this training step here we get our gradient tape, this is the gradient tape under the gradient tape. So, what happened get your prediction this is the model supply the input x into it, this is a function we are just writing this function graph from then we compute our loss using this loss object you know we have calculated the loss object here.

So, we call it takes this value, then we scale our loss with what we have been talking about before. So, this optimizer from the mixed precision loss scale optimizer we use it to scale. So, we get the scale loss here, after we get the scale loss, then we also get the what the scale gradient as well the scale gradients using tape loss scale gradient add your scale loss and the model variable add trainable into it.

Then after we have gone that is our scale gradient then we will get our re gradients. So, what happened after your scale your gradient then you what on scale back? So, our optimizer dot on scale the gradient back. So, we will scale the gradient back then we can

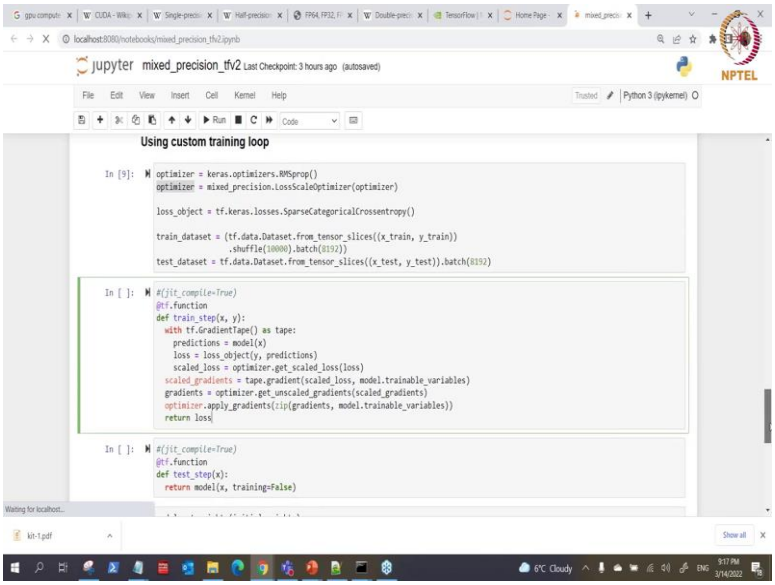
now use our optimizer to apply to our normal trainable gradient. So, we return the loss here.

(Refer Slide Time: 15:31)



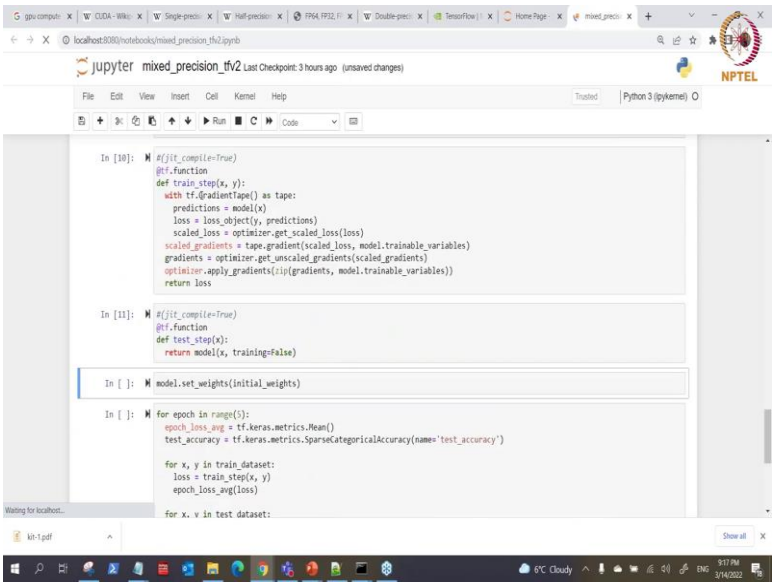
So, as we have done that we need to do also for testing, but our testing we do not need to go through this. So, we just write the model testing this way that you see sorry about this for now I do not include this here and here also I do not want to include this here.

(Refer Slide Time: 15:50)



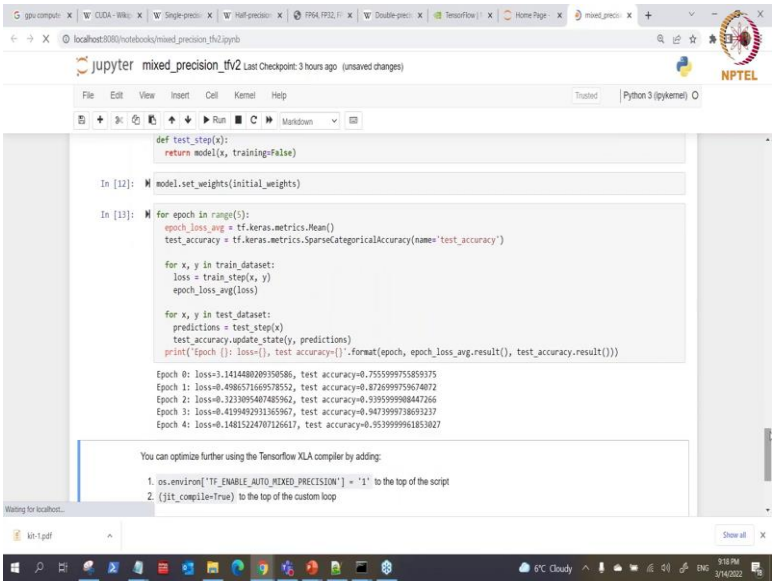
So, this @tf function is showing this is a graph formats of writing your model. So, then ok let me run this as well if I run the first one here.

(Refer Slide Time: 16:19)



So, we run that. So, run this cell which I have explained then the one also that compute for our test step. So, remember this is for training step and this is for test step.

(Refer Slide Time: 16:26)

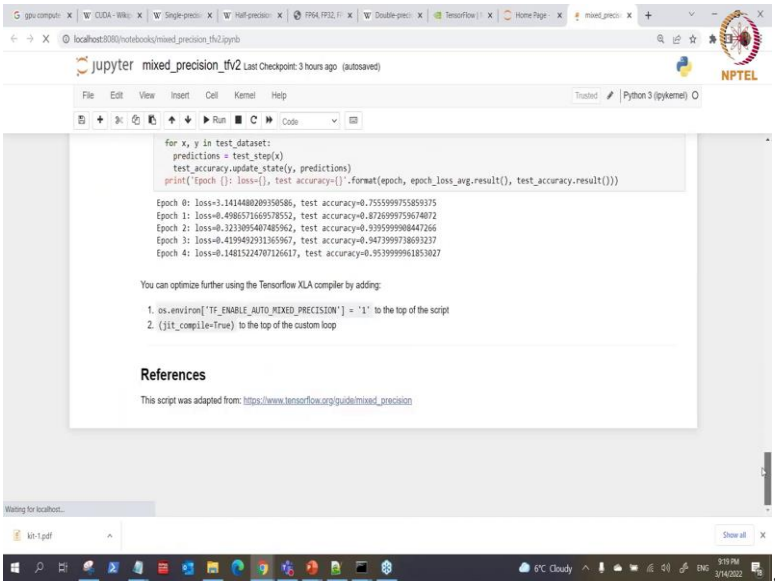


Then the weight we set our weight to initialize our weight again, then this is the customized loop now. So, we are inside the loop. So, inside the loop we calculate our epoch loss average here using the metric.Mean() then our test accuracy is set here. So, in each value inside our training data set we train with these steps get the loss and the loss average loss is calculated here.

Also inside our test set so we get the training test step here training step and then we our test accuracy will update the states and then we can compute here we can print all the results here. So, when you do that so you can see also it is a very fast order we did not calculate time here, but if you test this with bigger model you will be able to see and that it run faster than when you are not using mixed precision.

So, I have just showed you 2 for when you are using custom loop training loop and when you do not use custom training loop. So, with custom training loop you know you have to set your mixed precision loss scaling all of that then you have to scale you have to unscale back or scale gradients and all that which all the theoretical which we have explained before.

(Refer Slide Time: 17:54)

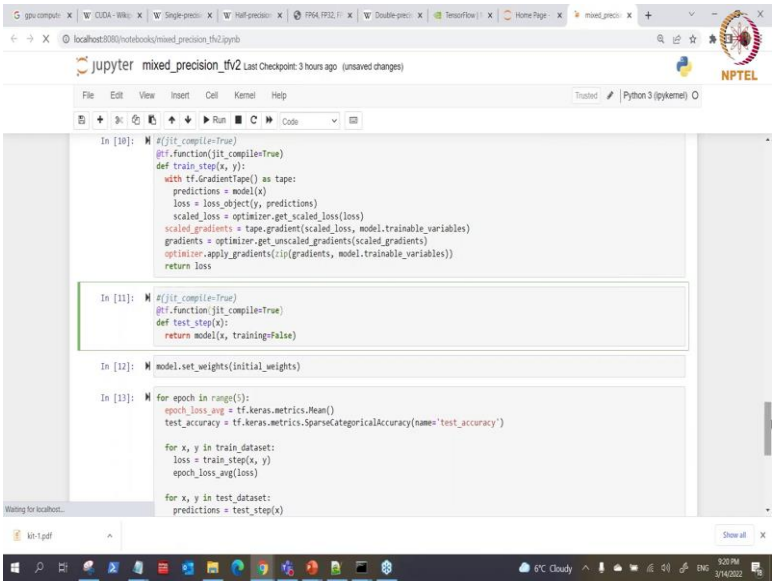


So, here one thing you can do again is to optimize for that is what we call the XLA compiler. So, you can do use the XLA compiler by what adding this to you add this to the environment variable then you enable a jit compiler on each of the top of custom loop that you have. That is you know our custom loop exist between on the each of this function.

So, what we need to do here is our first thing is to have that is let us come here then I can enable the mixed precision here. Remember we do not get confused this the enablement we are doing here is for the XLA. So, we enable it like this then let me run this as I have enabled that. So, we are still on the float 16 then I can come here then add the jit to it as

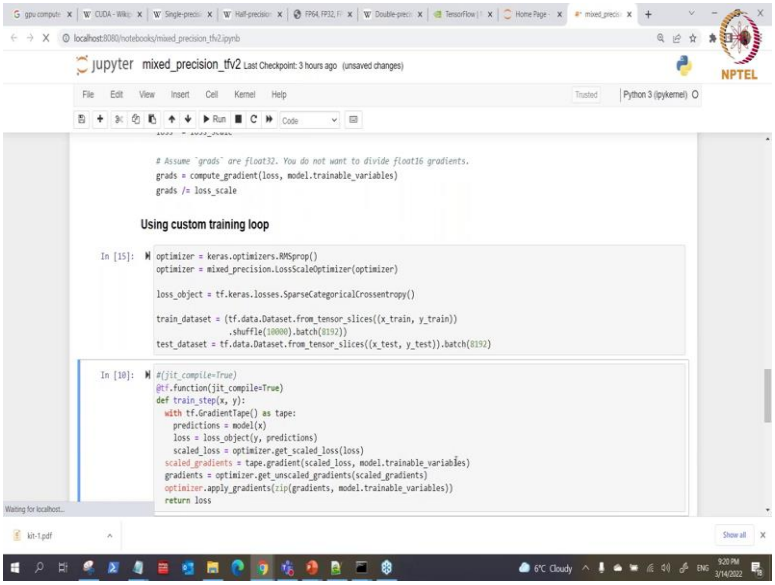
well, just to show you that you can have it in front of the function here just in time also you can add that here just in time and add that here.

(Refer Slide Time: 19:10)



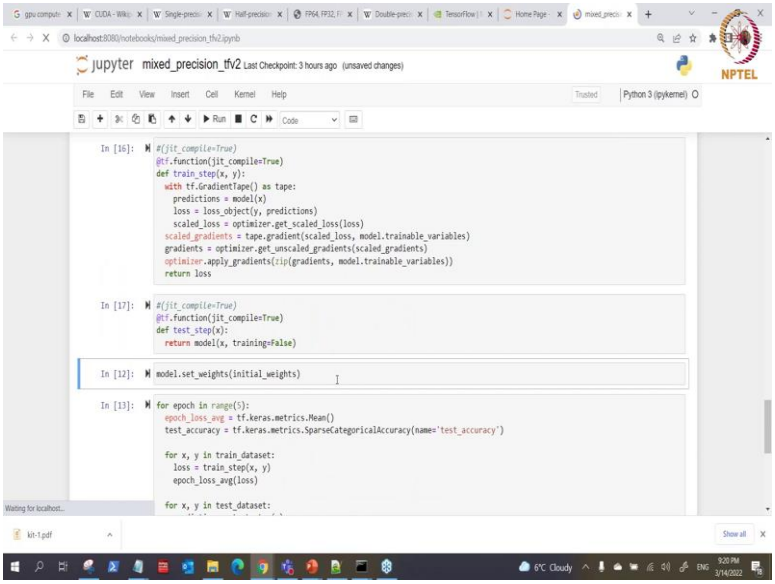
So, then I can run this over again and run this over again.

(Refer Slide Time: 19:18)

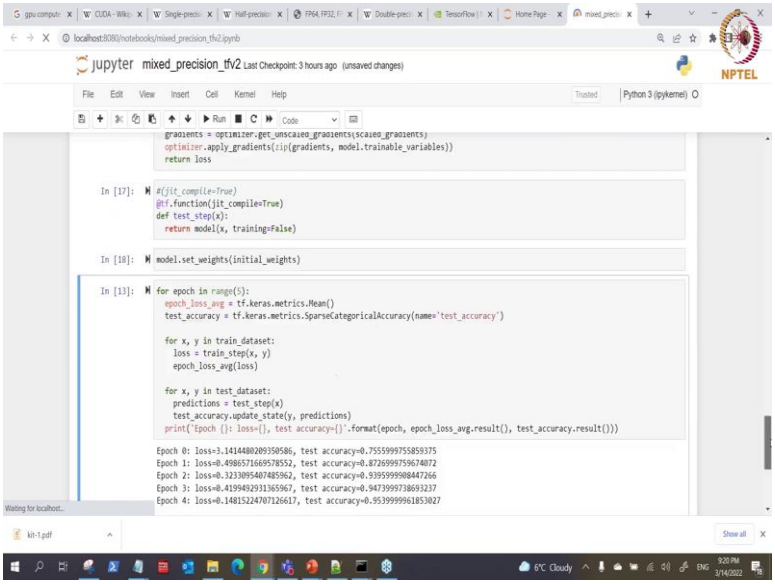


Then run this also over and run this as well.

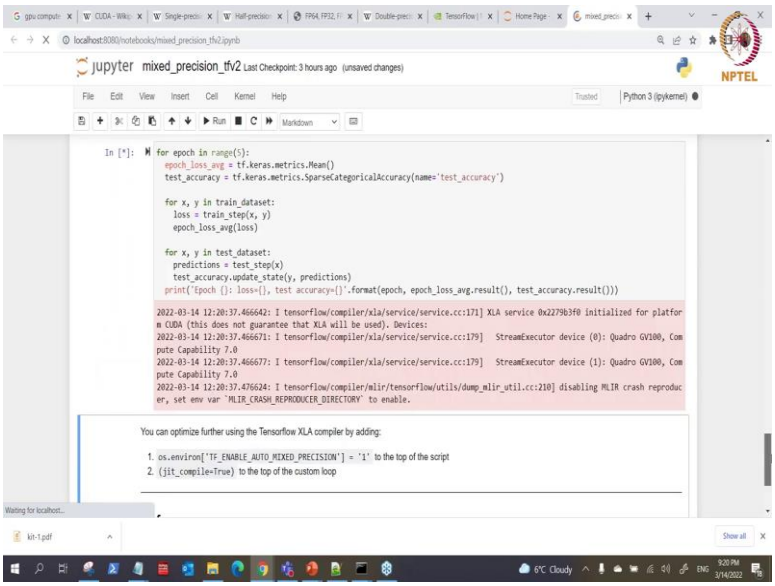
(Refer Slide Time: 19:20)



(Refer Slide Time: 19:27)

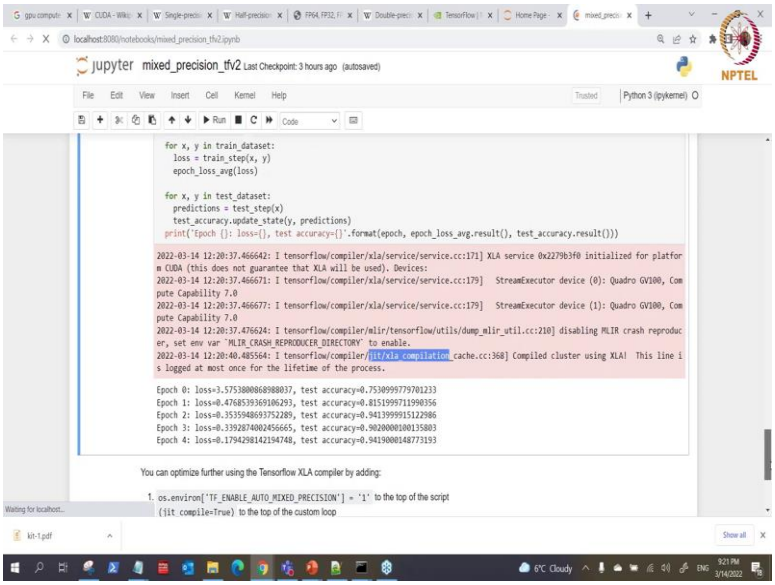


(Refer Slide Time: 19:31)



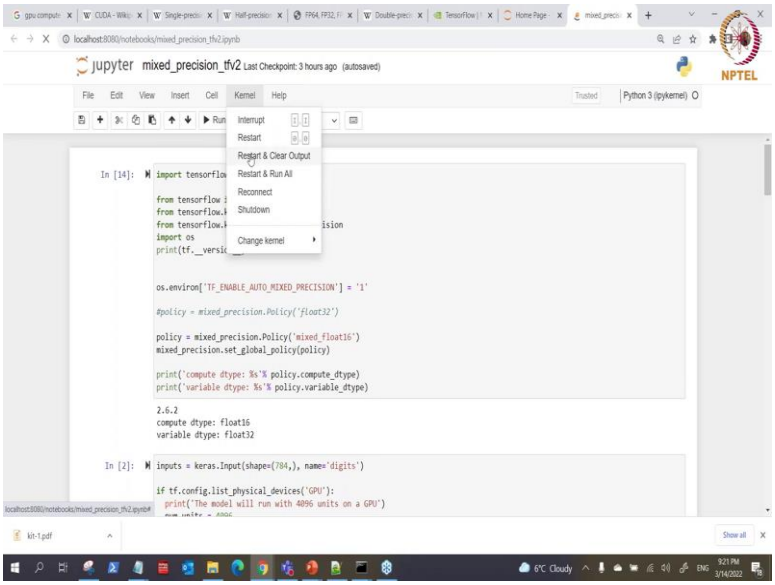
Run this as well then here. So, we can also run this as well. So, it will make it faster you can see if you can see here.

(Refer Slide Time: 19:35)

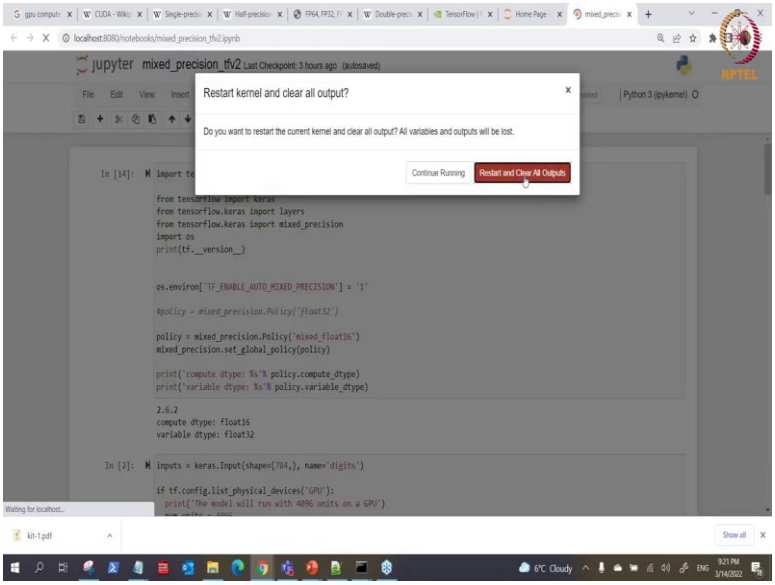


These are enabling the what the jit XLA compilation this is tensorflow special tensorflow compiler which will run with what using the jit there. So, you can also use this and if you use it that is this is combination of mixed precision and the XLA. So, which give more optimization.

(Refer Slide Time: 20:01)

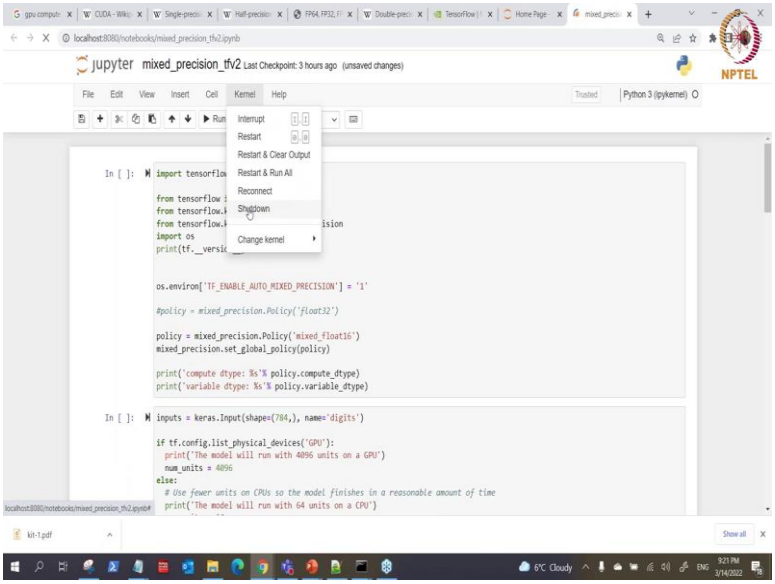


(Refer Slide Time: 20:07)



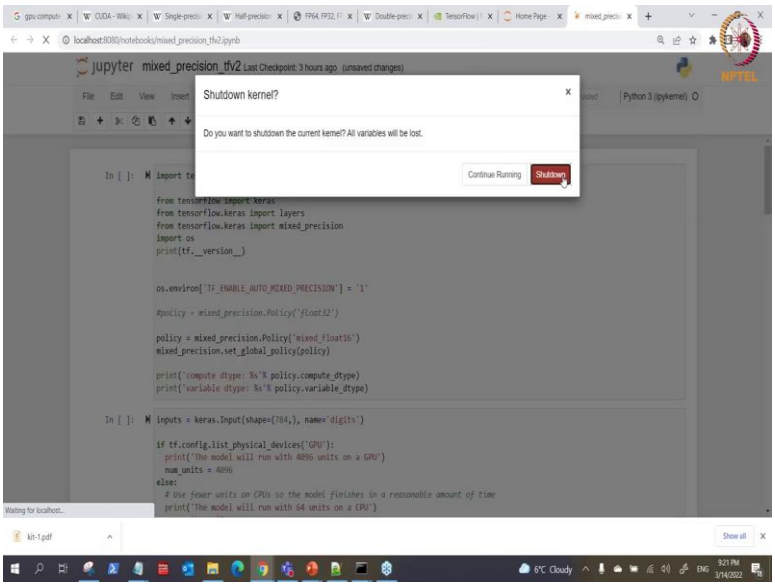
So, this is it on mixed precision then.

(Refer Slide Time: 20:12)



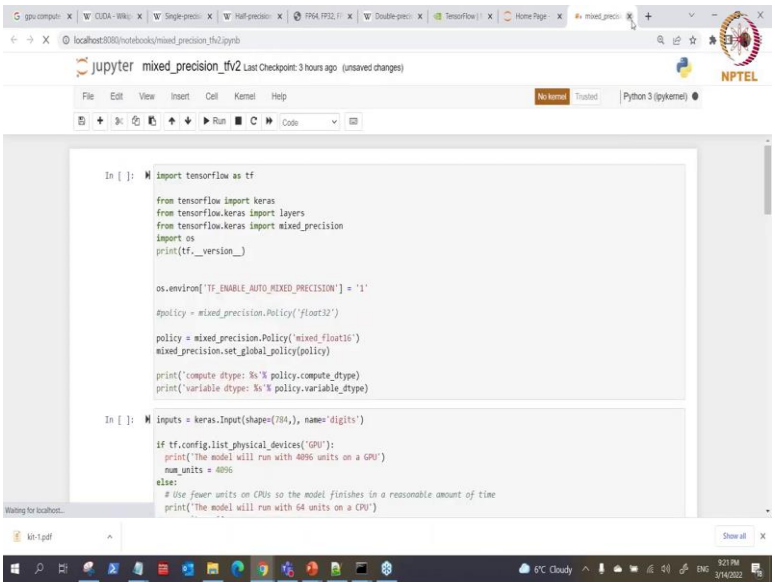
So, I am about to round up just 2 more slides.

(Refer Slide Time: 20:19)



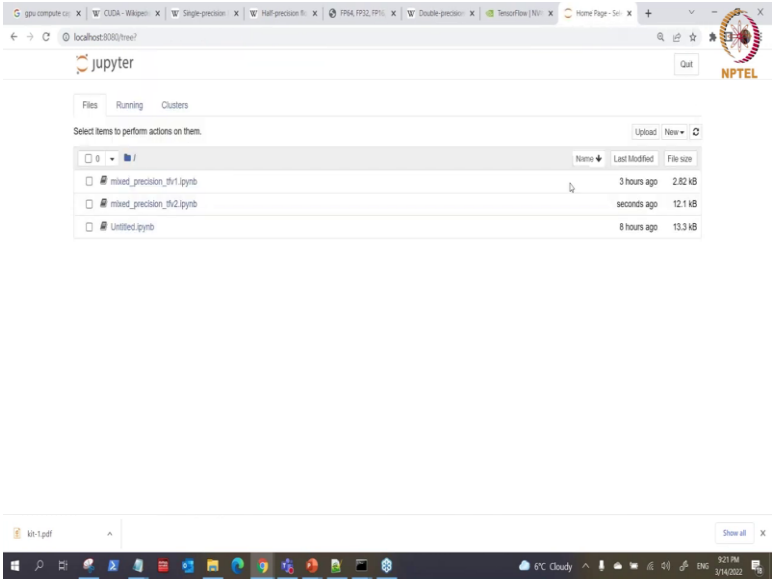
Let me shut down this here.

(Refer Slide Time: 20:20)

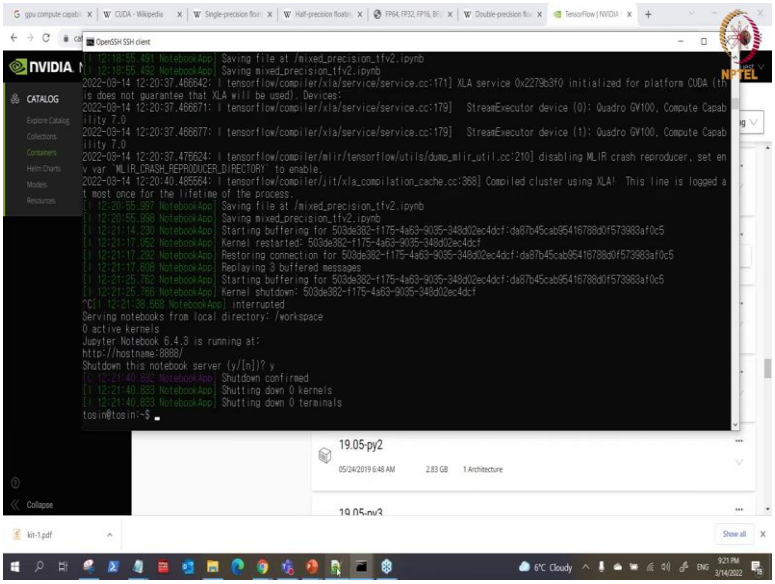


Then you can leave also I leave here.

(Refer Slide Time: 20:24)

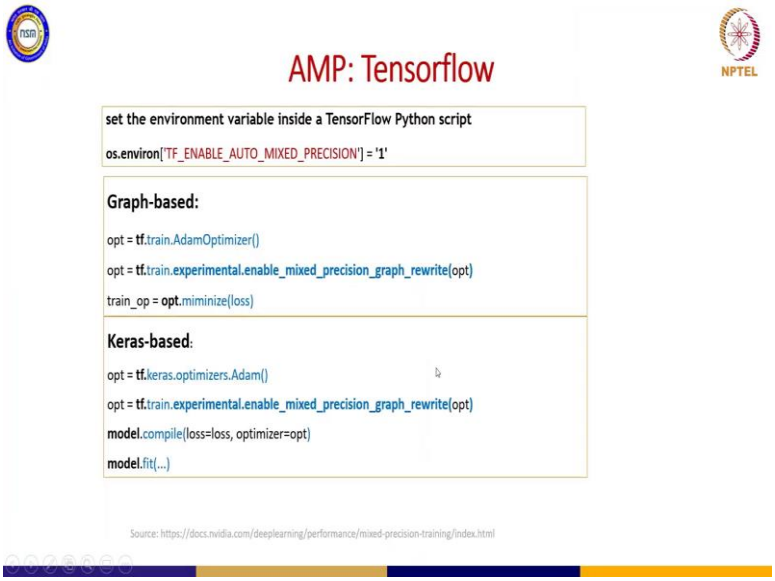


(Refer Slide Time: 20:32)



So.

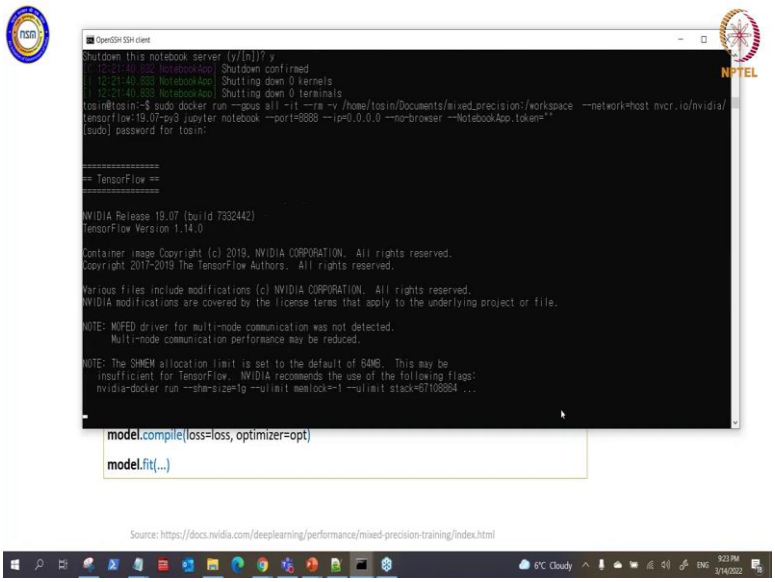
(Refer Slide Time: 20:43)



So, back from the demo, so there is another better way to do what we have done or so in that other way we which to use now automatic mixed precision with our tensorflow is that you set this environment although. We have seen how we can set this environment. So, if you are working with graph base what you need to include is just this enable_ mixed_ precision_ graph_ rewrite you use that to wrap your optimizer and it is done.

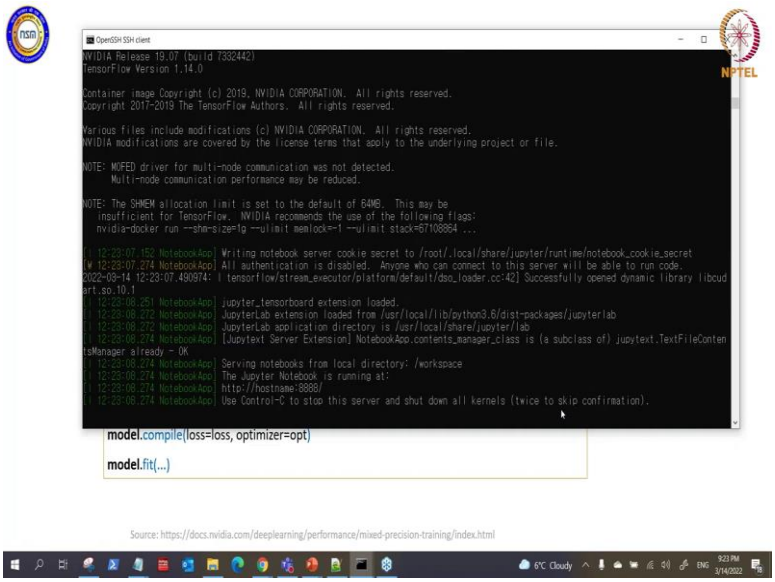
So, you do not need to on scale on the scale you do not need to do that, that is how automatic mixed precision is very easy to use. And if you are using keras based as well you can just do that as well enable_ mixed_ precision_ graph_ rewrite to wrap your optimizer and everything is done. It does the scaling and on scaling itself within it does that within.

(Refer Slide Time: 21:43)



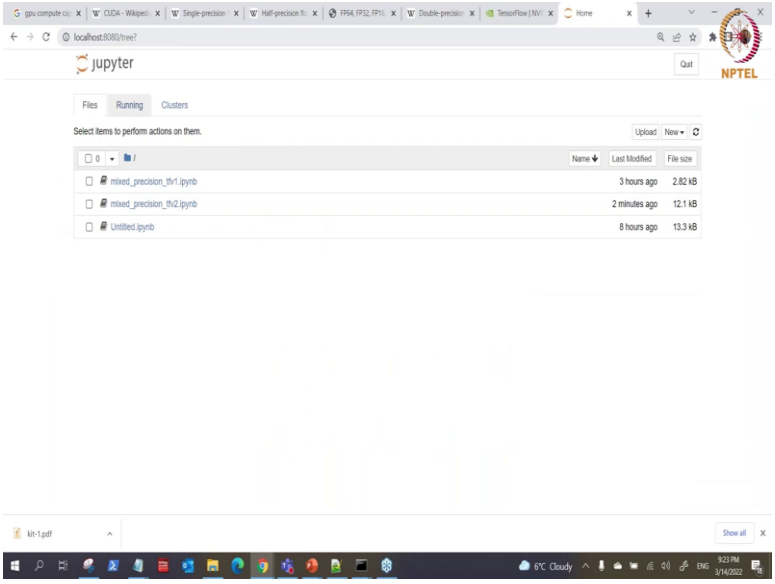
So, to show you that I can just because we can only do that with tensorflow version 1, if you can see here this is version 1 here container. So, I will run this again. So, we would not waste time on this ok.

(Refer Slide Time: 22:00)



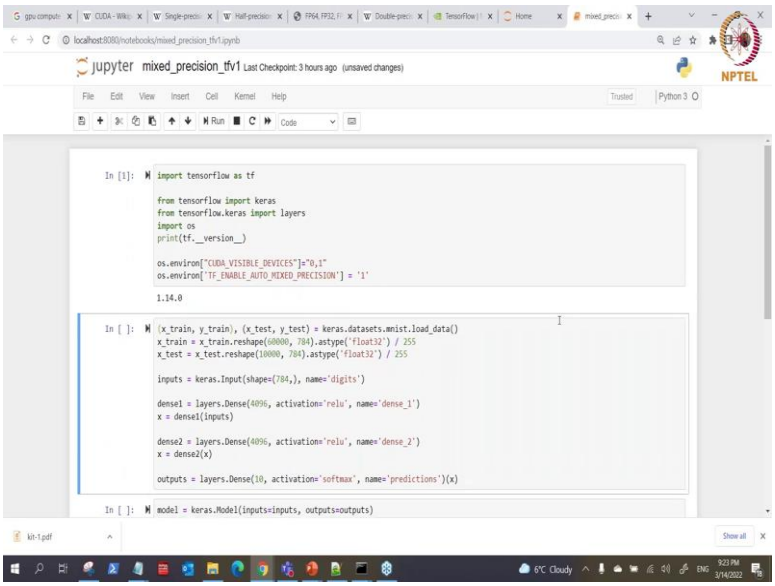
I think I can come here then let me have that is ok.

(Refer Slide Time: 22:08)



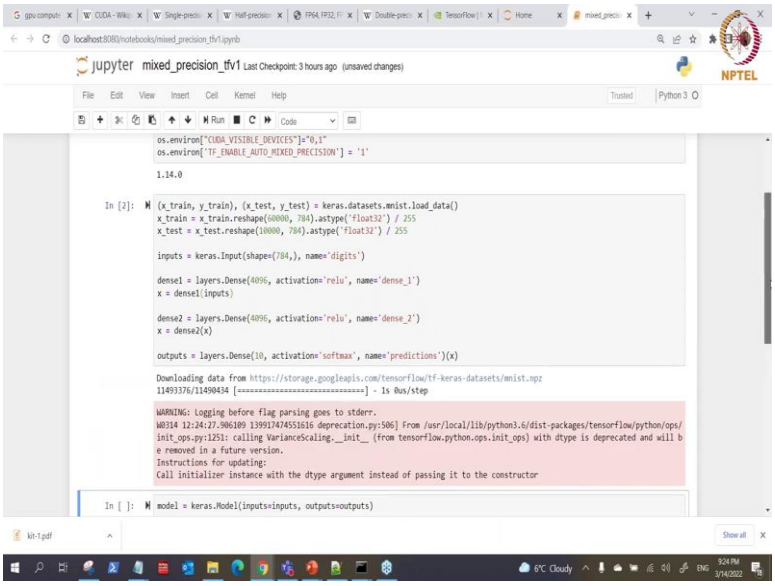
Now, I will run this is our version 1 here.

(Refer Slide Time: 22:18)



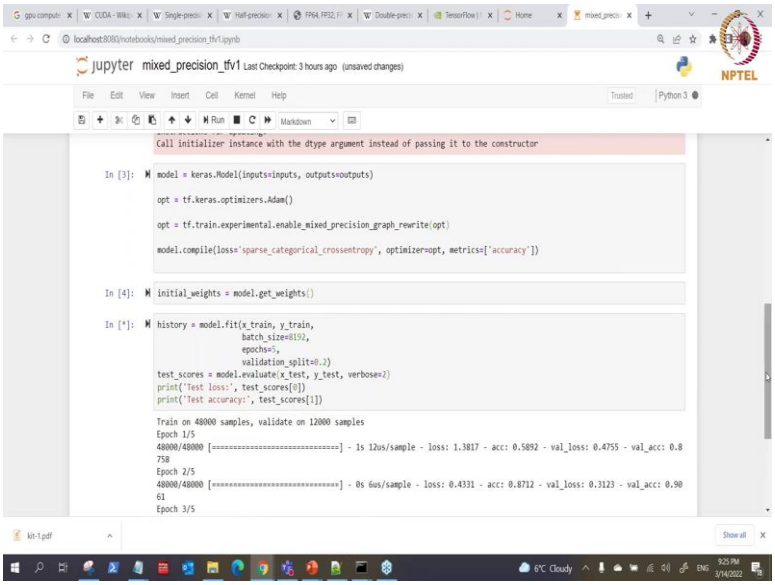
So, in the version 1 this is what with import tensorflow we have import our keras, our layer, OS do you see we do not import mixed precision here. So, what you can do here is that here I have set my environment for cuda device because I want to use GPU, then I set the environment here. So, after setting the environment here let me just run this that is done. So, you can see the version of the tensorflow is 1.14 yeah for automatic mixed precision.

(Refer Slide Time: 22:49)



So, here we are getting our value here loading the data from mnist, then the training the train and the test data then the input size which is the shape of the input is 784. Then we have our 2 dense layer and the output layer. So, here we care less whether our activation for sure it is on float 32 or is on float 16 we do not need to bother about that. So, I run that so downloading putting our download the data that we need.

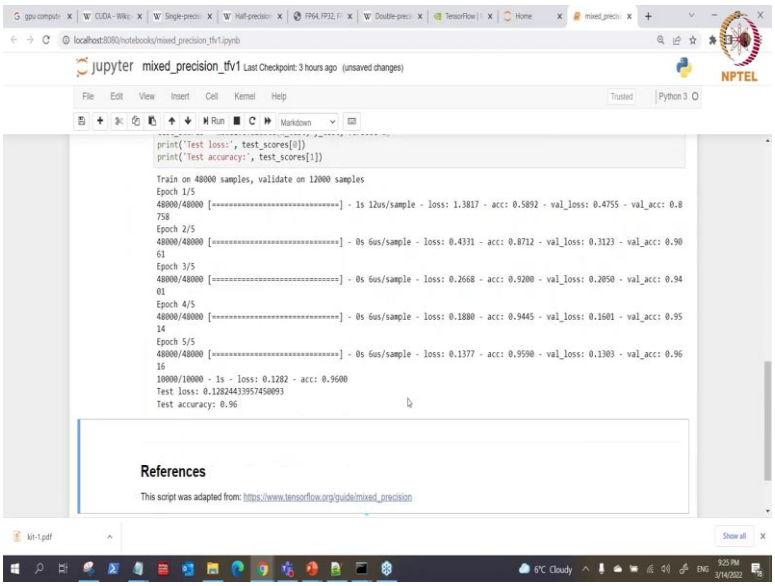
(Refer Slide Time: 23:24)



So, this is the crucial part here. So, we define our model here the input and output. So, then we load what our optimizer. So, after loading optimizer the next thing to do so to wrap the optimizer under tensorflow dot train dot experimental, then you wrap it with enable mixed precision graph rewrite. So, you wrap that against your optimizer. So, once you do that this is just a single line here.

So, which makes automatic mixed precision makes life easier. So, you wrap that here then you compile this is the normal line for to compile, you have your loss your optimizer your metrics that you want to use to compile. So, let me run that cell after running that cell then we can initialize when you initialize or initialization is done then.

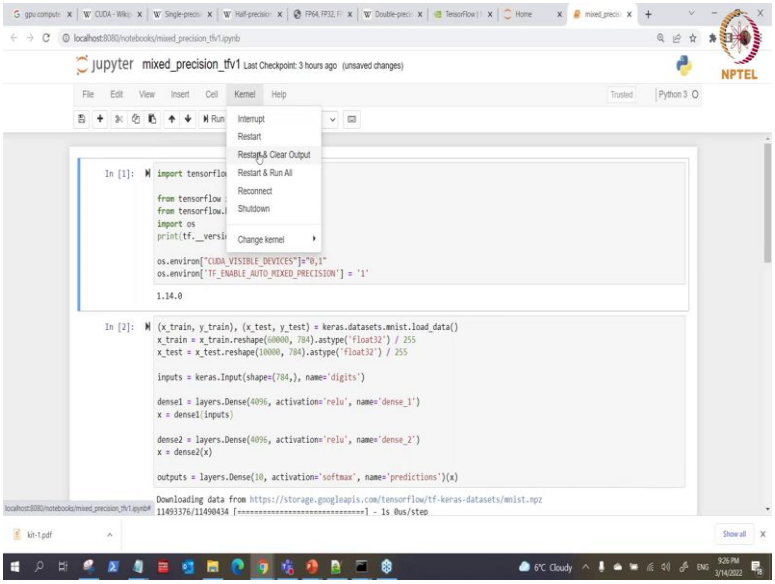
(Refer Slide Time: 24:27)



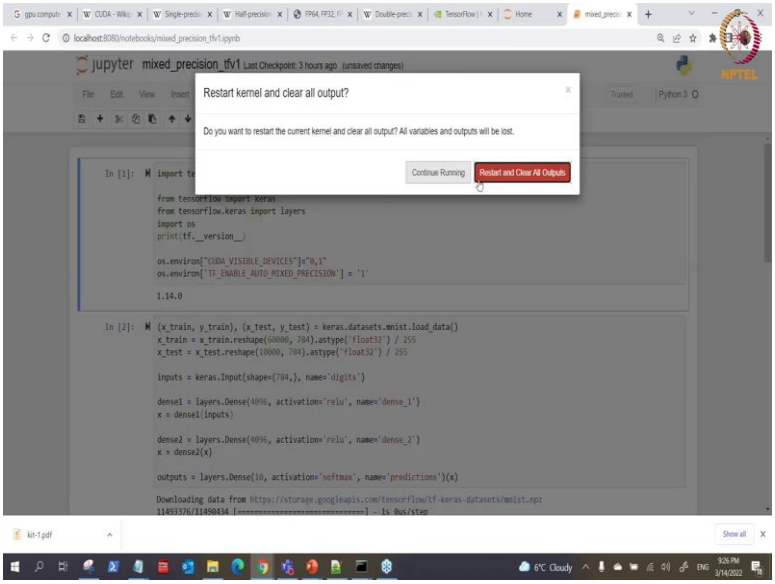
So, we can now run our model as well. So, you see we do not need to do anything for that here as well. So, just run the model you can see how the model runs, you see it is even almost it is faster a little bit faster. But we cannot judge based on the little more that we have done is 0. This is first one took 1 seconds and all that subsequent epoch, this number of steps just to 0 0 seconds and you can see this is just micro seconds yes 6 6 whether I want more than that ok.

So, this is just how to use automatic mixed precision. So, how do we check with this tensorflow one if we are not using automatic mixed precision or let us compare with when no automatic mixed precision is used or mixed precision is used.

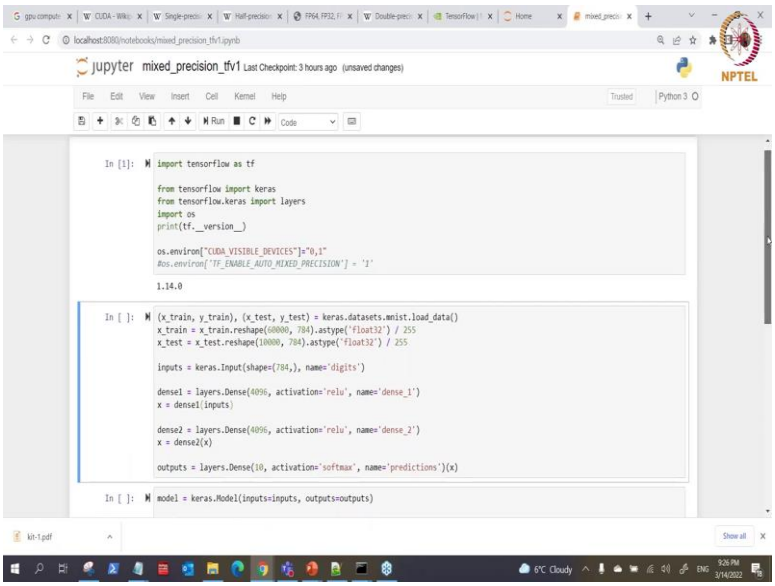
(Refer Slide Time: 25:09)



(Refer Slide Time: 25:13)

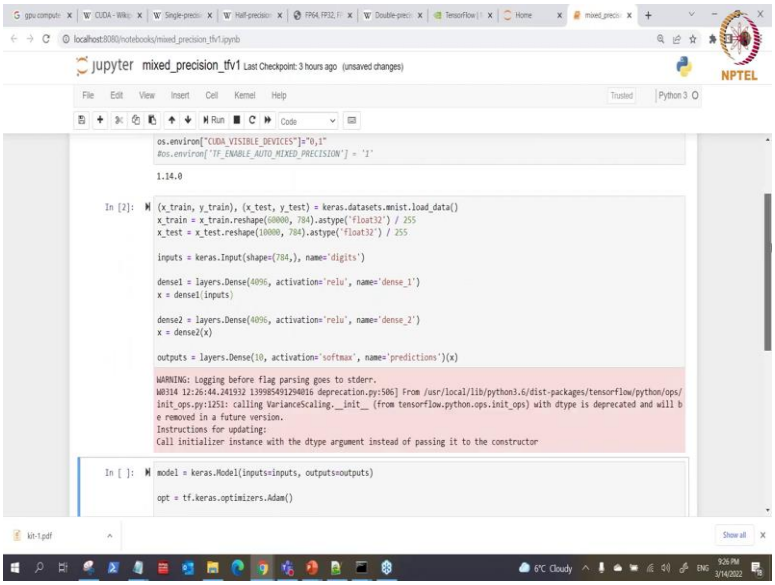


(Refer Slide Time: 25:15)



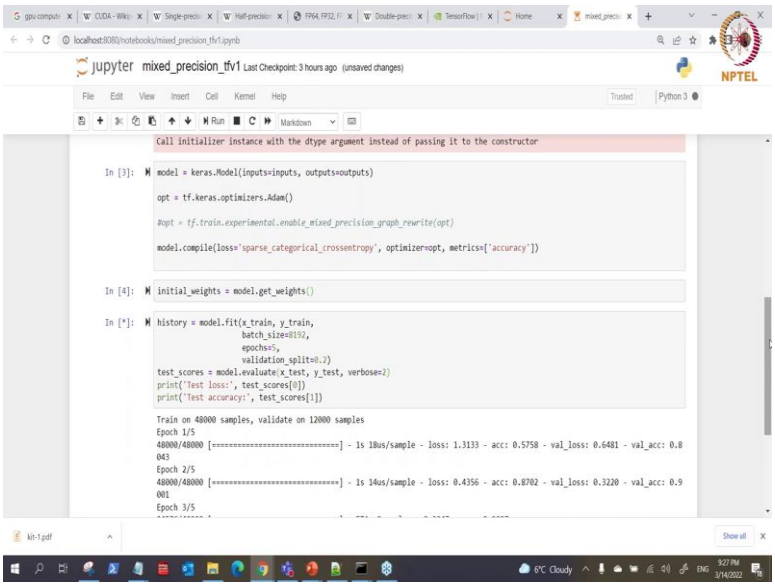
So, if I clear the output here I cleared then I would comment out this, that it is not used it is not enabled.

(Refer Slide Time: 25:22)



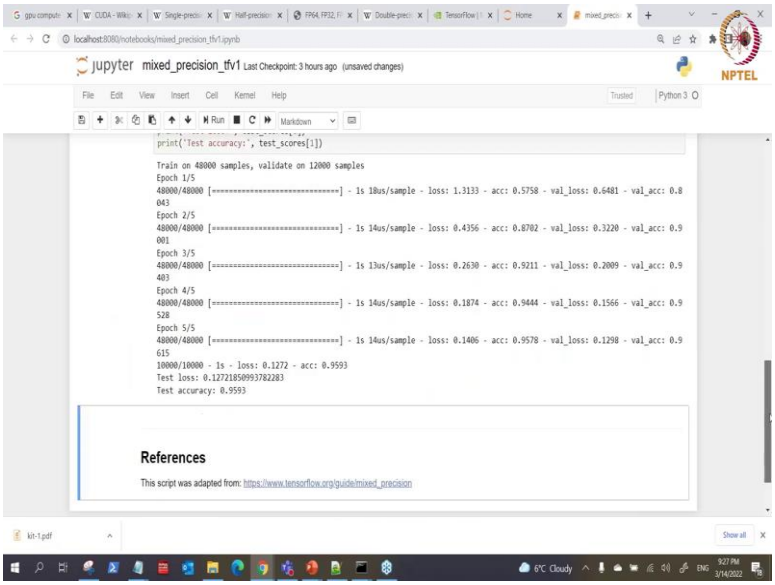
And I will also come here this is the line here and I comment this line out yeah. So now, we are just run in our normal tensorflow. So, we can see here I run this slide ok and also run this.

(Refer Slide Time: 25:41)



So, we do not actually need to put initialization we might not run that let us run it. So, and then let us see how long we took that.

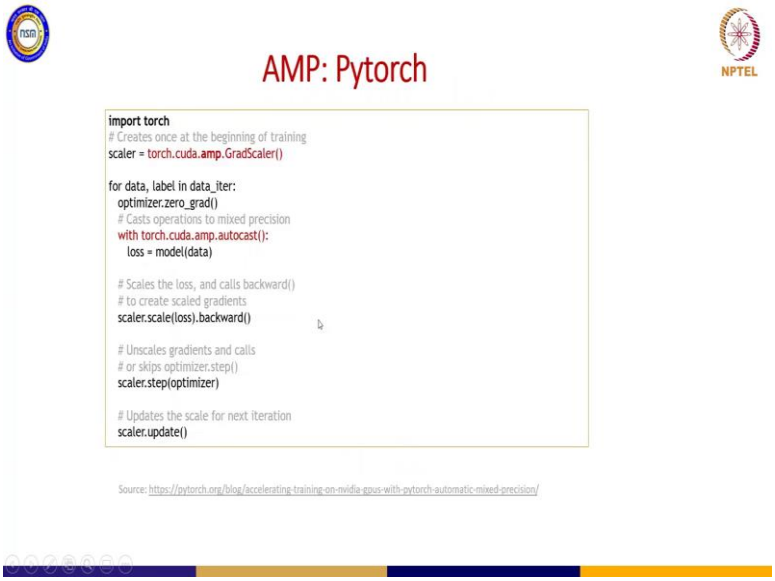
(Refer Slide Time: 26:05)



So, you can see when you run it normally this is steps it takes 1 second one seconds 14 microseconds 1 seconds second. So, this shows the efficacy imagine why you are running a bigger model, the scale up will be the differences will be very glaring that you can very easily see as compared to small model this is just a small demo I must say.

So, from here so we are back here, so I have explained how to use the automatic mixed precision with tensorflow rather it also exists in pytorch. So, this is how you can do that in pytorch. So, you import from cuda your automatic mixed precision with grad scalar, then for optimization here you auto cast as well you get your loss.

(Refer Slide Time: 26:37)



Then also you scale the loss the backward then you also would get your scaler dot step and then you update the weight as well. So, for more details you can check more details here and how to do this well, I did not prepare for the PyTorch version. That is why but I will if you check on this link it is very explanatory you can easily get how to do it very well so.

Thank you.