Design Document

Deep Learning on Embedded Platform

Version 1.0

Prepared by Christopher Johnson, Luay Alshawi, Gabe Morey

CS 461 Fall 2016

November 29, 2016

In this document we describe the design of a deep learning system developed to learn how to play the arcade game Galaga. Galaga is an arcade shooter which was released in 1981 by Namco [1]. This system is designed with the ultimate goal of being turned into a course for the NVIDIA Deep Learning Institute. The documentation will outline exactly what hardware was used, how the system was put together, and what methods were used in a way that allows others to recreate this project.

Contents

1 Overview

1.1 Purpose

The main aim of this document is to give others a thorough enough understanding of the project and its design. The reader of this document should walk away able to understand how the system works and why it was designed this way. The document must be thorough enough for NVIDIA to recreate the project and package the design into a training course on deep learning.

This project is to be used to produce a training course for NVIDIA's Deep Learning Institute. After the project is complete it will be handed off to NVIDIA for the purposes of accomplishing this goal. To that end the documentation has been designed to provide readers a clear path towards building this system themselves.

1.2 Scope

This document outlines the design of a system for teaching a deep learning neural network how to play the game Galaga. The sections of this document will outline the system details, setup, and all things necessary to understand how and why the project is designed. Hardware required, API's used, and setup will be detailed as part of the project design.

1.3 Context

This system is not designed for an average computer user. This system is a showcase of possibilities when working with Neural Networks. As such, no real UI has been developed for the purposes of this project. Average people may see the result of the project by viewing the network play Galaga.

This system is designed for developers. Developers should be able to do a project like this as an introduction to deep learning neural networks. The documentation provided here reflects this focus and is written with developers in mind.

1.4 Summary

The neural network created for this project will be able to play Galaga. Its design is based on GoogLeNet, a convolutional neural network designed in coordination by people at Google Inc., University of North Carolina, Chapel Hill, University of Michigan, Ann Arbor, and Magic Leap Inc. [2]. This neural network will undergo two main phases of training. One with just images, and a second hooked up to the game Galaga. The system will run inference on the Jetson TX1 developer kit, which will interface between a computer playing

the game with a controller. Cloud computing GPUs will be used to handle the meat of the training. In total, the system should be able to learn how to play Galaga, and then continually improve as it plays.

2 Definitions

3.1 Neural Network: A multilayered graph structure containing nodes, referred to as neurons, and weights that affect how likely the node is to be chosen when confronted with a decision. The weights of nodes are decided through training the network, and a traversal of the graph acts as a large chain of decisions as the neural network is confronted with situations.

3.2 Deep Learning: Using a layered neural network constructed by GPU processing to teach a computer how to accomplish a specific task.

3.3 Convolutional Neural Network: A neural network specifically designed to take images as input. It uses convolutional layers to process the image into a more manageable chunk of data and extracts the information relevant to the network.

3.4 Caffe: A code framework that allows for complex mathematical operations that are required for training a neural network.

3.5 API: Application Programming Interface

3.6 Jetson TX1: Quad core embedded system designed for power efficiency and deep learning projects

3.7 OS: Operating system

3.8 OpenCV: (Open Source Computer Vision) is a library that can be used with different languages(C, C++, Java, Python, etc.). It provides standard functionalities such as image capture, Faces recognition, Gesture recognition, Motion tracking, Mobile robotics, Object identification and Image manipulation.

3.9 Galaga: Galaga is an arcade video game released in 1981 by Namco [1]. The player takes control of a small space ship and must shoot at and dodge waves of alien space ships. Victory in a level is achieved by destroying all enemy ships.

3 Design

3.1 Introduction

This section lays out several viewpoints from which we have approached the design of this project. Each component will be matched under an appropriate viewpoint that gives the reader a clear idea of how the design was apparoached. Each subsection details a different set of components, many of which will be lumped together, which constitute a major piece of the final design. There are three main viewpoints from which we have approached the design:

- Developer Viewpoint: Design elements approached from the eyes of a developer trying to recreate this project.
- Data Viewpoint: Design elements approached from a perspective of data flow operations and data processing.
- Hardware Wiewpoint: Design elements approached from the perspective of the hardware level. Particularly focused on hardware setups required for the rest of the process.

Each viewpoint is covered in more depth in its respective section.

3.2 Development viewpoint

3.2.1 Set up image dataset

OpenCV is a useful tool for this deep learning project. It will be used as a helper tool to prepare the images for each object in the game such as players, enemies, and bullets. A script will be written in python which will use an OpenCV to crop, resize, and filter the game images when needed. The goal is to have a full set of the game images and their labels which will be used to train the neural network. However, using OpenCV at this phase is optional, but due to the need of fast image manipulation, OpenCV can be used to do this task easily as it will help to save time and efforts. Although, collecting objects images can be done manually, eventually, OpenCV will be used to resize the collected images to a consistent size.

3.2.2 Neural network setup

The neural network set-up itself is a multi-layered combination of components. At the basic level of design, our neural network follows the architecture known as GoogLeNet. GoogLeNet is designed, as a convolutional neural network, specifically to process images

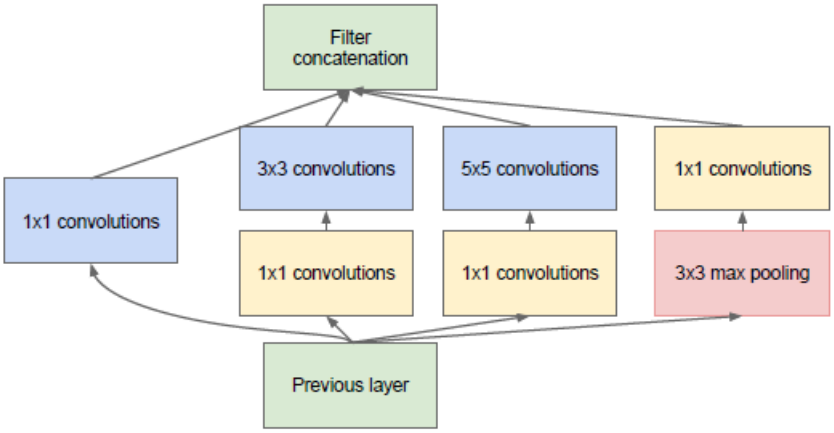

Figure 3.1: GoogLeNet Module

and categorize them. At the architectural level it has 22 layers, each consisting of special modules [2]. These modules have a structure similar to that outlined in figure 3.1.

The network set-up provided by GoogLeNet needs to be accessed with a framework, as well as modified slightly for this project's needs. For the framework this project uses Caffe. Caffe allows for command line instructions to be sent to define the network. To set up the network and Caffe properly we are using a program called DIGITS. DIGITS is a neural network set-up application designed and released by NVIDIA. It gives an easy interface for setting up the exact parameters for learning as well as building the network up. The exact parameters we need for the set-up of this net are unknown at this time and will be added when available.

Training of the neural network will be done on NVIDIA GPUs on the Amazon Web Services cloud. The network will run its calculations on the cloud and send its response down to the Jetson TX1. There the response will be interpreted into the proper action.

The structural goal of the neural net changes between two stages. In the first stage, the structure of the neural net is meant purely to teach it how to recognize and categorize in game objects from images. In the second stage, the neural network will be deployed on the Jetson TX1 and will take active game information. It will use that information to make one of three choices: move left, move right, or shoot. More detail on the training will be given in section X.X.

3.3 Data viewpoint

The data in this section consists of image processing of the game being played as well as the data of the trained neural network. This section will go in depth of both how image processing will be done as well as different ways to train and improve the neural network. A pseudocode will be included to give an overview of where these data will be used in this project. Also, the pseudocode will include a general algorithm of how the Jetson will be able to play the game Galaga using the image capture data and the neural network.

3.3.1 Image processing

After having a trained neural network, a software written in python will start by loading the necessary dependencies which are OpenCV, Caffe, and the trained neural network. At this step, its important to convert the given neural network to an object that OpenCV can understand. Then, a video capture of the game will take a place to record the monitor

where the game is being played using OpenCV's API [1]. Since videos are just consecutive of images, OpenCV will be used to process the images by comparing them with the trained neural network. The idea is to identify the location of each dangerous objects that would reduce the player's life so the Jetson TX1 can send a command to the game controller which moves the player away from the harmful object.

3.3.2 Neural training methods

The training of the neural network will be done in two stages. In the first stage, we will be using images created from the game to teach the network how to recognize in game objects. Given that GoogLeNet was designed with image categorization in mind, this set-up will be easy to create and bring to functionality. We will feed the network with images doctored by OpenCV and it will interpret them and label them. Once it has a high degree of success in correctly identifying in game objects, we will take the training to stage two.

In the second stage of training we will be feeding the network images directly from live game progress. The game will be running and the network will be deployed on the Jetson. The network will be given the ability to take actions based on the objects it identifies on screen. Positive feedback will be given to the neural network when it correctly kills an enemy or completes a level. Negative feedback will be given to the network when it dies, gets a game over, or loops too long doing nothing. As the neural network plays the game, it will improve performance and eventually be considered proficient.

3.3.3 Data visualization

We also needed to figure out how we were going to display the results of each round. We decided it would be best to have a summary graphical display of the data at the end of the playthrough. Once the program runs out of lives a screen will appear graphing statistics about the machines performance. These statistics will include things such as how long the machine lasted, how many levels it completed, which enemies or objects took away the most lives, and so on. Having detailed and easily readable data is key to knowing how to train the machine to keep performing better in the game.

3.4 Hardware viewpoint

While working with the Jetson TX1 there are several things that we needed to decide. These included our methods to communicate with our client, and the hardware used with the Jetson. Choosing out hardware was key to determining how each piece would fit into the design.

For communication purposes, our team decided to stick with using skype. We will conduct regular meetings with our client and continue using skype, which everyone is familiar with. Skype will allow us to video call as well, in case any visuals need to be explained.

Our project will actually need to be able to control the game. Inputs from the neural net need to be received by the game so that it can be played. To do this we decided not to use

[1] The hardware connection of the camera as well as the control mechanism of the game shall be explained in a different section.

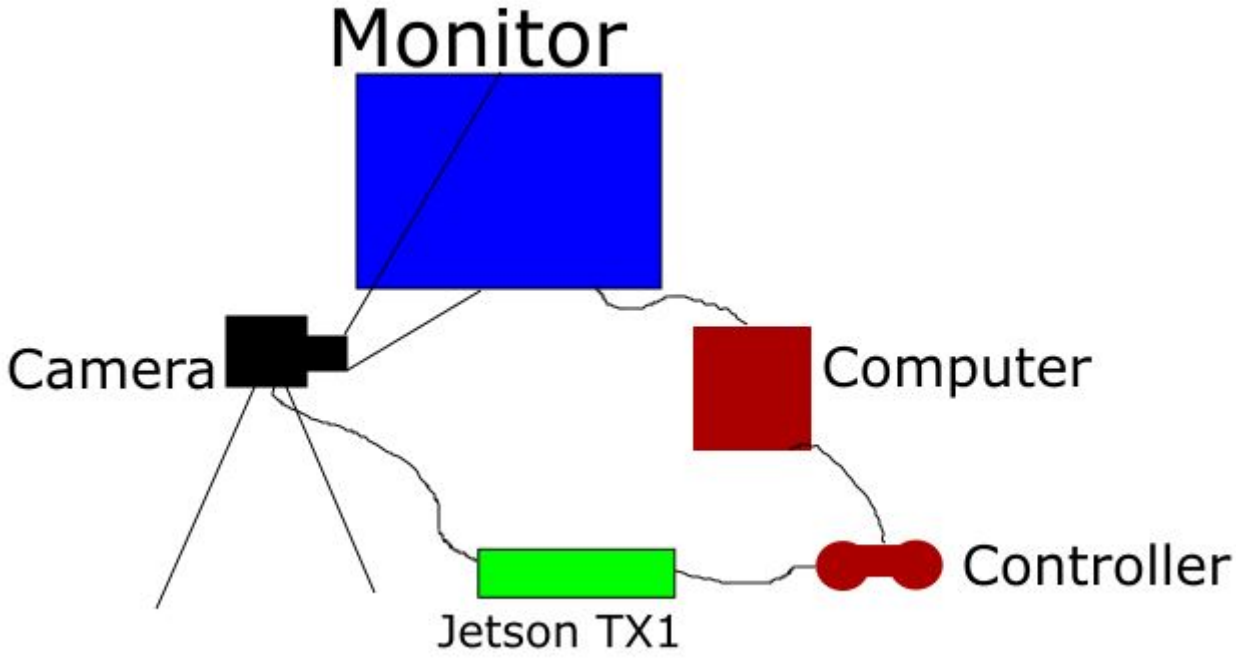

Figure 3.2: Hardware Layout

the standard keyboard. Using a keyboard requires a lot of additional coding to map the Jetson to the controls. Instead we decided to find a game controller that we will hardwire to the Jetson TX1.

3.4.1 Hardware setup

A game controller will significantly reduce the work load because it will already contain a lot of built in functionality that we need to connect it to the Jetson. The Jetson will be able to control the game using the controller's built in functionality.

Figure 3.2 shows the basic layout.

Since the Jetson has to learn how to play the game it needs to be able to see the game. For this we decided to use the camera sent to us by our client. This was the simplest option seeing as we can connect the camera directly to the Jetson. The camera will be connected to the Jetson and aimed at the screen showing the programs gameplay. The program will first be trained by being given the video input to recognize and characterize objects within the game. We will set up tables so that the neural net will be able to store information such as the locations of objects. Once those things are set up it will be able to train itself by playing the game using the controller. It will base its decisions on the table of information that it stores and will increasingly be able to survive the game longer. Eventually it will learn to respond to certain patterns in the table in order to become better at the game.

4 Conclusion