JETSON DEVELOPER KITS

for Engineers, Makers, and Learners

JETSON NANO 2GB

5W | 10W
0.5 TFLOPS (FP16)
$59

JETSON NANO

5W | 10W
0.5 TFLOPS (FP16)
$99

JETSON XAVIER NX

10W | 15W
6 TFLOPS (FP16) | 21 TOPS (INT8)
$399

JETSON AGX XAVIER

10 | 15W | 30W
11 TFLOPS (FP16) | 32 TOPS (INT8)
$699

Multiple developer kits - Same software

Full specs at developer.nvidia.com/jetson

Lesson Plan

Artificial Intelligence

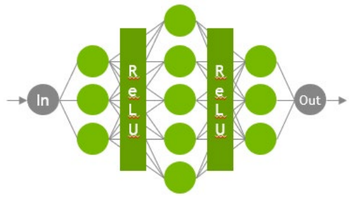
- Introduction to Recent Developments in Artificial Intelligence
- Machine Learning, Deep Learning, and Reinforcement Learning
- Statistics and Machine Learning Tools
- RL Agents: Markov Decision Processes
- Q-Learning and Deep Q-Network (DQN) Agents
- Actor-Critic Approach and Policy Gradient
- Supervised Artificial Neural Networks
- Unsupervised Artificial Neural Networks
- Workflow Application in Deep Learning
- Object Detection
- Image Classification
- Transfer Learning
- Pose Estimation
- Case Studies and its Walkthrough

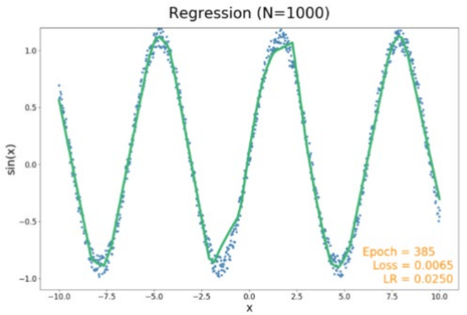
Modules Overview

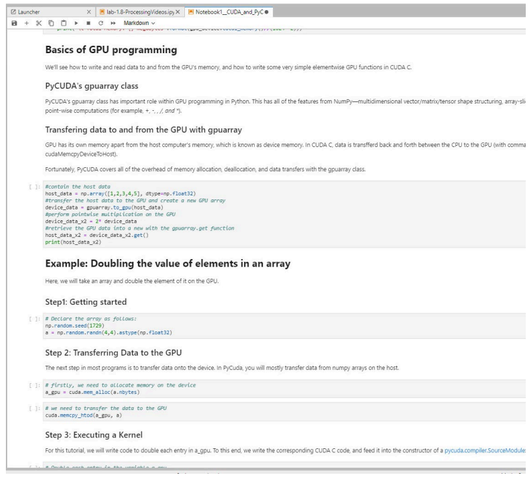
Introduction to Recent Developments in Artificial Intelligence

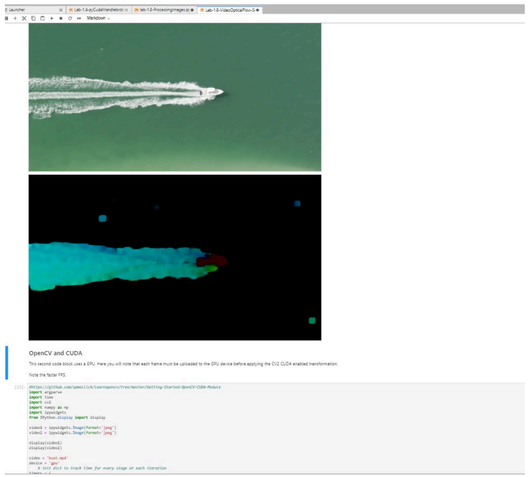
1.1: Teaching Kit Modules Overview
1.2: AI Introduction
1.3: AI Hardware
1.4: Introduction to Jetson Nano
1.5: Edge, AIoT, Big Data

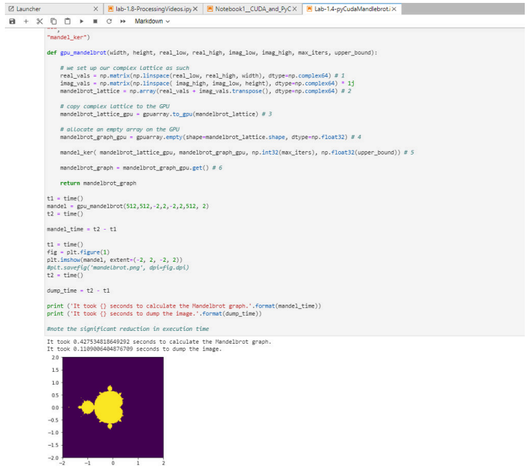
Lab: Mandelbrot CPU vs. GPU
Lab: Parallel Programming with CUDA
Lab: Image and Video Processing
Lab: Prediction and Classification using Machine Learning

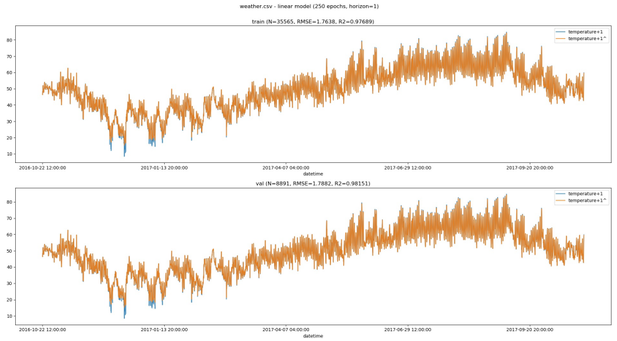












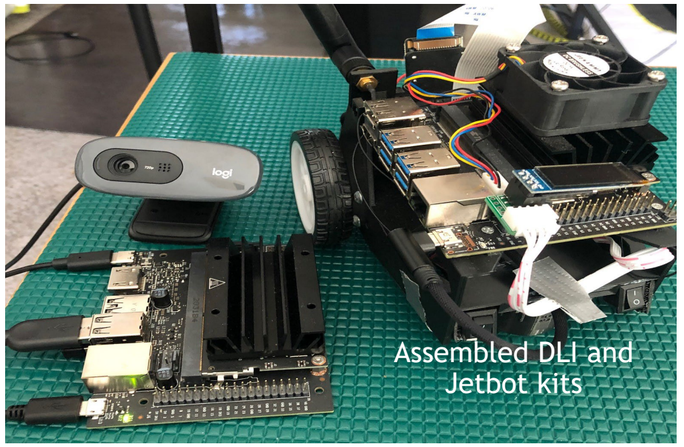
Teaching Kit Modules Overview

Introduction to Jetson Nano

- Jetson Nano 2 GB Developers Kit at core of content
- Sparkfun DLI Add on Kit https://www.sparkfun.com/products/16389
- Option to use Jetbot Kits https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetbot-ai-robot-kit/
- Add on kits provides essential accessories such as camera, power cables, SD cards as an all-in-one kit rather than purchasing each item separately
- Most kits supply 32GB SD cards but consider purchasing 64GB or higher SD cards

Sparkfun DLI kit

Waveshare Jetbot kit



Assembled DLI and Jetbot kits

Teaching Kit Module Overview

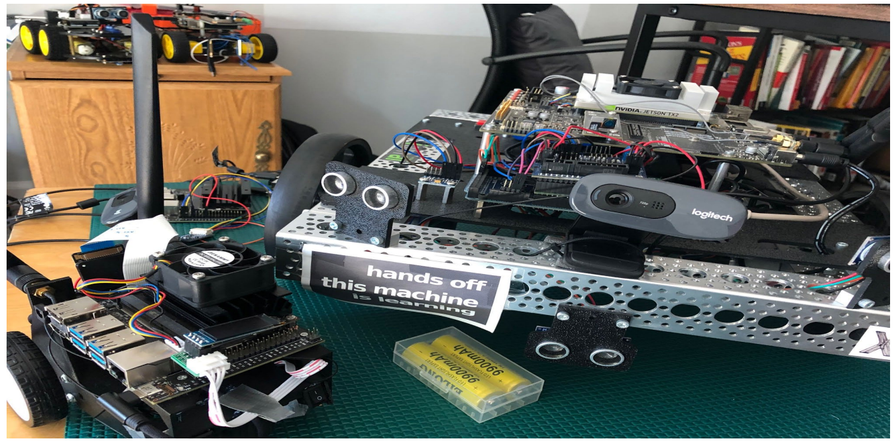
Jetson Hardware Differences from first Teaching Kit

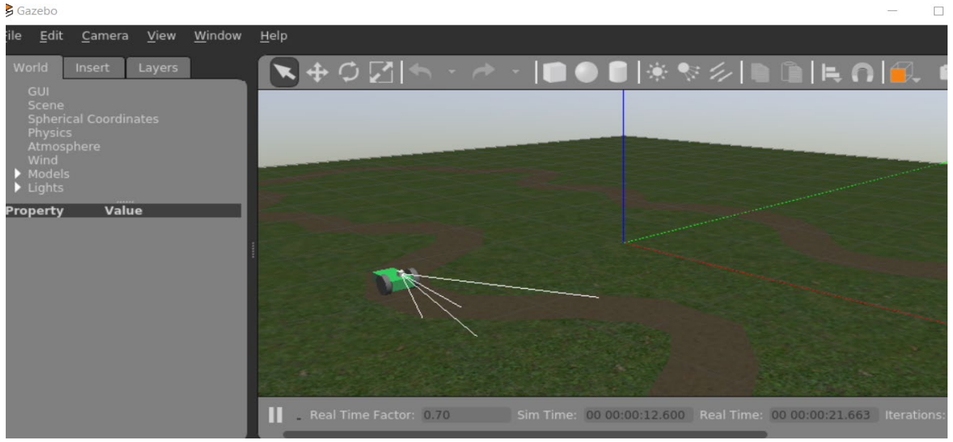
Affordable, minimum requirement is Jetson Nano 2GB plus accessories (camera, cables, SD card)

Reduced hardware build and software configuration:

- Less than 30 minutes for minimum Jetson DLI kit
- Less than 2 hours for Jetbot
- Headless connection from host to Jetson
- SD images and NVIDIA prebuilt containers

Option to work with virtual Jetbot





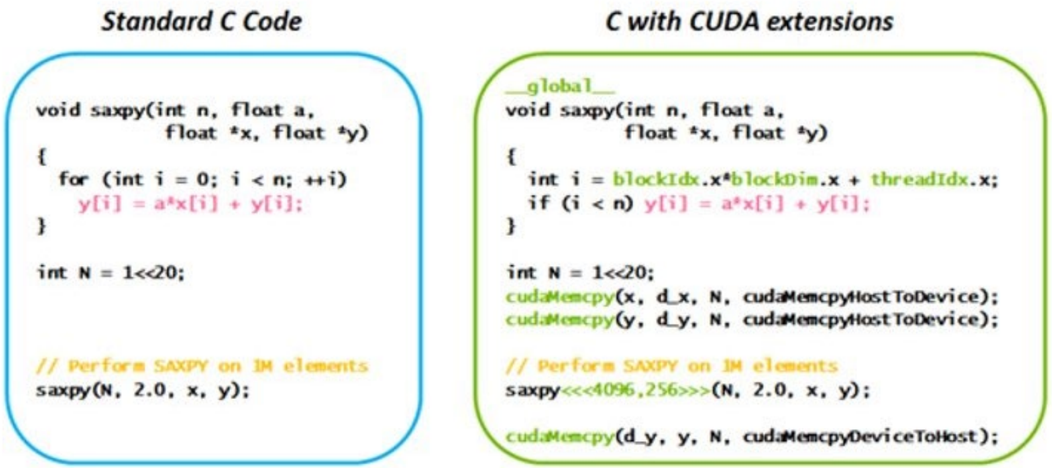
CUDA

Kernels: CUDA kernels are C functions that get executed in parallel on GPU.

Threads: Kernels execute as a set parallel threads.

Blocks: A group of threads that can execute task in parallel.

Grids: CUDA blocks are grouped into grids with each block having same number of threads.



A CUDA kernel is executed by an array of threads.

- All threads run the same code
- Each thread has an ID that it uses to compute memory addresses and make control decisions

CUDA accelerated libraries

JetPack SDK

Code in Action

Introduction to CUDA and PyCUDA

PyCUDA gives you easy, Pythonic access to Nvidia's CUDA parallel computation API.

- Abstractions make CUDA programming easier
- Full power of CUDA API if needed
- Automatic error checking and cleanup

Initialization

A few modules have to be loaded to initialize communication to the GPU:

- import pycuda.driver as cuda
- import pycuda.autoinit

The pycuda.driver module has methods that:

1. Allocate memory on the GPU (cuda.mem alloc())
2. Copy numpy arrays to the GPU (cuda.memcpy htod())
3. Execute kernels on the GPU described by CUDA code (see compiler.SourceModule)
4. Copy data from the GPU back to numpy arrays (cuda.memcpy dtoh()).

[ ]:
```
# Import PyCUDA and several modules associated with the PyCUDA
!pip install pycuda
import pycuda
import pycuda.driver as cuda
cuda.init()

import pycuda.autoinit
from pycuda.compiler import SourceModule
import pycuda.gpuarray as gpuarray
from pycuda.curandom import rand as curand
from pycuda.elementwise import ElementwiseKernel

import numpy as np
import numpy.linalg as la
```

CUDA device query

A GPU query is a very basic operation that will tell us the specific technical details of our GPU, such as available GPU memory and core count.

[ ]:
```
print('CUDA device query (PyCUDA version) \n')
print('Detected {} CUDA Capable device(s) \n'.format(cuda.Device.count()))
for i in range(cuda.Device.count()):

    gpu_device = cuda.Device(i)
    print('Device {}: {}'.format( i, gpu_device.name() ) )
    compute_capability = float( '%d.%d' % gpu_device.compute_capability() )
    print('\t Compute Capability: {}'.format(compute_capability))
    print('\t Total Memory: {} megabytes'.format(gpu_device.total_memory()//(1024**2)))
```

Basics of GPU programming

We'll see how to write and read data to and from the GPU's memory, and how to write some very simple elementwise GPU functions in CUDA C.

Introduction to CUDA Python with Numba

CONCEPT 1: What is Numba?

Numba is a **just-in-time, type-specializing, function compiler** for accelerating **numerically-focused** Python. Let's break down those terms:

- **function compiler:** Numba compiles Python functions, not entire applications, and not parts of functions. Numba does not replace your Python interpreter, but is just another Python module that can turn a function into a (usually) faster function.
- **type-specializing:** Numba speeds up your function by generating a specialized implementation for the specific data types you are using. Python functions are designed to operate on generic data types, which makes them very flexible, but also very slow. In practice, you only will call a function with a small number of argument types, so Numba will generate a fast implementation for each set of types.
- **just-in-time:** Numba translates functions when they are first called. This ensures the compiler knows what argument types you will be using. This also allows Numba to be used interactively in a Jupyter notebook just as easily as a traditional application
- **numerically-focused:** Currently, Numba is focused on numerical data types, like int, float, and complex. There is very limited string processing support, and many string use cases are not going to work well on the GPU. To get best results with Numba, you will likely be using NumPy arrays.

The most common way to use Numba is through its collection of decorators that can be applied to your functions to instruct Numba to compile them. When a call is made to a Numba decorated function it is compiled to machine code "just-in-time" for execution and all or part of your code can subsequently run at native machine code speed!

Numba supports CUDA GPU programming by directly compiling a restricted subset of Python code into CUDA kernels and device functions following the CUDA execution model. Kernels written in Numba appear to have direct access to NumPy arrays. NumPy arrays are transferred between the CPU and the GPU automatically.

Documentation

Terminology

- **host**: the CPU
- **device**: the GPU
- **host memory**: the system main memory
- **device memory**: onboard memory on a GPU card
- **kernels**: a GPU function launched by the host and executed on the device
- **device function**: a GPU function executed on the device which can only be called from the device (i.e. from a kernel or another device function)

Most CUDA programming facilities exposed by Numba map directly to the CUDA C language offered by NVIDIA. Therefore, it is recommended you read the official CUDA C programming guide.

CONCEPT 2: How does Numba work?

Numba reads the Python bytecode for a decorated function and combines this with information about the types of the input arguments to the function. It analyzes and optimizes your code, and finally uses the LLVM compiler library to generate a machine code version of your function, tailored to your CPU capabilities. This compiled version is then used every time your function is called.

Other things of interest:

Numba has quite a few decorators:

- `@jit` - compile the decorated function on-the-fly to produce efficient machine code. Docs are here
- `@njit` - this is an alias for `@jit(nopython=True)` as it is so commonly used!
- `@vectorize` - produces NumPy `ufunc` s (with all the `ufunc` methods supported). Docs are here.
- `@guvectorize` - produces NumPy generalized `ufunc` s. Docs are here.
- `@stencil` - declare a function as a kernel for a stencil like operation. Docs are here.
- `@jitclass` - for jit aware classes. Docs are here.
- `@cfunc` - declare a function for use as a native call back (to be called from C/C++ etc). Docs are here.
- `@overload` - register your own implementation of a function for use in nopython mode, e.g. `@overload(scipy.special.j0)` . Docs are here.

Introduction to Parallel Programming

Traditionally, computers were only focused on computer programs serially. These programs were broken into series instructions that are executed on a single processor sequentially. In contrast with parallel programming, you can break down the computer programs into the sub-programs which will be executed on multiple processor in parallel.

CUDA® is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs.

The CUDA Toolkit from NVIDIA provides everything you need to develop GPU-accelerated applications. The CUDA Toolkit includes GPU-accelerated libraries, a compiler, development tools and the CUDA runtime.

Getting started

In this tutorial, we will perform some operations with traditional computing and parallel computing.

[ ]:
```
# import libraries
import numpy as np
from numpy import testing
from numba import cuda, types
```

Array summation: Traditional computing

In this section, we will create two arrays using NumPy, we will sum them up and check the time.

[ ]:
```
# declare an array
n = 2048
x = np.arange(n).astype(np.int32)
y = np.ones_like(x)
z = np.zeros_like(x)
```

[ ]:
```
# print out the result
print(x)
print(y)
```

[ ]:
```
# create a function that will sum up the arrays
def add_arr(x, y, z):
    for idx in range(0,len(x)):
        z[idx] = x[idx] + y[idx]
    return z
```

[ ]:
```
# print out time
%time
z = add_arr(x, y, z)
z
```

Array summation: Parallel computing

Support for NumPy arrays is a key focus of Numba development and is currently undergoing extensive refactorization and improvement. Most capabilities of NumPy arrays are supported by Numba in object mode, and a few features are supported in nopython mode too (with much more to come).

A few noteworthy limitations of arrays at this time:

- Arrays can be passed in to a function in nopython mode, but not returned. Arrays can only be returned in object mode.
- New arrays can only be created in object mode.
- Currently there are no bounds checking for array indexing and slicing, although negative indices will wrap around correctly.
- A small number of NumPy array ufuncs are only supported in object mode, but the vast majority work in nopython mode.
- Array slicing only works in object mode.

Performance best practices

Array operations with GPUs can provide considerable speedups over CPU computing.

CuPy is an open-source array library for GPU-accelerated computing with Python. CuPy utilizes CUDA Toolkit libraries including cuBLAS, cuRAND, cuSOLVER, cuSPARSE, cuFFT, cuDNN and NCCL to make full use of the GPU architecture.

- Most operations perform well on a GPU using CuPy. CuPy speeds up some operations more than 100X.
- CuPy's interface is highly compatible with NumPy. CuPy supports various methods, indexing, data types, broadcasting and more. This comparison table shows a list of NumPy and their corresponding CuPy implementations.

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Benchmarking speed- NumPy vs CuPy

In this tutorial, we will perform some opeations using NumPy and CuPy library and we will benchmark the time.

[ ]:
```
# import libraries
import numpy as np
import cupy
```

Let's start with creating an array using NumPy and CuPy and compare the time.

Creating an array with NumPy

[ ]:
```
%%time
np_var1 = np.random.random((10000, 10000))
np_var2 = np.random.random((10000, 10000))
```

Creating and array with CuPy

[ ]:
```
%%time
cp_var1 = cupy.random.random((10000, 10000))
cp_var2 = cupy.random.random((10000, 10000))
cupy.cuda.Stream.null.synchronize()
```

[ ]:
```
# print the numpy array
np_var1
```

[ ]:
```
# print the cupy array
cp_var1
```

Trigonometric function

Let's do some trigonometric operations and compare the time.

[ ]:
```
%timeit bool((np.sin(np_var1) ** 2 + np.cos(np_var1) ** 2 == 1).all())
```

[ ]:
```
%timeit bool((cupy.sin(cp_var2) ** 2 + cupy.cos(cp_var2) ** 2 == 1).all())
```

ML Fundamentals

Learning Objectives

- Define artificial intelligence
- Define machine learning
- Define deep learning
- Define neural network and types of layers
- Explain convolution neural network
- Explain types of artificial neural networks

Code in Action

- **Weather Forecasting**
- **Solar Power Prediction**
- **Space Shuttle Classification**

pytorch-timeseries

Time-series forecasting and prediction on tabular data using PyTorch. Supports Jetson Nano, TX1/TX2, AGX Xavier, and Xavier NX.

Starting the Container

```
$ git clone https://github.com/dusty-nv/pytorch-timeseries
$ cd pytorch-timeseries
$ docker/run.sh
$ cd pytorch-timeseries
```

Weather Forecasting

In this hourly weather dataset, we are forecasting the future weather of a city based on it's past weather. The following data is provided on 1-hour intervals (over a timespan of 40,000 hours):

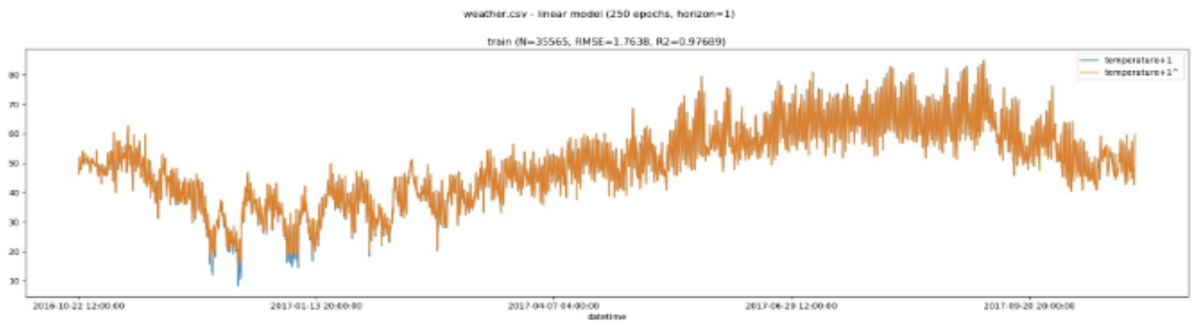
```
temperature
humidity
pressure
wind_direction
wind_speed
```

Let's start by forecasting just the temperature:

```
$ python3 train.py --data data/weather.csv --inputs temperature --outputs temperature --horizon 1

train RMSE: [1.7638184641368702]
train R2:   [0.9768890819461414]

val RMSE:   [1.788160646675811]
val R2:     [0.9815134518452414]
```

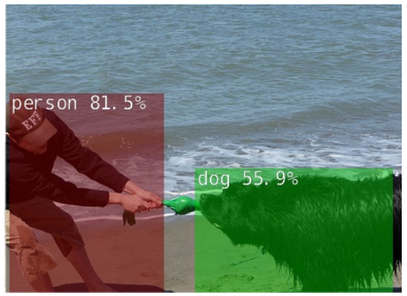


Teaching Kit Modules Overview

Vision Deep Neural Networks (DNNs)

2.1: Image Classification
2.2: Object Detection
2.3: Semantic Segmentation
2.4: Pose Estimation
Lab: Hello AI World

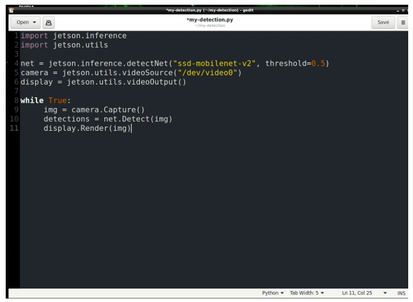
Image Classification



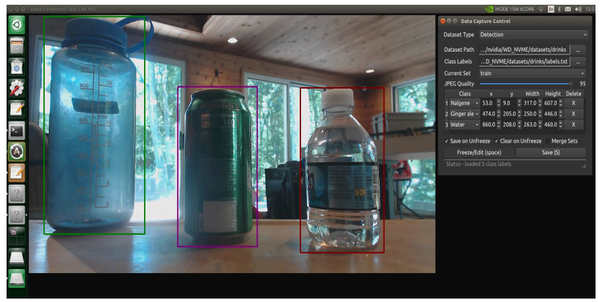
Object Detection

Semantic Segmentation

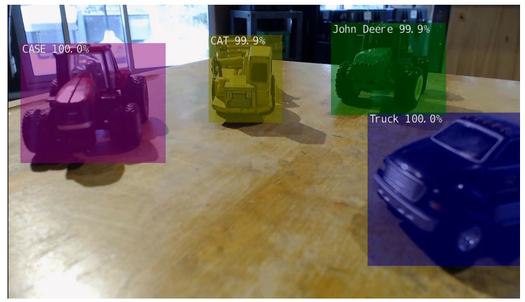
Pose Estimation



Code Your Own Programs
(Python or C++)



Collect Your Own Datasets



Train Your Own Models

Image Classification

Learning Objectives

- Define image classification
- Explain challenges in image classification
- Describe different dataset for image classification
- Describe image classification architectures
- Discuss performance and accuracy benchmarks

Objective:

Recognize the main subject in the image and assign a label to the image

Datasets for Image Classification

Architectures

- LeNet
- AlexNet
- VGG-16
- Inception V1
- ResNet-50
- Xception
- MobileNet

Object detection

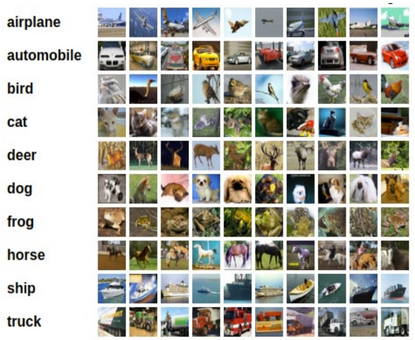
Learning Objectives

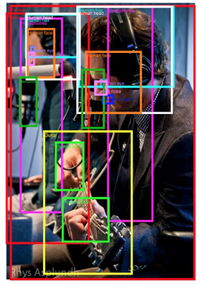
- Define object detection
- Describe different datasets for object detection
- Describe object detection architectures
- Discuss performance benchmarks

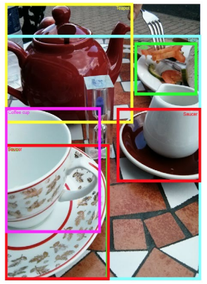
Objective:

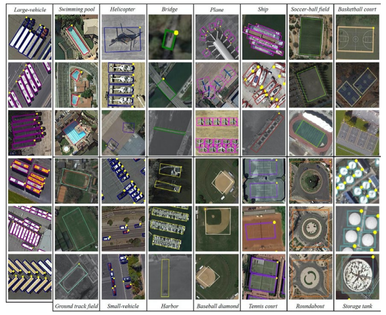
Predict the class and the position of multiple objects in the image

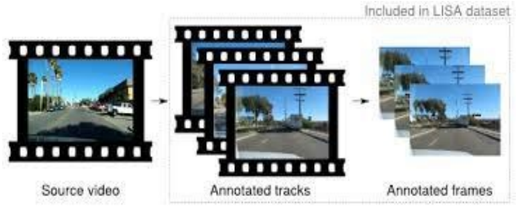
Datasets for object detection











Architectures

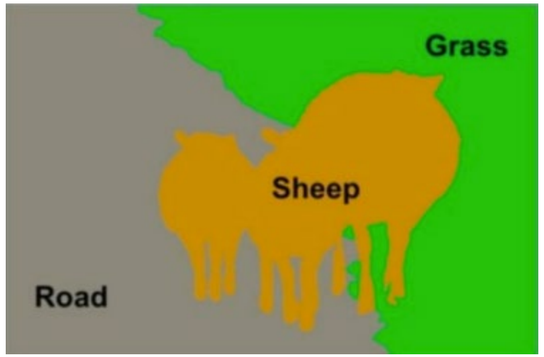
- Sliding Window
- R-CNN
- Fast R-CNN
- Faster R-CNN
- YOLO
- SSD
- Transformers

Semantic Segmentation

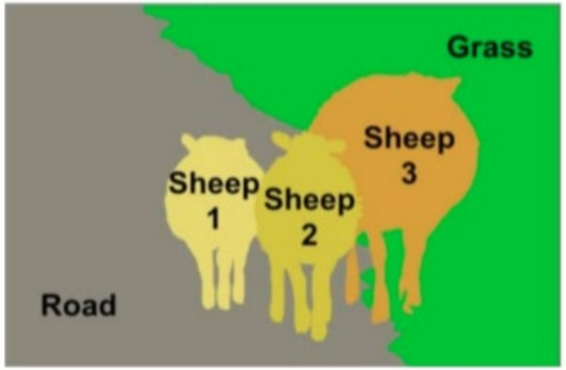
Learning Objectives

- Define image segmentation
- Define semantic segmentation
- Define instance segmentation
- Describe image segmentation architectures
- Learn about popular datasets and applications
- Discuss about performance

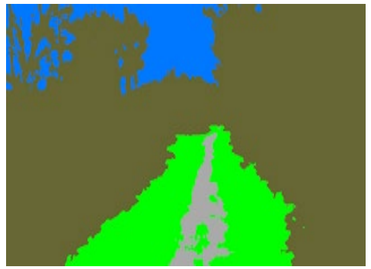
Semantic Segmentation Objective:
Classify all pixels
Don't differentiate instances.



Instance Segmentation Objective:
Classify pixels but differentiate instances (predict segmentation mask)



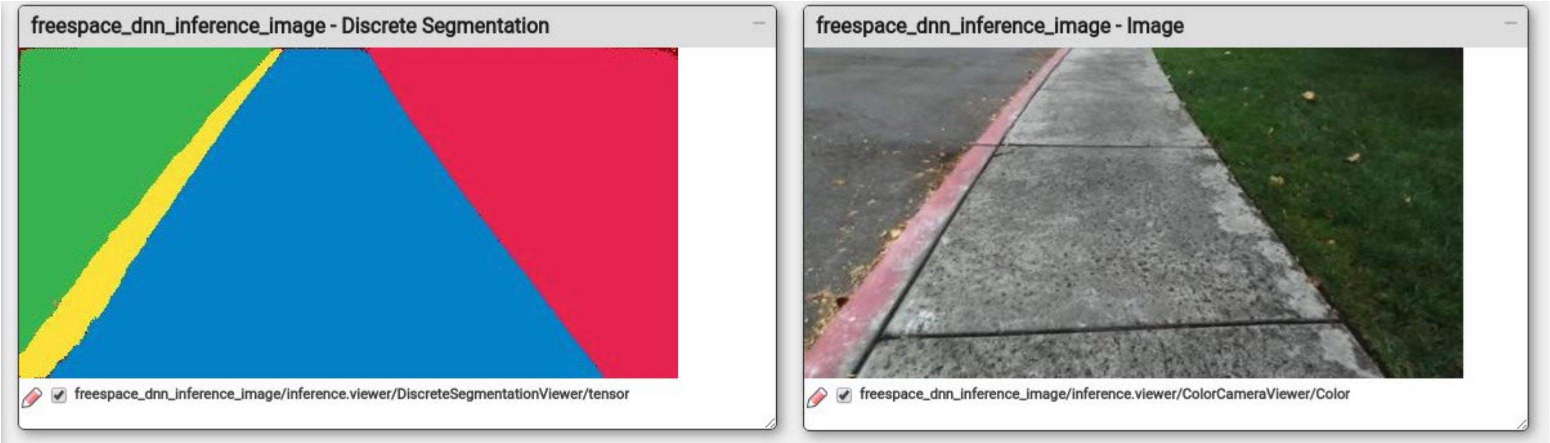
Popular Datasets



Architectures

- Fully Convolutional Networks
- SegNet
- U-Net
- RefineNet
- Learning Deconvolution Network
- Mark R-CNN

Applications: Free Space Detection

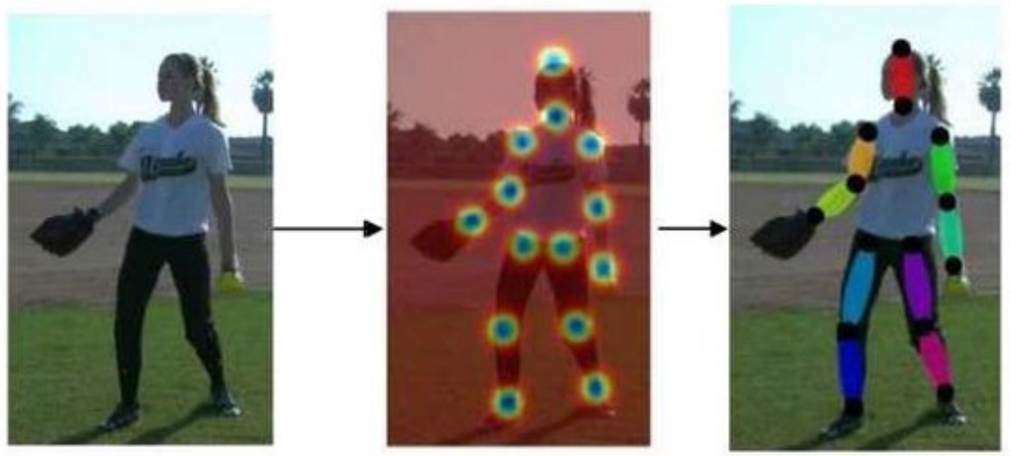


Pose Estimation

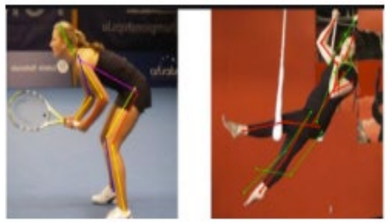
Learning Objectives

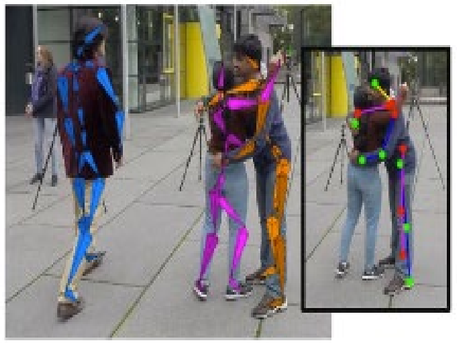
- Define pose estimation
- Describe pose estimation architectures
- Learn about popular datasets and applications
- Discuss about performance

Objective:
Predict the pose of a person/object from an image or a video



Datasets for Pose Estimation



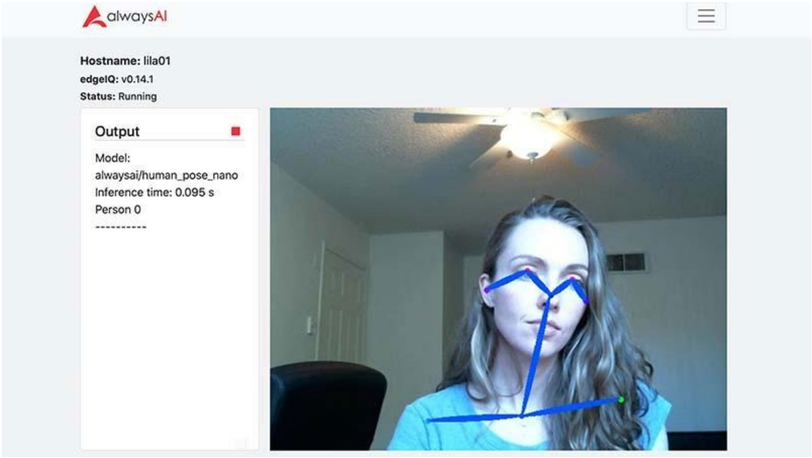


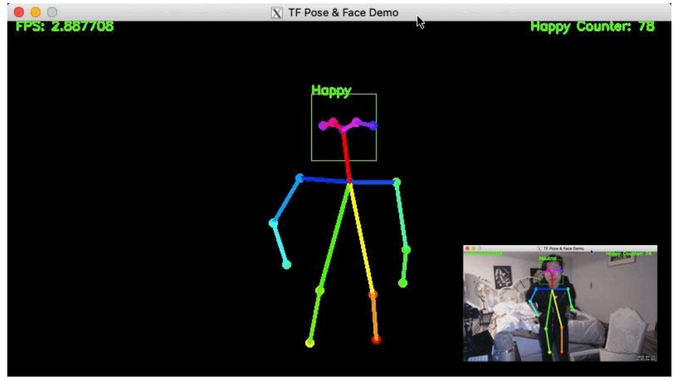
List of architectures models

- OpenPose
- High Resolution Net (HRNet)
- DeepCut
- Regional Multi-Person Pose Estimation (AlphaPose)
- Deep Pose:
- PoseNet:
- DensePose

Applications

- Gesture Recognition
- Virtual Reality
- Activity Recognition
- Posture Analysis

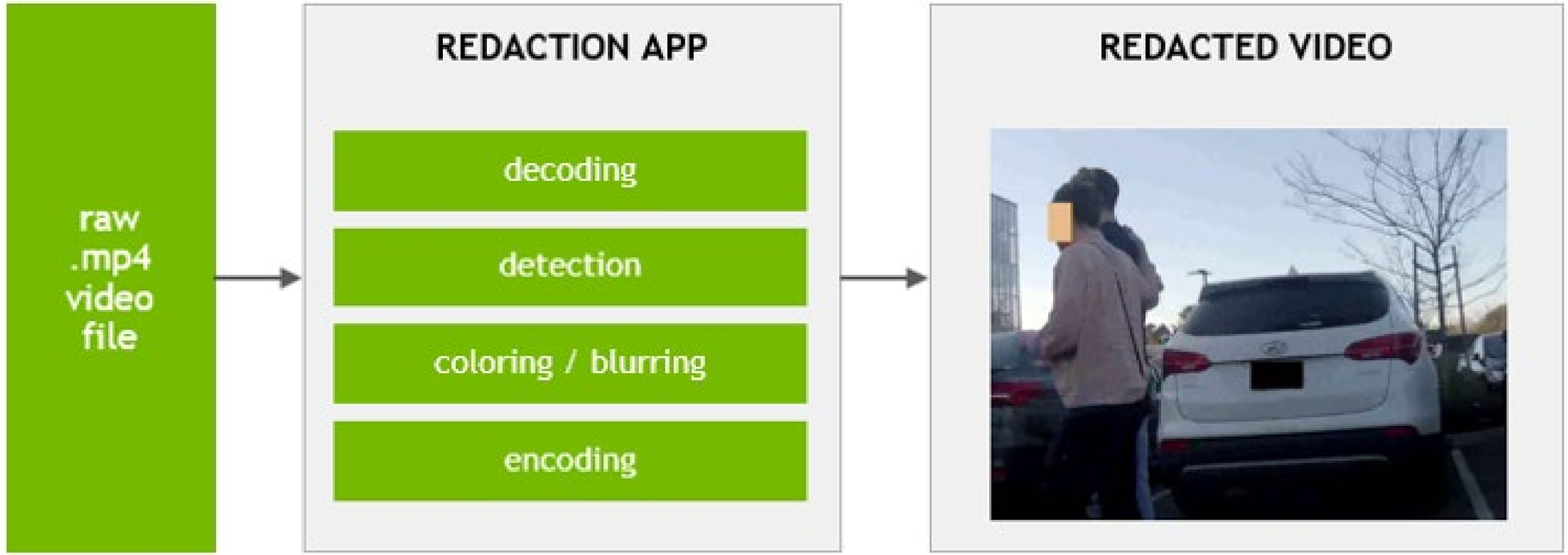




Teaching Kit Modules Overview

Diversity, Ethics, and Security

3.1: Diversity and Bias in Datasets
3.2: Ethics and Security
Lab: Image and Video Redaction



Teaching Kit Module Overview

Module 3 - Diversity, Bias and Security

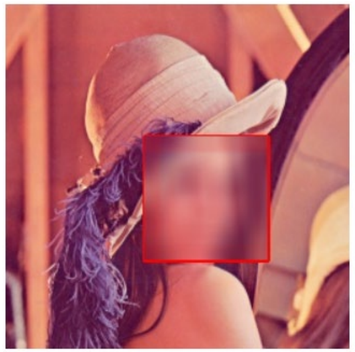
Outlines issues of Data Bias and ways to solve

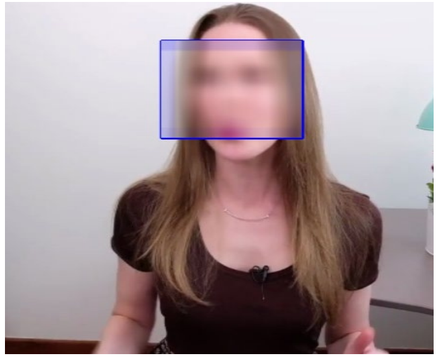
Outlines principles of Responsible AI and importance of transparency

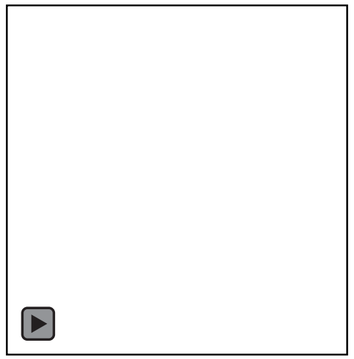
Discusses ethics and influences of AI

Highlights security considerations and data governance

Provides hands on example using OpenCV CUDA libraries and CUDA enabled haar cascades algorithms to detect and redact faces from images and video



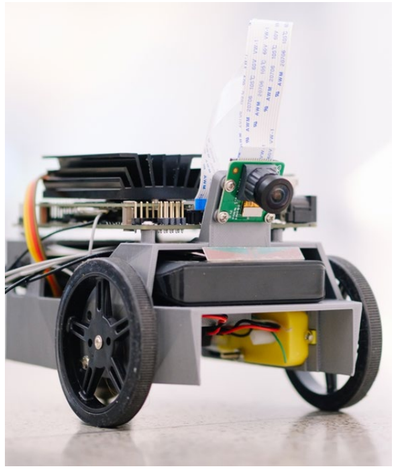


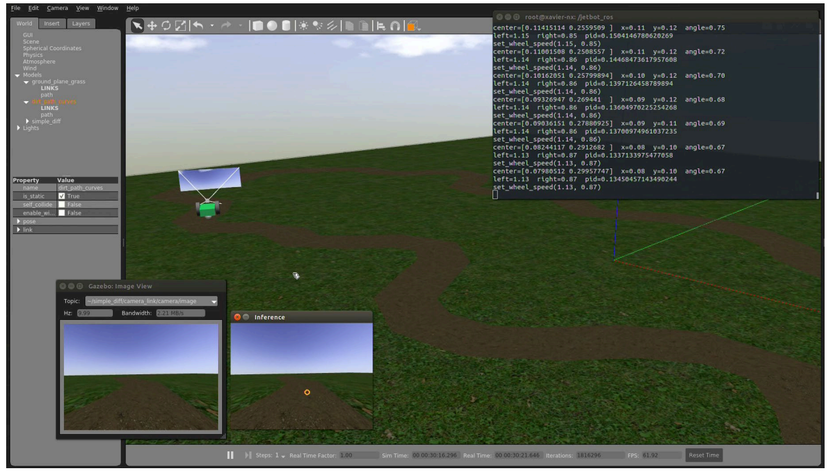


Teaching Kit Modules Overview

Module 4:
Autonomous Robotics

4.1: Agent Architectures
4.2: Teaching Machines to Act
4.3: Introduction to ROS2
4.4: Robot Navigation
Lab: Autonomous JetBot Navigation with ROS and Gazebo



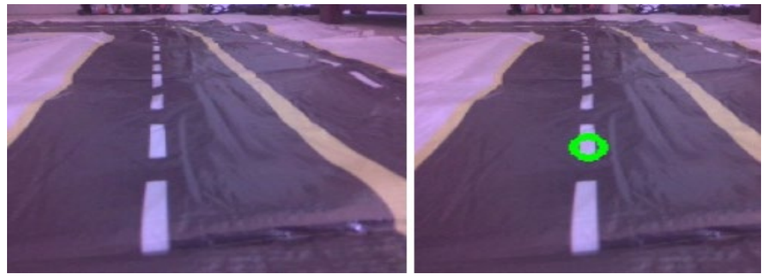


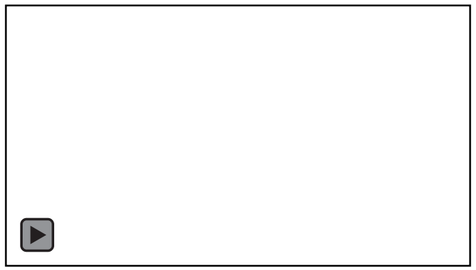
Teaching Kit Module Overview

Module 4 Autonomous Vehicles

Introduces the JetBot Getting Started Kit and covers essentials of:

- Robot as an Intelligent Agent
- Control and training machine to act
- Collects image data and trains to avoid obstacles and follow identified path
- Leverages Pytorch to train data and Tensort to optimize





Teaching Kit Overview

Module 4 ROS2 and Gazebo

The virtual JetBot uses both Robot Operating System 2 (ROS2) and Gazebo

Leverages the prebuilt NVIDIA Jetbot_Ros container

Provides an overview of the ROS2 commands

Lab exercises cover obstacle avoidance and road following using ROS2 and Gazebo

Launch Gazebo

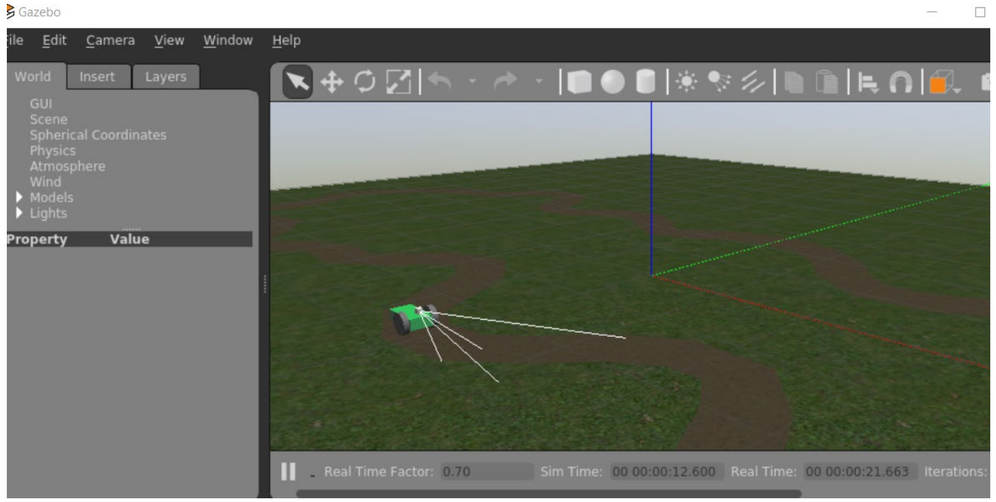
```
ros2 launch jetbot_ros gazebo_world.launch.py
```

Then to run the following commands, launch a new terminal session into the container:

```
sudo docker exec -it jetbot_ros /bin/bash
```

Test Teleop

```
ros2 launch jetbot_ros teleop_keyboard.launch.py
```

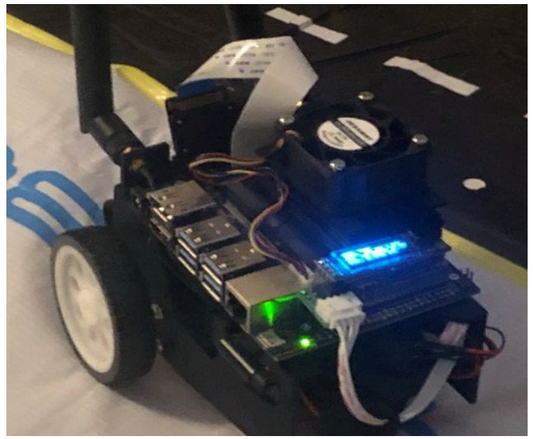


Teaching Kit Tips

Feedback from the classroom

UMBC Data 690 Applied AI for Data Science Practitioners

- Early content
- Spring 2021 Virtual Course
- Fall 2021 In Person Course



Challenges:

Linux may be new to many students

Difficult to troubleshoot hardware remotely

Streaming video and images requires extra setup

Software ecosystem is still emerging

Tips:

Use containers help reduce setup time

Allow extra time for hardware setup

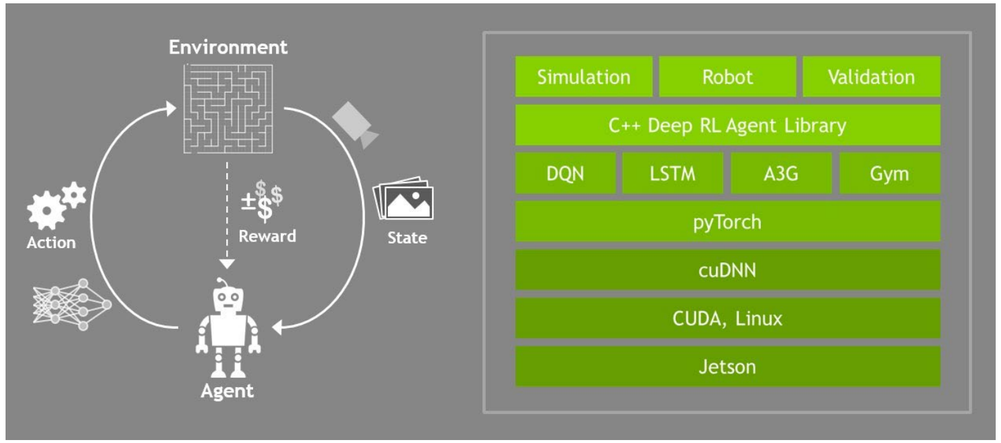
Have students demonstrate in person

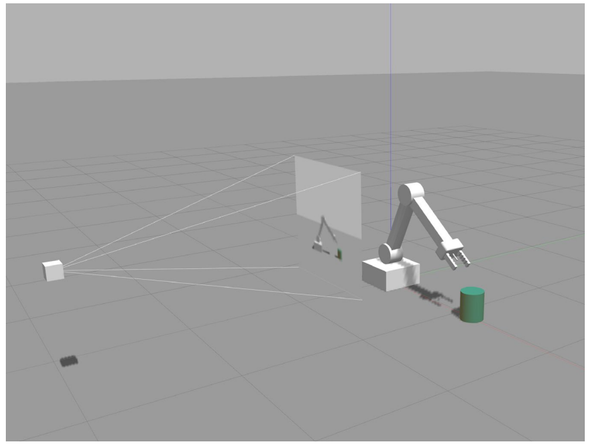
Have spare Jetson kits and accessories on hand (SD cards, cables, Nanos)

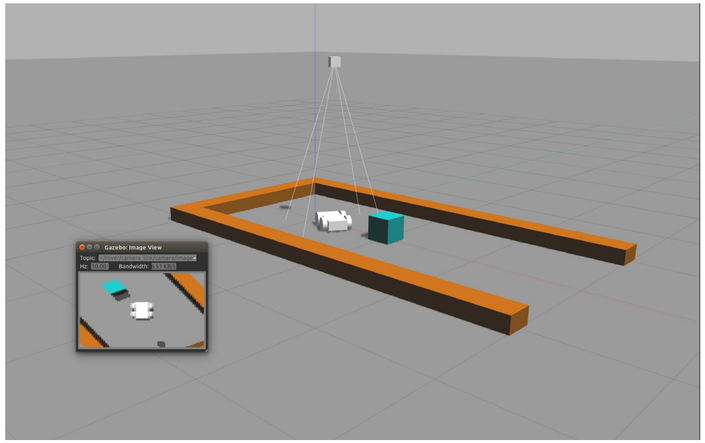
```
git clone https://github.com/dusty-nv/jetbot_ros
cd jetbot_ros
docker/run.sh
```

Teaching Kit Modules Overview

Reinforcement Learning	5.1: Introduction to RL Agents 5.2: Deep RL for Vision **Lab: OpenAI Gym (Deep Q-Network)** **Lab: Robotic Arm Manipulation in Gazebo**

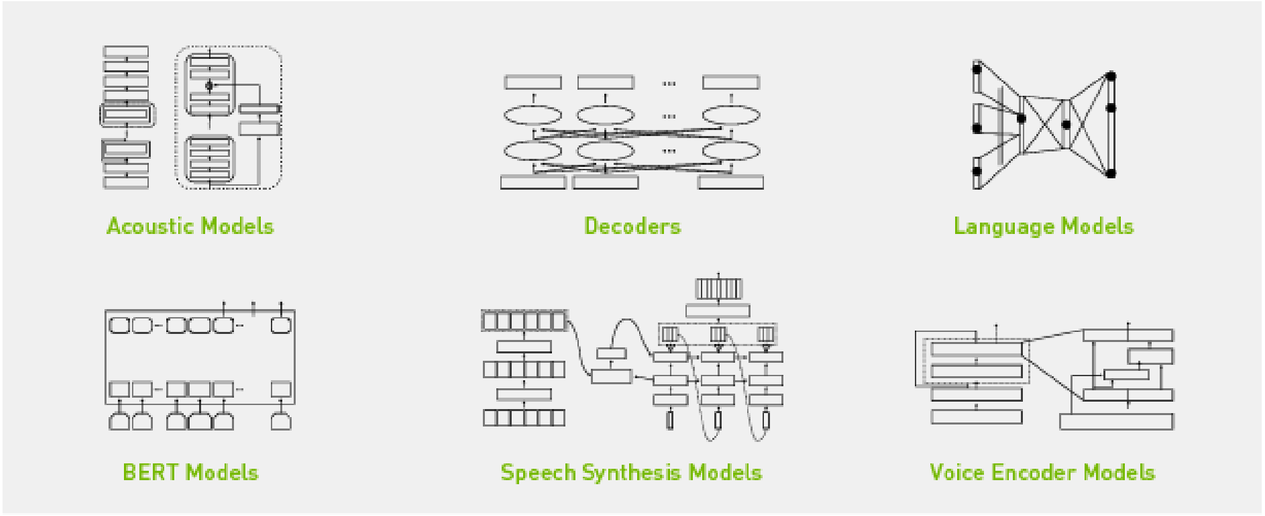


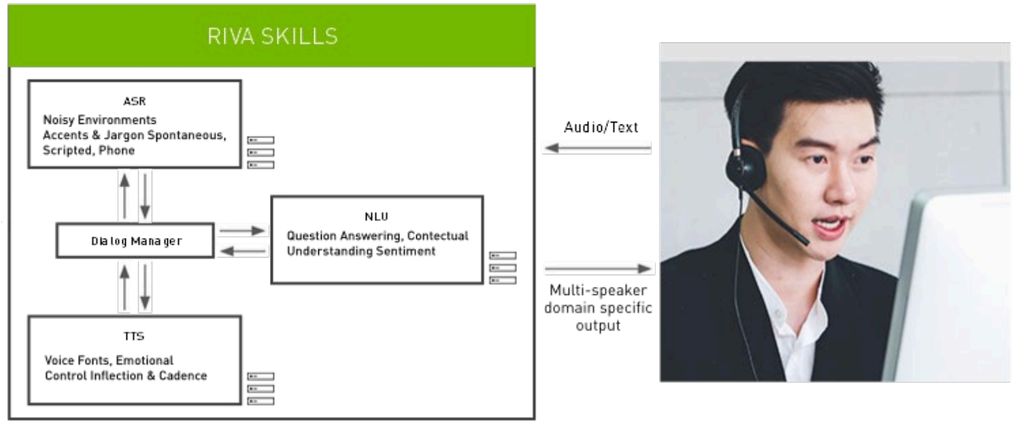




Teaching Kit Modules Overview *(For Future Release)*

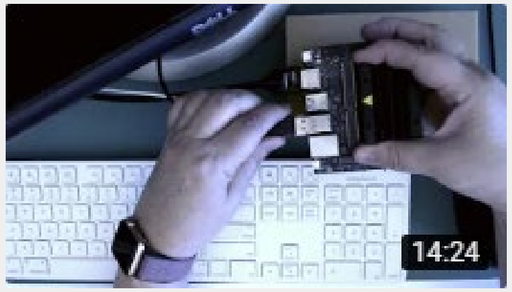
Module 6: Natural Language Conversational AI	6.1: Natural Language Processing (NLP) 6.2: Speech Recognition (ASR) 6.3: Text-to-Speech (TTS) **Lab: Python NLP** **Lab: ASR, NLP, TTS Chatbot**



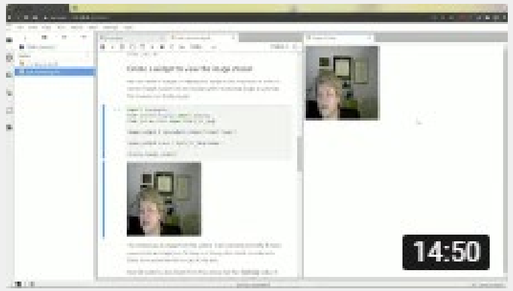


AI Fundamentals

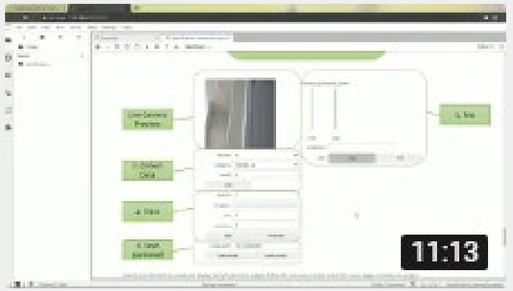
Getting Started with AI on Jetson Nano



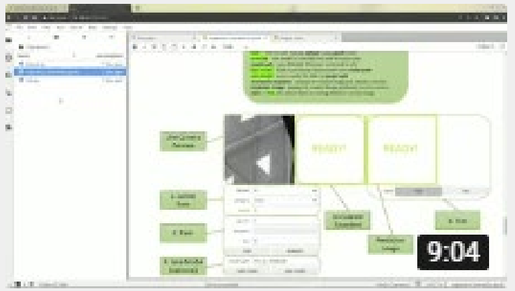
System Setup



Camera Capture



Classification



Regression

Hello AI World

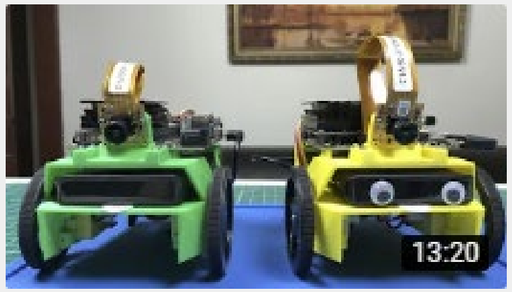
Image Recognition

Object Detection

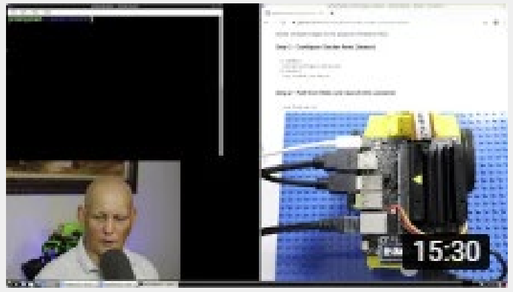
Data Collection + Training

Semantic Segmentation

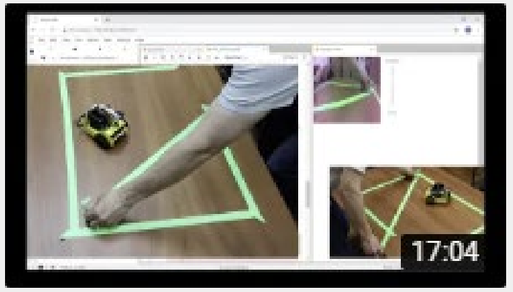
JetBot



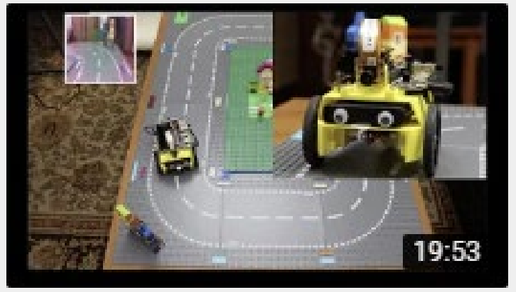
Hardware Setup



Software Setup



Collision Avoidance



Road Following