**Project Summary**

Project Title: AutoInputCPP

Author: William Reindl

Description: A proof of concept to demonstrate how the ResNet50 Convolutional Neural Network could be used for automated video game testing.

User: Video game QA testers, 20-34 years of age

Problem: Automate basic gameplay for QA testing which can improve development time

Technology:

C++

dlib

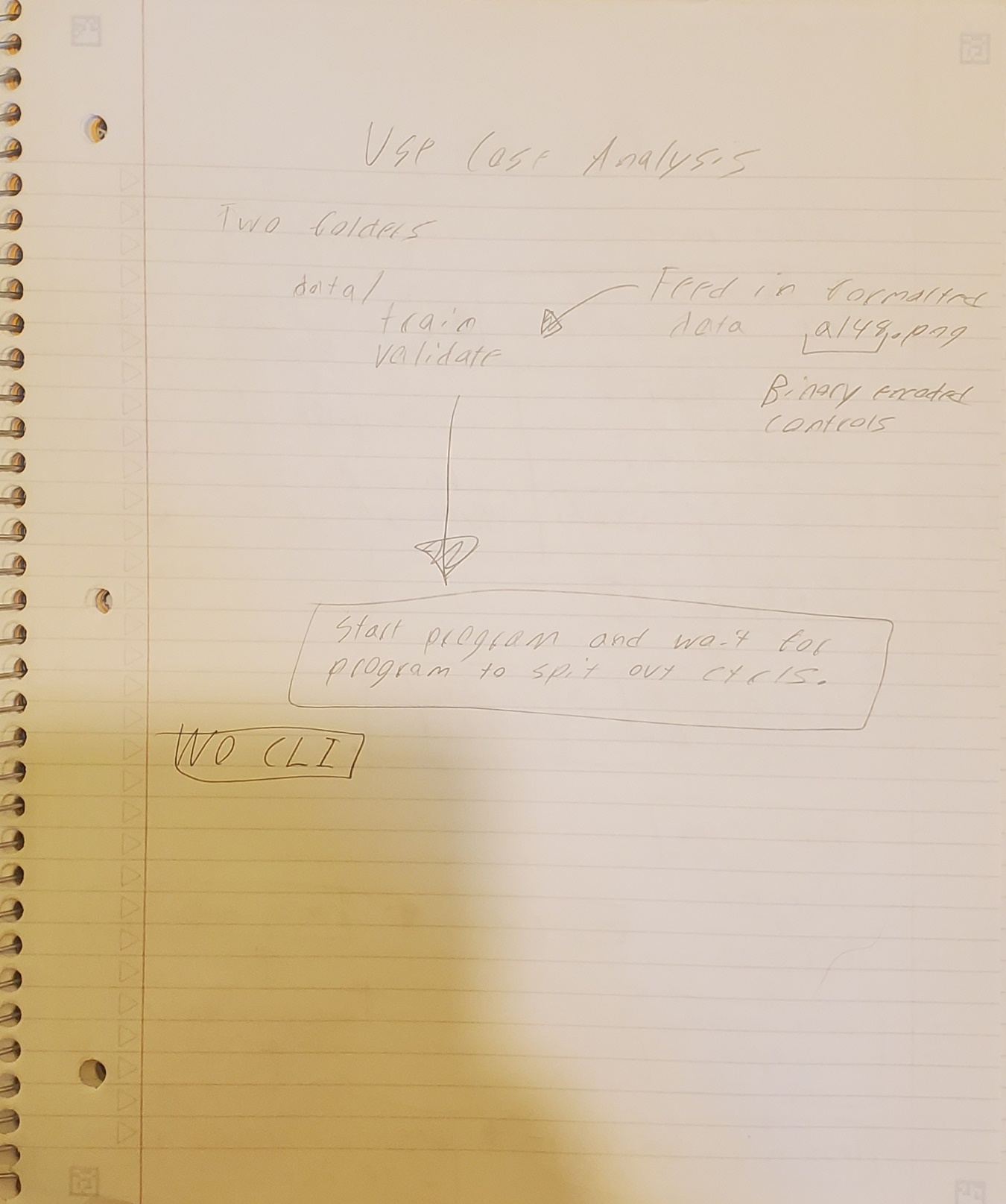
ResNet-50

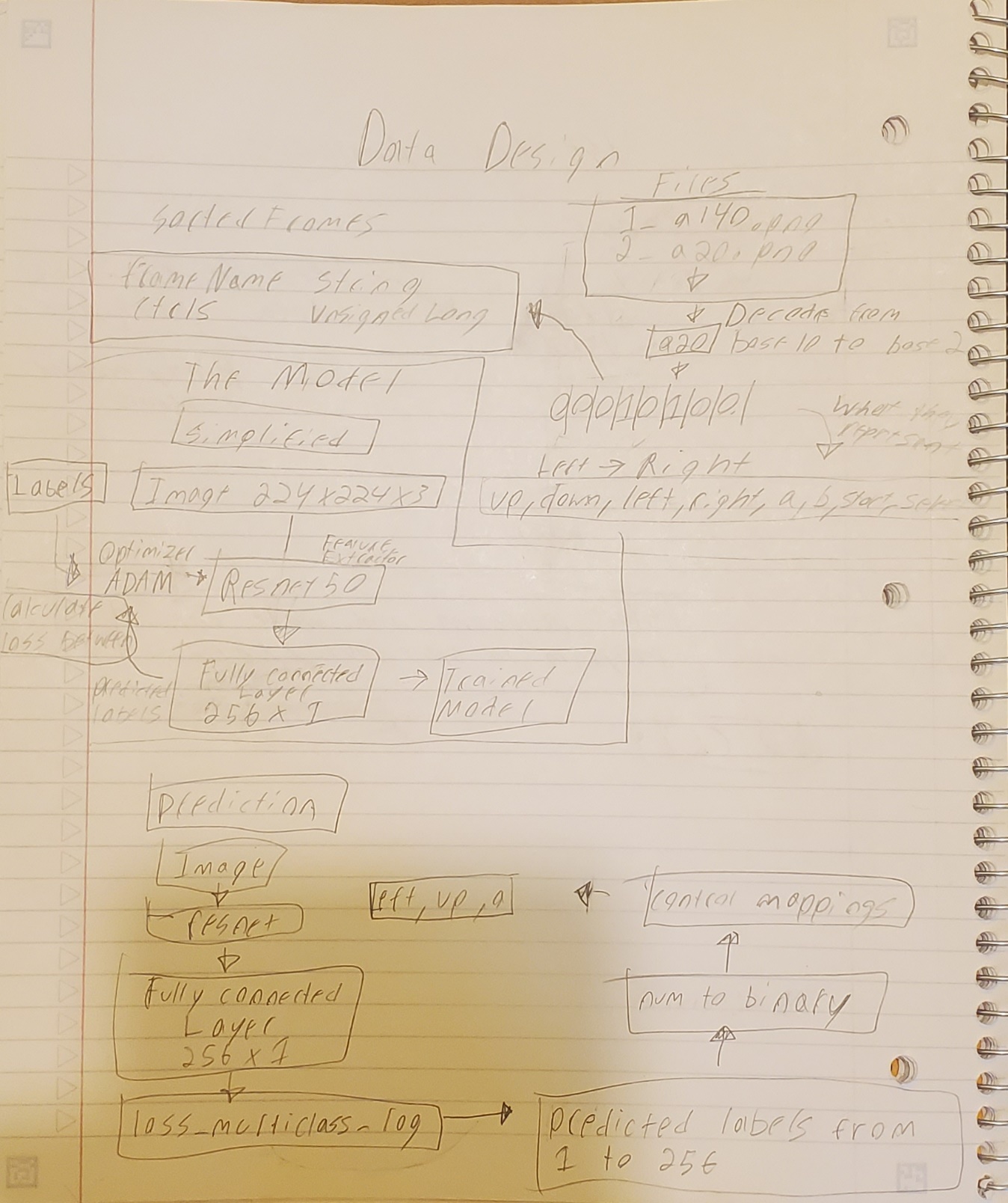
Cmake,

CUDA,

CuDNN,

OpenCV





**Algorithm**

Dispatch Class:

1. Load in inputDataset
2. Cycles through every nested frame inside train folder and pushes the filename and extracts the image label.
3. TranslateCtrls splits filename at delimiter ‘a’ and trims off the file extension leaving just the control format “a20”. a20 is the decimal representation of the binary encoded controls (see data page).
4. Converts the integer to a unsigned long and stores it along with the image frame name in the frames.h structure.
5. Load in validationSet using exact same process
6. Call train class and pass in inputDataset
7. Call tester class and pass in validationSet

Train Class:

1. Define resnet 50 train network from resnet.h
2. Load in pretrained resnet50 model
3. Chop off the last two layers of the resnet model (loss layer and classifier)
4. Addon our own loss layer (loss\_metric) and classifier (fc<256) on top of resnet backbone.
5. Initialize network
6. Set optimizer and its corresponding weight decay and momentum parameters (ADAM).
7. Set a checkpoint file named resnet\_game\_sync. This will allow us to resume training at this save point if a crash occurs.
8. Set learning rate (How much it should step each iteration/epoch)
9. regularLoad(), loads in the raw pixel values into memory and resizes the frames to a 224x224x3 format. Labels have already been loaded in so they are just returned to caller.
10. Start training loop.
    1. Runs training loop until learning rate has shrunken 4 times from 0.1 to 0.00001
    2. Randomly select images and labels from loaded in dataset and feed it into current iteration of model.
    3. Keep cycling through model until either pictures are exhausted or model has finished training (or if just feels like throwing another cuda:memory\_error).
11. Clean residual data left in network and save just the weights into game\_classifier\_network\_resnet.dat

Testing class:

1. Load in predictionFrames into memory and resize them to 224x224.
2. First define resnet work same as in training class, careful to load in resnet50 model first.
3. Attach new loss layer and classifier to resnet model temporarily.
4. Load in saved neural network.
5. Chop off last layer (loss layer) and attach a new loss\_multiclass\_log layer
6. Send predictionFrames into loss layer and save results into unsigned long vector.
7. Convert frames from base 10 numbers into binary.
8. Cycle through each character in binary string and any number that is 1, means that control was pressed on that frame.
9. Output predicted frames to console.

Other minor classes:

Resnet.h:

A copied textbook definition of resnet50, used to load in pretrained model.

This code is required and would not be possible to tweak without affecting pretrained model I’m using for feature extraction.

Model.h:

Purpose was to be a spot for common debugging methods and other shared code to be used between the training and testing side. Only purpose is to give status updates on the models state.

Frame.h

Storage structure for easier handling of frames and labels in-between classes.

**How To Run + General Comments**

This program is very heavy to run on modern hardware for the training part. Prediction/Inference should work on CPU however resnet50 is still somewhat heavy so might be a little slow.

Minimum specs to train:

GPU: Nvidia GPU with VRAM >= 8GB

Supported CUDA version: 11.8

Ram: 32GB

CPU with AVX-512 instruction or SSE 4.1

dlib version: 19.24

Cmake 3.26.3

**Notes**

1. Install CUDA and cuDNN.
2. Copy the contents of lib and include cuDNN into lib and include locations of CUDA install directory.
3. Download resnet\_50 model from [here](https://github.com/davisking/dlib-models) and extract into main program directory.
4. Data is stored in the format of “1\_a20.png” with 1 being the frame number, a20 the binary number encoded in base10 and .png as the accepted file format although any should do.
5. Data is stored inside same directory of the program in the format

data/train/<foo\_frames.png>

data/test/<test\_foo\_frames.png

1. cuda:memory\_errors can range from misconfigured shape of network, running debug version of library on release, missing files, CPU not loading in frames fast enough into GPU, or a slight breeze.
2. Dlib libraries are stored in the same directory as program.

**Useful Commands**

Cmake build commands

// Builds Cmake files for visual studio 2022 x64

cmake -G "Visual Studio 17 2022" -A x64 ..

//Build with AVX and AVX2

cmake -G "Visual Studio 17 2022" -A x64 -DUSE\_AVX\_INSTRUCTIONS=true -DUSE\_SSE4\_INSTRUCTIONS=true ..

//Builds library in Release

Cmake –build . --config Release

//Install in debug mode

cmake --install . --config Debug

//Install in release mode

cmake --install . --config Release

**CITATIONS**

@misc{Pinto2021,

author = {Pinto, R.C.},

title = {Super Mario Bros. Gameplay Dataset},

year = {2021},

publisher = {GitHub},

journal = {GitHub repository},

howpublished = {\url{https://github.com/rafaelcp/smbdataset}}

}

[Resnet50\_1000\_imagenet\_classifier.dnn.bz2](https://github.com/davisking/dlib-models/blob/master/resnet50_1000_imagenet_classifier.dnn.bz2)

Author: https://github.com/davisking