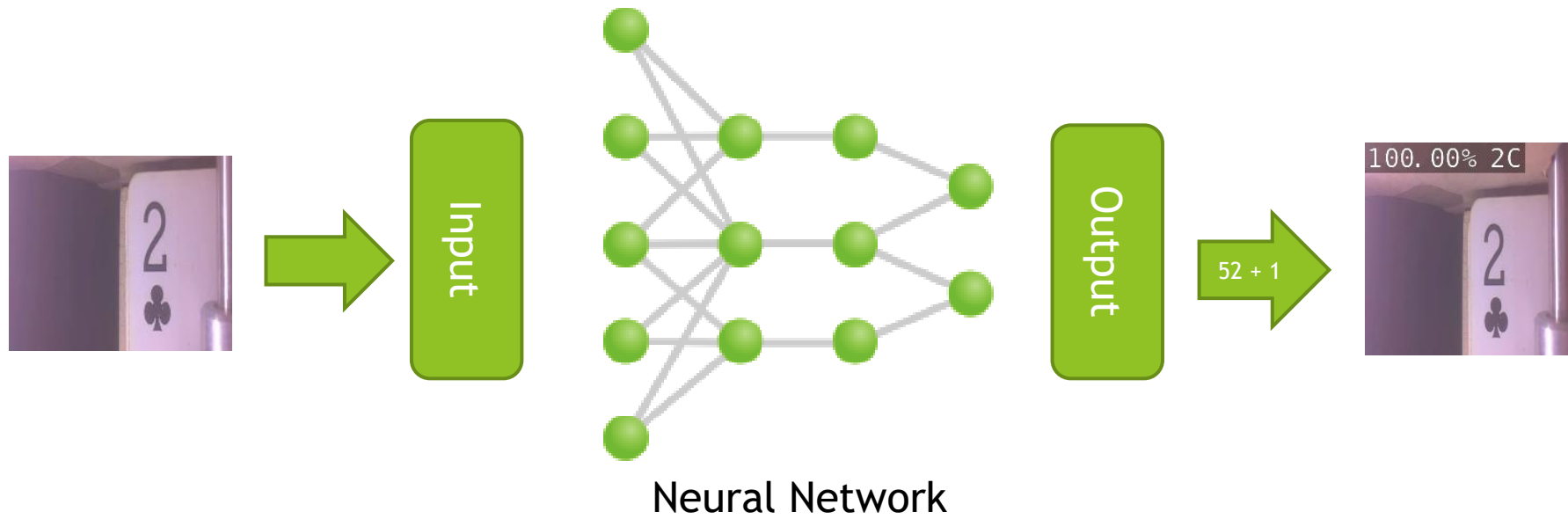


Playing Cards Recognition using Machine Learning (ML)

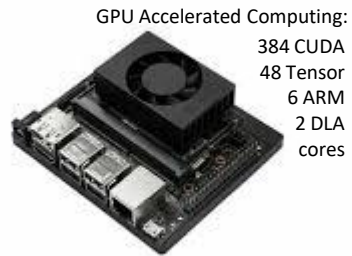
Project Goals



- ▶ Get into Machine Learning (ML)
- ▶ Develop Neural Network (NN) model for playing card recognition
- ▶ Test model on real-time system

System Setup / Learning ML Basics

- Acquired / installed Jetson hardware and development tools:



- Completed training and quick start tutorials:

Jetson AI
Fundamentals


Jetson AI Fundamentals - excellent course for beginners to quickly start ML development on Jetson boards. Includes Docker container with Jupyter lab and notebooks. I used this platform to develop my first ML model and train it for playing cards classification.

HELLO AI WORLD
NVIDIA JETSON

Hello AI World - quick start (with Docker container) for inferencing with TensorRT and Jetson. Learned how to collect datasets, train / 'transfer learn' off-the-shelf image classification models with PyTorch and convert / optimize them for embedded platform deployment.

Neural Network - Design

- Selected ResNet-18 based network (pre-trained on [ImageNet](#)) also, experimented with other network types pre-trained on ImageNet: AlexNet, GoogLeNet and ResNet-50)

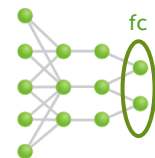
IMGENET dataset that spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images

- Used Transfer Learning technique (with PyTorch) to re-train ResNet-18 on playing card images

Transfer Learning is a technique for re-training a DNN model on a new dataset, which takes less time than training a network from scratch. With transfer learning, the weights of a pre-trained model are fine-tuned to classify a customized dataset.

- Defined Dataset Categories: ['2C', '3C', '4C', '5C', '6C', '7C', '8C', '9C', '10C', 'JC', 'QC', 'KC', 'AC', '2H', '3H', '4H', '5H', '6H', '7H', '8H', '9H', '10H', 'JH', 'QH', 'KH', 'AH', '2S', '3S', '4S', '5S', '6S', '7S', '8S', '9S', '10S', 'JS', 'QS', 'KS', 'AS', '2D', '3D', '4D', '5D', '6D', '7D', '8D', '9D', '10D', 'JD', 'QD', 'KD', 'AD', 'NONE']

- Defined the neural network and adjusted the fully connected layer (`fc`) to match the outputs required for the project:



RESNET 18

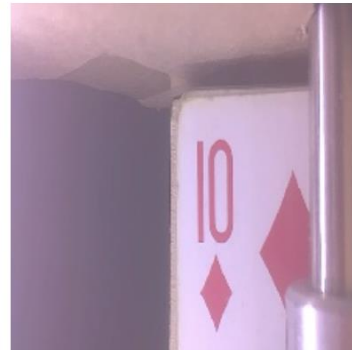
```
model = torchvision.models.resnet18(pretrained=True)
model.fc = torch.nn.Linear(512, len(dataset.categories))
```

Neural Network - Data Collection / Training

Data Collection:

- Initially, I used Jetson tools (from AI tutorials) to manually capture card images for model training, then (when training accuracy reached levels > 85%) I automated data collection with python scripting and playing card dealing machine

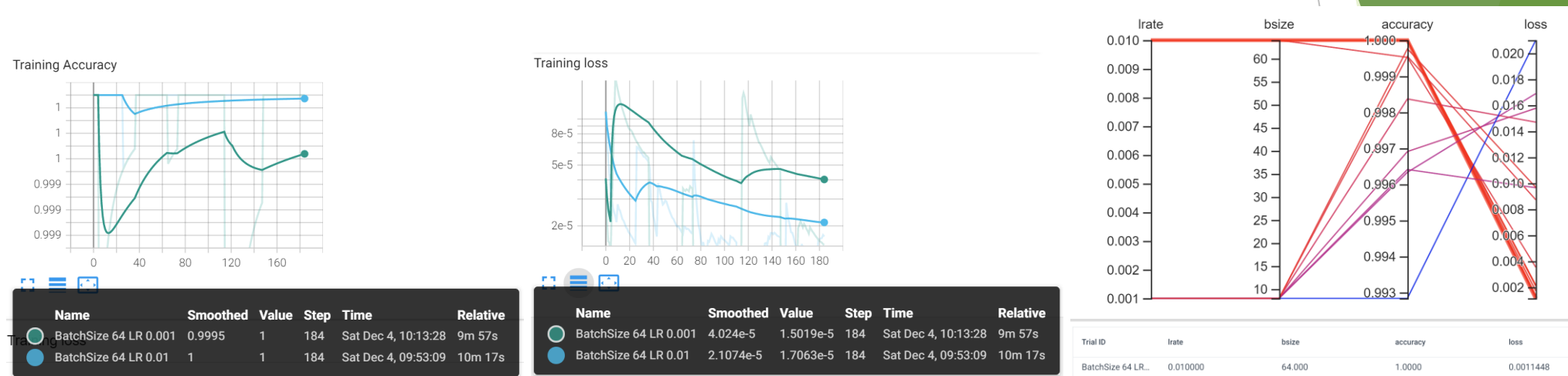
dataset	A	▼
category	10D	▼
count	58	▲▼
<input type="button" value="add"/>		
epochs	1	
progress	<div></div>	
loss	0.003134633822333011	
accuracy	0.9949302915082383	
<input type="button" value="train"/>		<input type="button" value="evaluate"/>
model path	card_detection_data/my_card_mc	
<input type="button" value="load model"/>		<input type="button" value="save model"/>



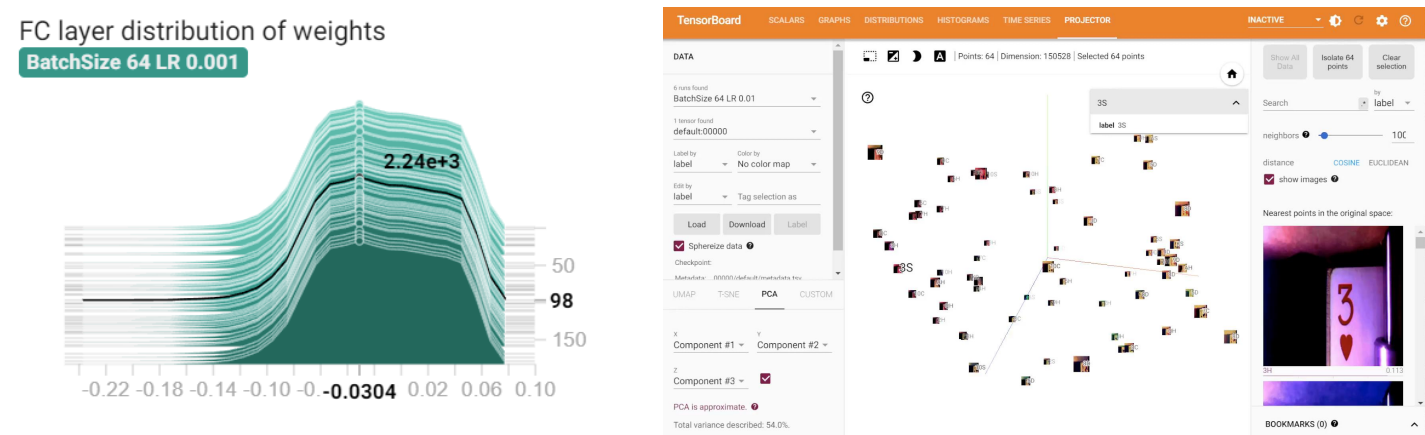
prediction	10D
score	0.9999996423721313
state	<input type="button" value="stop"/> <input checked="" type="button" value="live"/>

Neural Network Fine-Tuning with TensorBoard

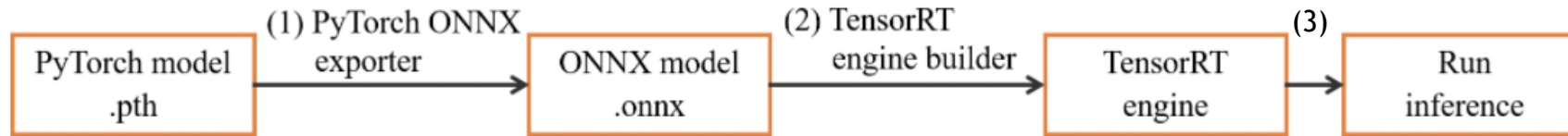
- Finding best training accuracy / loss for different batch size / learning rate values



- Plotting distribution of weights in FC layer for each training step and 3D model projector:



Model optimization for embedded deployment

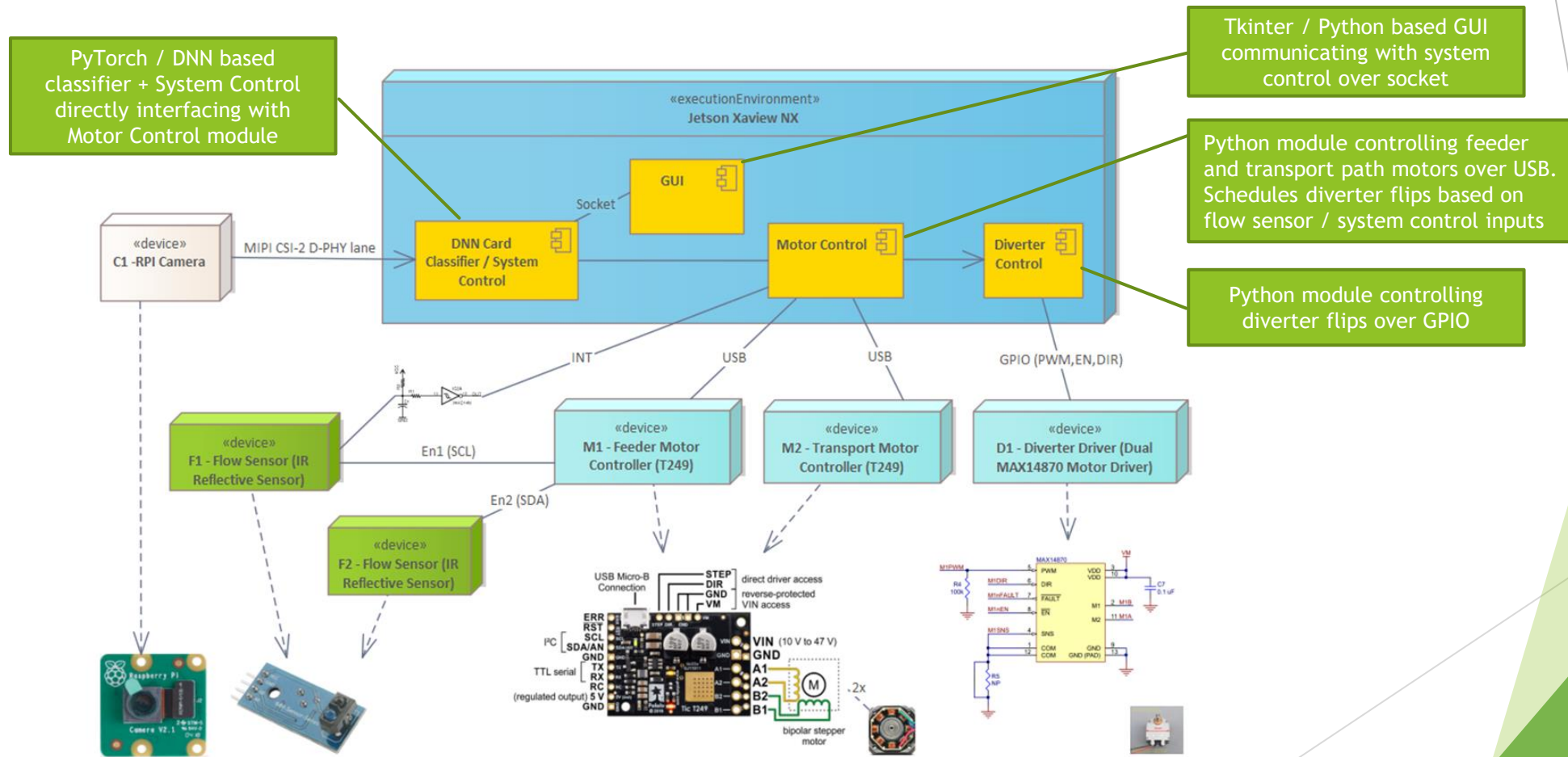


1. After training model using PyTorch framework, model files are in PyTorch model format (.pth)
2. In order to optimize inference speed on Jetson, the PyTorch model needs to be converted to TensorRT engine format. The conversion is done via an intermediate format called ONNX (Open Neural Network Exchange)
3. Accuracy and inference time comparison after TensorRT engine conversion:

Model	Image Size	PyTorch model		TensorRT model		
		Accuracy [%]	Frames/s	Precision	Accuracy [%]	Frames/s
Resnet-18	224x224	100	30	Float32	> 97	> 100

Software Deployment View

GUI, NN Card Classifier and System Control



Validation / Testing

The screenshot displays a computer interface for a bridge machine, titled "My Bridge Machine". The interface includes a "Machine Control" section with buttons for "Degl", "Stop", "Open", "Connect", and "Shutdown". Below this is a "DNN Model / Card Classification Control" section with checkboxes for "Validate" and "Collect Data". A central area shows a card layout with suits and ranks, including a highlighted section for "S" (South) with cards "AK7", "K10852", "QJ104", and "Q". To the right, a "Card Detector" window shows a card image and the text "100.00% JC". A terminal window at the bottom left displays a list of cards and their classification percentages, such as "30 100.00% 4D", "31 100.00% 9C", "32 100.00% 8C", "33 100.00% KD", "34 100.00% 5D", "35 100.00% 6C", "36 100.00% 10D", "37 100.00% 8D", "38 100.00% 7C", "39 100.00% 3C", "40 100.00% AC", "41 100.00% 10D", "42 100.00% 3D", "43 100.00% 10C", "44 100.00% 9C", "45 100.00% 7S", "46 100.00% 10C", "47 100.00% 4C", "48 100.00% KS", "49 100.00% 10S", "50 100.00% 4S", "51 100.00% JH", and "52 99.75% AS". A green arrow points from the highlighted card section in the center to a photograph of the physical bridge machine hardware, which includes a card deck and a card detector. The background of the interface is a green abstract pattern.

My Bridge Machine

Machine Control

demo1.pbn
demo2.pbn
demo3.pbn

Degl
Stop

DNN Model / Card Classification Control

☐ Validate
☐ Collect Data

Open
Connect
Shutdown

9 5 4 2
9 6
9 8 7
10 9 5 2

N
S

8 6 3
A J 7 3
5 2
K J 8 4

AK7
K10852
QJ104
Q

Card Detector | 30 f

100.00% JC

zbysek@zbysek-desktop: ~

30 100.00% 4D
31 100.00% 9C
32 100.00% 8C
33 100.00% KD
34 100.00% 5D
35 100.00% 6C
36 100.00% 10D
37 100.00% 8D
38 100.00% 7C
39 100.00% 3C
40 100.00% AC
41 100.00% 10D
42 100.00% 3D
43 100.00% 10C
44 100.00% 9C
45 100.00% 7S
46 100.00% 10C
47 100.00% 4C
48 100.00% KS
49 100.00% 10S
50 100.00% 4S
51 100.00% JH
52 99.75% AS

Profile: U
Scene Colle

Lessons Learned

- ▶ Neural net model design with PyTorch framework, Tkinter / Python GUI design
- ▶ How to develop, train optimize model for embedded platform
- ▶ How to accelerate ML / neural net inference with TensorRT and NVIDIA GPUs
- ▶ How to use TensorBoard to fine-tune, visualize and analyze machine learning / model data

Budget / Scope Review Against Plan

Budget Review

Item	Plan	Actual
Jetson Xavier NX development Kit	\$399	\$399
Arducam OV9281 1MP Global Shutter Camera Module	\$51	
Hardware mounting brackets and miscellaneous	\$25	
Seagate NVMe drive (32G SD card was too small for Linux, tools and model / image data)		\$62.99
Total	\$475	\$461.99

Scope Review

Work Item	Plan	Actual
Mechanical	Jetson mounting	More work than I planned. Had to add extra flow sensor mounting and S player stacker wall
Replace rolling shutter camera with global shutter one	Eliminate rolling shutter distortion	Managed to work with rolling distortion by filtering out transient frames and increasing spacing between cards
Hardware enhancements	Replace RPI3 with Jetson, reuse stepper motor controllers, flow sensors and power supply	More work than originally planned: added extra flow sensor to control spacing, extra work to drive diverter motor plus debouncing logic for sensor output to drive interrupt line
Software development	Port over RPI software to Jetson platform, Develop NN based classifier	Software development as planned. Disk space required for training and testing NN turned out to be bigger than expected.

What would I have done differently?

- ▶ Use CAD tools to design sensor and Jetson mountings
- ▶ Use 3D printing for mechanical parts
- ▶ Add overcurrent protection for diverter motor
- ▶ Do better project planning
- ▶ Order parts ahead of time

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.

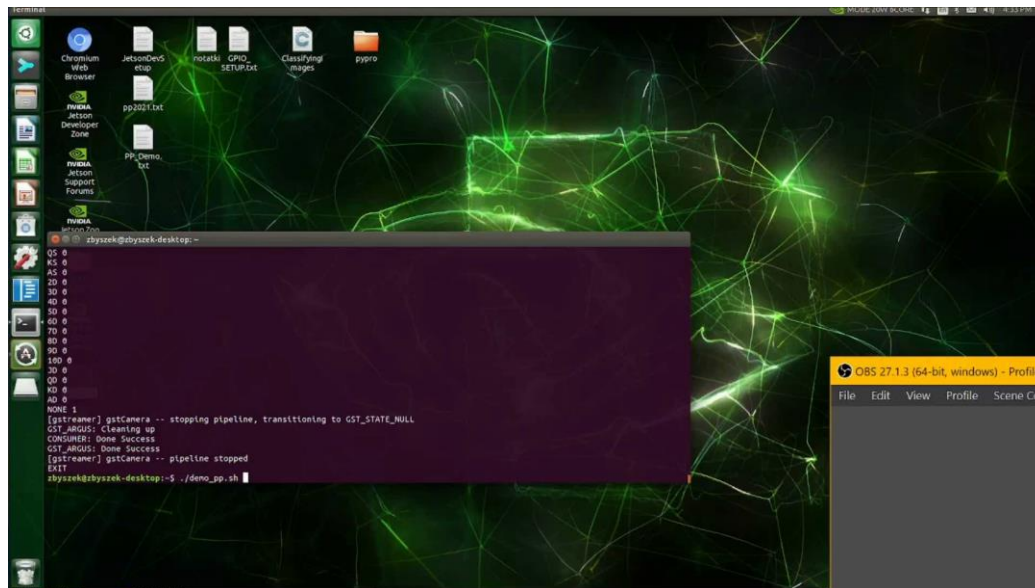
Q&A

Appendix A - NN Validation / Testing

Transient Cases



NN Validation



Appendix B - TensorBoard 3D Projector

