Machine Learning and Neural Net (COMP-5011-FB)

Final Project Report - Topic 1

Amirmohammad Shahbandegan (1172613)

Aditya Singhal (1154832)

10 December 2021

Problem statement

The main idea of this project is to work with CUDA to speed up the classification of the provided dataset while using ELM as the classifying algorithm.

Data Preparation

Caltech101

- 9144 images (3,030 for training and 5,647 for testing)
- classes: 101

Caltech256

- 30607 images (7,680 for training and 22,100 for testing)
- classes: 256

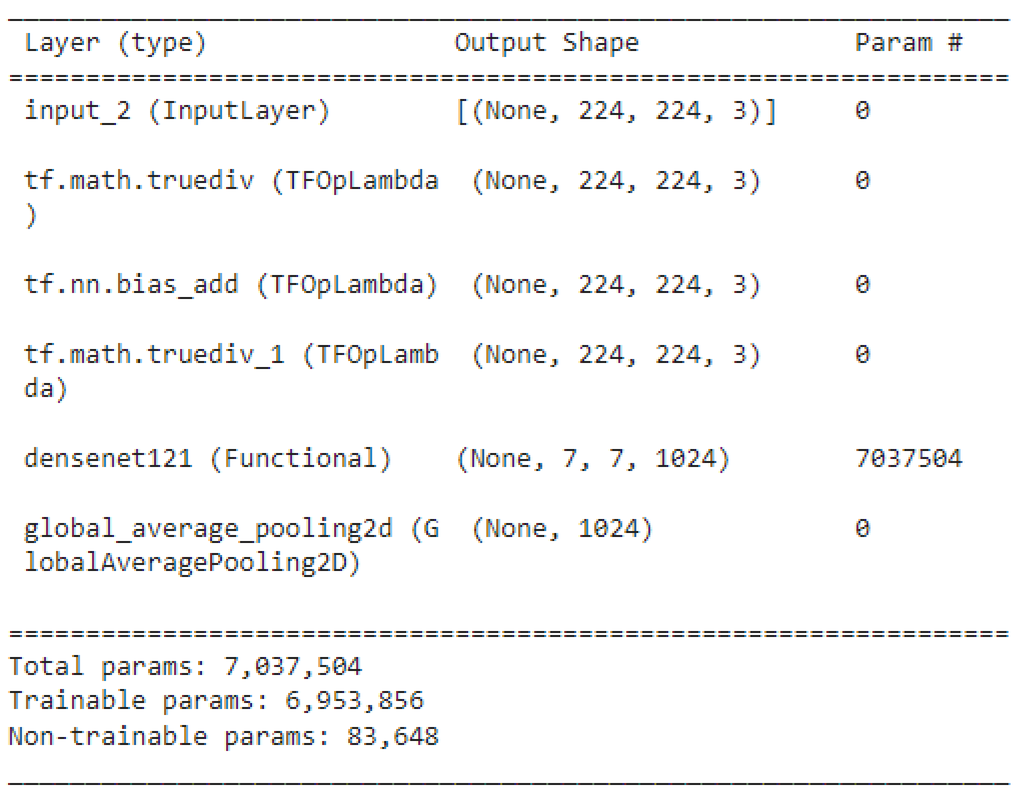
Algorithm

Step 1: Transfer learning and Feature combination: Each image dataset in the dataset is converted to a numerical feature vector. This stage is done using Python and stored in a file for later use in the CUDA C++ code.

Step 2: ELM-based classifier (CUDA C++ code) to get the final performance. All of the training and prediction are done in this part.

Methodology

- Open Google Colab.
- Load respective dataset
- Preprocess the data (Resize the images to 224x224 which is the same as DenseNet input)
- Divide in train-test set: Build the (X,y) pairs
- Create feature vectors using 'Densenet101' with ‘imagenet’ weights

- 

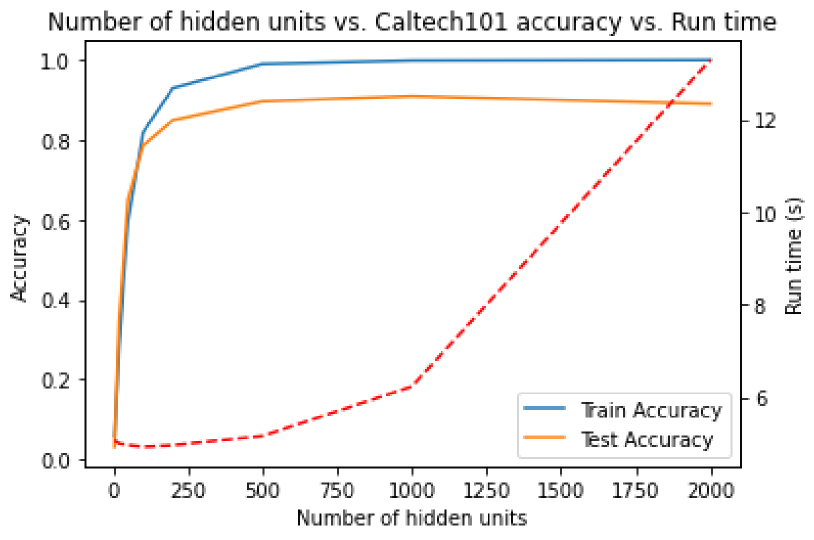
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.math.truediv (TFOpLambda  (None, 224, 224, 3)      0
 )

 tf.nn.bias_add (TFOpLambda)  (None, 224, 224, 3)      0

 tf.math.truediv_1 (TFOpLamb  (None, 224, 224, 3)      0
 da)

 densenet121 (Functional)    (None, 7, 7, 1024)        7037504

 global_average_pooling2d (G  (None, 1024)             0
 lobalAveragePooling2D)

=================================================================
Total params: 7,037,504
Trainable params: 6,953,856
Non-trainable params: 83,648
_________________________________________________________________
```

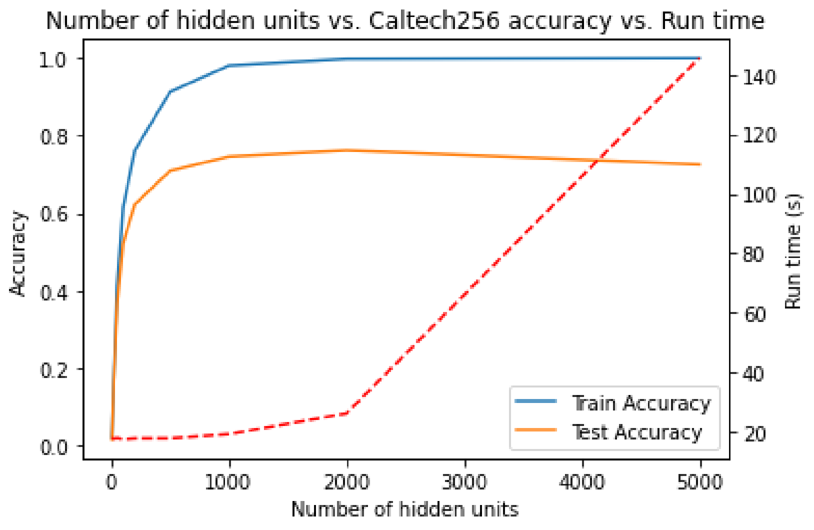
- Encode feature vectors generated as CSV and save them
- Install CUDA Version 11.2
- Implement CUDA Extreme Learning Machine (ELM) (Functions are implemented using CuBlas library with GPU memory
- The following functions are implemented using CUDA to build the ELM
  - Matrix multiplication
  - Matrix inverse
  - Moore-Penrose inverse
  - Sigmoid activation
- The results are stored in a CSV file and then analyzed using a simple python script

Experimental Results

Hyperparameter tuning

To find the best number of hidden units in the ELM, different values for this parameter are tested. The following charts show the results of these experiments.





As shown in the above tables, the best accuracy is achieved with 768 and 2000 hidden units for Caltech101 and Caltech256, respectively.

Table 1: Caltech 101 (768 hidden units)

S.No.	**Time**	**Training Accuracy**	**Testing Accuracy**
1st run	05.58s	99.99%	94.51%
2nd run	05.61s	99.14%	94.12%
3rd run	05.48s	99.63%	94.43%

Average Testing Accuracy = **94.35%**

Table 2: Caltech 256 (2000 hidden units)

S.No.	**Time**	**Training Accuracy**	**Testing Accuracy**
1st run	30.32s	99.96%	76.61%
2nd run	29.94s	99.81%	76.93%
3rd run	26.31s	99.79%	76.47%

Average Testing Accuracy = **76.67%**

Average top-1 accuracy of object-centric Image recognition task is

(94.35+76.67)/2 = 85.51%