CS Capstone Design Document

February 17, 2017

AI Gaming

Prepared for

NVIDIA

Mark Ebersole _____ _____
Signature *Date*

Prepared by

Group 14

Deep Learning on Embedded Platform

Christopher Johnson _____ _____
Signature *Date*

Gabe Morey _____ _____
Signature *Date*

Luay Alshawi _____ _____
Signature *Date*

Abstract

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 will used, how the system will be put together, and what methods will be 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 to replicate the project. 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 to be 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 will be developed for the purposes of this project. Average people may see the result of the final 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 with Google Inc., University of North Carolina, Chapel Hill, University of Michigan, and Magic Leap Inc. [2]. This neural network will undergo two main phases of training. One training phase will be done with just still images, while the second will involve live image feed from the game Galaga. The system will run inference on the Jetson TX1 developer kit, which will interface between a seperate computer via a controller. The Jetson will be wired to the controller to control the game on the seperate computer. Cloud computing GPUs will be used to handle the training. In total, the system should be able to learn how to play Galaga, and then continually improve as it plays.

1.5 References

[1] F. Rojas, "Galaga (Namco)," in gaminghistory101.com, Gaming History 101, 2012. [Online]. Available: https://gaminghistory101.co

[2] C. Szegedy et al., "Going Deeper with Convolutions," in cs.unc.edu. [Online]. Available: https://www.cs.unc.edu/ wliu/papers/GoogLeNet.pdf.

1.6 Definitions

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.

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

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.

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

API: Application Programming Interface

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

OS: Operating system

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.

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.

2 DESIGN

2.1 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:

- Structure Viewpoint: Design elements approached from a structural level.
- Information Viewpoint: Design elements approached from a perspective of data flow operations and data processing.
- Interaction Viewpoint: Design elements relevant to getting systems to integrate with each other. Particularly focused on hardware setups required for the rest of the process.

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

2.2 Structure viewpoint

This section covers integral structural elements of the system and how they come together. Tools used will be covered to give readers an idea of how we set up the system structure. The heart of this structure is our neural network set-up.

2.2.1 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 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 1.

The network set-up provided by GoogLeNet needs to be accessed with a framework, as well as modified slightly

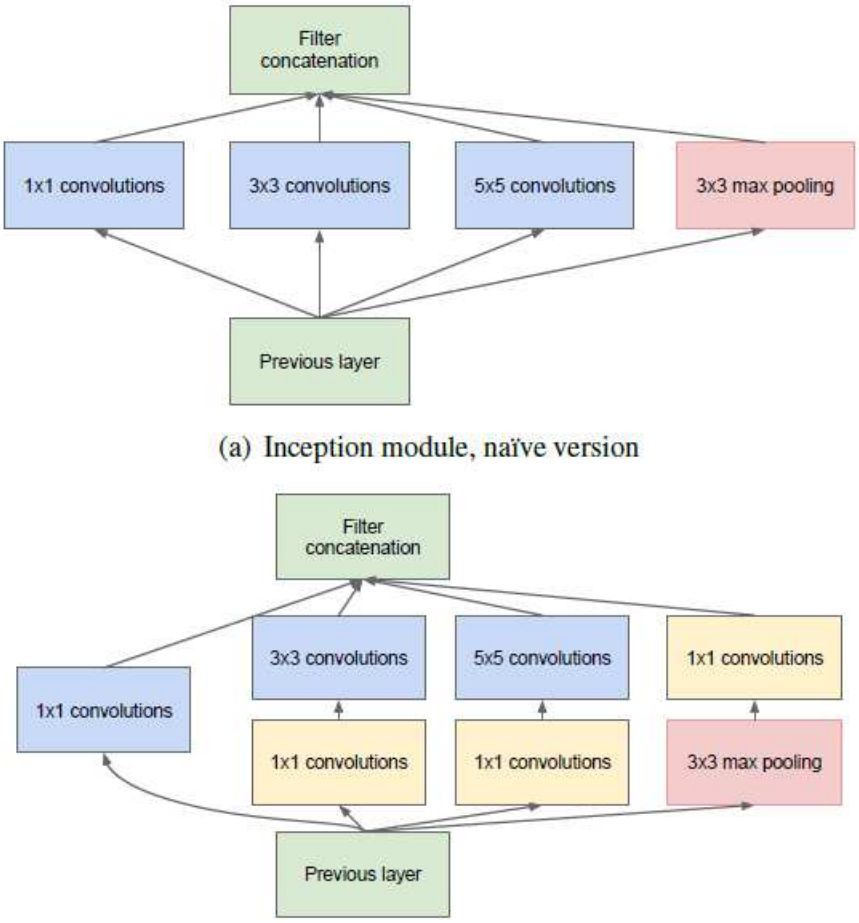

Figure 1. GoogLeNet Module

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 3.

3 INFORMATION VIEWPOINT

The data in this section consists of image processing of the game being played as well as 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 in 3.6 will include a general algorithm to show how output decision's layer works.

3.1 Set up image dataset

Setting up image set for training and testing requires using annotation methods that can be used to label and identify objects from any given image. In this project, Pascal VOC annotation is being used because it's the most common and supported by the majority of deep learning frameworks as well as neural networks. The deep learning framework we are currently using supports Pascal VOC and Coco but only Pascal VOC is being used. However, in order to prepare the annotations we would need to separate train validation images and test images so we can easily list them in a text file along with their path and XML which is the annotation file. In addition, we would need to write down a label map file that lists all the objects that need to be identified by the neural network. Overall, the files are needed in order to create the LMDB database are trainval.txt, test_name_size.txt, test.txt, lebelmap_voc.prototxt. Examples of how to write those files are in 3.5.

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 Input image processing

After having a trained neural network, a software written in python will start by loading the necessary dependencies, which are OpenCV. 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. The hardware connection of the camera, as well as the control mechanism of the game, shall be explained in section 4. Since videos are just consecutive images, OpenCV will be used to get each frame and then pass the image to the input of neural network. Then we would need to do a forward call to the network's API using Caffe which give us the output from the neural network.

3.4 Data visualization

It was necessary to figure out how 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 machine's 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.

3.5 Setup Image Data Sets Example

3.5.1 trainval.txt

train/image1.png train/image1.xml

3.5.2 test.txt

test/test1.png test/test1.xml

3.5.3 test_name_size.txt

imageName imageWidth imageHeight

test1 1080 1920

3.5.4 labelmap_voc.prototxt

{

name:"player",

label: 0,

display_name:"player"

}

Note: Label is a numerical value and should be incremented every time a new label is added.

3.6 Pseudocode

Algorithm 1: Decision Layer

Reshape the Layer to set output one thing of size 1 - for possible outputs such as left,right,shoot;
In Forward Function:;
$player_x, player_y \leftarrow CoordinatesOfthelabelPlayer$;
$topData \leftarrow buttom[0] -> date()$;
foreach *object on the game* **do**
  if *(object.y* $\geq$ *300) and and (object.x* $\leq$ *player_x+200)* **then**
    $topData[0] \leftarrow 0$;
  end
  else if *(object.y* $\geq$ *300) and (object.x* $\leq$ *player_x+200)* **then**
    $topData[0] \leftarrow 1$;
  end
  else if *(object.x* $\leq$ *player_x+100) and (object.x* $\geq$ *player_x-100)* **then**
    $topData[0] \leftarrow 2$;
  end
end

Algorithm 2: Jetson TX1 plays Galaga

Load necessary packages for openCV, and Caffe;
Load the trained net and label map;
Do input preprocessing using Caffe Transformer API;
Start video capture;
foreach *frame in the video capture* **do**
    Load the frame to data layer which is the input of the nerual network;
    Call Caffe forward API to get decision output from the network;
    Send the decision through the network to the game hosted computer with a command such as left, right and shoot;
end

4 Interaction Viewpoint

This section details how our physical hardware components interact to form a cohesive system. It will outline several pieces of hardware used to realize the goal of playing galaga.

4.1 Hardware setup

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.

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 the standard keyboard. Using a keyboard requires a lot of additional coding to map the Jetson to the controls. We had originally decided that we would use a gaming controller and wire the Jetson to it so that the neural net could send the game commands. A game controller would have significantly reduced the work load because it would have already contained a lot of built in functionality that we needed to connect it to the Jetson. However instead of requiring another piece of hardware we decided to just use Windows APIs and a client server system. The Jetson will be able to send commands to the game through an ethernet cable. This setup is outlined in figure 2.

Since the Jetson has to learn how to play the game on its own it needs to be able to see the game. By see I mean that video has to be captured and translated into data the neural net can understand. We were able to buy a pretty inexpensive camera that could capture more than enough detail. It only cost about $25. This option had not been in the tech review originally but after some work on the project we realized that in order for the neural net to play the game it didn't need incredibly high quality images. The neural net only needed images where it was obvious to see what enemies were being recorded. Also to react to the game quickly enough to beat the first level we realized the neural net wouldn't have to process more than 30 frames per second. The camera will be connected to the Jetson and aimed at the monitor, which is connected to the computer running the game. 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 the objects and tables are set up the neural net will be able to train itself by playing the game, sending commands through an ethernet cable. Its decisions will be based on the table of information that it stores and will increasingly be able to survive the game longer. Eventually the neural net will learn to respond to certain patterns in the table in order to become better at the game.

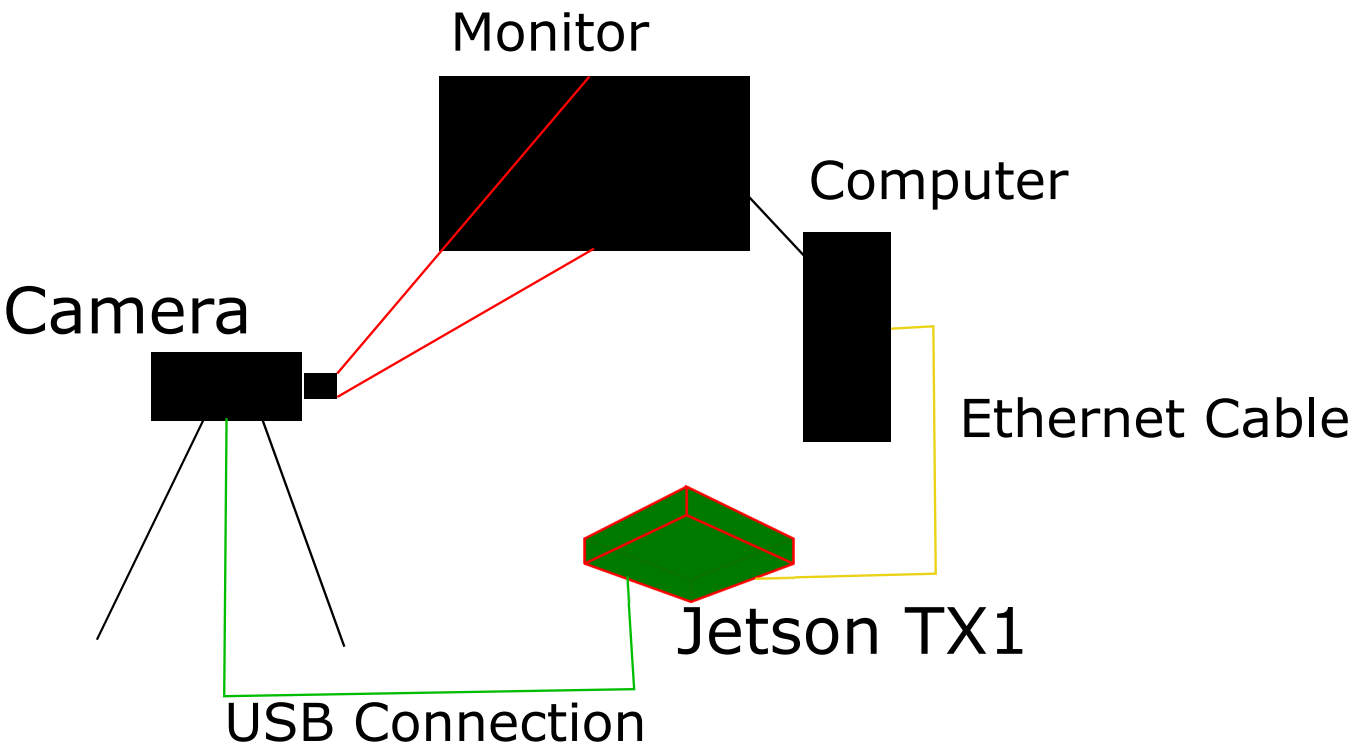

Figure 2. Hardware Layout

5 CONCLUSION

The goal of this design document is to provide the how to of this deep learning project which will be turned to the NVIDIA's deep learning institute to use it in a deep learning course. The overall steps for this document are to teach the Jetson TX1 to play the game Galaga by training a neural network to identify game objects and become effectively able to move the player away from enemies. This can be done based on three major phases which are preparing images for the game objects, training a neural network based on the set of images with labels, and finally by using a camera connected to the Jetson TX1 while the game hosted on a different machine that is being controlled using the Jetson TX1 and a controller input.