Hewlett Packard
Enterprise

Machine Learning in The Chapel Programming Language

Iain Moncrief
Oregon State University
Computer Science BS & Mathematics Minor, 2024
Computer Science MS, 2025

Chapel Team, Summer Intern

Chapel

- Built for high performance computing
  - Includes many language constructs that support parallel computing
- Distributed array programming
  - Many computers and CPUs can operate on the same array
- Automatically utilize available resources on a machine or cluster
  - Programs will scale to maximize the allotted CPUs or nodes
- Easily write parallel programs
  - Significant speedup just by rewriting a program in Chapel

- Applying Chapel to machine learning
  - Chapel does not have an existing machine learning library
  - Utilize Chapels array programming features

Background

NumPy

Lightweight tensor and linear algebra library

TensorFlow

Extends NumPy to support ML needs
(backpropogation, CUDA implementation)

Keras

PyTorch

Fully featured ML tools (uses TensorFlow)

My Project

- Implement ML programs in Python, only using NumPy
  - Basic ML library in a style like PyTorch or Keras
- Translate the Python programs to Chapel programs
  - Using parallel features whenever possible
- Performance comparisons between the two versions
  - How do these novel implementations perform with respect to one another?
- See what aspects of Chapel make it easier/harder to implement ML programs
  - Compile a report that summarizes this and my experience learning Chapel

My Chapel ML Implementation

- Tensor Library (intended NumPy equivalent)
  - Attempts to replicate much of NumPy's functionality
  - Arithmetic and linear algebra operations
  - Supporting helper functions
- Machine Learning Library (translation of my Python library)
  - Implements various layer types: Dense, Conv2d, MaxPool, SoftMax, ReLU, TanH, Sigmoid, Flatten, ...
  - Offers similar user interface as PyTorch or Keras

```
model = Sequential(
    Dense(4),
    Sigmoid(),
    Dense(3),
    Sigmoid(),
)
```

```
var model = new Sequential(
    new Dense(4),
    new Sigmoid(),
    new Dense(3),
    new Sigmoid()
);
```

Performance Comparisons

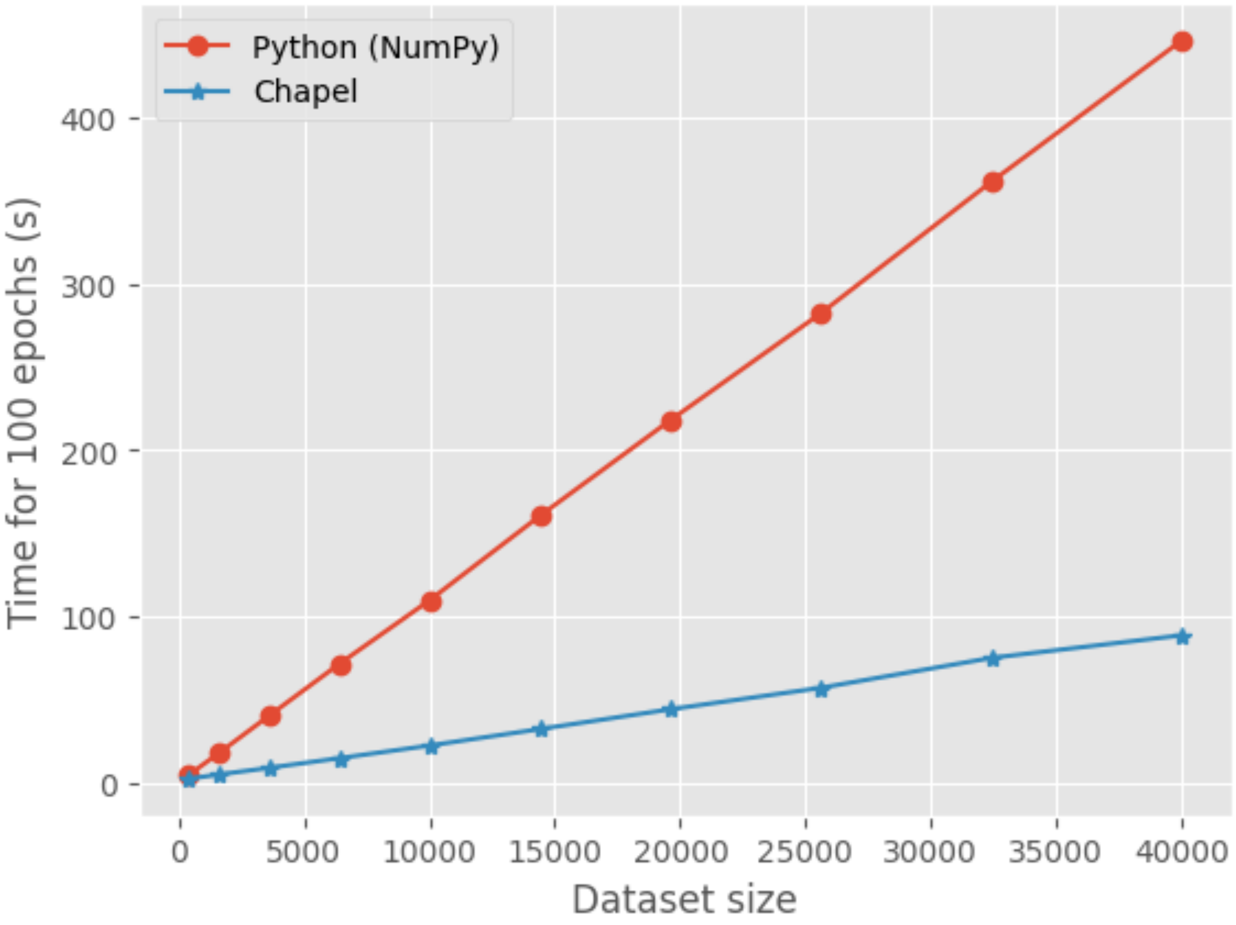
Classifier implementation via Python
and NumPy (using C backend)

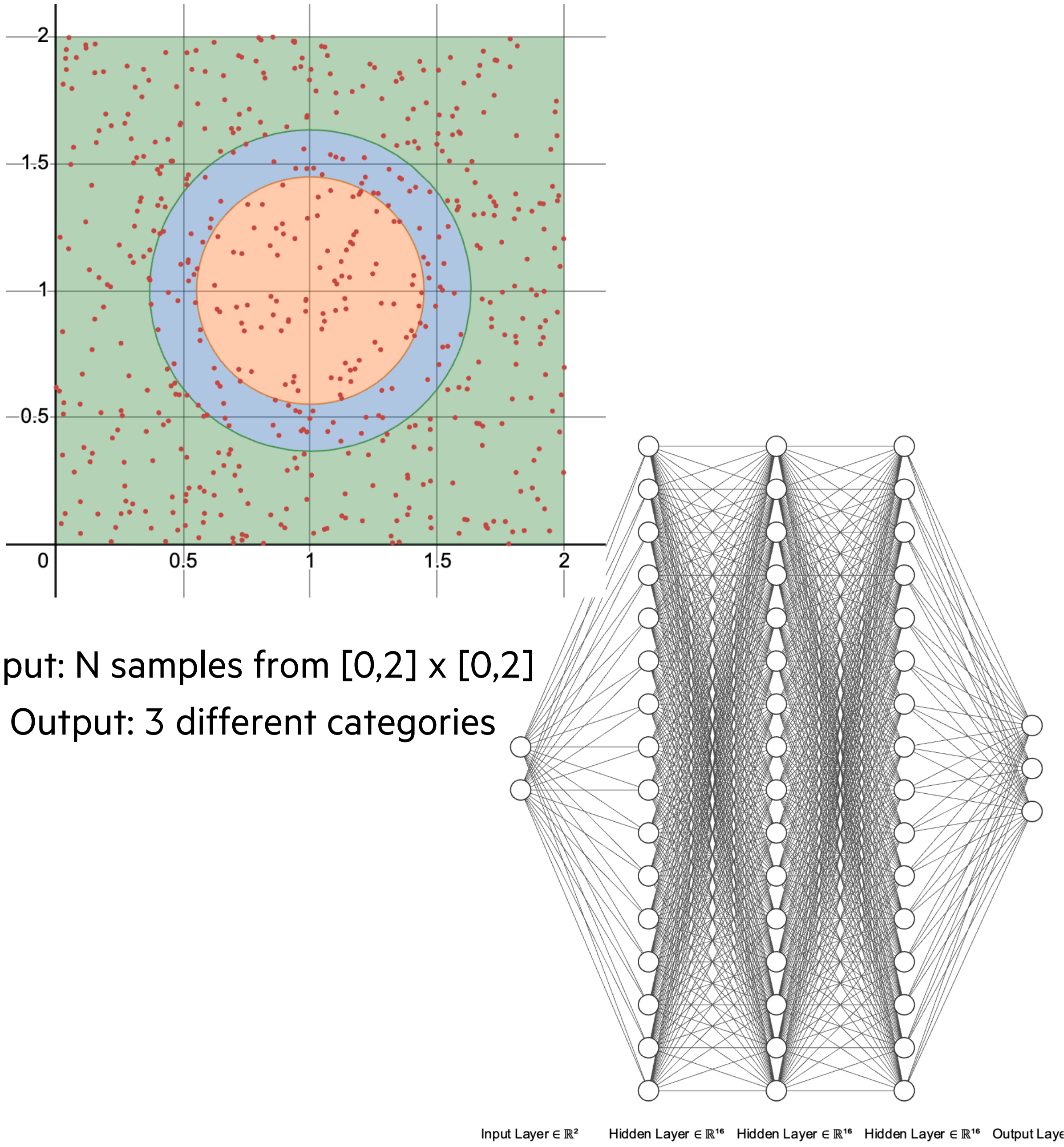
github.com/Iainmon/ml-chapel-summer-23/python

Classifier implementation using Chapel ML library

github.com/Iainmon/ml-chapel-summer-23/chapel

Speed Comparison (Simple Classification)

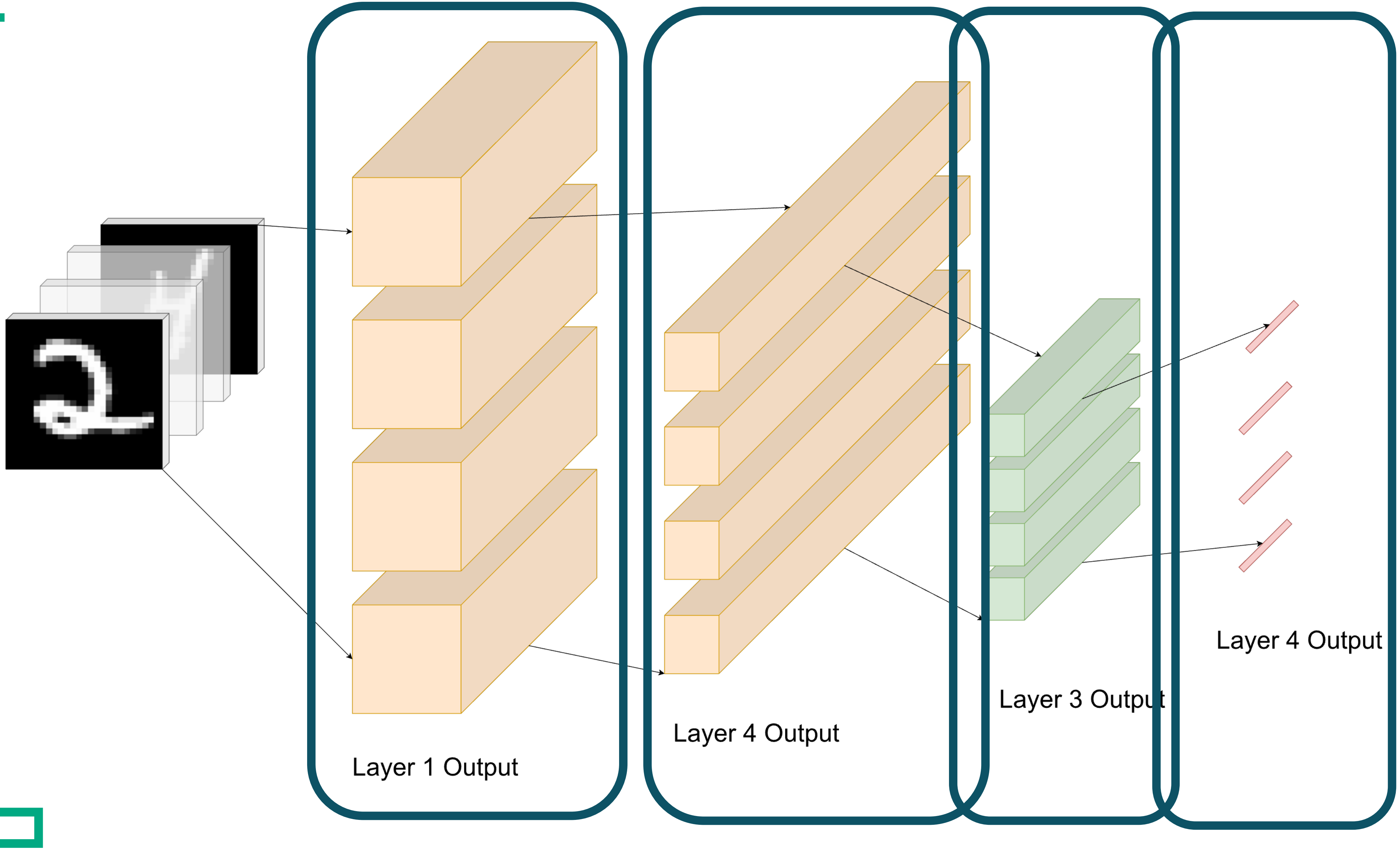




Input: N samples from [0,2] x [0,2]

Output: 3 different categories

Utilizing Chapel’s Parallelism Constructs

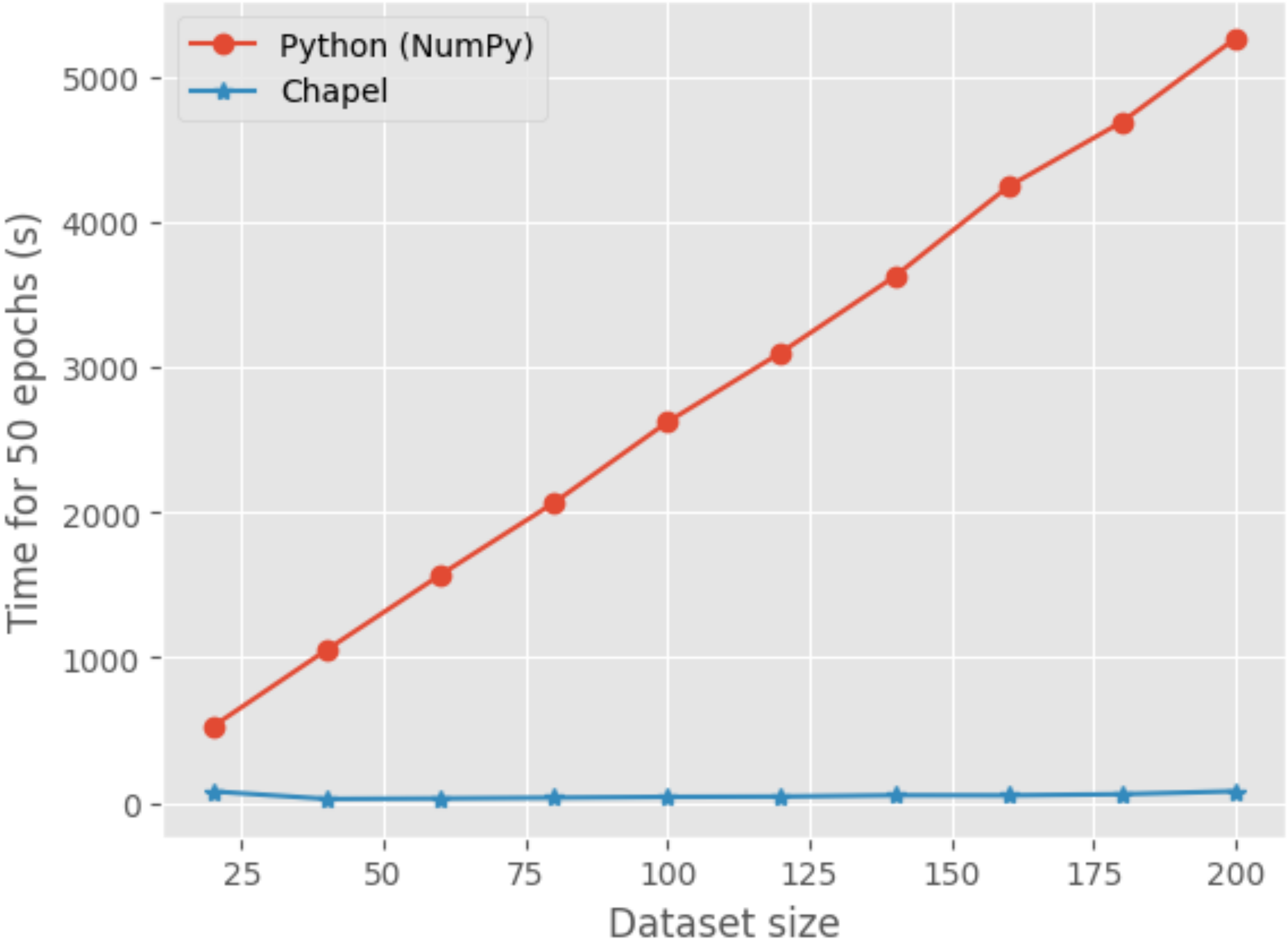


Easy Parallelism

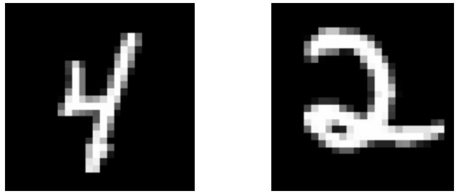
```
// Processes single input vector
proc forwardProp(input: Tensor(1)): Tensor(1) {
      return (weights * input) + bias;
}


// Processes a batch of input vectors in parallel
proc forwardPropBatch(batch: [?dom] Tensor(1)): [] Tensor(1) {
   var activations: [dom] Tensor(1);
   forall i in dom do
      activations[i] = forwardProp(batch[i]);
   return activations;
}
```

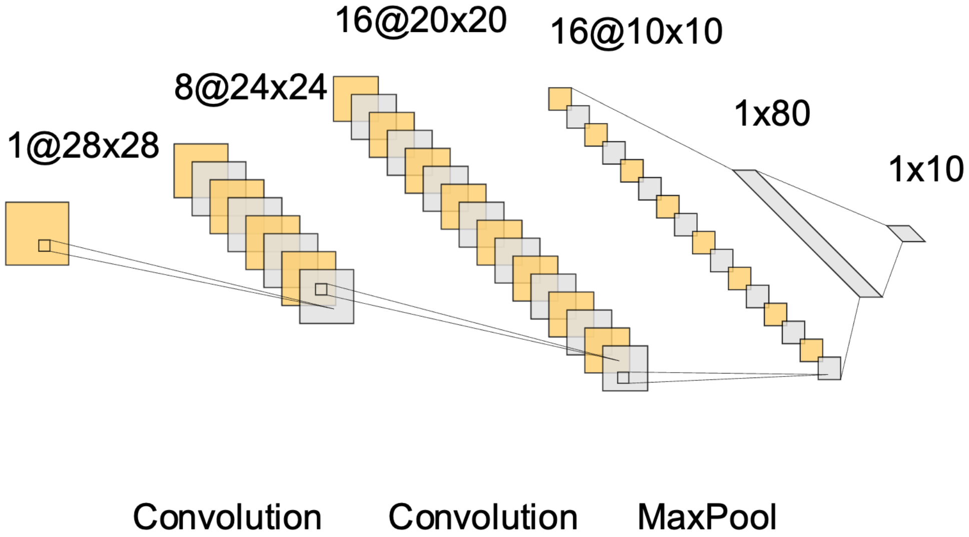
Speed Comparison (MNIST Classification)



Input:



Output: 0,1,2,3,...,9



Thank you

Thank you to my mentor, Jeremiah!

Summary

- Proof of concept for ML library in Chapel
  - Submitted library to the Chapel repository
- Identified pain points of implementing ML programs
  - Influenced language design choices via this project
  - Participated in design discussions
  - Consolidated final report (in progress)

Resources

- CNN From Scratch With NumPy (https://www.kaggle.com/code/milan400/cnn-from-scratch-numpy)
- Neural Networks from Scrath (http://neuralnetworksanddeeplearning.com/chap1.html)
- Backpropagation Simplified (https://towardsdatascience.com/back-propagation-simplified-218430e21ad0)
- Dive Into Deep Learning (https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html)
- A Survey on the New Generation of Deep Learning in Image Processing (https://www.researchgate.net/publication/337746202_A_Survey_on_the_New_Generation_of_Deep_Learning_in_Image_Processing)