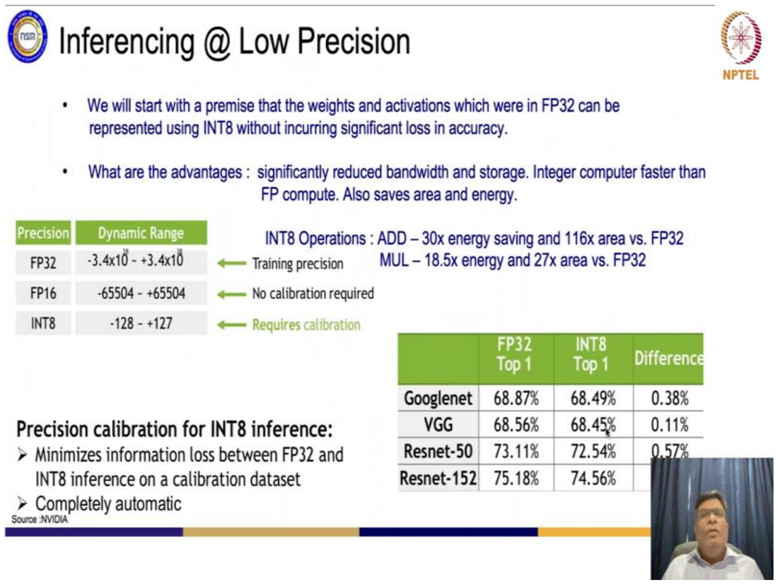
Applied Accelerated Artificial Intelligence
Prof. Satyadhyan Chickerur
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

Lecture - 41
Fundamentals of Accelerating Deployments Part 2

(Refer Slide Time: 00:15)



Inferencing @ Low Precision

- We will start with a premise that the weights and activations which were in FP32 can be represented using INT8 without incurring significant loss in accuracy.
- What are the advantages : significantly reduced bandwidth and storage. Integer computer faster than FP compute. Also saves area and energy.

Precision	Dynamic Range	
FP32	-3.4×10^{38} ~ $+3.4\times10^{38}$	⟵ Training precision
FP16	-65504 ~ +65504	⟵ No calibration required
INT8	-128 ~ +127	⟵ Requires calibration

INT8 Operations : ADD – 30x energy saving and 116x area vs. FP32
MUL – 18.5x energy and 27x area vs. FP32

	FP32 Top 1	INT8 Top 1	Difference
Googlenet	68.87%	68.49%	0.38%
VGG	68.56%	68.45%	0.11%
Resnet-50	73.11%	72.54%	0.57%
Resnet-152	75.18%	74.56%	[illegible]

Precision calibration for INT8 inference:
- Minimizes information loss between FP32 and INT8 inference on a calibration dataset
- Completely automatic

Source :NVIDIA

So now, let us try to understand this inferencing at low precision ok. So, again here we will start with the premise that the weights and activations which we used in during the training of our model was floating point 32, FP32. And it can be represented using INT8 without incurring significant loss in accuracy. This is a well known fact which people have been trying since long and it is possible to achieve this.

So, assuming that we know that when we convert our weights and activations from FP32 to INT8, we do not incur significant loss in accuracy ok. So, this is one way of trying to reduce in the memory; but it has got its disadvantages, we will see what are the disadvantages and what are the advantages, and how do we ensure that the accuracy is maintained ok.

Now, what are the advantages actually of doing this conversion from FP32 to INT8? What actually happens is when you do this you can significantly reduce the bandwidth and the storage. And your integer compute sorry; there is a typo here, it is basically integer compute is faster than the FP compute ok, and this also saves energy and area.

So, these are very very important advantages or very very significant advantages you reduce the bandwidth, you reduce the storage, your integer compute is faster than your floating-point compute, and you save area and energy as well. So, let us try to understand if you do a ADD operation as a INT8 operation instead of a FP32 operation, you are going to save 30 x energy and 116x area as against doing it on a floating point 32. And if you do a multiply operation using INT8 you are going to save 18.5 x energy and 27 x area versus floating point 32 operation.

So, now think of this just for one add and multiply like operation, you are saving so much energy so much area. And you are talking of deep neural networks which will use lot of such operations, you are talking of convolution neural networks right, they are deep neural networks we will have so many of such operations. And once you are able to reduce the energy the area ok, you will actually get ok lot of advantages in time as well ok.

So, the idea is that we should actually significantly use ok and try to use INT8 operations. So, there are so many other things wherein people talk of mixed precision, right automatically the software's can actually take in the required precision values and all of that. But we are not going into that what I am trying to tell you is if we do INT 8 operations at low precision, we can get good inferencing ok, as compared to if you just use FP32 right.

Now, what does it mean when we say precision from FP32 to INT8, where is the issue here? The issue is basically you are trying to do something called as quantization. If you are working with FP32, your dynamic range actually varies from -3.4×10^{38} to $+3.4\text{x}10^{38}$. This is the dynamic range in which your model would have got some values right, for the weights for the activation.

So, that is the whole dynamic range which you are working with, and this is what is let us assume to be the training precision. So, the training precisions dynamic range varies from $-3.4\text{x}10^{32}$ to $+3.4\text{x}10^{38}$ sorry 38 right. So, minus $3.4\text{x}10^{38}$ to $+3.4\text{x}10^{38}$, this is your training precision. Now, if you reduce it from FP32 to just FP16, your dynamic range changes from -65504 to +65504.

What does it mean, the dynamic range of your information or the values which have varied from this particular range to this particular range has to be condensed into -65504 to +65504 ok. So, what does it mean, here you are changing your precision from FP32 to FP16, your dynamic range reduces right. So, this basically means whatever information was put in that particular range should be compressed to this range right. And there are software's which are available which you try to use this particular step does not require any calibration.

So, you do not require any calibration ok, we will tell you what a calibration means in brief ok, why is it required and when you convert the FP32 to INT8 ok. You basically have to work with the dynamic range of minus 128 to plus 127. So, you would have trained your model with FP32 thinking that you have got this much precision, this much accuracy right, and this is the model which is giving me accuracy because I am using floating point 32.

And then now you try to convert it to INT8, you actually have to represent that information which you have thought from -128 to +127 and technically this is where some calibration is required right. We will not discuss it in today's class maybe we will tell you something about that in tomorrow's class. But the idea the next class the idea is this requires some calibration, because you will have to map it from a very very different dynamic range to a very small dynamic range of -128 to 127 right.

So, this basically means when you are trying to do this precision calibration for INT8 inferencing, it minimizes the information loss. Try understanding when you do the calibration what effectively happens is, we will minimize the information loss between FP32 and INT8 inference on a calibration data set. So, there has to be a specific calibration data set designed before you convert your FP32 into INT8.

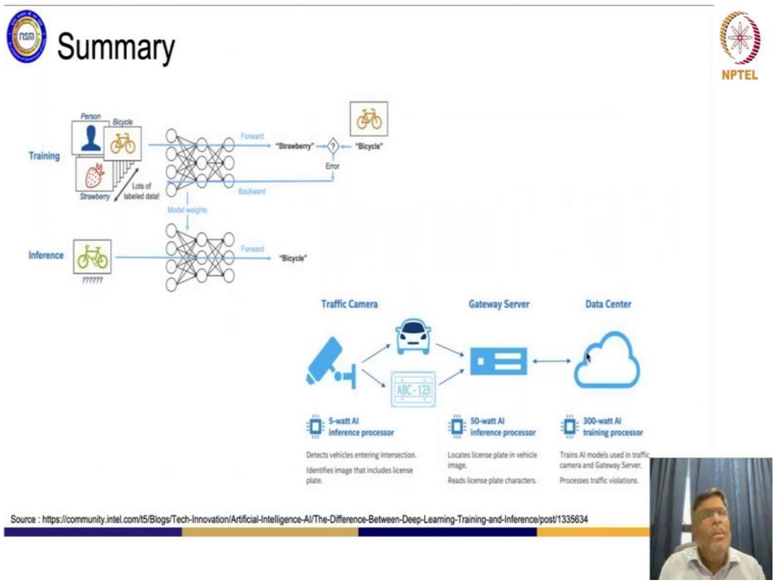
And that calibration data set is going to ensure that your precision your accuracy is not affected and this will be done automatically right; so, it is completely automatic ok. Now, let us try to understand the difference in the performance or accuracy of various networks ok. So, if you take FP32 and you consider top 1 results ok, whatever top results you consider for anything. This is the example where in Google net gives you 68.87 percent accuracy, VGG gives you this Resnet 50 gives you 73.11, Resnet 152 gives you 75.18.

So, when you do it using INT8, you optimize your model run it for inferencing ok. The INT8 model gives you accuracy of 68.49 as again 68.87. So, this is just a difference of

0.38 percent right, the same is the case with VGG you get a difference of 0.11 percent, 0.57 percent, for Resnet 50 and 0.61 percent for Resnet 152.

So, the idea here is that there will not be much of a difference of course, you can still go on improving and optimize it further. But this is not that much of a difference right, when you get the advantages of so many things like energy, bandwidth, storage, faster compute ok and everything. So, this is what is basically the gist of inferencing at low precision right ok. So, in summary let us try to understand what I have talked till now.

(Refer Slide Time: 11:19)



Summary

Training
Person
Bicycle
Strawberry
Lots of labeled data!
Forward
"Strawberry"
?
"Bicycle"
Error
Backward
Model weights
Inference
??????
Forward
"Bicycle"
Traffic Camera
Gateway Server
Data Center
ABC - 123
5-watt AI inference processor
Detects vehicles entering intersection.
Identifies image that includes license plate.
50-watt AI inference processor
Locates license plate in vehicle image.
Reads license plate characters.
300-watt AI training processor
Trains AI models used in traffic camera and Gateway Server.
Processes traffic violations.
NPTEL

Source : https://community.intel.com/t5/Blogs/Tech-Innovation/Artificial-Intelligence-AI/The-Difference-Between-Deep-Learning-Training-and-Inference/post/1335634

So, in essence the summary is whatever we have done till now points to this, we are actually training this type of a model. Let us say it is trained with lot of labeled data and then you have got this forward propagation, and then the backward propagation, error propagation everything of that. And your training model is able to actually predict something right, when you test it, when you do validation and everything.

Now, when you actually bring this model weights for that inferencing edge device or deploy the device or where you need to deploy. The inferencing is something like you give the input, it should give you the output this particular system right. For example, has to be implemented at traffic some traffic situation, right smart city situation.

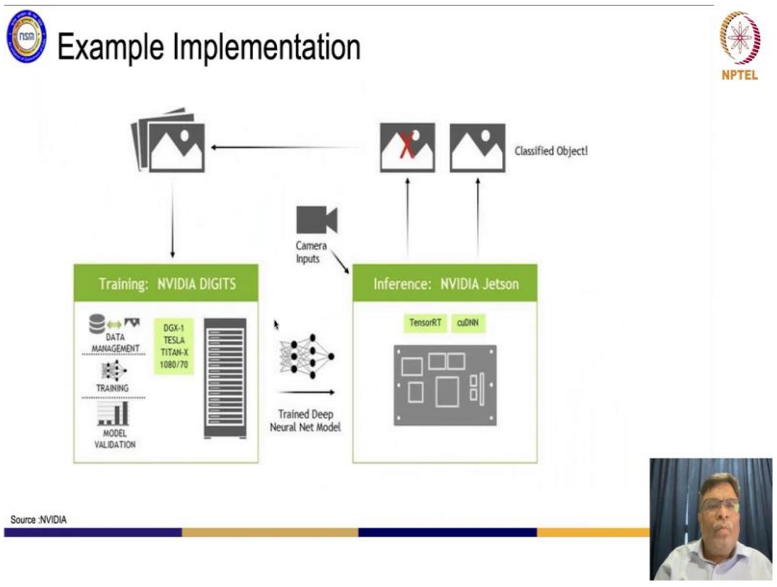
So, you have let us say a traffic camera, you have a gateway server, and you have a data center. So, the idea here is you have used and you have done training at the data center with GPU’s and let us say that this is a 300-watt AI training processor. What it does? It trains our AI model basically which is used in basically traffic camera and the gate way servers; so, that is the work.

Once it has trained it has to go for inferencing now right, and what does it process let us say it processes the traffic violations ok. Now, what does gateway server do? Gateway server is a 50-watt AI inference processor, here it is a inference processor with a 50-watt AI thing. What it does? It locates the license plate in vehicle image and it reads the license plate characters that is what a gateway server is going to do. What does a traffic camera do? So, a traffic camera is a 5-watt AI inference processor.

So, see the same trained model ok is running on a 300-watt AI training processor it is running on a 50-watt AI inference processor and then it is running on a 5-watt AI inference processor. So, in a 5-watt AI inference processor; the same trained thing, what it does? It detects the vehicle which is entering the intersection, it identifies the image that includes the license plate.

So, the inferencing here is done for this, the gateway server locates the license plate there and it detects the characters and here it detects the traffic violation. So, the idea is the level at which you are going to put in your model and the way in which it has to be used varies ok. And the idea is that here you are going to use something which is trained and something which has to be put or deployed for inferencing right.

(Refer Slide Time: 14:24)

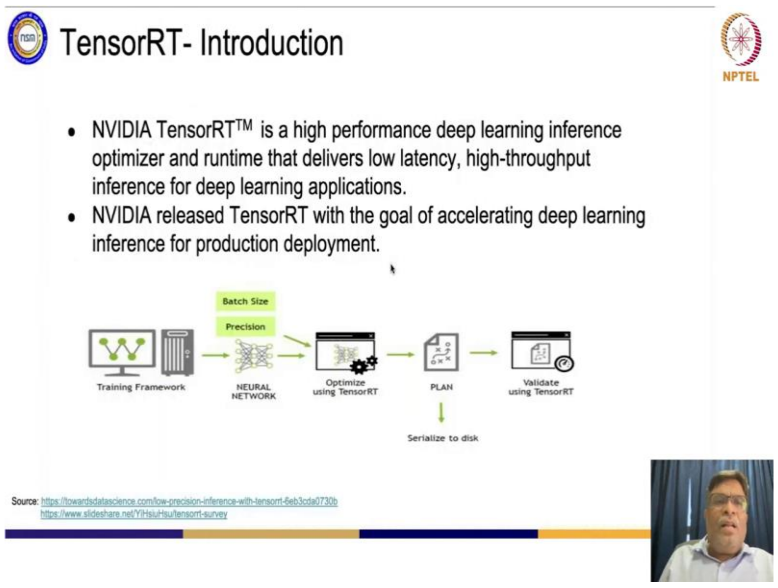


So, now, this is the actual example implementation, let us show you actually how we have implemented it in our lab right. So, what we have tried to do, we have tried to train our model right, let us say we have used Resnet 50. Here we are not using nvdia digits we are using Resnet 50, and Resnet 50 is basically trained ok on DGX 1. So, at this point in time we have got a trained deep neural network model which is a pre trained model ok, and we have NVIDIA jetson we have got NVIDIA jetson TX 1 TX 2; so, we will be showing you on TX 1 ok.

Now, that basically is a edge device right, and this edge device basically is used for inferencing. What does it have, we have to be very very clear that we are using tensorRT ok, we will tell you what tensorRT does and we have cuDNN also installed on our end jet circuit.

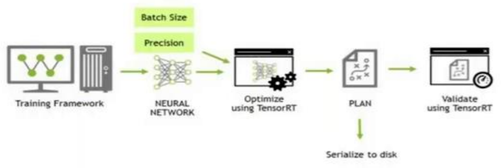
So, the idea is this model which has been trained for doing some classification tasks on the DGX is now to be converted optimized made smaller; so, that it can run on this jetson ok. Now, this jetson actually is able to classify the objects ok based on the inputs from the camera in the real-life setting. So, this is what is the situation right this is what we are going to actually do a implementation for ok.

(Refer Slide Time: 16:29)



TensorRT- Introduction

- NVIDIA TensorRT^{TM} is a high performance deep learning inference optimizer and runtime that delivers low latency, high-throughput inference for deep learning applications.
- NVIDIA released TensorRT with the goal of accelerating deep learning inference for production deployment.



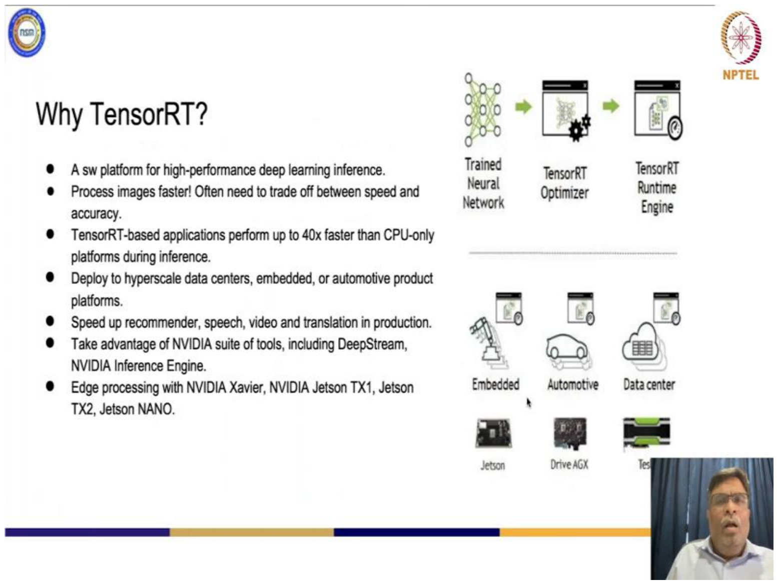
Source: https://towardsdatascience.com/low-precision-inference-with-tensorrt-6eb3cda0730b
https://www.slideshare.net/YiHsiuHsu/tensorrt-survey

So, now, let us try to understand the tensorRT thing right; so, what is tensor RT? I just showed you that we have something which is called as a tensorRT with cuDNN installed on our edge device, edge device is nothing, but our jetson ok. There are various other edge devices also, but we are showing you with NVIDIA jetson ok. So, this NVIDIA tensorRT is a high-performance deep learning inference optimizer and a runtime that delivers low latency high throughput inference for deep learning applications right.

So, what it means is we are trying to accelerate deep learning inferencing for production deployment, this is what is the basic essence. So, whatever we are; we were talking of all of this ok actually is taken care by this tensorRT. So, the goal is that we would be accelerating deep learning inference for production deployment.

Now, what does it mean, see here this is our training framework on DGX and all, our neural network is trained. We use the precision; we use the batch size from this we optimize it using our tensorRT and then we generate a plan. Now, this plan gives us a serial plan of plan 1, plan 2, plan 3 something like this, and then once we can validate it using tensorRT we actually would be using it for actually predicting ok.

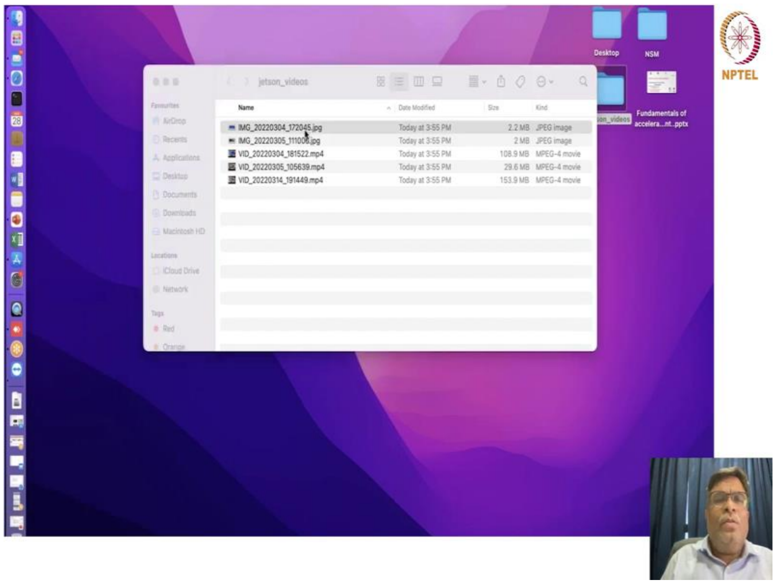
(Refer Slide Time: 18:40)


Why TensorRT?

- A sw platform for high-performance deep learning inference.
- Process images faster! Often need to trade off between speed and accuracy.
- TensorRT-based applications perform up to 40x faster than CPU-only platforms during inference.
- Deploy to hyperscale data centers, embedded, or automotive product platforms.
- Speed up recommender, speech, video and translation in production.
- Take advantage of NVIDIA suite of tools, including DeepStream, NVIDIA Inference Engine.
- Edge processing with NVIDIA Xavier, NVIDIA Jetson TX1, Jetson TX2, Jetson NANO.

So, now why tensor RT, and why it would is why it is required, and how do we use it maybe we will see it in tomorrow’s class. Now, we can do a video demonstration of what we have done I will just explain you in brief what we have done.

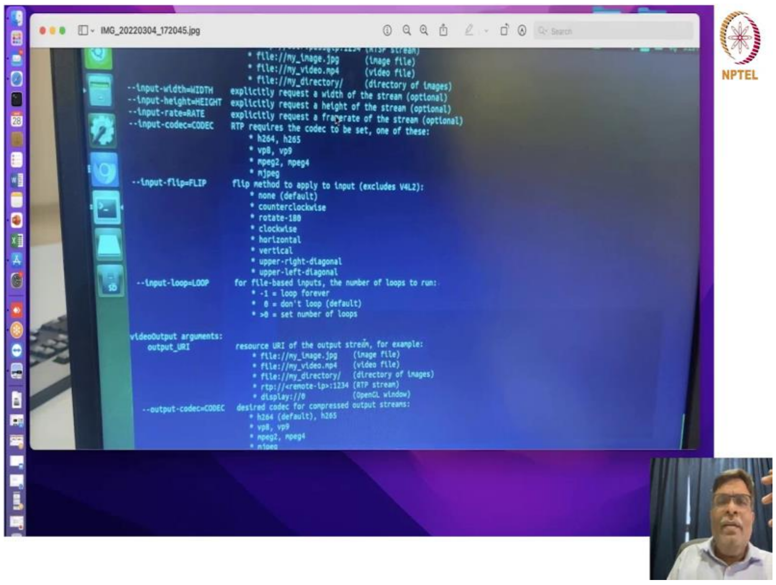
(Refer Slide Time: 19:05)



Name	Date Modified	Size	Kind
IMG_20220304_172045.jpg	Today at 3:55 PM	2.2 MB	JPEG image
IMG_20220305_111006.jpg	Today at 3:55 PM	2 MB	JPEG image
VID_20220304_181522.mp4	Today at 3:55 PM	108.9 MB	MPEG-4 movie
VID_20220305_105639.mp4	Today at 3:55 PM	29.6 MB	MPEG-4 movie
VID_20220314_191449.mp4	Today at 3:55 PM	153.9 MB	MPEG-4 movie

And let me just show you the images first ok.

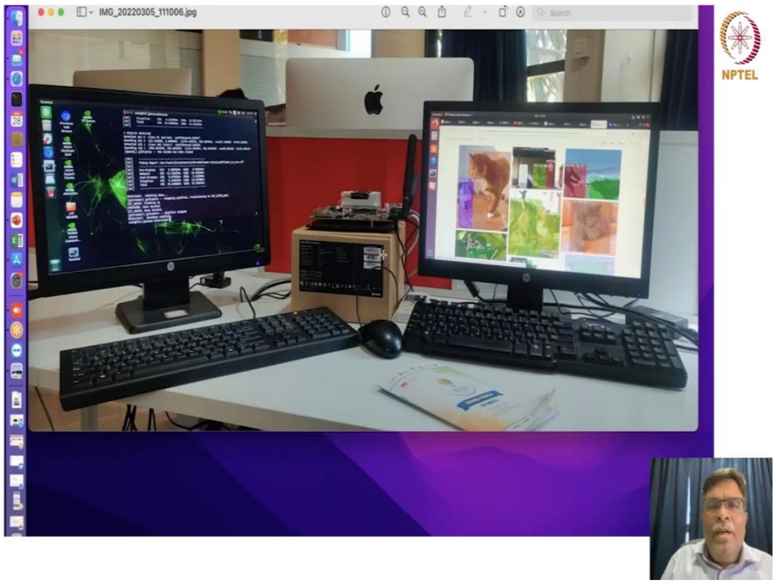
(Refer Slide Time: 19:11)



```
                         * file://my_image.jpg      (image file)
                         * file://my_video.mp4      (video file)
                         * file://my_directory/     (directory of images)
  --input-width=WIDTH    explicitly request a width of the stream (optional)
  --input-height=HEIGHT  explicitly request a height of the stream (optional)
  --input-rate=RATE      explicitly request a framerate of the stream (optional)
  --input-codec=CODEC    RTP requires the codec to be set, one of these:
                           * h264, h265
                           * vp8, vp9
                           * mpeg2, mpeg4
                           * mjpeg
  --input-flip=FLIP      flip method to apply to input (excludes V4L2):
                           * none (default)
                           * counterclockwise
                           * rotate-180
                           * clockwise
                           * horizontal
                           * vertical
                           * upper-right-diagonal
                           * upper-left-diagonal
  --input-loop=LOOP      for file-based inputs, the number of loops to run:
                           * -1 = loop forever
                           *  0 = don't loop (default)
                           * >0 = set number of loops

videoOutput arguments:
    output_URI           resource URI of the output stream, for example:
                           * file://my_image.jpg    (image file)
                           * file://my_video.mp4    (video file)
                           * file://my_directory/   (directory of images)
                           * rtp://<remote-ip>:1234 (RTP stream)
                           * display://0            (OpenGL window)
  --output-codec=CODEC   desired codec for compressed output streams:
                           * h264 (default), h265
                           * vp8, vp9
                           * mpeg2, mpeg4
```

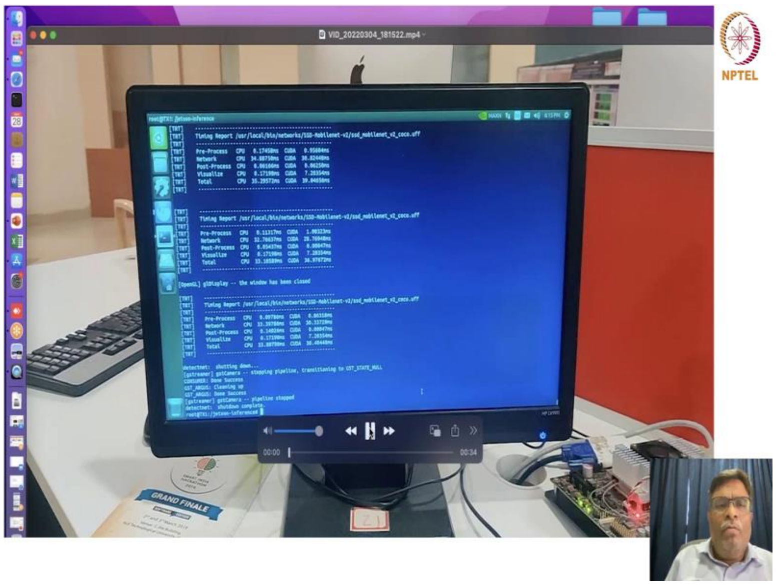
So, see here this is the output from the screen of our jetson device ok, this basically tells us about what are our video output arguments, what is the codec we are using, what is the width size ok, what are my image files and all of that right.

(Refer Slide Time: 19:40)



And this is being shown on a setup like this ok; so, this is our NVIDIA jetson TX 1, this is how it is running the program ok of various types right. We have done some semantic segmentation programs, we have done classification, we have done detection there are so, many things which run on this ok. Now, the idea is once you start working on a edge device, you will try to understand that there is a difference in which there is a way there is a difference in way your DGX does inferencing and the inferencing which this particular type of edge device does ok. So, there are three videos which I will show you today and maybe tomorrow we will see as to how that is to be achieved right.

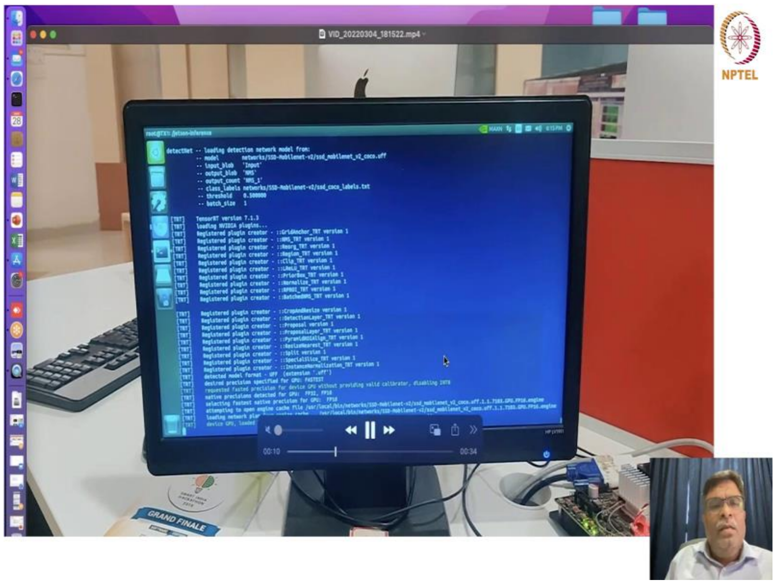
(Refer Slide Time: 20:37)



```
root@TX1: /jetson-inference
[TRT]    ------------------------------------------------
[TRT]    Timing Report /usr/local/bin/networks/SSD-Mobilenet-v2/ssd_mobilenet_v2_coco.uff
[TRT]    ------------------------------------------------
[TRT]    Pre-Process   CPU   0.17450ms  CUDA   0.95604ms
[TRT]    Network       CPU  34.88750ms  CUDA  30.82448ms
[TRT]    Post-Process  CPU   0.06166ms  CUDA   0.06250ms
[TRT]    Visualize     CPU   0.17198ms  CUDA   7.28354ms
[TRT]    Total         CPU  35.29572ms  CUDA  39.04656ms
[TRT]    ------------------------------------------------

[TRT]    ------------------------------------------------
[TRT]    Timing Report /usr/local/bin/networks/SSD-Mobilenet-v2/ssd_mobilenet_v2_coco.uff
[TRT]    ------------------------------------------------
[TRT]    Pre-Process   CPU   0.11317ms  CUDA   1.00323ms
[TRT]    Network       CPU  32.76637ms  CUDA  28.76948ms
[TRT]    Post-Process  CPU   0.05437ms  CUDA   0.00047ms
[TRT]    Visualize     CPU   0.17198ms  CUDA   7.28354ms
[TRT]    Total         CPU  33.10589ms  CUDA  36.97672ms
[TRT]    ------------------------------------------------

[OpenGL] glDisplay -- the window has been closed

[TRT]    ------------------------------------------------
[TRT]    Timing Report /usr/local/bin/networks/SSD-Mobilenet-v2/ssd_mobilenet_v2_coco.uff
[TRT]    ------------------------------------------------
[TRT]    Pre-Process   CPU   0.09780ms  CUDA   0.86318ms
[TRT]    Network       CPU  33.39780ms  CUDA  30.33729ms
[TRT]    Post-Process  CPU   0.14026ms  CUDA   0.00047ms
[TRT]    Visualize     CPU   0.17190ms  CUDA   7.28354ms
[TRT]    Total         CPU  33.80798ms  CUDA  38.40448ms
[TRT]    ------------------------------------------------

detectnet:  shutting down...
[gstreamer] gstCamera -- stopping pipeline, transitioning to GST_STATE_NULL
CONSUMER: Done Success
GST_ARGUS: Cleaning up
GST_ARGUS: Done Success
[gstreamer] gstCamera -- pipeline stopped
detectnet:  shutdown complete.
root@TX1:/jetson-inference#
```

So, let us start with this video which talks of ok how you can get the information from the camera ok.

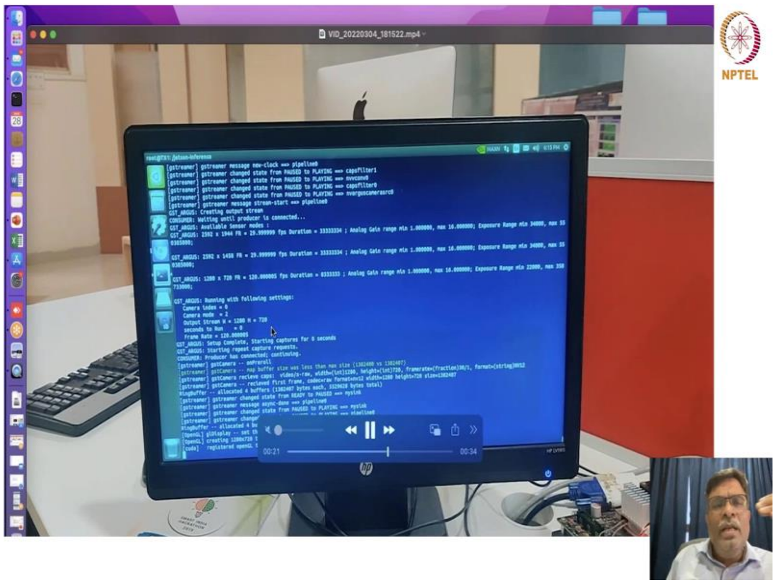
(Refer Slide Time: 20:45)



```
root@TX1: /jetson-inference
detectNet -- loading detection network model from:
          -- model        networks/SSD-Mobilenet-v2/ssd_mobilenet_v2_coco.uff
          -- input_blob   'Input'
          -- output_blob  'NMS'
          -- output_count 'NMS_1'
          -- class_labels networks/SSD-Mobilenet-v2/ssd_coco_labels.txt
          -- threshold    0.500000
          -- batch_size   1

[TRT]    TensorRT version 7.1.3
[TRT]    loading NVIDIA plugins...
[TRT]    Registered plugin creator - ::GridAnchor_TRT version 1
[TRT]    Registered plugin creator - ::NMS_TRT version 1
[TRT]    Registered plugin creator - ::Reorg_TRT version 1
[TRT]    Registered plugin creator - ::Region_TRT version 1
[TRT]    Registered plugin creator - ::Clip_TRT version 1
[TRT]    Registered plugin creator - ::LReLU_TRT version 1
[TRT]    Registered plugin creator - ::PriorBox_TRT version 1
[TRT]    Registered plugin creator - ::Normalize_TRT version 1
[TRT]    Registered plugin creator - ::RPROI_TRT version 1
[TRT]    Registered plugin creator - ::BatchedNMS_TRT version 1

[TRT]    Registered plugin creator - ::CropAndResize version 1
[TRT]    Registered plugin creator - ::DetectionLayer_TRT version 1
[TRT]    Registered plugin creator - ::Proposal version 1
[TRT]    Registered plugin creator - ::ProposalLayer_TRT version 1
[TRT]    Registered plugin creator - ::PyramidROIAlign_TRT version 1
[TRT]    Registered plugin creator - ::ResizeNearest_TRT version 1
[TRT]    Registered plugin creator - ::Split version 1
[TRT]    Registered plugin creator - ::SpecialSlice_TRT version 1
[TRT]    Registered plugin creator - ::InstanceNormalization_TRT version 1
[TRT]    detected model format - UFF  (extension '.uff')
[TRT]    desired precision specified for GPU: FASTEST
[TRT]    requested fasted precision for device GPU without providing valid calibrator, disabling INT8
[TRT]    native precisions detected for GPU:  FP32, FP16
[TRT]    selecting fastest native precision for GPU:  FP16
[TRT]    attempting to open engine cache file /usr/local/bin/networks/SSD-Mobilenet-v2/ssd_mobilenet_v2_coco.uff.1.1.7103.GPU.FP16.engine
[TRT]    loading network plan from engine cache... /usr/local/bin/networks/SSD-Mobilenet-v2/ssd_mobilenet_v2_coco.uff.1.1.7103.GPU.FP16.engine
[TRT]    device GPU, loaded
```

And then do things and optimized from the tensorRT thing right, we are using tensorRT version 7.1.3.

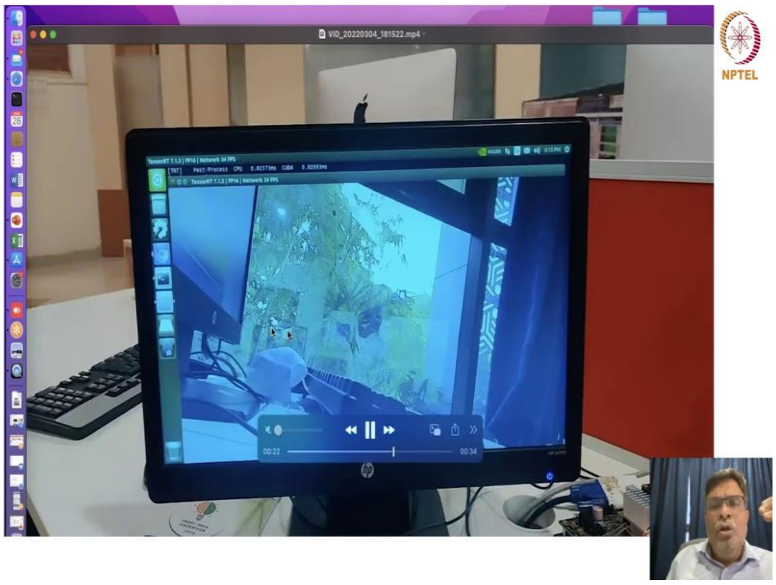
(Refer Slide Time: 20:58)



```
root@TX1: /jetson-inference
[gstreamer] gstreamer message new-clock ==> pipeline0
[gstreamer] gstreamer changed state from PAUSED to PLAYING ==> capsfilter1
[gstreamer] gstreamer changed state from PAUSED to PLAYING ==> nvvconv0
[gstreamer] gstreamer changed state from PAUSED to PLAYING ==> capsfilter0
[gstreamer] gstreamer changed state from PAUSED to PLAYING ==> nvarguscamerasrc0
[gstreamer] gstreamer message stream-start ==> pipeline0
GST_ARGUS: Creating output stream
CONSUMER: Waiting until producer is connected...
GST_ARGUS: Available Sensor modes :
GST_ARGUS: 2592 x 1944 FR = 29.999999 fps Duration = 33333334 ; Analog Gain range min 1.000000, max 16.000000; Exposure Range min 34000, max 55
0385000;

GST_ARGUS: 2592 x 1458 FR = 29.999999 fps Duration = 33333334 ; Analog Gain range min 1.000000, max 16.000000; Exposure Range min 34000, max 55
0385000;

GST_ARGUS: 1280 x 720 FR = 120.000005 fps Duration = 8333333 ; Analog Gain range min 1.000000, max 16.000000; Exposure Range min 22000, max 358
733000;

GST_ARGUS: Running with following settings:
   Camera index = 0
   Camera mode  = 2
   Output Stream W = 1280 H = 720
   seconds to Run    = 0
   Frame Rate = 120.000005
GST_ARGUS: Setup Complete, Starting captures for 0 seconds
GST_ARGUS: Starting repeat capture requests.
CONSUMER: Producer has connected; continuing.
[gstreamer] gstCamera -- onPreroll
[gstreamer] gstCamera -- map buffer size was less than max size (1382400 vs 1382407)
[gstreamer] gstCamera recieve caps:  video/x-raw, width=(int)1280, height=(int)720, framerate=(fraction)30/1, format=(string)NV12
[gstreamer] gstCamera -- recieved first frame, codec=raw format=nv12 width=1280 height=720 size=1382407
[gstreamer] gstCamera -- allocated 4 buffers (1382407 bytes each, 5529628 bytes total)
RingBuffer -- allocated 4 buffers (1382407 bytes each, 5529628 bytes total)
[gstreamer] gstreamer changed state from READY to PAUSED ==> mysink
[gstreamer] gstreamer message async-done ==> pipeline0
[gstreamer] gstreamer changed state from PAUSED to PLAYING ==> mysink
[gstreamer] gstreamer changed state from PAUSED to PLAYING ==> pipeline0
[gstreamer] gstreamer changed
RingBuffer -- allocated 4 bu
[OpenGL] glDisplay -- set th
[OpenGL] creating 1280x720 t
[cuda]   registered openGL t
```

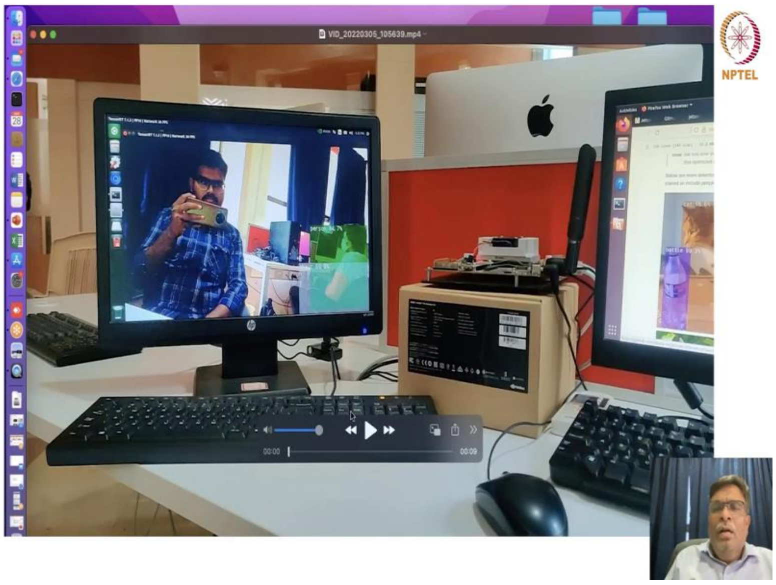
And there are various things which you will use.

(Refer Slide Time: 21:01)



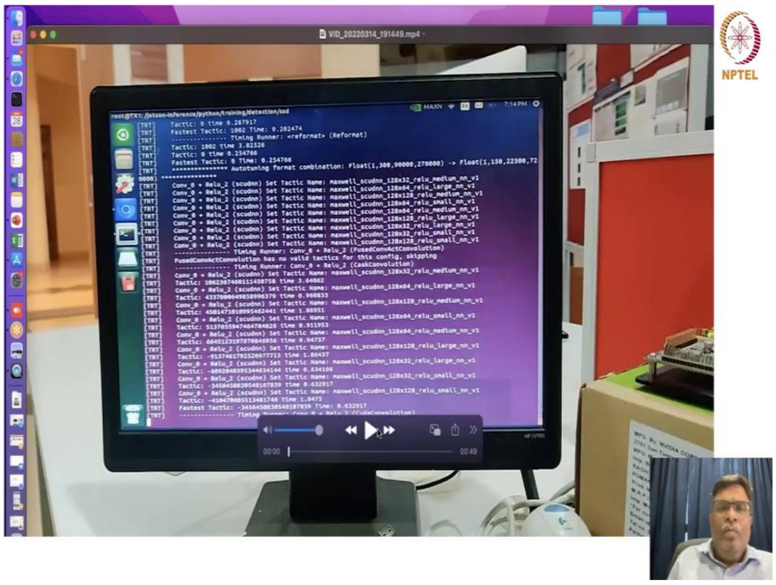
So, this is basically trying to do some object detection ok; now here the idea is we have tried to link the hand to a person ok. So, that is what we did some modification just wanted to show you like if you change the model, if you do the training in such a way right it can affect your inferencing a lot. So, this is our jetson TX 1 which we are working with.

(Refer Slide Time: 21:33)



Now, this is another video which talks of trying to do detection ok of person chair ok everything. So, see the way in which it is detecting right; the throughput rate, the speed at which it is detecting, the speed at which it is inferencing right, all of this actually is being executed on this jetson TX 1 ok so, right. So, this has got a small camera ok which basically we will show you tomorrow in detail, but this is the camera here which is trying to capture the image right of the person sitting here and here. So, this is how actually we are trying to do this thing ok.

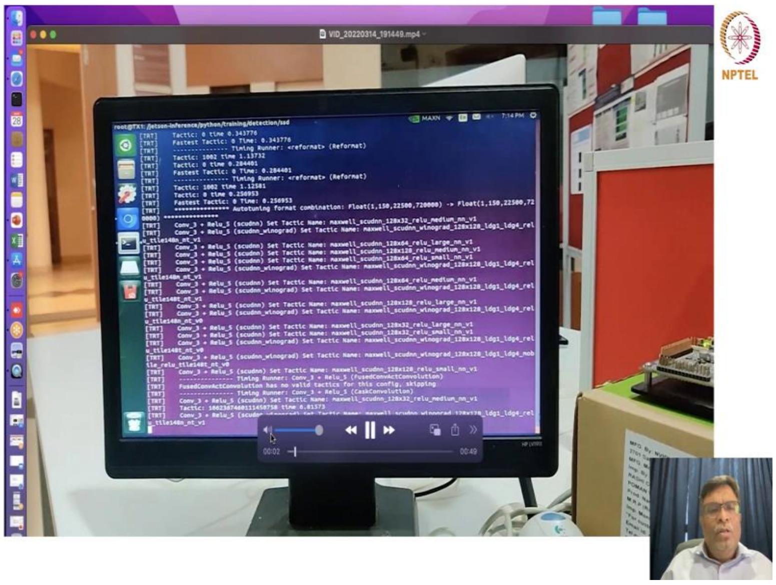
(Refer Slide Time: 22:33)



```
root@TX1: /jetson-inference/python/training/detection/ssd
[TRT]    Tactic: 0 time 0.287917
[TRT]    Fastest Tactic: 1002 Time: 0.202474
[TRT]    --------------- Timing Runner: <reformat> (Reformat)
[TRT]    Tactic: 1002 time 3.82526
[TRT]    Tactic: 0 time 0.254766
[TRT]    Fastest Tactic: 0 Time: 0.254766
[TRT]    *************** Autotuning format combination: Float(1,300,90000,270000) -> Float(1,150,22500,72
0000) ***************
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_medium_nn_v1
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_large_nn_v1
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_medium_nn_v1
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_small_nn_v1
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_medium_nn_v1
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_large_nn_v1
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_large_nn_v1
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_small_nn_v1
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_small_nn_v1
[TRT]    --------------- Timing Runner: Conv_0 + Relu_2 (FusedConvActConvolution)
[TRT]    FusedConvActConvolution has no valid tactics for this config, skipping
[TRT]    --------------- Timing Runner: Conv_0 + Relu_2 (CaskConvolution)
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_medium_nn_v1
[TRT]    Tactic: 1062367460111450758 time 3.64062
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_large_nn_v1
[TRT]    Tactic: 4337000649858996379 time 0.960833
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_medium_nn_v1
[TRT]    Tactic: 4501471010995462441 time 1.86951
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_small_nn_v1
[TRT]    Tactic: 5137655947464784826 time 0.911953
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_medium_nn_v1
[TRT]    Tactic: 6645123197870846856 time 0.94737
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_large_nn_v1
[TRT]    Tactic: -9137461792520977713 time 1.86437
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_large_nn_v1
[TRT]    Tactic: -6092040395344634144 time 0.634166
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_small_nn_v1
[TRT]    Tactic: -3456450830548107839 time 0.632917
[TRT]    Conv_0 + Relu_2 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_small_nn_v1
[TRT]    Tactic: -4104700055134817746 time 1.0475
[TRT]    Fastest Tactic: -3456450830548107839 Time: 0.632917
[TRT]    --------------- Timing Runner: Conv_0 + Relu_2 (CudaConvolution)
```

So, then there is another video which actually does how this convolution ok, 0 Relu and everything actually ok gets converted. And how much time it takes right all of that I am just trying to show you today.

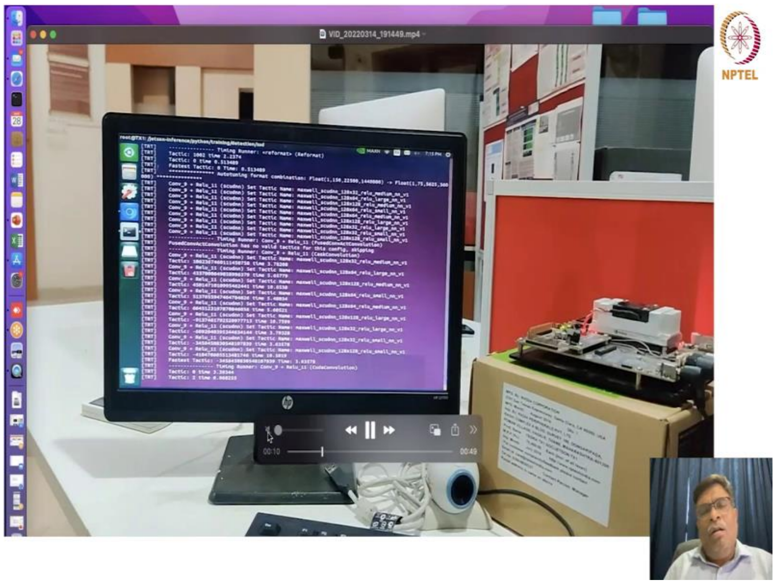
(Refer Slide Time: 22:51)



```
root@TX1: /jetson-inference/python/training/detection/ssd
[TRT]    Tactic: 0 time 0.343776
[TRT]    Fastest Tactic: 0 Time: 0.343776
[TRT]    --------------- Timing Runner: <reformat> (Reformat)
[TRT]    Tactic: 1002 time 1.13732
[TRT]    Tactic: 0 time 0.284401
[TRT]    Fastest Tactic: 0 Time: 0.284401
[TRT]    --------------- Timing Runner: <reformat> (Reformat)
[TRT]    Tactic: 1002 time 1.12581
[TRT]    Tactic: 0 time 0.250953
[TRT]    Fastest Tactic: 0 Time: 0.250953
[TRT]    *************** Autotuning format combination: Float(1,150,22500,720000) -> Float(1,150,22500,72
0000) ***************
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_medium_nn_v1
[TRT]    Conv_3 + Relu_5 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_rel
u_tile148n_nt_v1
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_large_nn_v1
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_medium_nn_v1
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_small_nn_v1
[TRT]    Conv_3 + Relu_5 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_rel
u_tile148n_nt_v1
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_medium_nn_v1
[TRT]    Conv_3 + Relu_5 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_rel
u_tile148t_nt_v1
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_large_nn_v1
[TRT]    Conv_3 + Relu_5 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_rel
u_tile148n_nt_v0
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_large_nn_v1
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_small_nn_v1
[TRT]    Conv_3 + Relu_5 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_rel
u_tile148t_nt_v0
[TRT]    Conv_3 + Relu_5 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_mob
ile_relu_tile148t_nt_v0
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_small_nn_v1
[TRT]    --------------- Timing Runner: Conv_3 + Relu_5 (FusedConvActConvolution)
[TRT]    FusedConvActConvolution has no valid tactics for this config, skipping
[TRT]    --------------- Timing Runner: Conv_3 + Relu_5 (CaskConvolution)
[TRT]    Conv_3 + Relu_5 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_medium_nn_v1
[TRT]    Tactic: 1062367460111450758 time 6.81573
[TRT]    Conv_3 + Relu_5 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_rel
u_tile148n_nt_v1
```

So, that the context is built; so, that you can actually try to understand things tomorrow in a better way right.

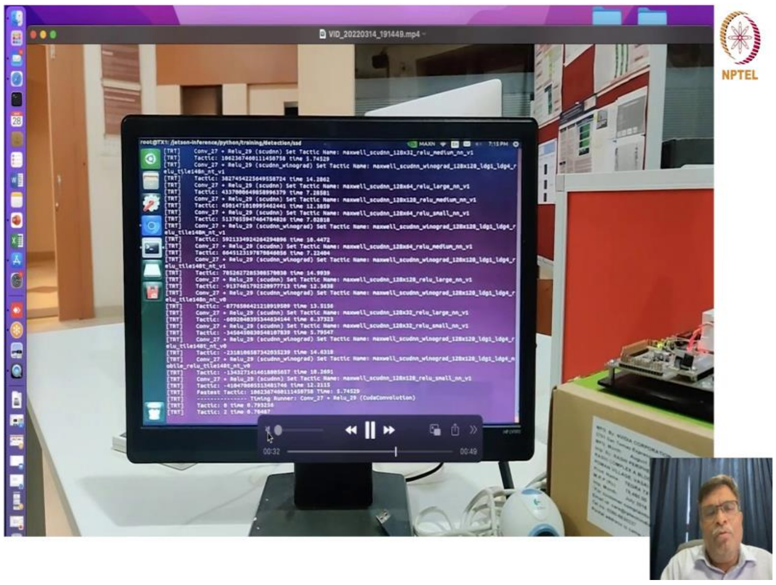
(Refer Slide Time: 22:59)



```
root@TX1: /jetson-inference/python/training/detection/ssd
[TRT]    --------------- Timing Runner: <reformat> (Reformat)
[TRT]    Tactic: 1002 time 2.2374
[TRT]    Tactic: 0 time 0.513489
[TRT]    Fastest Tactic: 0 Time: 0.513489
[TRT]    *************** Autotuning format combination: Float(1,150,22500,1440000) -> Float(1,75,5625,360
000) ***************
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_medium_nn_v1
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_large_nn_v1
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_medium_nn_v1
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_small_nn_v1
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_medium_nn_v1
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_large_nn_v1
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_large_nn_v1
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_small_nn_v1
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_small_nn_v1
[TRT]    --------------- Timing Runner: Conv_9 + Relu_11 (FusedConvActConvolution)
[TRT]    FusedConvActConvolution has no valid tactics for this config, skipping
[TRT]    --------------- Timing Runner: Conv_9 + Relu_11 (CaskConvolution)
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_medium_nn_v1
[TRT]    Tactic: 1062367460111450758 time 3.79200
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_large_nn_v1
[TRT]    Tactic: [illegible] time 5.65779
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_medium_nn_v1
[TRT]    Tactic: [illegible] time 10.0530
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_small_nn_v1
[TRT]    Tactic: [illegible] time 5.40834
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_medium_nn_v1
[TRT]    Tactic: [illegible] time 5.60521
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_large_nn_v1
[TRT]    Tactic: [illegible] time 10.7599
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_large_nn_v1
[TRT]    Tactic: [illegible] time 3.79328
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_small_nn_v1
[TRT]    Tactic: [illegible] time 3.63578
[TRT]    Conv_9 + Relu_11 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_small_nn_v1
[TRT]    Tactic: [illegible] time 10.1019
[TRT]    Fastest Tactic: [illegible] Time: 3.63578
[TRT]    --------------- Timing Runner: Conv_9 + Relu_11 (CudaConvolution)
[TRT]    Tactic: 0 time 2.38344
[TRT]    Tactic: 2 time 0.968255
```

When we start working on trying to show you these executions ok. So, this is how actually it will be processed and that particular video is actually going to be predicted right.

(Refer Slide Time: 23:21)



```
root@TX1: /jetson-inference/python/training/detection/ssd
[TRT]    Conv_27 + Relu_29 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_medium_nn_v1
[TRT]    Tactic: 1062367460111450758 time 5.74529
[TRT]    Conv_27 + Relu_29 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_r
elu_tile148n_nt_v1
[TRT]    Tactic: 3827454225649558724 time 14.2862
[TRT]    Conv_27 + Relu_29 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_large_nn_v1
[TRT]    Tactic: 4337000649858996379 time 7.28581
[TRT]    Conv_27 + Relu_29 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_medium_nn_v1
[TRT]    Tactic: 4501471010995462441 time 12.3859
[TRT]    Conv_27 + Relu_29 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_small_nn_v1
[TRT]    Tactic: 5137655947464784826 time 7.02818
[TRT]    Conv_27 + Relu_29 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_r
elu_tile148m_nt_v1
[TRT]    Tactic: 5921334924264294896 time 10.4472
[TRT]    Conv_27 + Relu_29 (scudnn) Set Tactic Name: maxwell_scudnn_128x64_relu_medium_nn_v1
[TRT]    Tactic: 6645123197878848856 time 7.22404
[TRT]    Conv_27 + Relu_29 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_r
elu_tile148t_nt_v1
[TRT]    Tactic: 7852627285308570838 time 14.9939
[TRT]    Conv_27 + Relu_29 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_large_nn_v1
[TRT]    Tactic: -9137461792520977713 time 12.3638
[TRT]    Conv_27 + Relu_29 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_r
elu_tile148n_nt_v0
[TRT]    Tactic: -8776586421218919589 time 13.5156
[TRT]    Conv_27 + Relu_29 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_large_nn_v1
[TRT]    Tactic: -6092040395344634144 time 6.37323
[TRT]    Conv_27 + Relu_29 (scudnn) Set Tactic Name: maxwell_scudnn_128x32_relu_small_nn_v1
[TRT]    Tactic: -3456430830548107839 time 5.79547
[TRT]    Conv_27 + Relu_29 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_r
elu_tile148t_nt_v0
[TRT]    Tactic: -2318106587342035239 time 14.6318
[TRT]    Conv_27 + Relu_29 (scudnn_winograd) Set Tactic Name: maxwell_scudnn_winograd_128x128_ldg1_ldg4_m
obile_relu_tile148t_nt_v0
[TRT]    Tactic: -1343271414618805657 time 10.2691
[TRT]    Conv_27 + Relu_29 (scudnn) Set Tactic Name: maxwell_scudnn_128x128_relu_small_nn_v1
[TRT]    Tactic: -4104700051134817746 time 12.2115
[TRT]    Fastest Tactic: 1062367460111450758 Time: 5.74529
[TRT]    --------------- Timing Runner: Conv_27 + Relu_29 (CudaConvolution)
[TRT]    Tactic: 0 time 0.793258
[TRT]    Tactic: 2 time 0.70467
```

So, that is how actually it is; so, I should have maybe shown you this before than the other video which I show you just now. So, that is how actually it is going to basically be executed ok and that is how you can develop in your own applications and put it on the edge device right this is how actually it is going to pan out ok. So, we have done it for cats, dogs we will show you tomorrow cats and dogs maybe; so, let us see, so yeah.

So, I suppose we are done with today’s session yeah, maybe any questions if you have we can start; so, maybe yeah; so, I will just wind off with y tensorRT just to complete the whole thing. So, what is actually tensor RT, and why we should use tensorRT is it is a software platform for high performance deep learning inferencing. You actually get a trained neural network, you optimize it using tensorRT optimizer, and then you use a tensorRT runtime engine to actually run right.

So, basically it is a software platform for high performance deep learning inference, it processes images faster that is what will happen right. So, often you need to trade off between speed and accuracy; obviously, as we discussed, tensorRT based applications perform up to 40x faster than CPU only platforms during inferencing. So, if you have a tensorRT based application developed, it is going to perform 40 x faster than just the CPU only platforms.

You can deploy to hyper scale data centers, embedded or automotive product platforms as shown here. So, you have jetson which we are working with, we have drive AGX which you can work with for automotive, and then you have tesla right basically on the data center. So, this tensorRT you can actually optimize your model and deploy it anywhere that is what it means right; so, it will be optimized according to where it basically is running right.

So, you can use the tensorRT for speeding up the recommender systems, speech processing, video translation. Now, you can use actually this tensorRT to take advantage of tools like DeepStream, and NVIDIA inference engines. You can do edge processing with NVIDIA Xavier's as I told you jetson TX 1, TX 2 and Jetson Nano all three ok; so, you can do everything using tensor RT. So, it is a deep learning inferencing software platform which we will show you tomorrow in the next class right.

So, yeah I think I am done now and we stop here yeah.