|  |
| --- |
| Photo displaying partial image of two pie charts on a canvas-textured page |
| **Intelligent algorithm in single board computer**  Machine Learning/Deep learning real time prediction implementation on single board computer or embedded module. |
| |  |  |  | | --- | --- | --- | | Golam Gause Jaman | Idaho State University | [Course title] | |

**TITLE:** Machine Learning/Deep learning real time prediction implementation on single board computer or embedded module.

Proposed by **Golam Gause Jaman**

**DEPARTMENT:** Mechanical Engineering, Idaho State University

**Objective**

Primary aim of this document is to state strategy for development of machine learning or deep-learning based procedure with capability of running in small form factored and resource constraint computer such as Beaglebone-Black or Raspberry Pie. The aim of this research is to develop a system where some form of local environment’s data being sampled and feed to embedded module followed by machine learning process to classify an event. This research can contribute to far bigger goal, for instance, development of autonomous wheelchair to assist patients. Following figure shows generic workflow.

Single Board Computer

Workstation

>>import torch.nn

Wheelchair





**Figure 1: Workflow**

**Hardware Requirements:**

* BeaglebonBlack or Raspberry Pie (Available at Biomed Lab @ MCERC)

**Software and tools requirements:**

* Python
* PyTorch
* Keras
* TensorFlow
* MATLAB®

**Introduction**

Deep learning gained exponential attention in recent times simply because of its capability of solving non-intuitive, non-linear complex problems. Furthermore, semiconductor industries enhanced so much so, we are living in a time where highest number of transistors are accommodated in an unit volume. That is, we are living in an era of fastest computer yet small sized and low power consuming, which makes tasks like machine learning or deep learning more popular, as powerful computers are getting more accessible. Single Board Computer (SBC) like Beagle Bone Black and Raspberry-Pie have same capabilities as average desktop computers from 10 years back. Therefore, a new need was born to run learning algorithm in small formed factor devices like SBC. Benefit of doing learning algorithm on the SBCs is that it allows embedded system to learn from events, portable low power devices can communicate with others and share information to learn and asses events autonomously, collaboratively and more accurately. Figure 1 shows generic workflow this research aims to accomplish.

**Approach**

The work done can be segmented into three phases.

*Phase 1*

The research initially investigated typical computational resource requirements to perform classification problem using machine learning or deep learning. Commercially available SBC typical configuration as follows:

* **CPU** – Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz.
* **RAM** – 1GB, 2GB or 4GB LPDDR4-2400 SDRAM (depending on model)
* **Wi-Fi** – 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE.

*Phase 2*

Development of Convolutional Neural Network for time series data. Most sensor reading comes in time series data and there are enormous tools or pretrained model or transfer learning opportunity in CNN. However, CNN are often used for image input. In this work, CNN was explored in time series data that needed translation of 1-D data to image data format. A test CNN was developed on generated times series data and performed training process followed by testing.

*Phase 3*

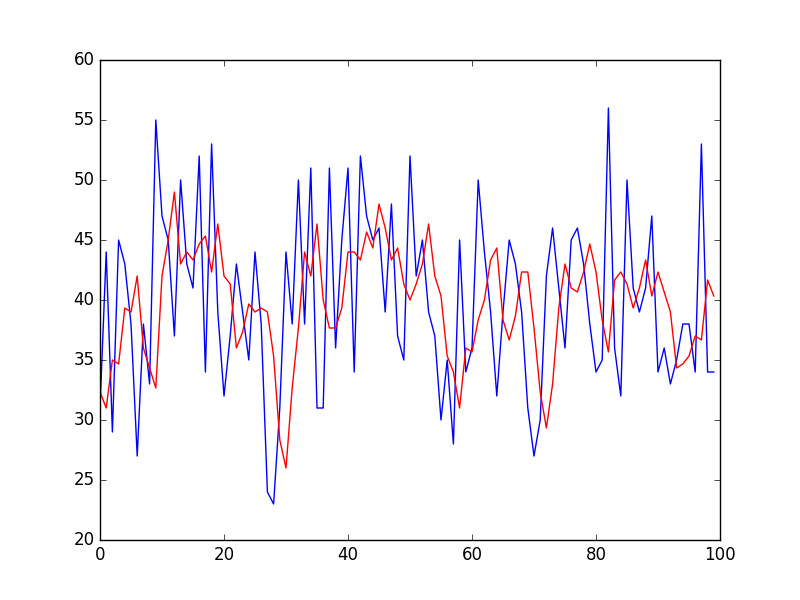
In this phase, path was explored where model generated in python run in MATLAB® Simulink to take advantage of rich set of tools that upload model to various SBCs.

**CNN with time series data**

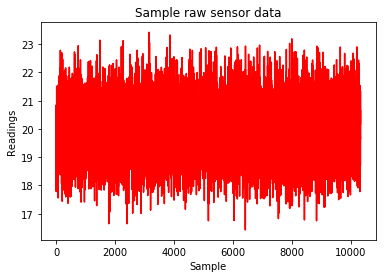
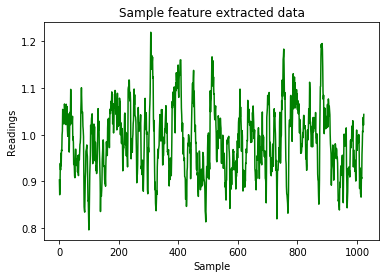
Followings are key steps considered

* Generate data that mimics photo sensitive resistor readings due light reflection from color surface.
* Apply feature extraction of the raw data on pre-defined sized window (e.g. zero-crossing or moving average).
* Controlled event will be generated (e.g. certain color light or surface) to complete training and test datasets.
* Re-format the training and test datasets to make compatible with torchvision datasets.
* Defined a simple Convolution Network followed by feed forward network.
* Once trained, validation exercise was conducted to analyze performance.

Sliding direction



Feature extraction moving window

**Figure 2: Feature extraction process on sampled data**

Presenting time series data in terms of image often seems like abstract art. Idea is to take extracted features from striding window and populate a matrix where each element can be treated as pixel. Figure 3 shows how filled contour map of extracted features looks like.

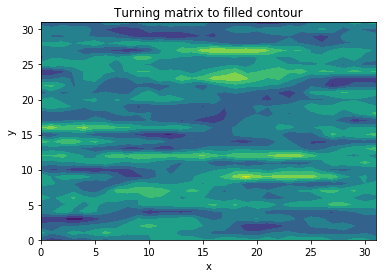


Figure 3: Image equivalent of time series data

Convolution Neural Network used in this work takes single channel 32 by 32 pixels image equivalent data and trained on two stage convolution process followed by three-layer linear network as shown in figure 4.

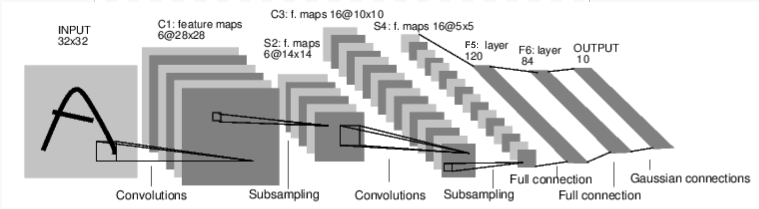


Figure 4: Convnet

**Result**

At this point, concept was tested with data that has distinct characteristics and training has driven loss function to near zero. However, in real sampled data this may not be the case and even if it does, that is not necessarily a progress rather overfitting as real data often have noises.

**Pytorch 🡪 Keras 🡪 MATLAB®**

Pytorch and keras both are popular tool when it comes to deep learning. However, due to more opportunity in pytorch to customize model parameters, this work was done using pytorch tools. MATLAB® does not communicate with models written in pytorch but works with keras. Therefore, Pytorch to Keras conversion needed. Fortunately, there is good amount of work done by nerox8664 from GitHub, details are there.

1. First of all, convert the pytorch model to Keras with this converter:

k\_model = pytorch\_to\_keras(model, input\_var, [(10, 32, 32,)], verbose=True, names='short')

1. Save it as h5 file and then convert it with tensorflowjs\_converter but it doesn't work sometimes. As alternative, you may get Tensorflow Graph and save it as a frozen model.

Once, successful conversion, keras layers can be imported in MATLAB® using importKerasLayers.

[layers = importKerasLayers(modelfile)](https://www.mathworks.com/help/deeplearning/ref/importkeraslayers.html;jsessionid=922bba1d5b0d52a74cd74fa076d1#d117e113681)

imports the layers of a TensorFlow™-Keras network from a model file. The function returns the layers defined in the HDF5 (.h5) or JSON (.json) file given by the file name modelfile.

This function requires the Deep Learning Toolbox™ Importer for TensorFlow-Keras Models support package. If this support package is not installed, then the function provides a download link. Following is sample workflow.

Import the network layers from the model file digitsDAGnet.h5.

modelfile = 'digitsDAGnet.h5';

layers = importKerasLayers(modelfile)

layers =

LayerGraph with properties:

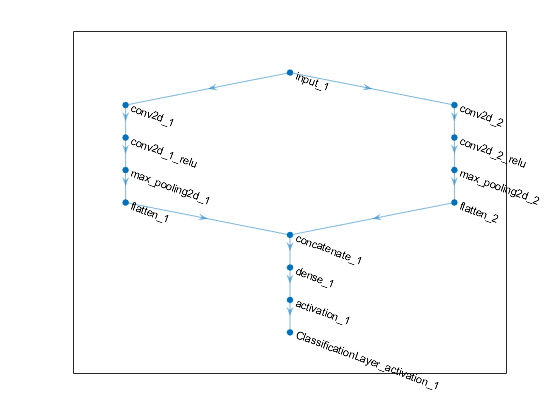
Layers: [13×1 nnet.cnn.layer.Layer]

Connections: [13×2 table]

Plot the network architecture.

figure

plot(layers)



Once, model is successfully imported to MATLAB®, function can be defined and called as function block in Simulink that enables further uploading the model to target hardware. Importing to hardware part was not done during the period of this work due to time constraint.

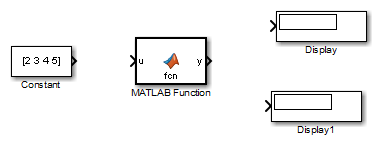


Figure 5: MATLAB® function as block in Simulink

**Budget**

This work can be used towards part of on-going research at ISU and all equipment and computer resources are provided by Mechanical Engineering Department, ISU.

**Notes:**

Outcome of this work will also benefit separate active research on limb motion prediction using surface electromyography signals, Stall warning detection in jet engine and real-time fault detection in additive manufacturing.

**Conclusion:**

This work completed a key piece of work towards real-time, time series data classification. Translation to datasets compatible with torchvision, enables series of new experiments. The research also address bridging between pytorch model and Simulink that creates new window of implementation in hardware with pre-trained or transfer deep learning models.

**References**

<https://www.mathworks.com/help/deeplearning/ref/importkeraslayers.html>

<https://github.com/nerox8664/pytorch2keras>