BINARY IMAGE CLASSIFICATION COMPARISON USING NEURAL NETWORKS AND RANDOM CONTRAST LEARNING

1. INTRODUCTION AND BACKGROUND

Although Deep Learning has enjoyed tremendous success and growth in recent years, it continues to be costly and time consuming. Computations in deep learning are so intensive that a suite of hardware processors has been adapted or developed to accommodate its computation needs. Most notably, Graphic Processing Unit (GPU) cards developed by NVIDIA and Tensor Processing Unit (TPU) cards developed by Google, continue to handle most of the deep learning loads in data centers around the world due to their ability to perform large numbers of calculations faster and more efficiently than computer CPUs (Central Processing Units).

As demand for more compute power and capacity grew, so did the demand for computer chips needed to manufacture GPU and TPU hardware. This caused GPU and TPU hardware prices to skyrocket in recent years which resulted in a rise in deep learning operating costs. Although GPU and TPU prices have come down slightly in the past year, they are still relatively high.

Described as the "alternative to Deep Learning", Random Contrast Learning or RCL, is a novel approach to supercomputing and machine learning developed by Lumina Research (a project of Lumina Analytics), headquartered in Tampa, Florida. Training RCL models does not require costly GPU or TPU hardware. They are trained on CPUs which are abundant and less expensive and promise to outpace neural networks in training and inference performance, as well as accuracy. In this work, we will test RCL on binary medical image classification and compare its results to neural networks.

2. DESCRIPTION OF DATASETS

The datasets used in this study were acquired from public sources. The following three datasets were selected for our experiments:

- **Breast Cancer Biopsy Dataset (7,500 samples, 2-class, balanced)**
- **Lymph Node Cancer Biopsy Dataset (600,000 samples, 2-class, balanced)**
- **Brain Tumor MRI Scan Dataset (9,561 samples, 2-class, balanced)**

2.1 Breast Cancer Biopsy Dataset Overview

The Breast Cancer Biopsy Dataset contains 7,500 images in two classes: benign and malignant. The images are in the PNG format and are relatively large at 700x460 pixels. The dataset is balanced.

2.2 Lymph Node Cancer Biopsy Dataset Overview

The Lymph Node Cancer Biopsy Dataset contains 600,000 images in two classes: benign and malignant. The images are in the PNG format and are relatively small at only 96x96 pixels. The dataset is balanced.

2.3 Brain Tumor MRI Scan Dataset Overview

The Brain Tumor MRI Scan Dataset contains 9,561 images in two classes: normal and tumor. The images are in the PNG format and are average size at 256x256 pixels. The dataset is balanced.

3. IMAGE CLASSIFICATION METHODS

We will attempt to train two different methods to classify the samples in the selected datasets:

- **Random Contrast Learning Classifier** (RCLC) – an untrained alternative to deep learning language translation model developed by Lumina Research
- **Four-Layer Convolutional Neural Network** (CNN-4) – an untrained deep learning convolution neural network

Several experiments are conducted using each method, but only the best results will be reported in this work. Trained models will be evaluated based on their test accuracies.

3.1 Random Contrast Learning Classifier (RCLC)

Lumina Research describes Random Contrast Learning (RCL) as “a new approach to supercomputing and machine learning. RCL employs a novel use of randomness that may enable it to outperform neural networks in training speed, inference speed, and accuracy.”

RCL was developed by the Lumina team in 2022. We will use the RCL Classifier (RCLC) module in our experiments, which were conducted on Windows computers with INTEL i9 or Xeon Silver processors. The RAM on the computers ranged from 64 GB to 512 GB. RCLC is branded as PrismRCL on Windows.

3.2 Four-Layer Convolutional Neural Network (CNN-4)

The four-layer neural network used in the experiments is a simple network comprised of four convolution layers, with max pooling and dropout, as well as a final dense layer at the output. The final layer uses ReLU activation followed by Sigmoid activation. The architecture of this network is provided in the Appendix on page 9.

The neural network experiments were conducted on Google Colab, which is a cloud-based Python Notebook environment with a lot of built-in support for machine learning. The Colab account used was configured with 54 GB of GPU and a similar amount of CPU RAM.

4. IMAGE DATASETS PRE-PROCESSING AND PREPARATON

RCL does not require much pre-processing or data preparation. The algorithm has two main requirements:

1. The images must be in the PNG format. The images for one of the acquired datasets were in the JPG format, but using a simple Python script, we were able to convert the images to PNG easily.
2. The raw images can be fed to the RCL algorithm in their class folders, even if they are of different dimensions. RCL accepts training and testing image data as follows:
   Parent_Folder/
   Class1_Folder/img1000.png, img1001.png, …
   Class2_Folder/img2000.png, img2001.png, …
   Note: it is important to make sure that image names are unique across all class folders.

For the neural network, the image datasets must be converted to the NumPy array format. The training images and their class information are typically saved in separate arrays. Also, all images must be of the same shape and size. NumPy arrays can be saved to disk and used in future experiments, unless the dataset structure is modified, or if data is added or removed. In that case, the arrays must be generated again.

5. IMAGE CLASSIFICATION EXPERIMENTS

In this section, we briefly describe the experiments that were carried out, starting with the four-layer neural network then RCLC. Although many experiments were conducted on each dataset, using both RCLC and the neural network, we present only the best set of results from all three datasets and both sets of experiments.

5.1 Four-Layer Convolutional Neural Network Experiments and Results

Setting up a neural network training and testing pipeline in Python requires a decent amount of programming knowledge and dozens of lines of code. Assuming all your images have the same dimensions, first you need to load the training data, convert it to NumPy arrays and save those arrays to disk (for future use). Then you need to set up your neural network of choice, and define your training parameters like number of epochs, batch size, learning rate, early stopping rules, etc. Once training is complete, you then have to evaluate your model using the test data, before finally determining if you ended up with a good model. Table 5.1.1 below shows the test accuracies obtained on our three datasets.

Dataset	Samples	Classes	Test Accuracy
Breast Cancer Biopsy	7,500	benign / malignant	94.20%
Lymph Node Cancer Biopsy	600,000	benign / malignant	94.50%
Brain Tumor MRI Scan	9,561	normal / tumor	97.30%

Table 5.1.1 – CNN-4 test accuracy results on three medical imaging datasets

5.2 Random Contrast Learning Classifier Experiments and Results

Compared to a neural network, there is no doubt that RCL is much simpler and easier to use. Although it is a native Windows application, PrismRCL does not have an interactive user interface. It is run via the command line, usually requiring only one line of code to start a training or an inference session. No programming experience is needed. Before initiating your training session, you will probably want to optimize your training parameters, which can be done via a single line of code as well. It is also a multi-threaded application that takes advantage of all the CPU power a computer has. Table 5.2.1 below shows the results of the best training sessions conducted on all three datasets included in this study.

Dataset	Samples	Classes	Test Accuracy
Breast Cancer Biopsy	7,500	benign / malignant	99.50%
Lymph Node Cancer Biopsy	600,000	benign / malignant	99.80%
Brain Tumor MRI Scan	9,561	normal / tumor	99.10%

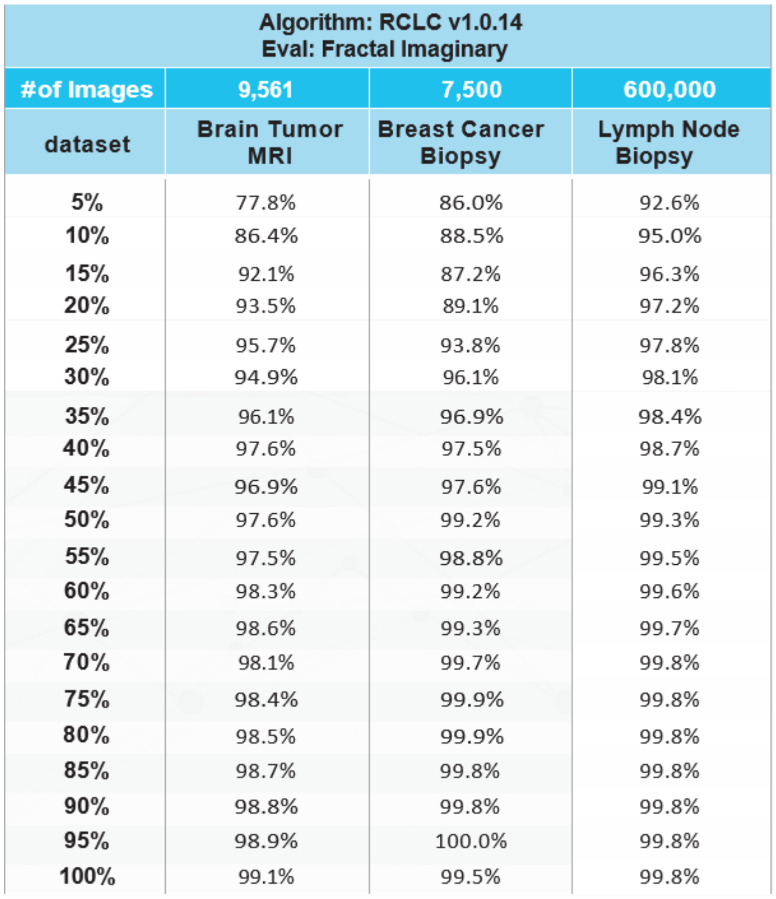
Table 5.2.1 – RCLC near perfect test accuracy results on three medical imaging datasets

It is evident that in these experiments, RCLC exceeded the performance of the neural network and produced accuracies that are near perfect. Add to this the simplicity, ease of use and modest hardware requirements, and you've got promising technology in RCL.

6. RANDOM CONTRAST LEARNING BENEFITS AND FEATURES

When compared to neural networks, the current state of the art, RCL has shown advantages in the following areas:

- **CPU-based Training** - AI-specific chips such as GPU, NPU, and TPU are limited in quantity and cost-prohibitive for most organizations. RCL's use of CPU hardware minimizes capital expense and allows for democratization of machine learning technology.
- **Generalization** - RCL can generalize on smaller datasets than required by neural networks. We can see from Table 6.1 below that RCLC's test accuracy exceeds 90% on small portions of the training data.
- **Speed of Data Preparation** – RCL can ingest PNG images, tabular data and text without the pre-processing work required by neural networks. While deep learning algorithms require all images to be of the same shape, RCL accepts PNG images of different sizes.
- **Sensitivity to Weak Signals** - RCL's use of randomness as a filter allows for patterns to be illuminated at their earliest point of significance.
- **Combining Models for Inference** - RCL appears to have the ability to combine models that have been trained separately and thus make the larger, disparately trained models accessible through a single improved model.



Algorithm: RCLC v1.0.14 Eval: Fractal Imaginary			
#of Images	**9,561**	**7,500**	**600,000**
dataset	**Brain Tumor MRI**	**Breast Cancer Biopsy**	**Lymph Node Biopsy**
5%	77.8%	86.0%	92.6%
10%	86.4%	88.5%	95.0%
15%	92.1%	87.2%	96.3%
20%	93.5%	89.1%	97.2%
25%	95.7%	93.8%	97.8%
30%	94.9%	96.1%	98.1%
35%	96.1%	96.9%	98.4%
40%	97.6%	97.5%	98.7%
45%	96.9%	97.6%	99.1%
50%	97.6%	99.2%	99.3%
55%	97.5%	98.8%	99.5%
60%	98.3%	99.2%	99.6%
65%	98.6%	99.3%	99.7%
70%	98.1%	99.7%	99.8%
75%	98.4%	99.9%	99.8%
80%	98.5%	99.9%	99.8%
85%	98.7%	99.8%	99.8%
90%	98.8%	99.8%	99.8%
95%	98.9%	100.0%	99.8%
100%	99.1%	99.5%	99.8%

Table 6.1 – Detailed results for all three datasets showing the percentage of training data that was needed to achieve high test accuracies

Supported Data Types:

RCL supports three types of data:
(1) Image data in PNG format stored in class folders. Images in other formats can be easily converted to PNG.
(2) Text data stored in text files which are in turn stored in class folders. Each text file can contain one or more lines and will be treated as one sample.
(3) Tabular data, space separated and stored in text files which are in turn stored in class folders. One feature vector per text file.

Automatic Parameters:

Before you start training RCL on your data, you will need to set some parameters that will optimize the training and produce the best possible model. These parameters are: **evaluation**, **rclticks**, **boxdown**, and **imaginaryslice**. RCL offers a special feature that can find the optimal training parameters for your dataset automatically. We call it autooptimize. No more trying random parameters endlessly. Let RCL do the work for you!

Features:

Access to the RCL algorithm is available through a public API as well as a Windows Desktop application that is very simple to use. Here is what you can expect from the API and the Desktop application:

- **Simplified Dataset Management** – The API comes with a frontend web application that allows you to manage your datasets. Upload your dataset once, use it as many times as you need. Delete your datasets if you no longer need them.
- **Intuitive Job Monitoring** – The web application provides an easy way for you to monitor your training and inference jobs. Re-use your API requests through an easy-to-use copy and paste feature.
- **Model Management** – The API application will store your models for future use. Train once. Run inference as many times as needed. Delete the models you no longer need.
- **In-Memory Inference** – Load your favorite model into memory once and run inference on it through the API as many times as you need.
- **Full Control** - With the desktop application, branded as PrismRCL, you can do all the above and much more. Run it on your hardware using single line commands. Schedule training jobs or run them in batch mode. The application is built to run from the Windows terminal so you can automate your jobs and run them unattended. More importantly, keep your datasets safe and secure. No uploading or sharing.

7. SUMMARY AND CONCLUSIONS

In this work, we ran binary medical image classification experiments on two systems: RCLC, an untrained alternative to deep learning, and a four-layer convolutional neural network.

The imaging datasets that we used ranged from 7,500 images for the Breast Cancer Biopsy dataset to 600,000 images for the Lymph Node Cancer Biopsy dataset.

RCLC, which does not require a GPU, and runs on any CPU (via a public API or natively on Windows), offers very simple parameter optimization and training, and yields results that exceed the best performance of the neural network used in our experiments, all without the benefit of any transfer learning. We also demonstrated the rate at which RCLC's test accuracy improves using only a fraction of the whole dataset, as shown in Figure 7.1 below.

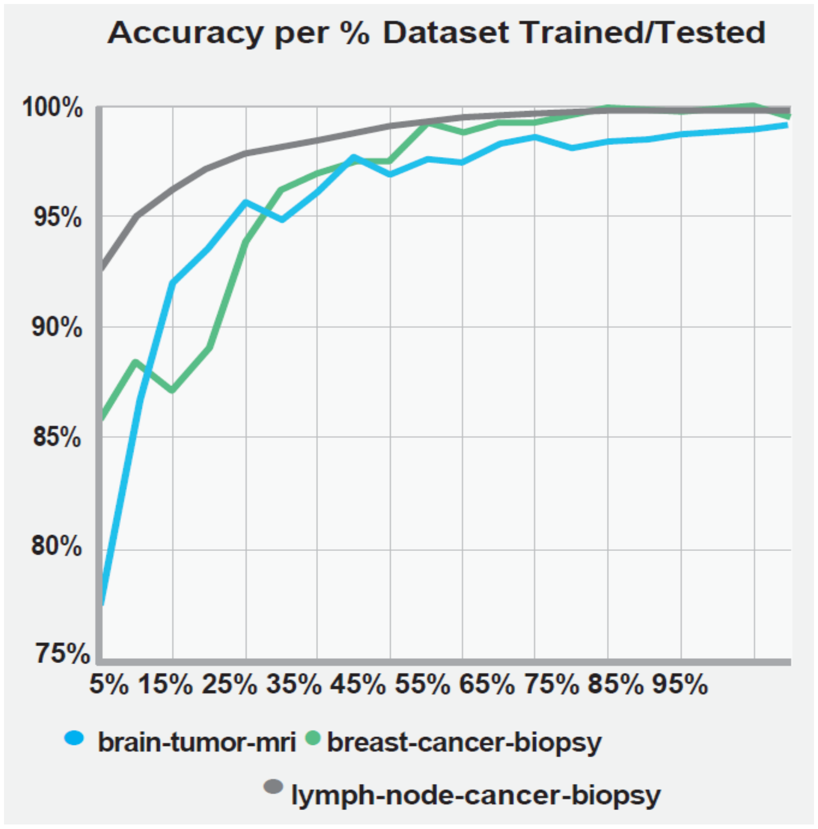


Figure 7.1 – Graphical illustration showing how quickly RCL reaches near perfect test accuracies on all three datasets

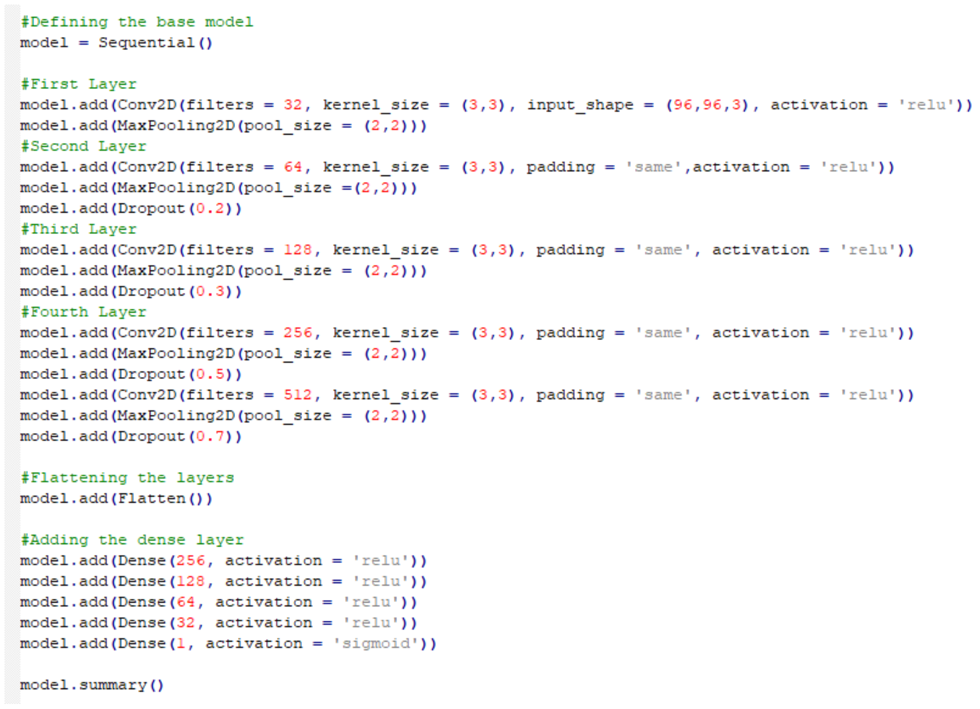
RCLC's best test accuracies were 99.1% for the Brain Tumor MRI scans, 99.5% for the Breast Cancer Biopsy and 99.8% for the Lymph Node Cancer Biopsy. Using the neural network, the accuracies were as follows: 97.3% for the Brain Tumor MRI scans, 94.2% for the Breast Cancer Biopsy and 94.5% for the Lymph Node Cancer Biopsy.

RCLC was run on Windows computers running with INTEL i9 or Xeon Silver processors, with memory ranging from 64 GB to 512 GB or DDR4.

Our four-layer neural network was run in the Google Colab environment which is equipped with an NVIDIA A100 or V100 GPU card and 54 GB of GPU RAM.

APPENDIX

I. Four Layer Neural Network used in experiments described in this document:



```
#Defining the base model
model = Sequential()

#First Layer
model.add(Conv2D(filters = 32, kernel_size = (3,3), input_shape = (96,96,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
#Second Layer
model.add(Conv2D(filters = 64, kernel_size = (3,3), padding = 'same',activation = 'relu'))
model.add(MaxPooling2D(pool_size =(2,2)))
model.add(Dropout(0.2))
#Third Layer
model.add(Conv2D(filters = 128, kernel_size = (3,3), padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.3))
#Fourth Layer
model.add(Conv2D(filters = 256, kernel_size = (3,3), padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.5))
model.add(Conv2D(filters = 512, kernel_size = (3,3), padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.7))

#Flattening the layers
model.add(Flatten())

#Adding the dense layer
model.add(Dense(256, activation = 'relu'))
model.add(Dense(128, activation = 'relu'))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(32, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

model.summary()
```