**Chapter 7**

**Experiment and Result**

This chapter is about description of proposed experiment and results. In the first part is software tools in this study. Next part is description of software and hardware configuration of testing equipment. Next phase preparation of dataset is described. The last portion of this chapter is dedicated to specifics of implementation and results.

**7.1 Software:**

In order to select suitable tool for implementation of CNN for classification, search of available software tools and libraries was conducted. Now days there are different software tools available for machine learning. Some of these are universal tools for machine learning, but some are exactly designed for deep learning. For the last decade the software tools for machine learning has undergone a renaissance. There is wide selection of them available and new tools are announced quite frequently. For example, Caffe21 was presented very recently on April 18th. Almost every frequently used programming language has either some software addon library or at least some available Application Programming Interface (API). The choice of the software tool was influenced by several aspects. Firstly, the implementing language had to be well known and somewhat majority. Enough of available learning documentation materials had to be available, preferably in form of videos. The most significant factor was good support for learning on GPU.

**Theano**

It is tested python library. Designed to describe, optimize and evaluate mathematical expression with multi-dimensional arrays. This makes it suitable for machine learning needs. Theano is made on top of Numpy, which is python module that enables effective operation with tensors and basic image processing procedure. Mixture of Numpy and Scipy brings rich set of tools for image processing and data processing. Its abilities can arguably rival MatLab, while being open source and free of cost. Theano’s major rival is currently Tensorflow development. One of the drawbacks of Theano is its low-level nature. Development of machine learning algorithms directly can be very complicated. This is maybe reason it slowly falling by the way side. This is also the reason why Theano as a tool is not fit for direct implementation of CNN models.

**Tensor flow**

Tensor flow is similar tool like Theano. As the name suggest this tool is focused on effective work with tensors. It was originally created for internal use in Google for machine learning project, but it was lunched as open source in 2015. Tensor flow calculations are expressed as stateful dataflow graphs, which enables efficient support for GPU supported computation. Tensor flow is currently advertised as one of the fastest frameworks for deep learning needs. Its drawback is like Theano, in the fact that it is very low level and direct usage for operation of Deep learning models is not perfect.

**Caffe**

Caffe is a deep learning tool that goals to be modular and fast. It is created by Berkeley AI Research and by community contributors. Yangqing Jia developed the project during his PhD at UC Berkeley. C++ is programming language is used to implemented, but it also available API for several other languages as for example python. Its main drawback is its lack of quality documentation and material. This fact is partially improved by the existence of Model Zoo, which is collection of favorite models that are available freely. Caffe was in the last years used by Facebook for example mainly because its performance capabilities. Caffe is more geared towards the development of large production application than it is for study purposes.

**Keras**

Keras is new software for machine learning but developed project written in python. It is high lever neural network API. It is built capable of running on top of either Theano or Tensor flow libraries. It is very simple with emphasis on quick model development. It is very simply extensible. At present time Keras has one of the largest communities among similar tools for deep learning. It has very good documentation and materials which containing many code demonstration and other resources that help users to get started very rapidly.

**7.2 Hardware and Software Configuration**

Training of Neural Networks notoriously computational expensive and it required a lot of resources. From bottom level perspective it translates into many multiplications of matrices. Modern Central Processing Units (CPUs) are not made of such computations and therefore are not very efficient. On the other hand, modern GPUs are designed to preform exactly these operations.

At present on the market there are two main parallel computing platforms CUDA and OpenCL. They both have their own advantage and disadvantage, but the major difference is that CUDA is proprietary, while OpenCL is available free. This divide translates into hardware productions as well. CUDA is mostly supported by NVIDIA and OpenCL is support by AMD. NVIDIA with its CUDA platform is presently leader in the domain of deep learning. Therefore, for training of CNN models was selected GPU from NVIDIA. Selected training model was GIGA BYTE GeForce GTX 1080. Details information about hardware configuration is in Table 7.1.

Table 7.1: Hardware Configuration

|  |  |
| --- | --- |
| GPU | GeForce GTX 1080 4GB |
| CPU | Intel(R) Core(TM) i7-8550 CPU @ 2.00GHz |
| Memory | DIMM 1333MHz 8GB |

From the list of considered software tool was selected Keras. The reason being that Keras satisfied all consideration factors and because it was written in python which was most aware to the me. Support of efficient GPU in Keras is relying on either Tensor flow or Theano back-end. From the different user perspective, it doesn’t really mater either way, but Tensor flow was selected because it was observed as faster of the two. GPU-accelerated library package of primitives for deep neural networks. Details information about software configuration is brief in table 7.2

Table 7.2 Software Configuration

|  |  |
| --- | --- |
| Keras | 2.04 |
| Tensorflow | 1.1.0 |
| CUDA | 7.5 |
| Python | 3.53 |
| Operating System | Window 10 |
| Open CV | 2.0 |

**7.3 Model Structure:**

In this study We applied Keras sequential model for CNN, which is a concept that is suitable for modeling of feed forward network. Definition of the network is made of layers. Concept of layer in Keras sequential model doesn’t fully map into already described definition of layer from topological viewpoint. Keras layers are finer grained and in order to create equivalent topological layer it is essential to use multiple Keras layers. Model is created simply by calling Sequential constructor

from keras.models import Sequential

model = Sequential()

Layers are added by calling an add method on object of sequential model

model.add(NumbersOfLayer),

where NumbersOfLayer is definition of the layer.

**7.3.1 About Keras Layers**

All models were created by configuration of following layers.

**Convolutional layer**

Convolutional layer architecture was of following structure

*Conv2D(filters=num, kernel\_size=(x, x), strides=(s,s), padding=’valid’, input\_shape=shape)*

where num is number of filters that the layer will have, x is size of kernel, s is number of pixels in stride and input\_shape describes size of input matrix.

**Activation Function**

To create activation function on the output of the layer user can require parameter activation of the layer itself or create activation as a layer

*Activation(acitvation\_function)*

where activation\_function cab be ’softmax’ or ’relu’. Both specifications are equal because Keras automatically applied linear activation function for each layer.

**Pooling Layer**

Pooling layer can be definite as

*MaxPooling2D (Pool\_Size=(z, z), strides=(s, s))*

where Pool\_Size requires size of pooling kernel and strides requires number of pixels in vertical and horizontal direction that are traversed in between application of individual pools.

**Fully Connected Layer**

Fully connected layer is created using below function

*Dense(total\_units)*

where total\_units is total number which is a fully connected neurons in specific layer.

**Dropout**

Similarly, to activation function to apply a drop out regularization on a layer it wants to be added after it as another layer.

*Dropout(prob)*

where prob is both probability that any unit is dropped and also the coefficient by which are the outputs multiplied through forward evaluation.

**Other**

Feature extraction layers are n dimensional. Specifically, Convolutional and Pooling layers are 2D (two dimensional). Classification layers that are created by fully connected layers are 1D (one dimensional). To join the two, it is required to create mapping between them. For this purpose, it needed to use following layer

*Flatten()*

which takes care of necessary connections between these layers.