ID5130 Course Project Report

Parallelised Neural Networks for Image Classification using OpenACC

Raghav Mallampalli, MM18B109
Abhimanyu Swaroop, MM17B008

May 25, 2021

1 Abstract

Image classification algorithms have a several use-cases in industry and academia, seeing application in microscopy, biotechnology, medicine, astrophysics, materials engineering etc. The industry standard is to use a deep learning (neural network) based model to classify images. An example is given below.[1]

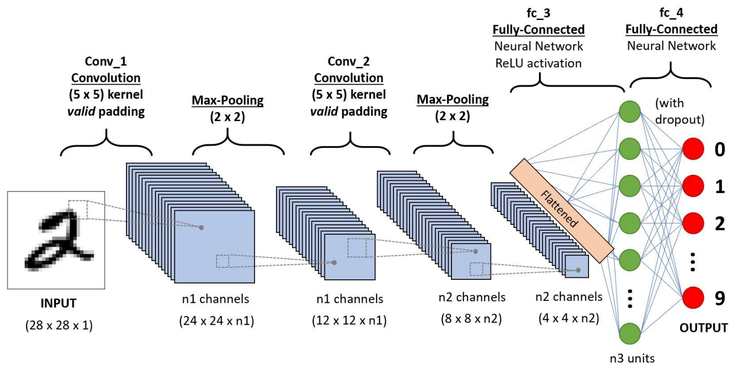

Figure 1: Example deep learning model architecture

However, for a low visual information density problem like microstructure classification or number classification, convolution and a “shallow” neural network should suffice to obtain reasonable accuracy.

[1]Sumit Saha: A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, medium post

2 Introduction

We use convolution, pooling and a dense neural network in sequence for the task of classifying microstructures. Our chosen dataset is the NIST Ultra High Carbon Steels micrograph dataset. The microstructure type (pearlite, martensite, etc) is the label to be classified into. 5766 images were extracted from 961 micrographs and the model classifies them into one of the 7 microstructure type labels.

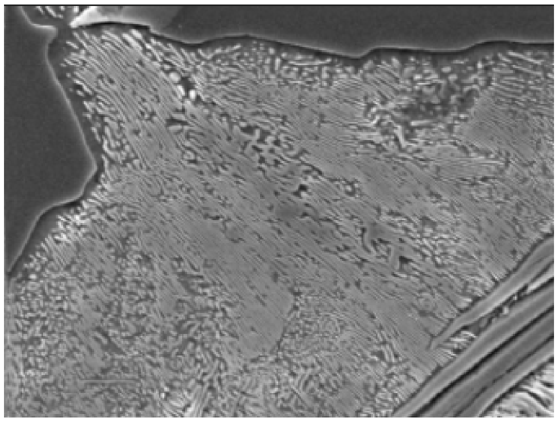

Figure 2: Sample micrograph from dataset

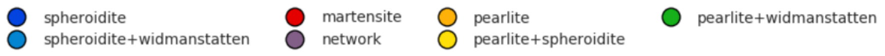

Figure 3: Labels

OpenACC GPU parallelisation will be carried out.
Time performance profiling and test accuracy will be used to judge the implementation. A successful implementation would yield identical or sufficiently similar results in a lesser amount of time for the entire gamut of training operations.

3 Model architecture

3.1 Convolution kernels

Sobel: The Sobel kernels to be used in the edge detection convolution are:

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

They detect edges along vertical and horizontal direction respectively.

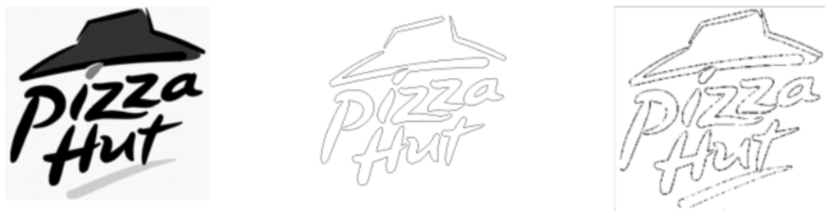

Figure 4: Comparison of algorithms: Clockwise; original image, Sobel algorithm, Fuzzy relative pixel algorithm

Sharpen: The standard sharpen kernel was used. It will increase the contrast of the detected edges in a lower dimension representation of the image.

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & +5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Manual: A manually tuned kernel was also used. The form of the manual kernel turned out to be similar to the contrast kernel for best performance. Note that only centre symmetric kernels were explored for the tuning. It is possible that greater performance can be obtained if more eccentric kernels are used.

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Neural Network: The neural network consists of one hidden nodes and one output layer. The input for the neural network is obtained by pre-processing the images using the above mentioned kernels. The output layer consists of 7 nodes (each corresponding to a class label) and the number of nodes in the hidden layer can be varied according to the need of the user.

The neural network uses softmax activation function to compute the "belongingness" of a particular image to all classes. The vector output by the softmax function sums to 1 and can be mathematically described as,

$$y_i(z_i) = \frac{\exp z_i}{\sum \exp z_i}$$

Cross-entropy loss function is used to calculate the loss for any given iteration as it is known to outperform other cost functions for a multi-class classification problem. Cross-entropy is calculated by adding the product of true probability and negative log of the predicted probabilities. Mathematically speaking,

$$H(y, \hat{y}) = -\sum y_i log(\hat{y}_i)$$

4 Implementation notes

Extracting input images from the micrographs was done using Python. The images were saved into space separated files (1 for each image). One hot encoding was created for the labels and it was stored as a 5766*7 array (each column represents the class - 1 implies image belongs to that class, 0 implies it does not).

Pre-processing was done on an image-by-image basis. The file was opened and saved into an array. Once processed, the data was saved into another array.

To ensure fair bench-marking, only arrays were used for both the serial and parallel versions. No libraries were used apart from the built-in C ones. The time taken to load the data is included in our computations since it is identical for serial and parallel and makes no difference.

5 Observations

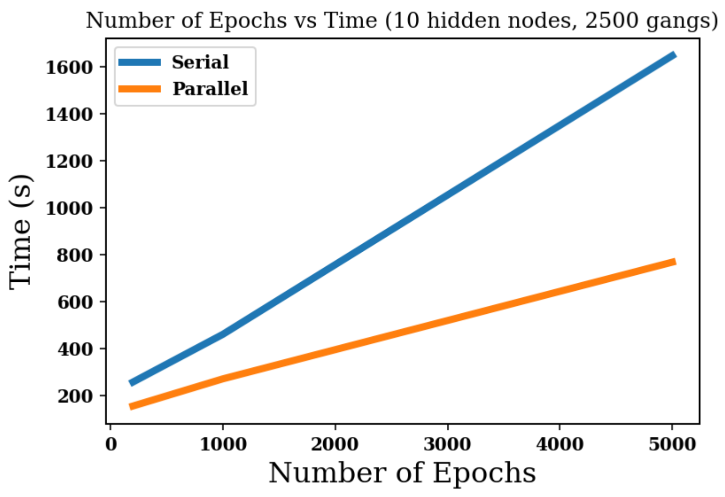

Figure 5: Comparison of serial and parallel run time with varying number of epochs

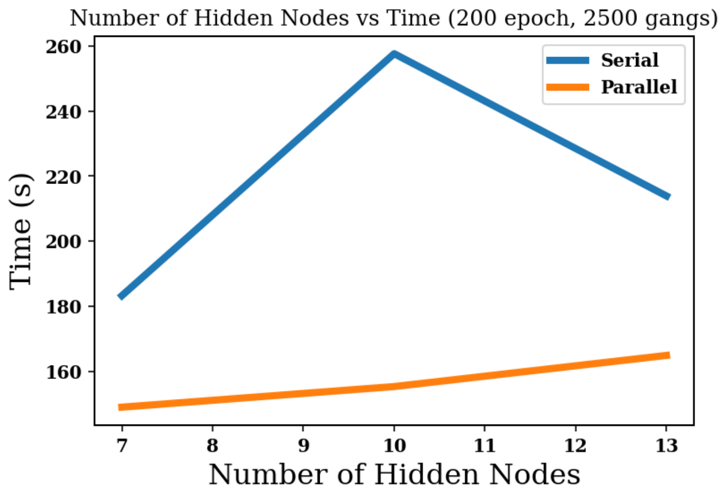

Figure 6: Comparison of serial and parallel run time with varying number of nodes in the hidden layer

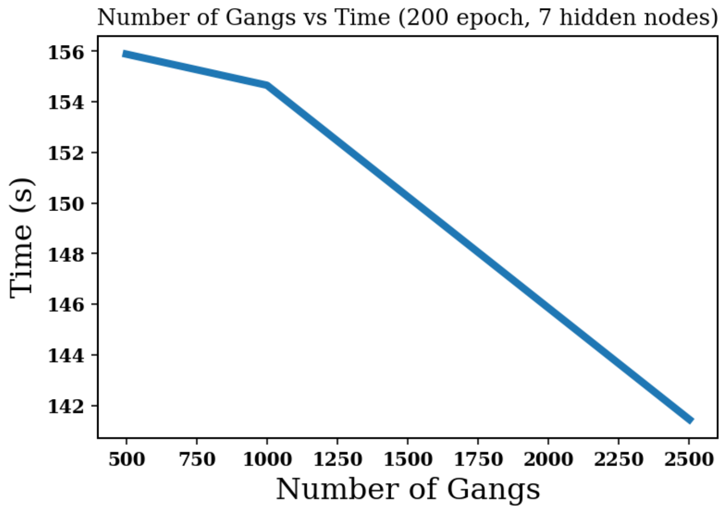

Figure 7: Illustration depicting the parallel run time with varying number of gangs

Maximum accuracy of approximately 39% was obtained for all tested neural network architectures and maximum epoch combinations. The accuracy obtained in the serial and parallel mode of the code was the same giving confidence that the parallel execution of the code is correct. The figures above show the variation of execution time for the code in different parameter settings. The figures also compare the parallel execution time with the serial execution time.

6 Inferences

- The implemented architecture was found to be highly parallelisable, with the parallel version performing better than the serial for all combinations of tested parameters.
- For high epoch and hidden layer nodes, the speed-up was ≈ 3.
- The accuracy saturation at a low epoch number seems to point out that the pre-processing is not carrying over the entire information present in the image.
- A possible next step would be do reduce the number of pooling layers or increase pooling stride to allow greater information pass-through.
- Obtaining maximum ($> 95\%$) accuracy will likely require a full convolutional neural network.
- Further scope for parallelisation: It may be possible to parallelise the pre-processing code over the input_size loop and the label loading loop, however it was avoided as it would result in the same file pointer attempting to read the same file at different locations.

7 Conclusion

The encoded architecture has performed satisfactorily. Scope for parallelisation was explored to the extent possible and methods not explored were listed. Parallelisation greatly improved performance.