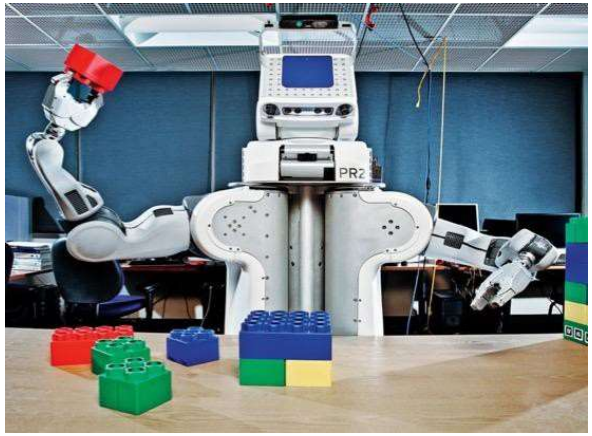
Lightweight Deep Learning on Smart Sensors for Real-Time Data Collection, Analysis, Decision Support and Control

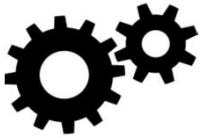
Dantong Yu, Associate Professor
Graduate Program Director
School of Management
New Jersey Institute of Technology

Outline

- Motivation Use Cases
- End-to-End Machine Learning/Deep Learning from Data Generation to Decision Making
- Proposed Smart Sensor Technology
  - Software
  - Hardware
- Use Case 1: NSLS-II Real-Time Experiment Prediction on Embedded System.
- Use Case 2: Skin Cancer Detection on Jetson TX2 and Raspberry Pi.
- Future Works

Motivation





Predictive Operations/ Maintenance

Infrastructure inspection

Predictive maintenance

Physical security

Intelligent Factory

Pick and place

Complex/custom tasks

Visual inspection

Task consolidation

Dynamic reconfiguration

Collaborative robotics

Efficiency optimization

Factory simulation

End-To-End Machine Learning Workflow

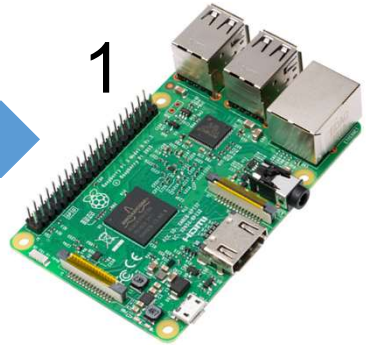
Stage 1: Off-line Training

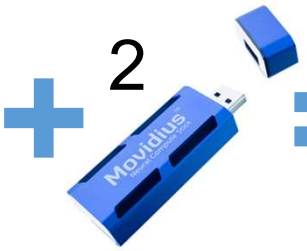
Stage 2: On-line Training /Transfer Learning

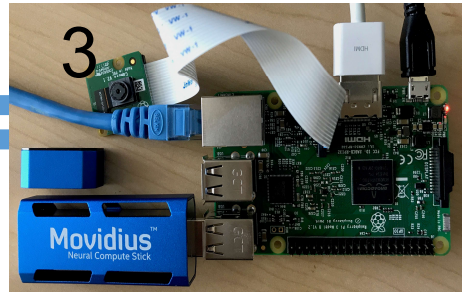
Stage 3: Real-Time Inference

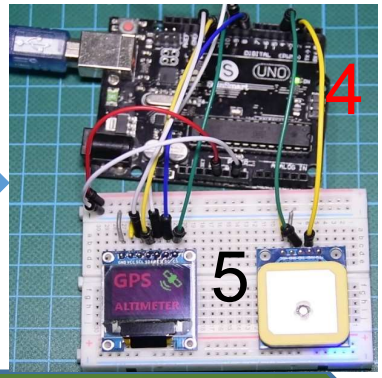
Stage 4: Light-Weight Real-Time Deep Learning & Inference Engine

Stage 5: Real-Time Sensing & Decision Making Engine









High Performance Computing — Edge Computing — Smart Sensors — Internet of Things (Sensing and Controlling)

Electricity Cost High — Electricity Cost Medium — Electricity Cost Low

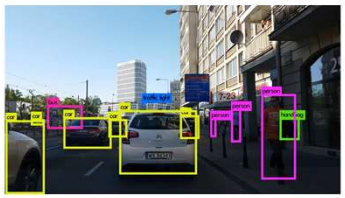
NVIDIA Jetson TX2
Supercomputer with a small footprint

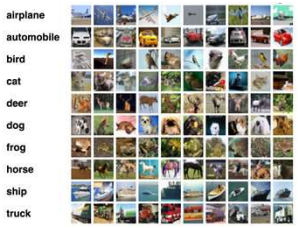
- Hardware
- Jetson TX2 (200 GPU cores =2 TFPS)
- Carrier Board (1Gb Ethernet, USB 3.0, HDMI, Wireless AP)
- Box container
- Portable (3.75” × 2.48” × 2.29”, Fits in a palm)
- Power efficient (15W maximum)
- 50000mAh Recharged Battery (12V/3A) lasts more 24 hours
- Complimentary by Solar Charger for 24 × 7 operations anywhere

Layered NVIDIA Jetson TX2 Software Ecosystems

Software Hierarchy from Lower to Higher Layers

Deep Learning Apps





Video Prediction

Experiment Data Analysis and Decision making

jupyter

Machine Learning & Deep Networks

TensorFlow

PYTORCH

Caffe2

scikits learn

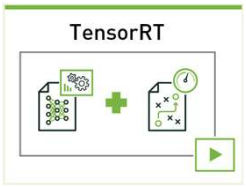
Keras

Critical Libraries

cuDNN

CUDA library

TensorRT



OpenCV Python

Video API

python

Operating Systems

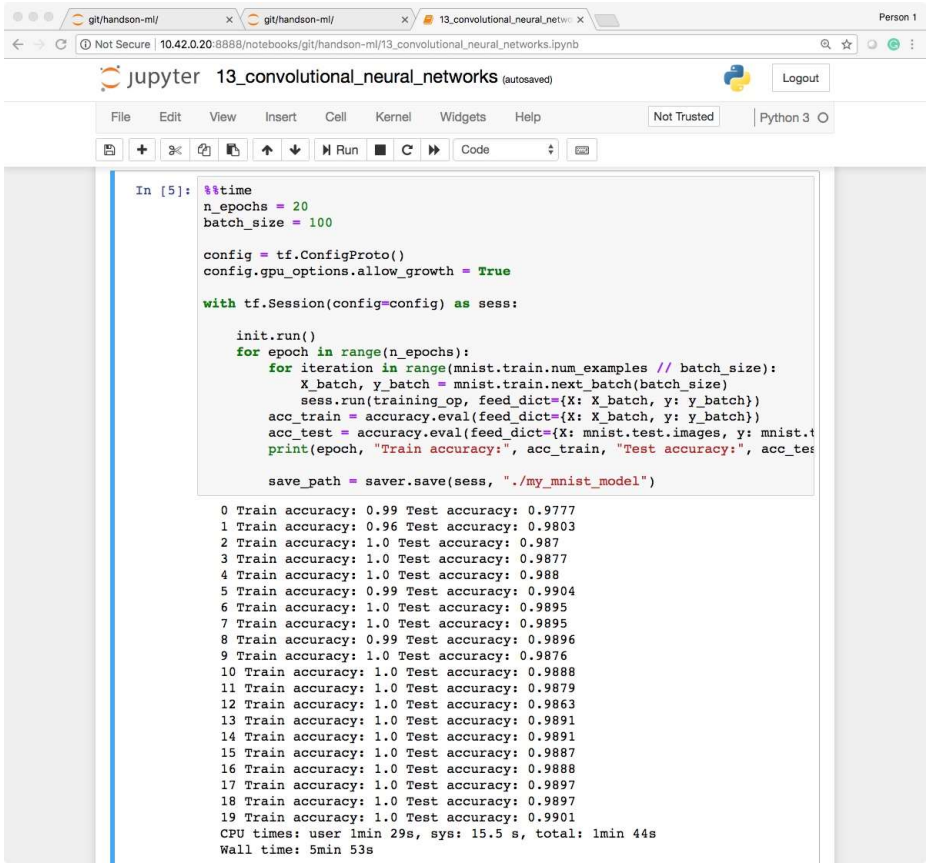
Carrier Board Support Package (BSP, USB, PCI-e, Display Port)

Hardware

Jetson TX2 Embedded System. GPU, ARM, Video CODEC

Jetson TX2 Supports Training Better Than Macbook (8 X speed up)

Jetson TX2



jupyter 13_convolutional_neural_networks (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

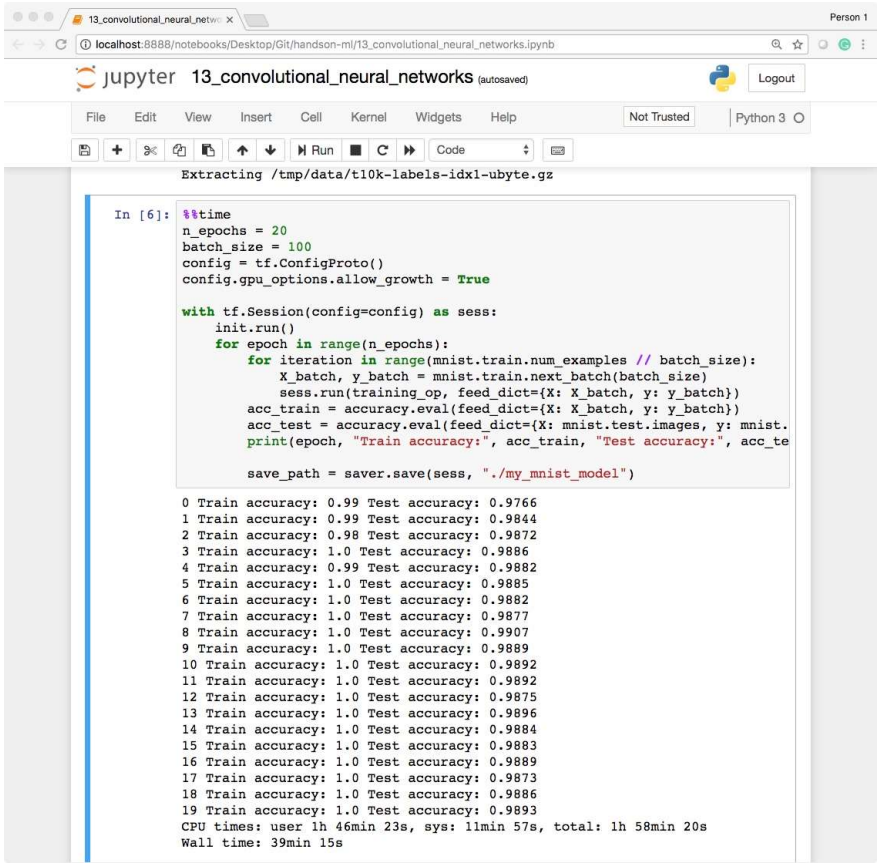
Run Code

In [5]:

```
%%time
n_epochs = 20
batch_size = 100

config = tf.ConfigProto()
config.gpu_options.allow_growth = True

with tf.Session(config=config) as sess:

    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: mnist.test.images, y: mnist.t
        print(epoch, "Train accuracy:", acc_train, "Test accuracy:", acc_tes

        save_path = saver.save(sess, "./my_mnist_model")
```

```
0 Train accuracy: 0.99 Test accuracy: 0.9777
1 Train accuracy: 0.96 Test accuracy: 0.9803
2 Train accuracy: 1.0 Test accuracy: 0.987
3 Train accuracy: 1.0 Test accuracy: 0.9877
4 Train accuracy: 1.0 Test accuracy: 0.988
5 Train accuracy: 0.99 Test accuracy: 0.9904
6 Train accuracy: 1.0 Test accuracy: 0.9895
7 Train accuracy: 1.0 Test accuracy: 0.9895
8 Train accuracy: 0.99 Test accuracy: 0.9896
9 Train accuracy: 1.0 Test accuracy: 0.9876
10 Train accuracy: 1.0 Test accuracy: 0.9888
11 Train accuracy: 1.0 Test accuracy: 0.9879
12 Train accuracy: 1.0 Test accuracy: 0.9863
13 Train accuracy: 1.0 Test accuracy: 0.9891
14 Train accuracy: 1.0 Test accuracy: 0.9891
15 Train accuracy: 1.0 Test accuracy: 0.9887
16 Train accuracy: 1.0 Test accuracy: 0.9888
17 Train accuracy: 1.0 Test accuracy: 0.9897
18 Train accuracy: 1.0 Test accuracy: 0.9897
19 Train accuracy: 1.0 Test accuracy: 0.9901
CPU times: user 1min 29s, sys: 15.5 s, total: 1min 44s
Wall time: 5min 53s
```

Macbook (2017)



jupyter 13_convolutional_neural_networks (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Code

```
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

In [6]:

```
%%time
n_epochs = 20
batch_size = 100
config = tf.ConfigProto()
config.gpu_options.allow_growth = True

with tf.Session(config=config) as sess:
    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: mnist.test.images, y: mnist.
        print(epoch, "Train accuracy:", acc_train, "Test accuracy:", acc_te

        save_path = saver.save(sess, "./my_mnist_model")
```

```
0 Train accuracy: 0.99 Test accuracy: 0.9766
1 Train accuracy: 0.99 Test accuracy: 0.9844
2 Train accuracy: 0.98 Test accuracy: 0.9872
3 Train accuracy: 1.0 Test accuracy: 0.9886
4 Train accuracy: 0.99 Test accuracy: 0.9882
5 Train accuracy: 1.0 Test accuracy: 0.9885
6 Train accuracy: 1.0 Test accuracy: 0.9882
7 Train accuracy: 1.0 Test accuracy: 0.9877
8 Train accuracy: 1.0 Test accuracy: 0.9907
9 Train accuracy: 1.0 Test accuracy: 0.9889
10 Train accuracy: 1.0 Test accuracy: 0.9892
11 Train accuracy: 1.0 Test accuracy: 0.9892
12 Train accuracy: 1.0 Test accuracy: 0.9875
13 Train accuracy: 1.0 Test accuracy: 0.9896
14 Train accuracy: 1.0 Test accuracy: 0.9884
15 Train accuracy: 1.0 Test accuracy: 0.9883
16 Train accuracy: 1.0 Test accuracy: 0.9889
17 Train accuracy: 1.0 Test accuracy: 0.9873
18 Train accuracy: 1.0 Test accuracy: 0.9886
19 Train accuracy: 1.0 Test accuracy: 0.9893
CPU times: user 1h 46min 23s, sys: 11min 57s, total: 1h 58min 20s
Wall time: 39min 15s
```

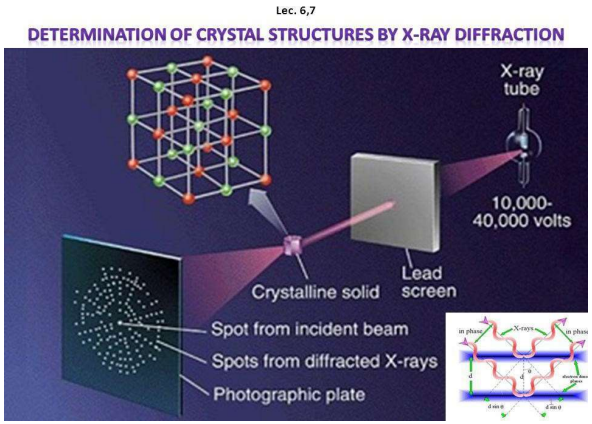
Limited Memory Becomes Bottleneck (TX2)

- 8GB Memory shared between CPU and GPU, greatly Impact the training.
- Large networks must be trained with Low-precision integer (Binarized Networks, 8Bit, 16Bit)
- Will be addressed in Jetson Xavier (Three-Generation)

In-Operando Tracking and Prediction of Transition in Material System using LSTM

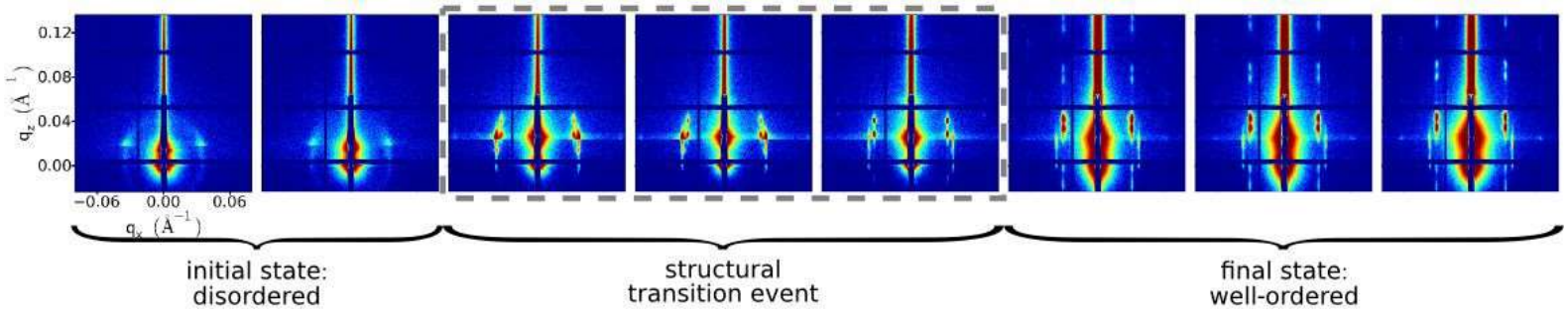
Background

A material system's structure is probed using x-ray scattering beams in NSLS-2.



The structures of many material systems evolve as they are treated with physical processing.

Background



The **structural transition event** in a material system being probed under X-ray is the most information rich.

In a material system undergoing the structural transformation, the peaks in the scattering images will sharpen, and the scattering rings will become increasingly 'textured'.

Motivation

Accurate identification of the transition frame in advance brings multiple benefits to the NSLS-II in-operando experiments such as:

- Minimal Beamline damage to samples
- Optimal sampling of material properties.

Approach

Formulate it as a **Future Frame Prediction** problem. (Sequence to Sequence)

Given a set of n images [X_1,X_2, .. X_n] from a video sequence, the objective is to predict (generate) the next m images [X^{n+1}, X^{n+2}, .. X^{n+m}] of the given video sequence as closely as possible to the ground truth frames [X^{n+1}, X^{n+2}, .. X^{n+m}].

Approach

We approach this problem in following two steps:

1. Identification of sequences having structural transition event.
2. Prediction of the future frame in the sequence based on historical image frames

We utilize an **LSTM** (Long Short-Term Memory) based model for this purpose.

Challenge

- Training an LSTM network (Deep Neural Network) requires a large amount of training data.
- Obtaining such amount of annotated training data from NSLS-2 is infeasible.

Solution

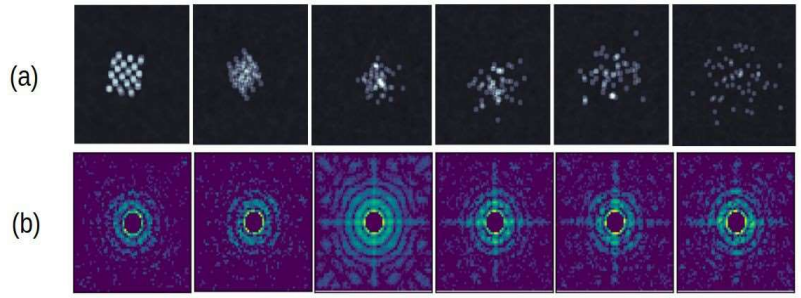
- Generate artificial data by simulating the diffraction process.

Synthetic Data

- Three crystal structures considered in our experiment namely BCC (Body-Centered Cubic), FCC (Face Centered Cubic) and SC (Simple Cubic) selected with random probability.

- A 3D box is filled with particles based on the chosen structure with particles location shifted by some random noise.

- Synthetic sequence of length 20-45 is obtained by repeatedly rotating the 3D box by a random angle.



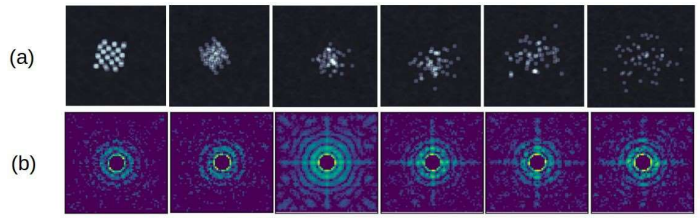
(a) Crystal structure
(b) X-ray scattering

Synthetic Data

- Index of Transition frame selected randomly. At the transition frame particles initiate to move in a random direction by some amount to simulate the Brownian motion.

- Structure is projected and FFT is applied on the 2D projection to obtain the magnitude image.

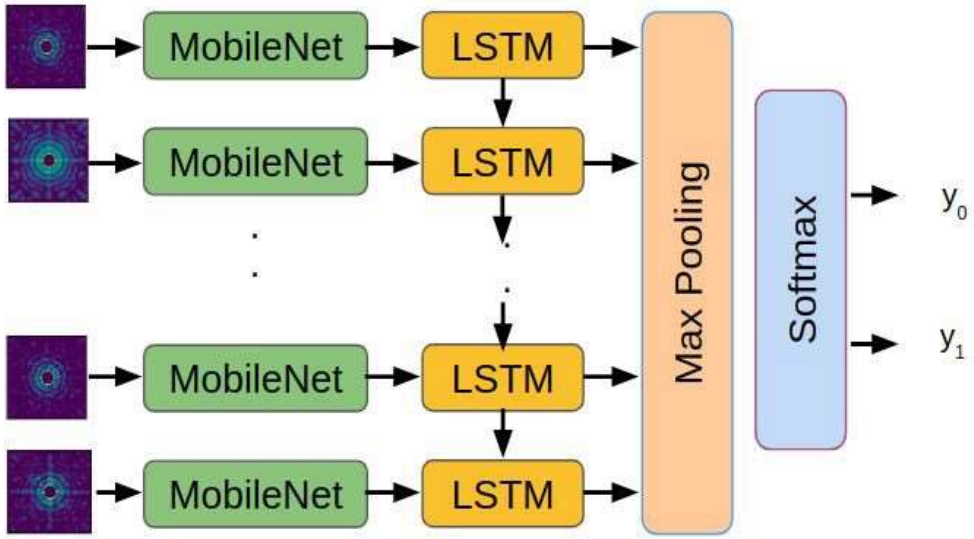
- 20k samples in the training split, with 10K samples containing transition frames and 10K samples without it. Similarly, 4800 samples were obtained in the testing split.



(a) Crystal structure
(b) X-ray scattering

Model

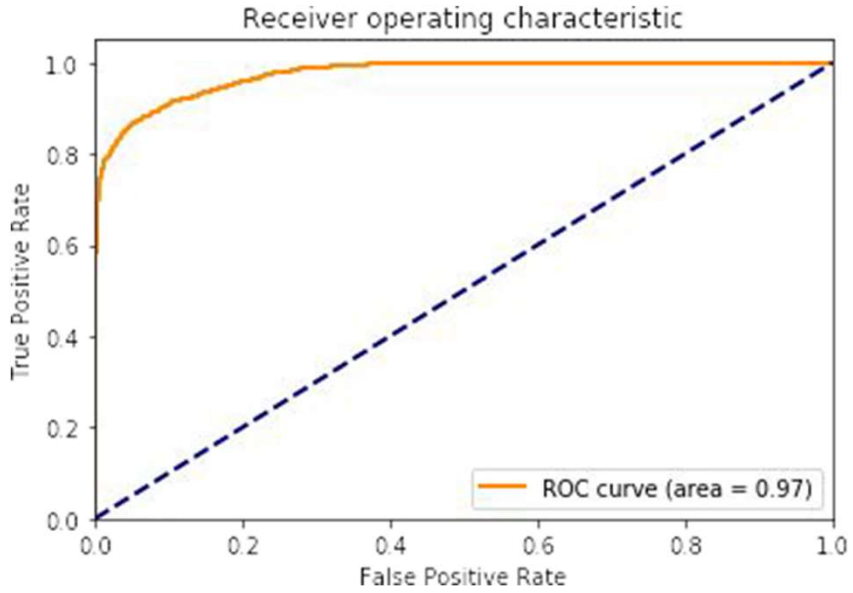
- We use a **CNN-LSTM**[1] model to exploit both temporal and spatial information.
- MobileNet CNN is used for doing transfer learning on the X-ray scattering images.



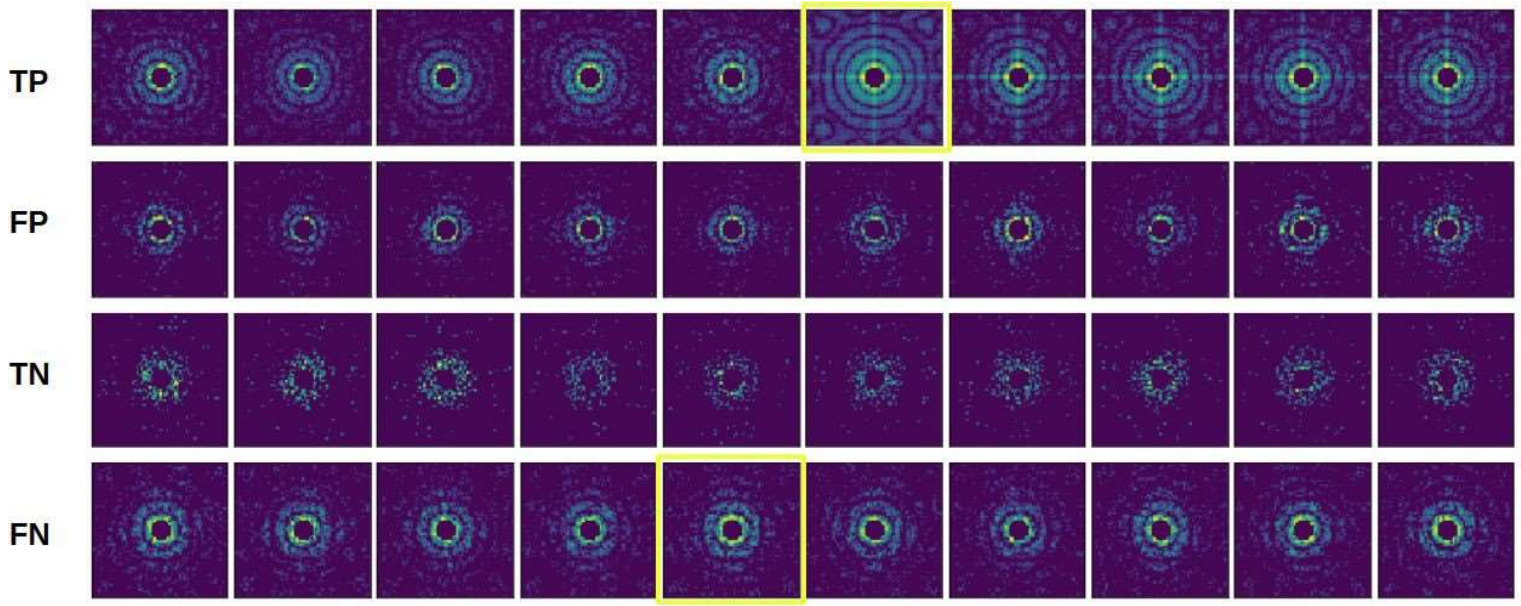
1. Beyond Short Snippets: Deep Networks for Video Classification

Results

- We obtain 0.97 AUC for a binary classification task (with transition sequence and without transition sequence)



Results



True Positive, False Positive, True Negative and False Negative from the test split

Future Work

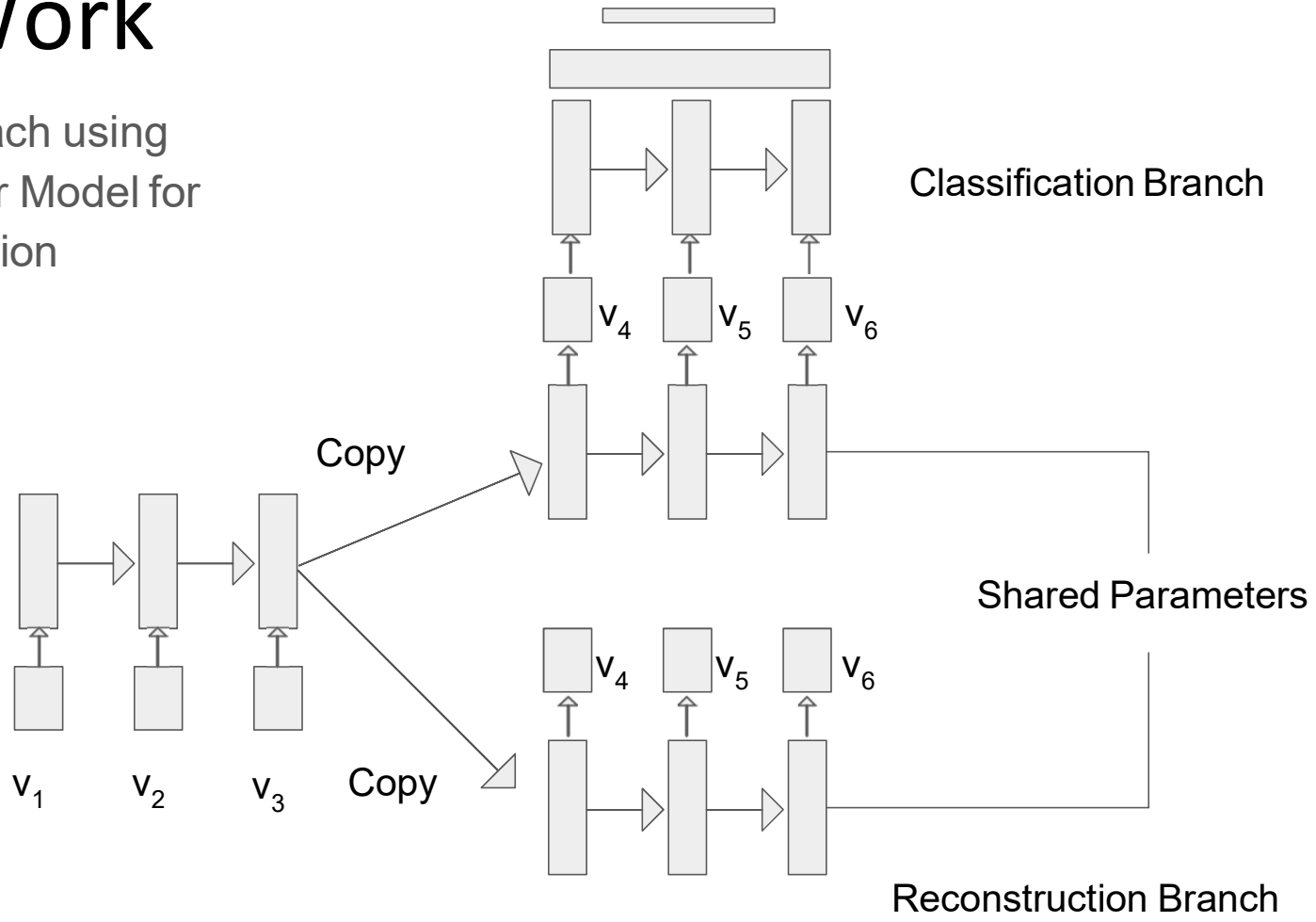
- Multi-task learning approach with the objectives to construct the future frames as well as identify the transition frame.

- Similar to an encoder-decoder model with two branches, one for classification and other for predicting (constructing) future frames[1].

1. Unsupervised Learning of Video Representations using LSTMs

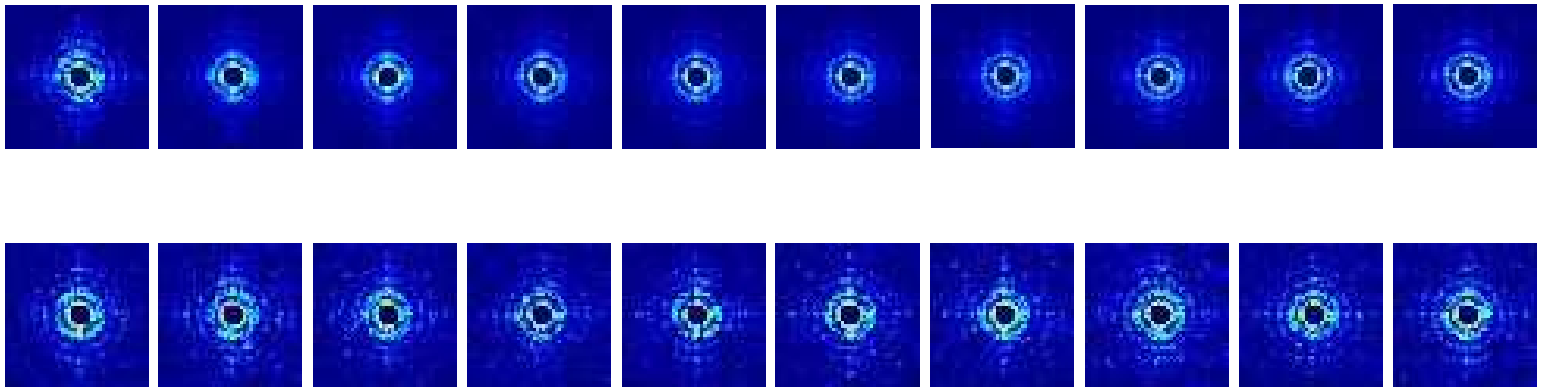
Future Work

Multi-task Approach using Encoder-Decoder Model for Transition prediction



Future Work

- Recently GAN based architectures have shown promising results in generating the future content[1] which could be incorporated in our model as well be replacing MSE loss by Adversarial loss.



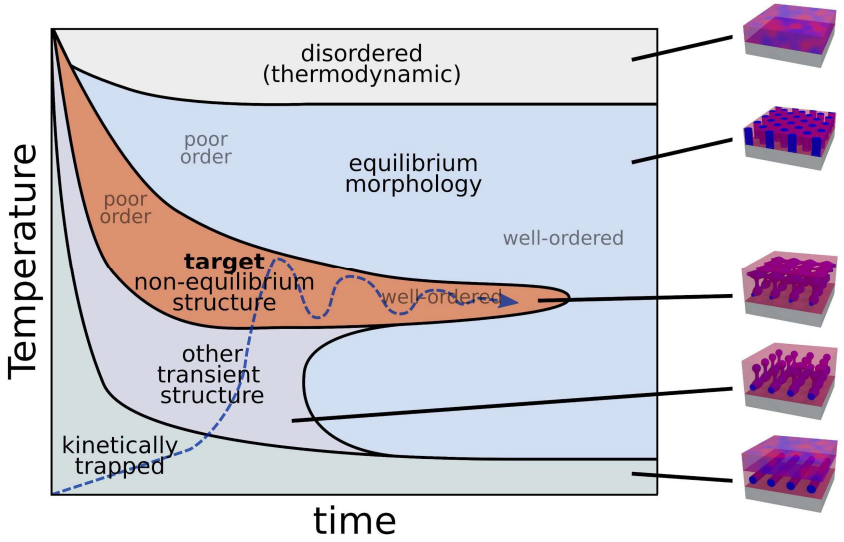
Constructed Sequence (MSE loss)

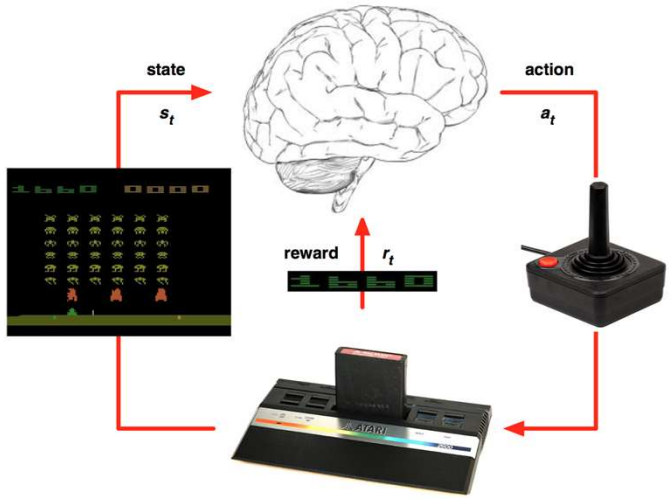
Real Sequence

1. Generating the future with adversarial transformers.

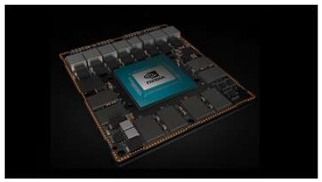
Future Work

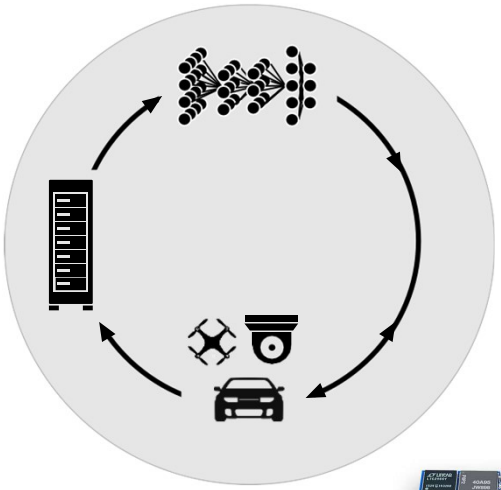
- Autonomous agent to explore the state space (Self-Driving Experiment).
- A computer-guided autonomous agent selects processing pathways
- During experiment, Search for target states while avoiding transition to undesired states.





Future Works: Integrated GPU and FPGA

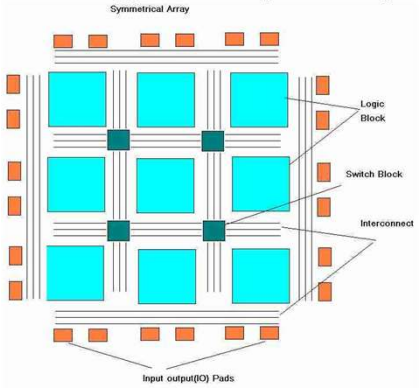




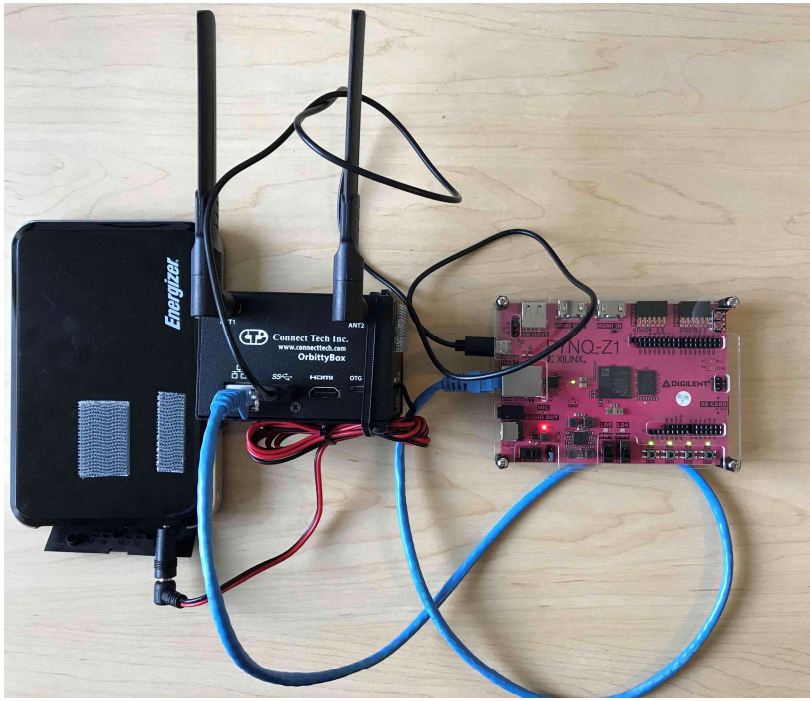
Integrated Portable Machine Learning Device that integrates training and real-time inference into one embedded system

- Training Engine Jetson TX2 (200 GPU cores =2 TFPS)
- 1Gbps Ethernet collects telemetry Data
- 4Gbps USB 3.0 for Interconnections with FPGA
- Inference Engine: FPGA (Altera or Xilinx)
- 13K Logic Blocks, 630KB Block Memory
- 1Gbps Ethernet collects telemetry Data
- 4Gbps USB 3.0 for Interconnections with Jetson TX2
- 200M operations for real-time high-frequency inference and prediction

+



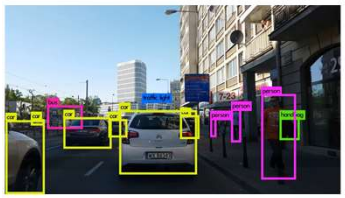
=

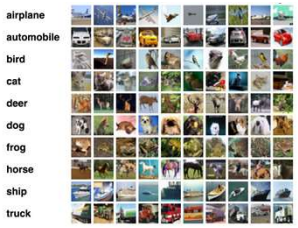


Layered NVIDIA Jetson + PYNY FPGA Software Ecosystems

Software Hierarchy from Lower to Higher Layers

Deep Learning Apps





Video Prediction

Experiment Data Analysis and Decision making

jupyter

Machine Learning & Deep Networks

TensorFlow

PYTORCH

Caffe2

scikits learn

K Keras

Critical Libraries

cuDNN

CUDA library

TensorRT

OpenCV Python

Video API

python

Operating Systems

Carrier Board Support Package (BSP, USB, PCI-e, Display Port)

Hardware

Jetson TX2 Embedded System. GPU, ARM, Video CODEC