# Project#1 Final Report

Team Member 1 Shivam Aggarwal (2016195)

# 1. Application and Convolutional Neural Network (CNN) Architecture Overview

Describe your chosen application and CNN architecture briefly. Provide appropriate reference(s) as well.

Application Area: Computer Vision

Object classification is a standard computer vision problem. CIFAR-10 as described below is well tried and tested dataset for the task. I initially chose Binary Neural Network based VGG-16/19 networks. However, due to their large volume and dependency on the BatchNorm layer, I switched to a conventional VGG-13 network without any batch-normalization or dropout.

CNN architecture: VGG-13

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

Dataset: The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

## 2. Training of Model and Key Results

Describe training process followed by you and key relevant results.

You should provide references for any implementation that you might have used for transfer learning. Do mention relevant framework(s)/libraries (Keras or any other) and any accuracy improvement technique(s) that you used, e.g. changing the architecture, hyperparameter tuning.

Also provide following key relevant information and results obtained during training process in the following format: -

**Dataset Information**

| **Dataset** | **CIFAR-10** |
|---|---|
| **Target number of classes** | 10 |
| **Training dataset details** | **5000 training images per class** |
| **Test dataset details** | **1000 test images per class** |

**Model Information**

(Expand number of rows and support with complementary diagram if it helps)

2D Convolution layers are as described above. I have trained various different iterations of VGG based networks on CIFAR-10

**Key Results**

**Standard Results referenced from various resources**

| Accuracy | Network | Precision |
|---|---|---|
| 82.9 | VGG16 | Ternary Values |
| 86.71 | BNN | Binary |
| 90.10 | BinaryConnect | 32 bit floating points |

**Results obtained through training from scratch**

| | |
|---|---|
| **87.98** | **VGG13 - Without BatchNorm** |
| **92.1** | **VGG13- Using BatchNorm** |
| **86.4** | **BNN-VGG19** |

# 3. OpenCL Implementation of Trained Model

### (3.a) Implementation details

Modules implemented:

Linear: Implements matrix multiplication operation between weights and input layers. Gives output after applying HardTanh operation.

BatchNorm1D: Standard 1d batchnorm function

Conv2D: Using output data decomposition to parallelize the convolution operation between input and weight kernels.

MaxPool2D: Similar to Conv2D

LogSoftMax: Each work-item calculates the total sum of exponentials followed by its respective output.

Support your design choices with appropriate quantitative results where applicable.

### (3.b) Functional results

**Results comparison**

| **Test accuracy using model training framework/libraries** (from section 2 above) | **Test accuracy achieved with OpenCL implementation** |
|---|---|
| 87.98 | 85.2 |

### (3.c) Performance results

Mention key runtime performance results that you achieved during performance optimization process.

At least, final runtime performance results for your optimized and correctly functional implementation should be provided.

I tested my results on NVIDIA GeForce 1050M. It is a 4 GB mobile desktop GPU. Since my architecture has around 10 Conv2D layers and 3 FC layers, it took around 0.47220 seconds to run inference on a single image.

## 4. Extra work (for bonus points eligibility)

Describe any relevant technique that you applied and was not discussed (at all or in detail) in the class. Do justify application of any such technique with appropriate quantitative results.

- I used 3D images to store my inputs and weights. Utilizing output data decomposition to calculate the output products in parallel.

| | |
|---|---|
| Image support | Yes |
| Max number of samplers per kernel | 32 |
| Max size for 1D images from buffer | 134217728 pixels |
| Max 1D or 2D image array size | 2048 images |
| Max 2D image size | 16384x32768 pixels |
| Max 3D image size | 16384x16384x16384 pixels |
| Max number of read image args | 256 |
| Max number of write image args | 16 |

- By looking at the specification of my device, I decided to store my weights in (KxK)xCinxCout format while input is stored in MxNxCin format.
- Moreover, I avoid external padding in convolution operation by defining the indices carefully.

## 5. References: -

[1] Binarized Neural Networks: Training Neural Networks with Weights andActivations Constrained to +1 or −1

[2] A Review of Binarized Neural Networks

[3] CIFAR-10 dataset

.

.