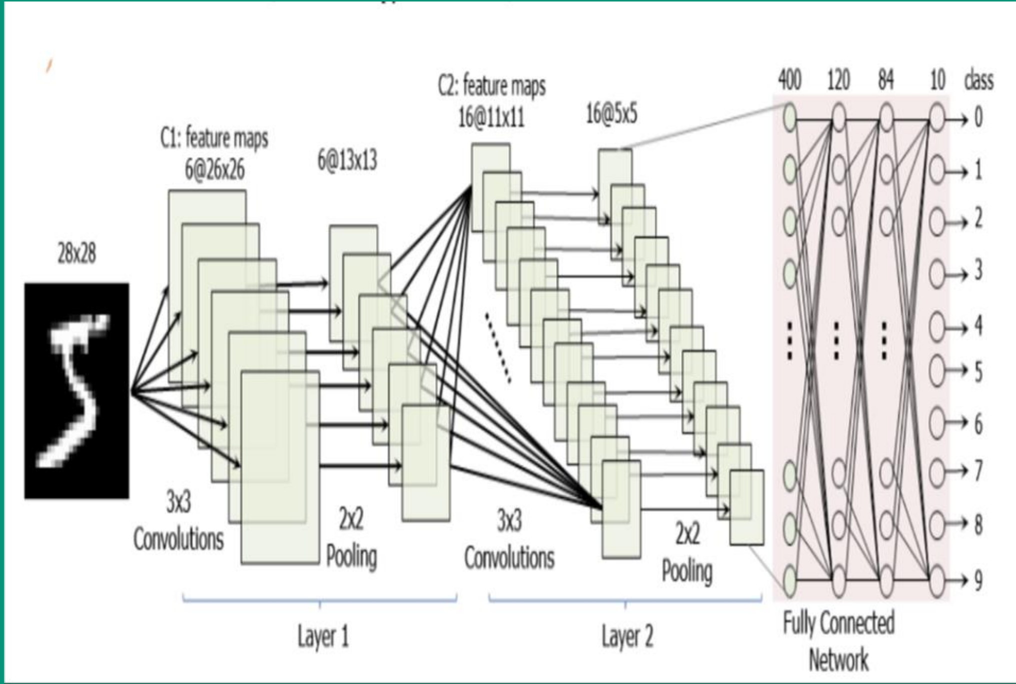
Convolutional Neural Network with Intel Thread Building Blocks

By Joshua Duncan & Matthew Irvine

What is a CNN

- CNNs are a class of deep learning neural networks, predominately used for processing data with a grid-like structure
- Applications
  - Image Recognition
  - Image Classification
  - Object Detection
- Key Components
  - Training set
  - Convolutional Layers
  - Fully connected network



Training Set Data

- Stored in multiple .bif files
- 10,000 28x28 Images
- 7.84 Million Elements
- Images are of handwritten number digits
- 10,000 Expected Outputs
- Corresponds to the expected number

Convolutional Layers

First Layer

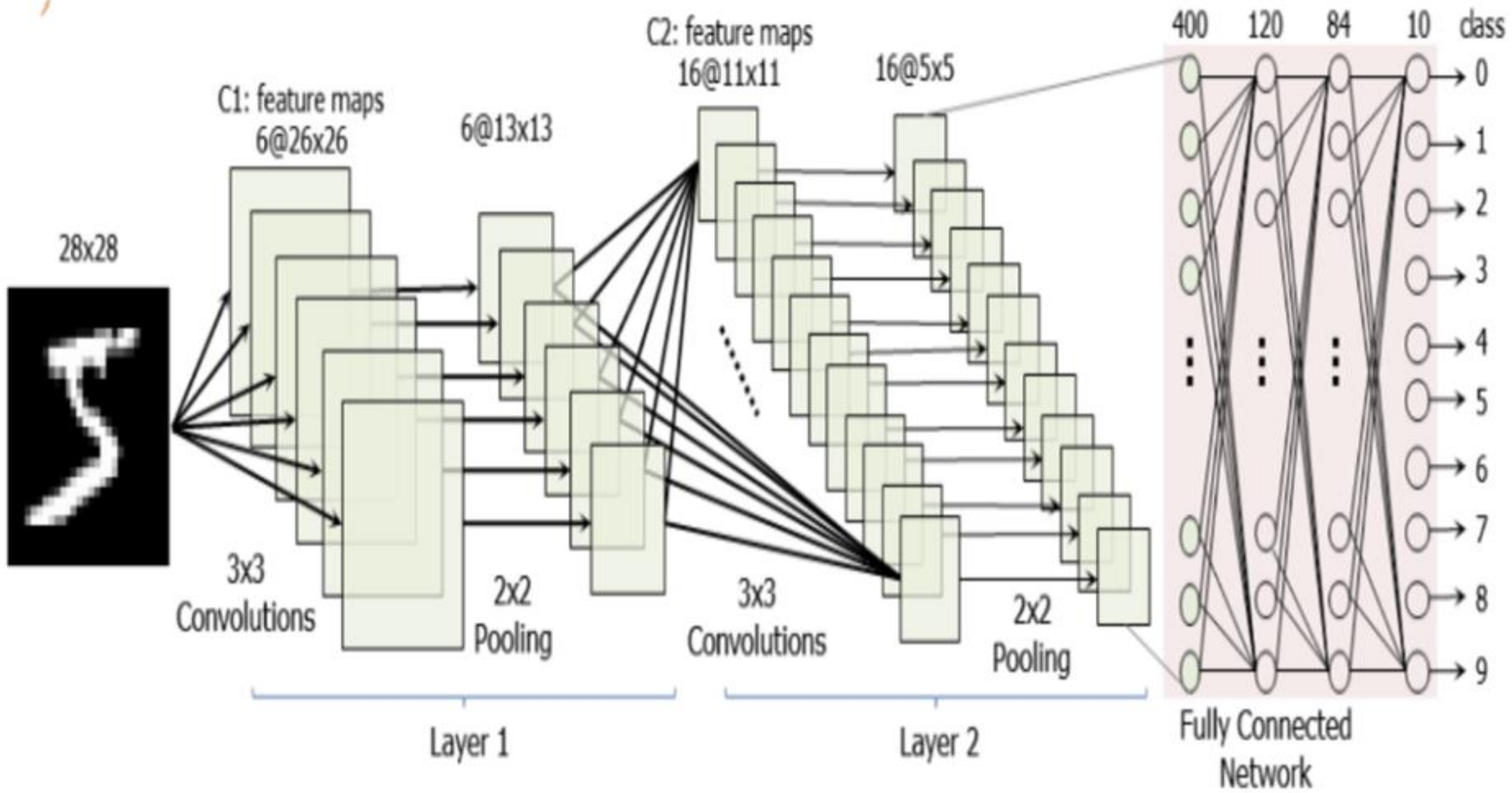
- Grabs first 28x28 image that is stored in a .bif file(binary input file)
    - This is the input map
- Contains 6 feature maps
- Each feature map has a distinct 3x3 kernel to extract certain features from the image
- 6 Narrow 2D convolutions will be done, creating 6 26x26 feature maps
- Bias and Rectified Linear Unit(ReLu) are applied to each pixel
- Max pooling is applied to only retain important features, outputting the result: 13x13

Note 1: The ReLu function converts any negative input (x) to zero to introduce nonlinearity so the network can learn complex patterns
Note 2: Bias ensures that if a feature is absent, a given neuron can still be active

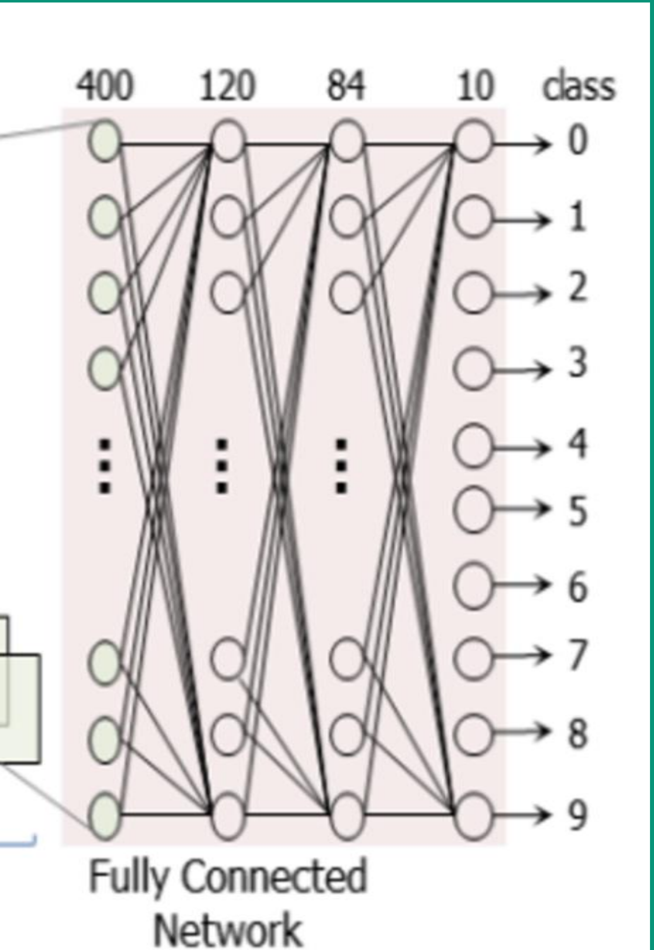
Second Layer

- Takes the 6 pooled outputs from first layer
    - These are the input maps
- 16 feature maps
- 96 kernels(6 for each feature map)
- Convolution will be done to the 6 input maps with their distinct kernels
- Once each input map is convoluted, they are summed together creating one feature map
- Bias and ReLu are then applied to each pixel
- Lastly max pooling is applied result: 5x5

Note 3: The convolution descibed is cross convolution since CNNs do not flip the kernels



28x28
C1: feature maps
6@26x26
6@13x13
C2: feature maps
16@11x11
16@5x5
400
120
84
10
class
0
1
2
3
4
5
6
7
8
9
3x3
Convolutions
2x2
Pooling
3x3
Convolutions
2x2
Pooling
Layer 1
Layer 2
Fully Connected
Network



Fully Connected Network

- Fully Connected Network Consists of 3 layers
  - 1st layer interprets the features extracted by the convolutional layers
  - 2nd layer narrows down the most important features
  - 3rd layer serves as the decision making component giving us 10 classified outputs

*Each of these layers have distinct weights and biases

Classified Outputs

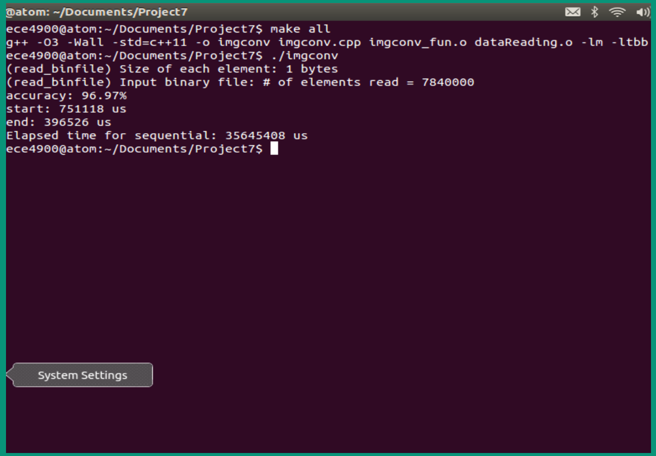
- The FCN outputs a vector containing 10 elements, each corresponding to a class label from 0-9
- These elements report the networks confidence scores for each of the 10 possible classes
- The class with the highest confidence score is considered the predicted class by the FCN

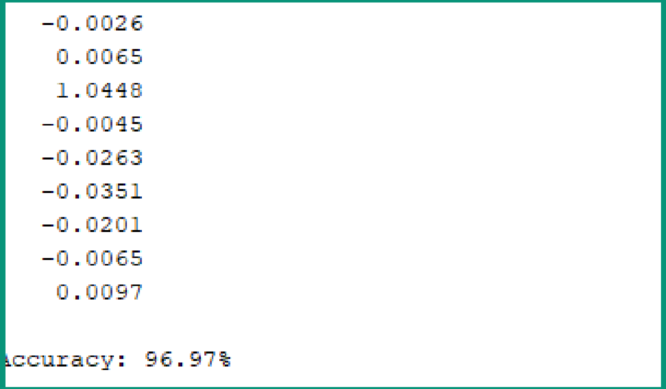
```
ece4900@atom: ~/4772
end: 290379 us
Elapsed time for Parallel_For: 4915 us
ece4900@atom:~/4772$ make all
g++ -O3 -Wall -std=c++11 -o imgconv imgconv.cpp imgconv_fun.o dataReading.o -lm
-ltbb
ece4900@atom:~/4772$ ./imgconv
(read_binfile) Size of each element: 1 bytes
(read_binfile) Input binary file: # of elements read = 7840000
0.019937
-0.002645
0.006472
1.044811
-0.004536
-0.026347
-0.035109
-0.020069
-0.006500
0.009695
Image 0: Predicted Class = 3, Expected Class = 3
accuracy: 0.01%
start: 957460 us
end: 963077 us
Elapsed time for Parallel_For: 5617 us
ece4900@atom:~/4772$
```

Initial Computation time/Accuracy

- The CNN’s accuracy was confirmed to be 96.97% meaning 9697/10000 datasets were accurately guessed by the CNN
- Accuracy was compared to matlab implementation of CNN
- The elapsed time was found to be 35.6 seconds on average

Elapsed Time





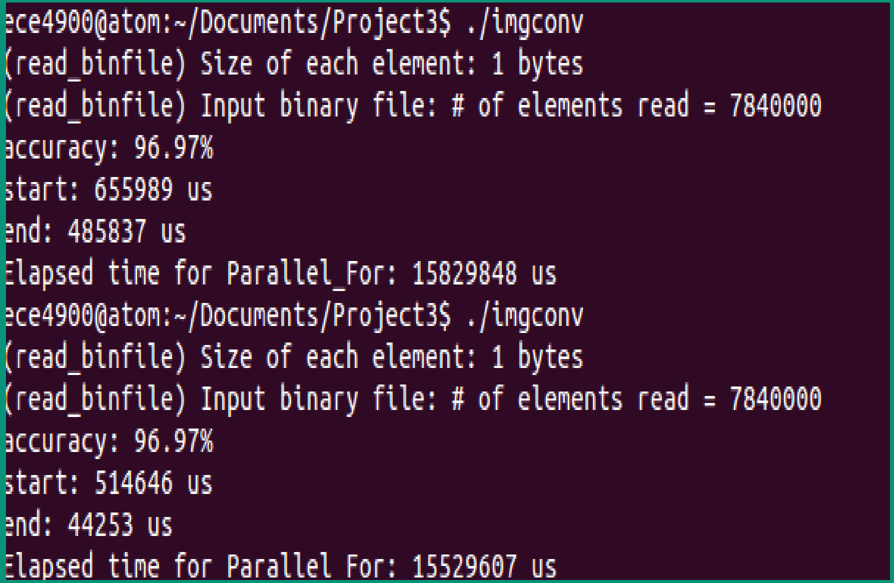
Opportunities for Parallelization

Parallel_For

- Since we are dealing with a large dataset with 28x28*10,0000 elements, parallel_for prevents the linear increase in computation time
- Parallel_for would also be beneficial in the second layer, since 16 feature maps need to be produced
- 2.28x increase cutting computation time in half

* Could theoretically use parallel for in the 2d convolution calculation itself but the input images and kernels are far too small to see any benefit

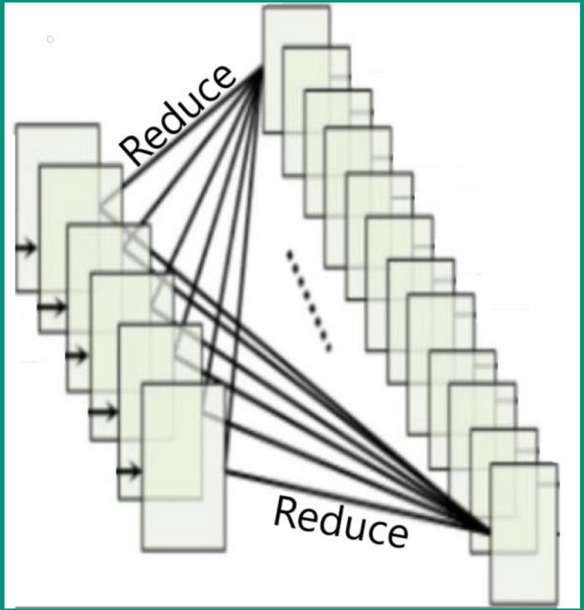
Elapsed Time



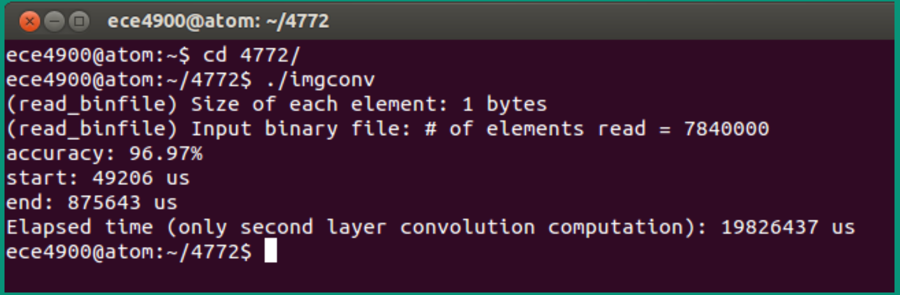
Avg. Time: 15.55s

Parallel_Reduce

- Implemented with parallel_for
- Used to sum convolutions in the second layer
- Each convolution is 11x11 or 121 elements
- There are 6 convolutions total for each parallel_reduce call
- Early implementations were slower because not enough elements were added



Elapsed Time



Avg. Time: 19.17s

(3-stage) Parallel_Pipeline

- In the pipelined implementation 3 stages were used
  - Stage 1 – Image loading: Continuously loads images sequentially feeding them into the pipeline without waiting, allowing for a constant stream of data
  - Stage 2: - Convolutional Processing: This stage applies the first and second convolutional layers in parallel and processes multiple images simultaneously to extract primary features
  - Stage 3: FCN stage: This stage operates in parallel to compute the final outputs

Elapsed Time

```
ece4900@atom:~/Documents/Project6$ make all
g++ -O3 -Wall -std=c++11 -o imgconv imgconv.cpp imgconv_fun.o dataReading.o -lm -ltbb
ece4900@atom:~/Documents/Project6$ ./imgconv
(read_binfile) Size of each element: 1 bytes
(read_binfile) Input binary file: # of elements read = 7840000
accuracy: 96.97%
start: 761433 us
end: 874473 us
Elapsed time For 3-stage Parallel_Pipeline: 16113040 us
ece4900@atom:~/Documents/Project6$
```

Avg. Time: 16.11s

(4-stage) Parallel_Pipeline

- In this pipelined implementation 4 stages were used
  - Stage 1: Image loading: Continuously loads images sequentially feeding them into the pipeline without waiting allowing for a constant stream of data
  - Stage 2: Processes the 1st layer to extract the primary features like normally
  - Stage 3: Processes the 2nd layer
  - Stage 4: Consists of the fully connected network to get the predicted outputs of the CNN

Elapsed Time

```
ece4900@atom:~/Documents/Project5$ make
g++ -O3 -Wall -std=c++11 -o imgconv imgconv.cpp imgconv_fun.o dataReading.o -lm -ltbb
ece4900@atom:~/Documents/Project5$ ./imgconv
(read_binfile) Size of each element: 1 bytes
(read_binfile) Input binary file: # of elements read = 7840000
accuracy: 96.97%
start: 714494 us
end: 386009 us
Elapsed time For 4-stage Parallel_Pipeline: 16671515 us
ece4900@atom:~/Documents/Project5$
```

Avg. Time: 16.67s

Implementations	Elapsed times(s)
Sequential	35.5
Parallel for	15.55
Parallel reduce	19.17
Parallel pipeline(3-stage)	16.11
Parallel pipeline(4-stage)	16.7

All times

Further Improvements

- In the parallel for implementation this code here takes place after the convolutional layers and FCN are finished,
- variable correct_predictions is being updated in the parallel loop
- This causes a potential race condition because multiple threads could try to update the thread concurrently
- To avoid potential race condition, use atomic operations

```
softmax(z3i,z3i,10);
int local_correct_predictions = 0;

 int predicted_class = argmax(z3i,10);

 if(predicted_class == expected_outputs[m]){
     local_correct_predictions++;
 }
 //printf("Image %d: Predicted Class = %d, Expected Class = %d\n",m,predicted_class,expected_outputs[m]);
    free(tempinput);free(pooledOutput); free(FCNi);free(z); free(z2i); free(z3i);
        correct_predictions += local_correct_predictions;
});
```