Classification of Wildfires from MODIS Data Using Neural Networks

PI: James MacKinnon
Co-Is: Troy Ames, Dan Mandl, Charles Ichoku, Luke Ellison
Interns: Jacob Manning, Baram Sosis

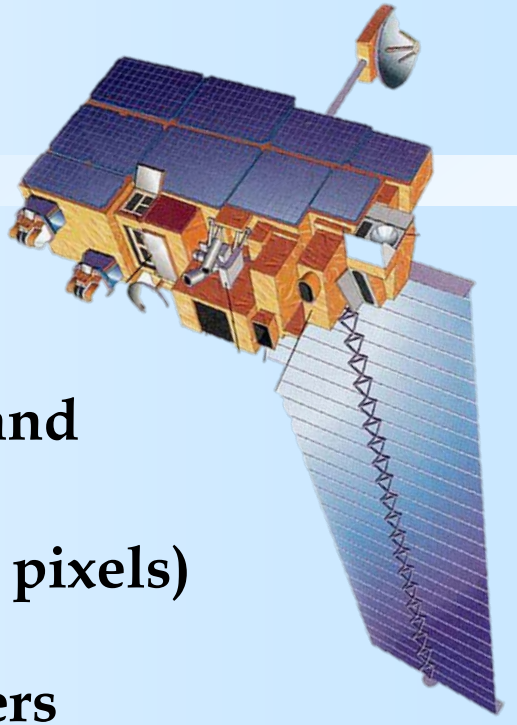
Overview

- Can we build a neural network that detects wildfires?
  - Attempted before in 1990's with limited success (They didn't have GPUs)
- Moderate-resolution imaging spectroradiometer (MODIS) data analyzed for this task
  - MODIS Active Fire and Burned Area Products (MODFIRE) ensures a wealth of labeled training data
  - MODFIRE algorithm intractable for embedded hardware
- Project resulted in a highly accurate neural network
  - Trained on ground using powerful GPUs (dual GTX 1080)
  - Inferences performed on:
    - Flight-like embedded hardware for CubeSat like testing
    - Nvidia Tesla P4 for ground testing

Motivations

- Wildfires pose danger to life and property
- Enable autonomous wildfire detection
  - Low cost detection platform (CubeSats)
  - Quick turnaround time for detection
- Embedded platforms lack adequate NN tools
  - For Training on Ground
  - For Inferencing in Space
  - Training in Space?
- MODIS data provides useful stand-in for future missions

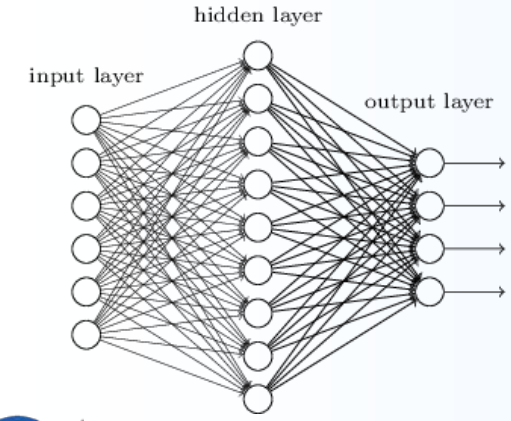
MODIS Data Format



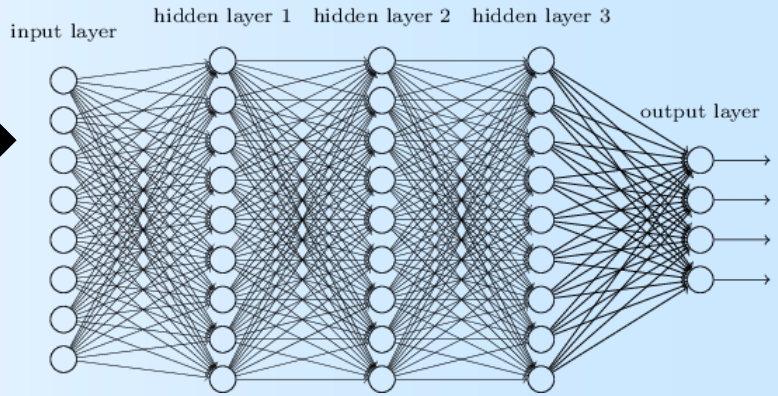
- Level 1B calibrated data products
  - 36 spectral bands
  - Radiance or Reflectance depending on band
  - Scaled Integer format
  - "Granules" – 5 minute swaths (2030x1354 pixels)
- Hierarchal Data Format (HDF) data containers
  - Easily parsed with Python scripts
  - Standard naming format
  - Contains meta-data in addition to spectral band data
- Data located at LAADS Distributed Active Archive Center
  - Local to Goddard, so data is provided over intranet!

Neural Networks Primer

- **Built out of many simple, highly interconnected neuron-like units**
- **Use pre-labeled datasets to learn associations and classifications in data and correctly classify new data**
- **Have been applied to medical diagnosis, social network filtering, quantum chemistry, and more**



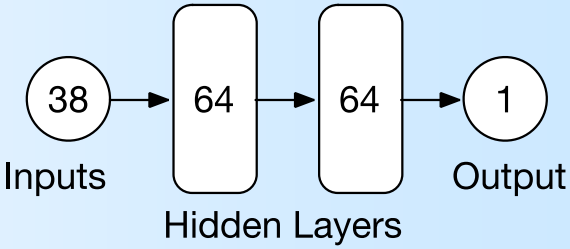
Deep Versus Shallow



Images from http://neuralnetworksanddeeplearning.com

Our Neural Network

- Initial attempt had:
  - 38 inputs
  - 2 hidden layers w/ 64 PEs
  - ReLU activations
  - Can we do better?

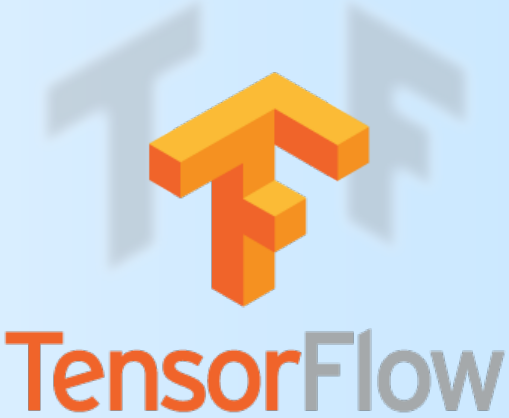


- Smaller network resulted in very similar accuracy, and faster execution
  - 38 inputs
  - Reduced PEs in hidden layers
    - 38 in first and 20 in second

- Finally, introduced meta-data as input
  - Solar Azimuth, Zenith, Time of day

Our Neural Network cont.

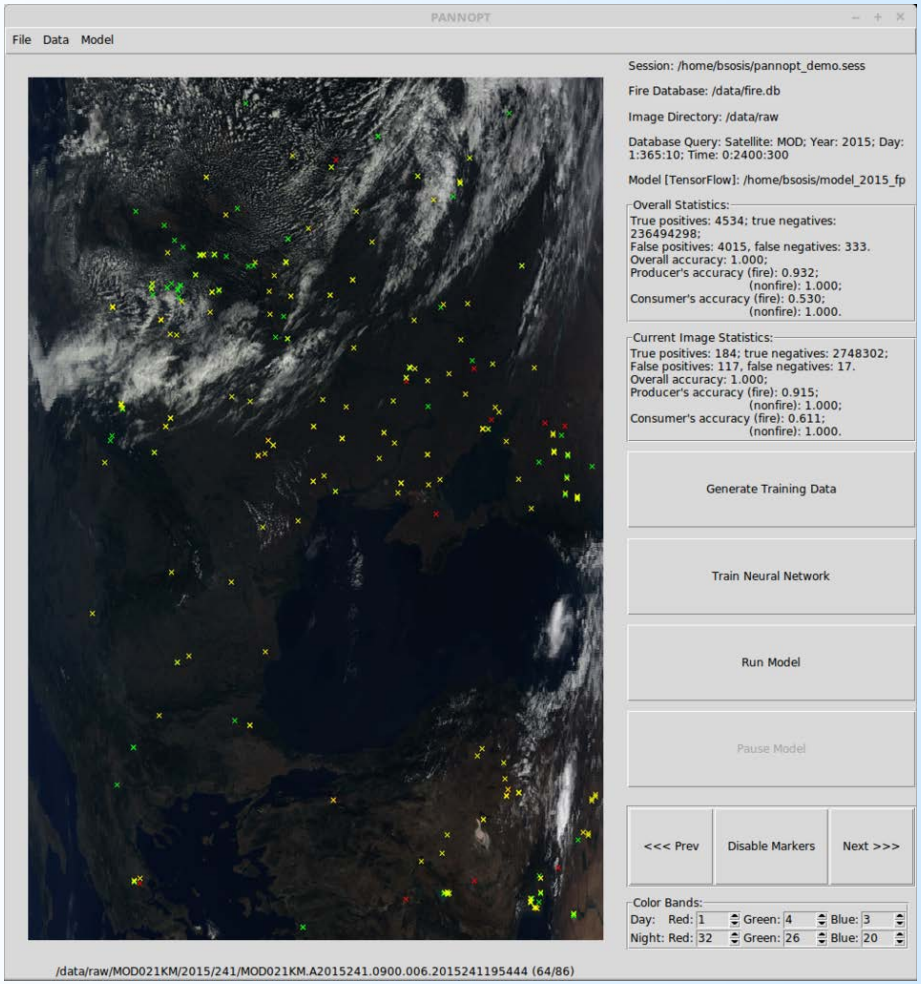
- Neural Network implemented using Keras w/ TensorFlow backend
- Keras is a high level neural network API, open source and written in python
  - Rapid prototyping made easy
  - Abstracts away complicated TensorFlow data graphs
- TensorFlow is Google's powerful graph processing library
  - Very fast, uses GPUs



Training the Network

- GUI developed in Python to automate training set generation, model running, and data visualization
- Tasks were many-fold
  - Grab batches of fire pixels from MODFIRE products
  - Use fire pixels to determine what data to retrieve and then download it locally
  - Pre process downloaded data
    - Handle error codes (e.g. fill values, saturation, etc)
    - Convert scaled integer products to floats
    - Perform feature scaling
- Result is a training set with 1 million total examples
  - 50/50 split of fires and non-fires

Training GUI



Validation Results

Validation Results (samples not in training set)

Accuracy: 99.59%

Precision: 99.03% (False Positives < 1% fires mistakenly detected)

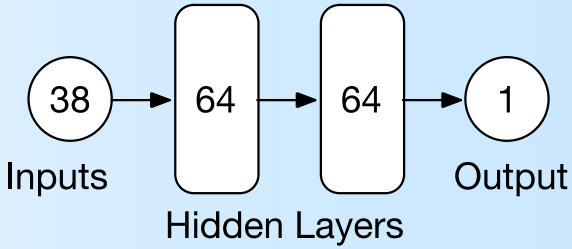
Recall: 99.05% (False Negatives < 1% or missed real fires)

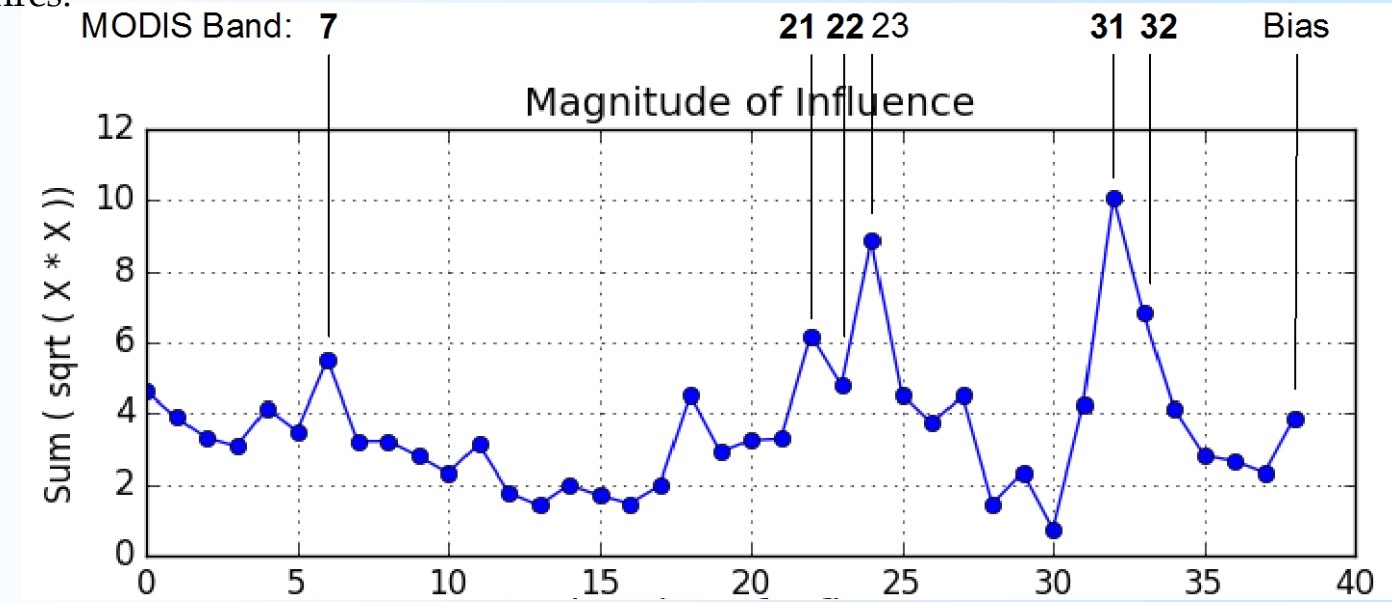
Loss: 0.319% (Mean Squared Error)

A preliminary look at the NN weights after training indicate that the MODIS 7, 21, 22, 23, 31, and 32 bands were the most influential for detecting fires.

Note the MODIS active fire detection algorithm uses bands: 1, 2, 7, 21, 22, 31, and 32.

Network Architecture





Results Cont.

Metric	Equation	Result
Overall Accuracy	(TP+TN)/(TP+TN+FP+FN)	99.96 %
Producer Accuracy (Fire)	TP/(TP+FN)	97.82 %
Producer Accuracy (Non-Fire)	TN/(FP+TN)	99.97 %
User Accuracy (Fire)	TP/(TP+FP)	**7.17 %**
User Accuracy (Non-Fire)	TN/(TN+FN)	99.99 %

TP: True Positive
FP: False Positive
TN: True Negative
FN: False Negative

Analysis of Results

- The neural network essentially learned the MODFIRE algorithm
  - Important bands in MODFIRE were import in the neural network as well
- False positive rates were relatively high
  - FP rate less important than FN
  - False alarms better then missing real fires
  - Partially mitigated by FP bootstrapping
- FP Boostrapping is running a trained model and taking FPs and adding them to training set
  - Can lead to overfitting

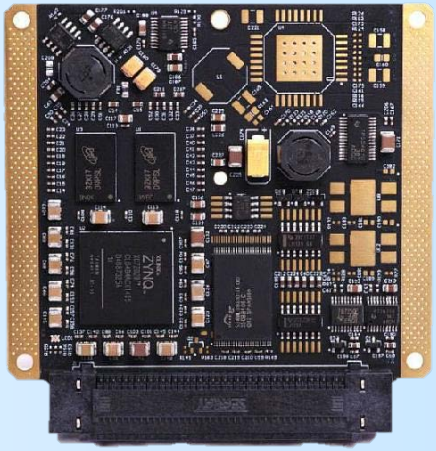
Results Cont. w/ FP Bootstrapping

Metric	Equation	Result
Overall Accuracy	(TP+TN)/(TP+TN+FP+FN)	99.99 %
Producer Accuracy (Fire)	TP/(TP+FN)	**94.10 %**
Producer Accuracy (Non-Fire)	TN/(FP+TN)	99.99 %
User Accuracy (Fire)	TP/(TP+FP)	**53.02 %**
User Accuracy (Non-Fire)	TN/(TN+FN)	99.99 %

- Adding false positives back into training set drastically increased the user fire accuracy at the cost of loss of generalization ability

Embedded Implementation

- Aim was to target flight-like hardware commonly found on CubeSats
- Targeted two platforms:
  - CHREC Space Processor (CSP)
    - ARM Cortex-A9 (dual core)
  - Raspberry PI
    - ARM Cortex-A53 (quad core)
- R-Pi as stand-in for upcoming next generation space computer containing Zynq UltraScale+ MPSOC



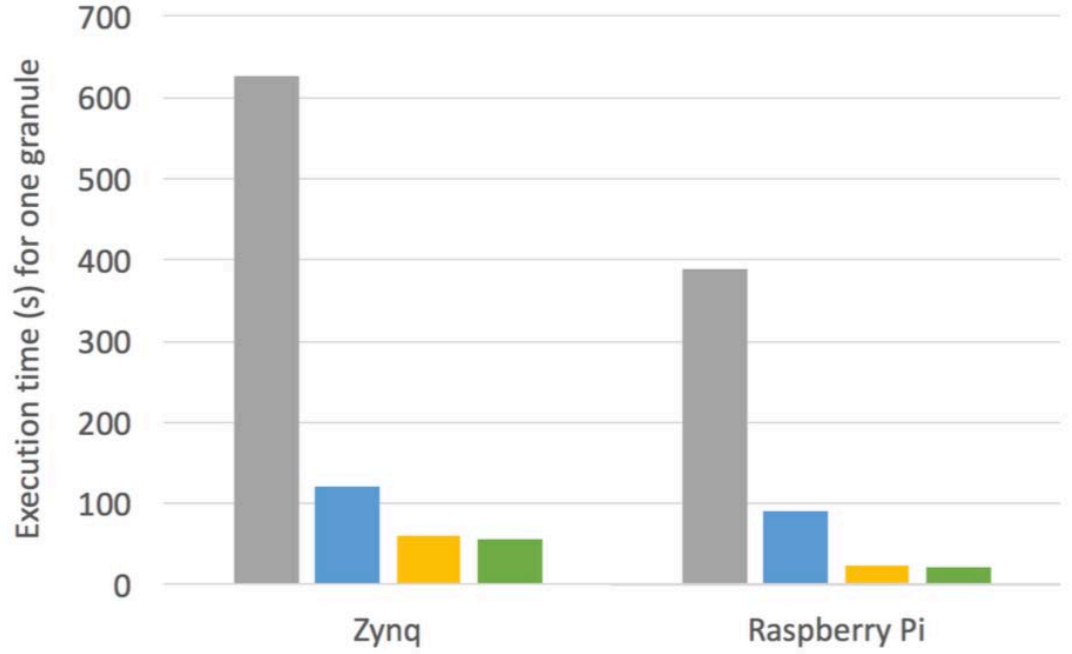
CSP

Embedded NN Implementation

- Three pronged approach:
  - Simple Cross compile of Tensorflow
    - Easiest to implement
    - Reuse models directly
    - But very slow
  - Optimized NumPy implementation
    - Mimic calculations with NumPy's parallelization capabilities
    - Still slow; hamstrung by Python's loop inefficiencies
  - Custom C implementation
    - More time consuming to implement
    - 165x improvement over TensorFlow!

Embedded NN Implementation

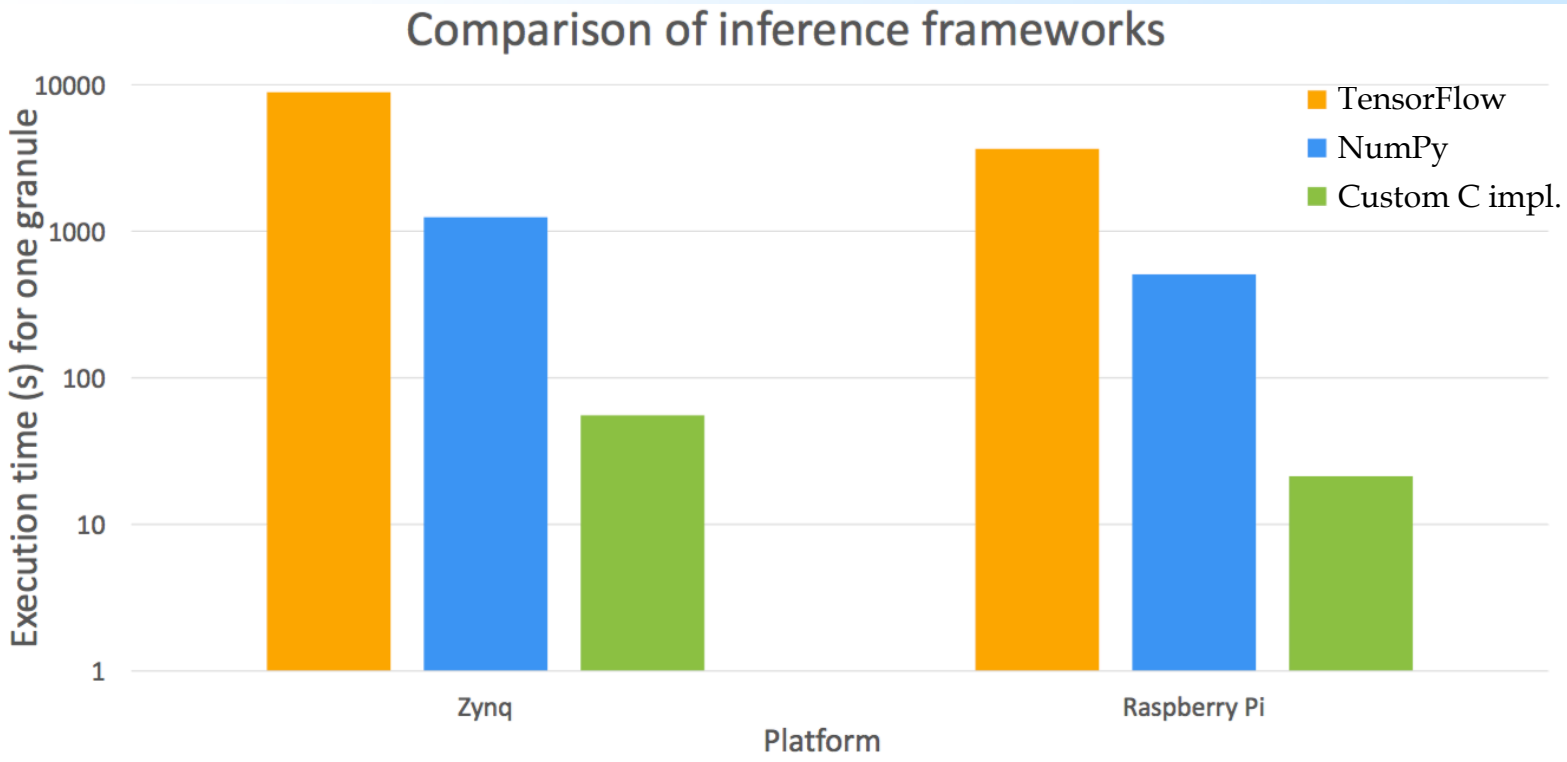
- **Improvements came from optimizing for the ARM architecture**
  - NEON - SIMD Vector instructions
  - OpenMP – Matrix Multiply scales well across cores
  - Ensuring arrays are optimally aligned for maximum cache hits
- **Ditching the Python interpreter was main driver of performance**



Note: R-PI has 4 cores and higher clock rate resulting in the better performance numbers

SIMD: Single Instruction Multiple Data

Embedded Implementation Results



Zynq: Dual core, ARM Cortex A9
Raspberry Pi: Quad core, ARM Cortex A53

Note: Logarithmic scale on Y axis!

Future Work

- Reduce false positive rate
  - False positive bootstrapping is a start
  - MODFIRE algorithm uses adjacent pixels
- Use more modern NN architectures
  - ”Slice” granules into regions and apply convolutional NN
- Apply NN to more data sources
  - Geostationary – e.g. GOES
  - Low Earth Orbit – Future CubeSat constellations
- Train on raw data
  - Level 1b data products not normally available onboard
  - NN would “learn” pre-processing, but require deeper net

Conclusions

- Designed a neural network that can detect wildfires with a high degree of accuracy from MODIS data
- Developed a suite of tools to facilitate rapid prototyping
  - Data gathering
  - Training set generation
  - Model running and testing
- Implemented Feed-Forward component of NN on low-power, low-cost embedded hardware
  - High performance without high cost
  - Perfect for CubeSats

Questions?