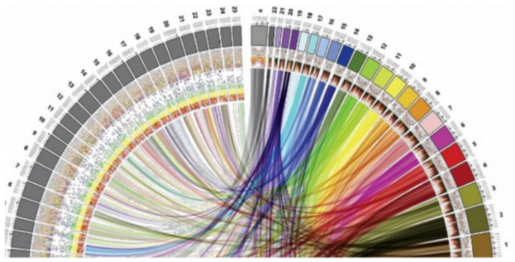


MSc in Bioinformatics

Nextflow and Singularity enabling heterogeneous CPU/GPU bioinformatics containers in workflows

Author: Kevin Sayers

A thesis submitted for the fulfillment of the Master's in Bioinformatics

in the

Faculty of Biosciences, University Autonoma Barcelona

September 2017

Project supervisors:

Cedric Notredame

Paolo Di Tommaso

Academic adviser

Miquel Angel Senar

Acknowledgements

I would like to thank Cedric Notredame and the rest of the Comparative Bioinformatics group at the Centre for Genomic Regulation. In particular, I would like to thank Paolo Di Tommaso for his input on the technical aspects of this project and his extensive help with learning Nextflow.

I would also like to thank my parents for their influence and support over the years. Lastly I would like to thank my fiancée for her support and willingness to move overseas to pursue this masters.

Abbreviations

GPU: Graphics processing unit

GPGPU: General-purpose GPU

CPU: Central processing unit

scRNA-seq: Single cell RNA sequence

HPC: High performance computing

DSL: Domain specific languge

VPC: Virtual private cloud

AWS: Amazon Web Services

IFC: integrated fluidic circuits

CUDA: Compute Unified Device Architecture

BWA: Borrows-Wheeler Alignment tool

BWT: Burrows-Wheeler transform

SAM: Sequence Alignment Map

BAM: Binary Alignment MapGATK: Genome analysis toolkit

AMI: Amazon Machine Image

SVM: Support Vector Machine

VM: Virtual machine

OS: Operating System

EFS: Elastic file system

EC2: Elastic compute cloud

S3: Simple storage solution

FPGA: field programmable gate array

Abstract

The growing amount of biological data which needs to be processed is pushing computational boundaries. This has necessitated new tools to process data in different computing environments such as on HPC clusters or in the cloud. These tools include workflow managers, containers, and even specialized hardware. Workflow managers provide a way to orchestrate the steps of a bioinformatics analysis, and can abstract away computational tasks such as submitting jobs to a cluster. Containers promote not only the portability of software, but also improve the reproducibility of the results obtained. Specialized hardware such as GPUs can substantially speedup certain computational tasks. Combining all of these tools will become increasingly necessary to perform bioinformatics.

The goal of this thesis was to explore ways in which Nextflow could be used to address these cutting edge computational challenges in bioinformatics. Nextflow already supports Singularity, a HPC container technology gaining traction in the scientific computing field. A model workflow for the analysis of single cell RNA-seq data was developed. Two processes in the workflow also made use of tools that performed computations on the GPU. The first process was read alignment to the reference file and the second process was classification of the data using machine learning models.

The workflow demonstrated the ability to use both Singularity and Docker containers together in a single workflow. Nextflow and Singularity can be used together to easily develop and deploy workflows with GPGPU tools. This work opens up some new opportunities for using Nextflow to develop workflows. This includes fields of bioinformatics that rely heavily on GPU tools such as structural bioinformatics. It also demonstrates the possibility of using machine learning as part of a Nextflow pipeline.

Table of Contents

Introduction

1.1 Project overview

The goal of this research project was to demonstrate new uses of the Nextflow workflow language. This was achieved through the development of a proof of concept scRNA-seq workflow utilizing GPU enabled tools for different processing steps. This includes the use of short read aligners which were able to run the alignments on a CUDA compatible graphical processing unit (GPU). Additionally, a basic machine learning classifier was developed using a GPU enabled machine learning library. The work demonstrates a novel use of Nextflow to handle GPU enabled containers and software packages.

As the size and complexity of omics datasets increases there is a need for tools to enable the analysis of this data. Efficient processing of data enables quicker turnaround times for research and faster clinical results. Additionally, less computational time can drastically reduce the cost of a project. The data being generated through sequencing experiments may lead to bioinformatics being one of the largest data processing fields (Stephens et al., 2015). Methods to improve bioinformatics analysis throughput will therefore become increasingly important.

Selecting appropriate computing hardware can be vital for efficiently processing these large datasets. This includes specialized hardware such as GPUs, coprocessor cards, and more recently field programmable gate array (FPGA) cards (Schmidt, 2010). These hardware options often come with configuration challenges. For example GPU tools are notoriously difficult to compile, and work with. The drivers and libraries for a given card dictate what GPU code can be compiled and run. Containers have provided a way to deploy software packages in a reproducible way, but GPU container support was largely experimental and error prone. The recent introduction of native support for Singularity containers with NVIDIA GPU libraries provided an easy way to build and distribute containers with GPU tools precompiled. As part of this project I built and published two Singularity containers with GPU enabled short read aligners to the public Singularity Hub repository. Additionally, I demonstrated the pulling of a publicly available machine learning container with Tensorflow compiled for usage with a GPU card.

The use of containers is important for reproducible research as they ensure that the results being generated are coming from tools in an identical state. Issues such as conflicting dependencies can also be avoided. Containers provide a way to deploy tools which may be challenging to compile, and removes the necessity of setting up a processing environment beyond installing container software (Boettiger, 2015).

GPU containers are only very recently gaining widespread adoption. This has largely been in the field of machine learning as a way to deploy models (Calmels & Abecassis, 2017) (Markovtsev, 2017). Additionally, an example of using Shifter and Docker to run TensorFlow or PyFR on an HPC with GPUs has been published (Benedicic, Cruz, Madonna, & Mariotti, 2017). The works using GPU containers have looked to containerize individual tools for deployment. A natural extension of this is the eventual need to containerize tools in workflows with some steps utilizing the GPU and others utilizing the CPU. These heterogeneous workflows will likely become increasingly common and providing a way to deploy them in different environments will be advantageous to the bioinformatics community.

A model workflow, SRAGPU-nf (https://github.com/KevinSayers/SRAGPU-nf), was developed which demonstrates the use of different containers within Nextflow. A sample single cell RNA-seq dataset was processed with variant calling software, and simplistic machine learning models were created using TensorFlow. The scRNA-seq and machine learning portions of this work were exploratory in nature, and used primarily as a way to demonstrate the GPU utilization. The

building of the GPU enabled containers and configuration of Nextflow to utilize these will hopefully serve as a useful model for future works using heterogeneous computing architectures.

1.2 Nextflow

Nextflow is a domain-specific language (DSL) and workflow management system. A DSL is a language which has specific functions or logic to handles tasks for a given field, in this case bioinformatics. A Nextflow script defines process blocks that represent each step in a workflow. The input and output of each step is then connected to any other process using channels. This ensures that each step of a workflow is receiving and outputting the expected data. Nextflow also has built in support for common bioinformatics tasks such as splitting FASTA files. Each process block can define a script block where the command line to be executed is explicitly written with variables being defined for text interpolation. Scripting languages such as R and Python can also be used within a process script block.

In addition to standard workflow features it abstracts away a number of common computational tasks. Nextflow will automatically execute those tasks which can be parallelized across multiple threads. This parallelization is automatically determined based on the inputs and outputs of a given processes. Nextflow also makes use of dataflow through the use of channels, a channel is opened between the output of one process and the input to another process. Once data is present in the channel the next process will consume it and begin processing.

Nextflow can be used with common HPC cluster managers to distribute tasks on a cluster. Cluster management tools such as SLURM, SGE, and Torque are supported. There is built in support for deploying either single AWS instances, or a virtual private cloud (VPC). The number of instances, AMI, spot prices, and storage can all be specified and Nextflow will automatically create and launch the desired instances. A configuration file can be used to define different parameters for Nextflow when executing locally, on an HPC cluster, or in the cloud. This enables a workflow to be prototyped on local hardware, and then distributed easily to a multicore environment.

Another useful feature of Nextflow is that storage such as AWS S3 and EFS can be easily utilized. S3 paths can be specified in the same way that a local path would be, and Nextflow will handle the reading or writing to the bucket. Likewise if an EFS share is available on AWS the shareid can be used to automatically mount the filesystem in instances launched by Nextflow.

Nextflow provides built in support for both Docker and Singularity containers. This enables the use of containerized tools which promotes the reproducibility of analyses. Containers can be automatically pulled from either personal repositories hosted on Docker Hub and Singularity Hub or from publicly available repositories such as BioContainers (http://biocontainers.pro/).

Workflows can be shared on GitHub and Nextflow will automatically pull the workflow for execution if a GitHub project is specified. This enables the easy sharing and versioning of workflows. (Di Tommaso et al., 2017)

An example workflow is shown in **Figure 1** which illustrates the components of a Nextflow script file. The workflow consists of two process blocks, the first downloads a transcriptome file the second uses BarraCUDA to index this file. Each process block uses a container specified on lines 2 and 18. There is an ‘input’ block in the ‘barracudaIndex’ process that defines the type and source of the inputs. Both processes have an ‘output’ block which specifies the type and output channel. The output block matches the specified filenames, such as ‘Homo_sapiens.GRCh38.cdna.all.fa’ on line 6 and then copies the file matching this mask into the output channel. This workflow also demonstrates the data flow concept used by Nextflow. As ‘reference’ is an output of the ‘setup’ process it can be inferred that ‘barracudaIndex’ must await completion of ‘setup’ process. Once data is received in the ‘reference’ channel the ‘barracudaIndex’ process will automatically start. Nextflow will parallelize tasks which are determined to be independent of others. The last component of this example script is the actual script block. The commands bound by the quotation marks are the actual commands executed. As shown by the ‘setup’ process multiple commands can

be strung together, or a single command such as the one executed in the ‘barracudaIndex’ step. As each process specifies a container the tools do not need to be installed locally. Nextflow will automatically configure the data volumes to be mapped into the containers so that the local data is accessible in the container.

```
process setup{
        container = "docker://sayerskt/samtools"
        publishDir './', mode: 'copy', overwrite: true

        output:
        file "Homo_sapiens.GRCh38.cdna.all.fa" into reference
        file "Homo_sapiens.GRCh38.cdna.all.fa.fai" into refindex

        """
        wget ftp://ftp.ensembl.org/pub/release-88/fasta/homo_sapiens/cdna/Homo_sapiens.GRCh38.cdna.all.fa.gz
        gunzip Homo_sapiens.GRCh38.cdna.all.fa.gz
        samtools faidx Homo_sapiens.GRCh38.cdna.all.fa
        """

}
process barracudaIndex{
        container = 'shub://KevinSayers/BarraCUDA_Singularity'

        storeDir 'index/'
        input:
        file ref from reference

        output:
        file "${reference.baseName}.*" into indexOut, indexFiles

        """
        barracuda index -p ${reference.baseName} ${ref}

        """

}
```

Figure 1: An example Nextflow workflow file. The script consists of two processes first the transcriptome file is downloaded and extract. The second process indexes the transcriptome file using BarraCUDA a short read aligner discussed in section 1.5 of this work.

1.3 High performance computing architectures

A number of computing architectures exist which are widely used in bioinformatics. Grid computing or clusters have most commonly been used to process bioinformatics data. Either consists of a large number of network connected CPU cores which tasks can be distributed to (Nobile, Cazzaniga, Tangherloni, & Besozzi, 2016). HPC clusters normally use batch schedulers such as Sun Grid Engine (SGE) or SLURM among others to manage the resource allocation within the cluster (Foster, Zhao, Raicu, & Lu, 2008).

A disadvantage to these setups is that the upfront cost to acquire and maintain such a cluster can be costly and inflexible. More recently cloud computing has been gaining popularity with infrastructure as a service. Cloud computing provides more flexibility than cluster computing as you can change the hardware to meet individual analyses requirements.

Cloud providers such as Amazon AWS, Google Cloud Platform , and Microsoft Azure provide hardware as a service by charging for the compute time utilized. AWS was the platform

utilized in this work for both computing and storage. AWS elastic compute cloud (EC2) provides a way to provision different hardware through specified instance types. During provisioning different operating systems can be chosen as the amazon machine image (AMI) or community AMIs can be selected which have various software packages already installed. Instances can be used as either single workstations in the cloud or through creation of a virtual private cloud. A VPC is a set of AWS EC2 instances that are connected to form a virtual compute cluster (https://aws.amazon.com/vpc/). Cloud computing also provides an affordable way to test new hardware optimizations. AWS offers a variety of different instances which can be selected based on the computational task. This includes instances with GPUs which can be used to improve the throughput of some bioinformatics tools (Nobile et al., 2016). Additionally, AWS is offering an FPGA specifically designed for NGS sequencing analysis (http://www.edicogenome.com/dragen-on-amazon-web-services/).

In addition to the compute resources provided by AWS EC2, storage is provided by the simple storage service (S3) platform. S3 is an object storage system that arranges data into buckets. Usage is charged in tiers based on the amount of data being stored (https://aws.amazon.com/s3/details/). AWS elastic file system (EFS) is another storage solution available which provides a filesystem which can be mounted in EC2 instances. EFS is useful when working with a VPC, as data such as images or the work directory which must be shared can be mounted on each of the instances (https://aws.amazon.com/efs/).

Another advance in parallel computing has been the usage of GPUs. Utilizing GPUs for non-graphics computations has been termed GPGPU (General purpose GPU) (Wu & Liu, 2008). Whereas a modern CPU may have tens of cores, modern GPUs have thousands of parallel cores capable performing mathematical computations. The GPU used in this work, the NVIDIA Tesla K80, has 4992 NVIDIA CUDA cores per card (http://www.nvidia.com/object/tesla-k80.html). GPGPU excels at mathematical computations that can be broken up into many individual calculations (Schatz, Trapnell, Delcher, & Varshney, 2007). Additionally, GPGPU has been adopted by the machine learning community as a way to speed up machine learning models as the math computations necessary work well with the GPGPU architecture (Bergstra et al., 2010).

A downside to GPGPU programming is that code must be modified to utilize GPUs unlike with other type of computing where more parallelization can be achieved through the addition of more nodes or instances. The code to be executed on the GPU is termed a kernel. The two most prominent libraries providing GPU programming support are OpenCL and CUDA. The CUDA library is an NVIDIA specific extension of C/C++ for developing GPU kernels (Nickolls & Dally, 2010) .

GPU versions of common bioinformatics tools exist such as BLAST (Vouzis & Sahinidis, 2011) and Smith-Waterman alignments (Manavski & Valle, 2008). NGS tools for short read aligning such as BarraCUDA (Klus et al., 2012) , Soap3-GPU (Liu et al., 2012), and a Bowtie2 implementation developed by NVIDIA (https://nvlabs.github.io/nvbio/nvbowtie_page.html). These tools have been demonstrated to have substantially performance improvements for certain analyses.

1.4 GPU short read aligners

Read alignment is a computational problem which can be parallelized as the alignment of a given read is independent of the other reads. This also makes it a process that can take advantage of the large number of parallel cores present in a GPU. This work focuses on implementing workflows for two GPU enabled short read aligners BarraCUDA and nvBowtie. Both BarraCUDA and nvBowtie are based on the Burrows-Wheeler Alignment tool (BWA) (Klus et al., 2012).

BarraCUDA is a short read aligner that utilizes the parallelization of GPUs to align sequences. The alignments are done in memory on the GPU card before being transferred to disk. The alignment is carried out in three steps by the software. Initially, BarraCUDA uses BWT to

index the transcriptome or genome file using the CPU. This is then loaded into the GPU memory along with the reads to be aligned. Reads of length greater than 38bp are split into shorter fragments, with the default being 32bp fragments. A GPU thread is then used to process each fragment ultimately creating the overall alignment for the read. Next the alignments generated from this step are converted to a SAM file (Klus et al., 2012)(Klus et al., 2012)

The mappings generated by BWA and BarraCUDA were compared to demonstrate that the GPU implementation had not substantially changed the output alignments. The original BarraCUDA paper benchmarked the alignments using different parameters, and datasets comparing both a single and six core CPU to a GPU. The authors observed that the GPU performed alignments in roughly half the time as the single core CPU. The six core and the GPU alignment were roughly equivalent (Klus et al., 2012). Of note though the GPU card used for these benchmarks had up to ten times fewer cores than a modern NVIDIA GPU. Additionally, the most recent version of BarraCUDA was further improved using genetic programming to modify the GPU kernels, this version had a further 62% speedup over the previous version of BarraCUDA (Langdon, Lam, Petke, & Harman, 2015).

NVIDIA has developed a set of projects, nvBio, which provide GPU implementations of bioinformatics tools. As part of this, nvBowtie was developed which took the Bowtie2 algorithm and modified it such that the steps were able to be run on a GPU. Unlike BarraCUDA which utilized the GPU only for alignment, nvBio includes nvBWT which is a GPU implementation of the BWT algorithm used to index the reference samples (http://nvlabs.github.io/nvbio/).

NVIDIA similarly did a comparison of the mapping of reads compared with Bowtie2. This again showed similar mappings between the two tools (http://nvlabs.github.io/nvbio/nvbowtie_page.html) . The indexing of a reference sequence is claimed to be twenty two times faster using nvBWT compared to the BWT implementation in Bowtie2 (http://nvlabs.github.io/nvbio/nvbwt_page.html). Mapping was shown to be eight times faster using nvBowtie for a given test dataset (http://nvlabs.github.io/nvbio/nvbowtie_page.html).

1.5 Single cell RNA-seq

Single cell RNA-seq (scRNA-seq) relies on the ability to isolate a given cell and then perform transcriptomic sequencing. Analysis of a single cell can provide insight into the heterogeneity of the cellular populations present. Cells are isolated through a variety of methods. Flow cytommetry can be used to sort cells based on fluorescent antibodies to either intracellular or extracellular proteins. Cells can also be separated physically using either micomanipulation or optical tweezers to capture a single cell from the population. (Saliba, Westermann, Gorski, & Vogel, 2014) Microfluidic devices with integrated fluidic circuits (IFCs) have also been used. These devices can separate each cell and perform the necessary reactions before NGS sequencing is performed. (Kolodziejczyk, Kim, Svensson, Marioni, & Teichmann, 2015)

After the cells are isolated they are lysed to release the intracellular RNA molecules. The first cDNA strand is generated using a reverse transcriptase. The second cDNA strand is then generated, this is commonly done using PCR to amplify the compliment to the first cDNA strand. (Saliba et al., 2014)

Once the cDNA library has been created it is then sequenced using an NGS platform. Short reads of 30-400 bp are generated. These reads are then aligned to a reference genome or transcriptome. A number of short read aligners exist, with a variety of advantages and disadvantages that must be considered. A special consideration for aligning RNA reads is that splice junctions must be taken into account as the sequenced read will not have an intron that is present in the reference genome. (Wang, Gerstein, & Snyder, 2009) RNA-seq aligners such as Tophat2 (Kim et al., 2013), STAR (Dobin et al., 2013), and others have been designed to take introns into account and can map RNA reads to a reference genome. Another approach to this problem is to align to a

reference transcriptome which has already had the introns spliced out of. This allows use of aligners such as Bowtie2 and BWA to align RNA reads to the transcriptome. (Zhao, 2014) A limitation of this method is that only annotated transcripts will be aligned to, and information regarding isoforms is lost. (Conesa et al., 2016)

Once the reads have been aligned different analyses can be used to gain insight from the data. This includes analyses such as variant calling (Quinn et al., 2013), differential expression of genes (Love, Huber, & Anders, 2014).

1.6 Opossum and Platypus

Opossum is a Python based preprocessing for RNA-seq data and variant calling. The combination of Opossum and Platypus is suggested for RNA-seq variant calling by the authors of Opossum. It is designed to handle all of the of the preprocessing necessary before variant calling with Platypus. (Oikkonen & Lise, 2017)

Opossum initially removes reads that have a mapping quality below a specified threshold. It also removes additional erroneous reads such as when paired-end reads are mapped to different chromosomes. The next preprocessing step removes duplicate reads (Oikkonen & Lise, 2017)

Platypus is a variant caller which utilizes three different calling methods to analyze variants in a given sample. Initially variants are determined using local alignments based on colored de Bruijn graphs of the reads and reference, and alignment of the reads to the reference sequence. Variants are compiled into a list of haplotypes, and the frequencies are calculated. The haplotypes are called based on a Bayesian model. Finally, Platypus filters out those variants which have either a significant bias or do not meet another QC metric (Rimmer et al., 2014)

1.7 Machine learning in bioinformatics

Machine learning is a way of using mathematical models and software to learn patterns from data. The two most common methods of machine learning are either supervised or unsupervised learning. In supervised learning data is broken into a training set with each sample having an associated known label. In unsupervised learning the training does not require a known label. A common form of unsupervised learning is clustering, where features of the data are learned to split the data into groupings (Kashyap, Ahmed, Hoque, Roy, & Bhattacharyya, 2015). An in-depth discussion of machine learning is outside of the scope of this work, but a general overview is provided.

Supervised learning is the method employed in this work, and therefore will be the focus of this section. Different algorithms can be used as the underling learning model for supervised learning. These include K-nearest neighbor (KNN), support vector machines (SVM), and neural networks (NN) (Kashyap et al., 2015).

Another important aspect of machine learning is defining the features from the data, and encoding these features in a way that can be used by the model. Feature selection is important for not only deciding the scientific question that can be learned, but also to ensure that the data is not overly sparse. The selected data for the training and testing of the model must also be reasonably distributed among the label classes (Libbrecht & Noble, 2015).

TensorFlow is an open-source machine learninng software package developed by Google which has become widely used. TensorFlow consumes a graph that is constructed in a programming language such as Python. The graph consists of nodes and edges which describe the relationships in the model. TensorFlow provides support for different devices to execute the mathematically operations, including GPUs (Abadi et al., 2015).

There are a variety of different parameters and models than can be used in TensorFlow. The two models that were used as part of this work were SVM and softmax regression. Both are aimed at learning linear relationships.

The methods share a couple common operations that enable the learning of a problem. The basis for learning of a problem is given a set of features, *x*, and a set of labels, *y*, what function can map the features that correlate with a given label. The predicted labels are based off a set of weights that are modified each iteration of the model which ideally eventually converges such that the weights can accurately predict an unlabeled dataset. The weights are modified using a optimization function which attempts to minimize the loss for a given dataset. The loss is the measure of correctly identifying the labels for the given dataset (Vapnik, 1999).

The SVM algorithm is based on constructing a hyperplane which can, ideally, completely separate classes from one another. SVM classifiers are normally binary classifier. The algorithm works by adjusting how the hyperplane divides the data. The best hyperplane is determined by analyzing the distance between the hyperplane and the two datasets. This is the maximum margin. The SVM algorithm can be extended to work with multiple classes by using a one vs. many strategy where each class is learned against the other classes (Noble, 2006).

Machine learning has been gaining usage in bioinformatics to tackle a number of problems. This includes work on protein interactions (Hamp & Rost, 2015), proteomic biomarkers (Swan, Mobasheri, Allaway, Liddell, & Bacardit, 2013), metagenomic sequence classification (Vervier, Mahé, Tournoud, Veyrieras, & Vert, 2016), and various other applications. Machine learning is efficient at analyzing the large number of features and data that is becoming common in bioinformatics (Kashyap et al., 2015).

1.8 Containers

Containers are a way of isolating software execution from the host operating system (OS). The underlying kernel is shared between the host and the container. Another type of virtualization is a virtual machine (VM), which differ in that a VM contains a guest operating system that has access to some of the hardware as controlled by the hypervisor. These differences are illustrated in **Figure 2** which shows the different software layers necessary to use either virtualization technology.

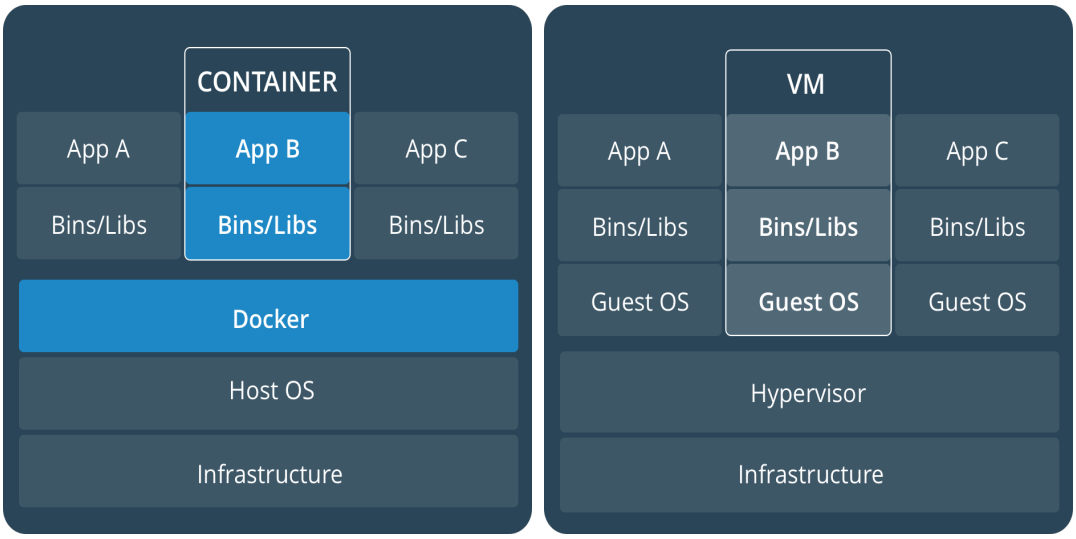


Figure 2: A schematic comparison of containers (left) and virtual machines (right). The lack of the guest OS is also shown to demonstrate that containers are more light weight in terms of size and can be more quickly started and stopped. (obtained from: https://www.docker.com/what-container)

The sharing of the host kernel in containers makes them smaller in size, and quicker to startup. The kernel is shared but only portions of the operating system are available to the container, this enables the software installed in a container to have conflicting dependencies with the host OS or other containers (Merkel, 2014). Each container launched from a given image are launched in the same state guaranteeing the same environment each time a container is created. As shown in **Figure 3** the base image is maintained each time a new container is launched the base image is used to create the container but the image itself can be maintained unchanged. Two common container implementations are Docker (Merkel, 2014) and Singularity (Kurtzer, Sochat, & Bauer, 2017)

Docker images are defined in a Dockerfile which is composed of a base image, and the commands needed to setup the image. Dockerfiles indirectly help with reproducibility as the file contains the exact steps needed to setup a specific software package. Docker Hub is a publicly available hosting repository for Docker images, these images can be pulled using the Docker command line (Boettiger, 2015). Similarly Singularity images can be created from a Singularity bootstrap file, and can be hosted on Singularity Hub (Kurtzer et al., 2017).

Bioinformatics tools, and software in general, are only reproducible if installed in the exact same way across computing environments. It is possible to run an analysis in differing computing environments and obtain dissimilar results (Di Tommaso et al., 2017) Containers provide a way to package a given software tool and make it easily distributable across computing environments. Additionally, conflicting dependencies can be used in different tools within the same workflow or computing environment. Containers have also been shown to not negatively impact the processing times of tools. (Di Tommaso et al., 2015)

A community based bioinformatics container service, BioContainers, provides publicly available Docker containers for common software tools. These containers are hosted on Docker Hub and can be automatically pulled using either Docker or programmatically from other tools such as Galaxy. (Leprevost et al., 2017)

Docker has been adopted both by the bioinformatics and general software community. A limiting factor of Docker is that it is commonly not usable in HPC clusters. This is a result of a potential underlying security issue with the Docker daemon which must be run with root privileges. Singularity is an alternative container technology developed at the Lawrence Berkeley National Lab. Singularity does not require root privileges to run a container, and therefore is better suited for running in HPC environments. Singularity also can be easily configured to enable GPU pass through from the host environment to the container. It achieves this by binding NVIDIA driver libraries on the host to paths within the image. This enables GPU software tools to be used more easily within a Singularity container (Kurtzer et al., 2017). Singularity images can be pulled directly from the Singularity Hub repository. Additionally, Singularity can also pull from Docker Hub repositories and build a Singularity image without the need for a local installation of Docker. (Kurtzer et al., 2017)

NVIDIA does provide a modified version of Docker which can be used to launch Docker GPU containers (https://github.com/NVIDIA/nvidia-docker). Though this still does not resolve the root privileges issue discussed previously. Another option is Shifter which removes the root privilege requirement, and also provides support for GPU access. (Benedicic et al., 2017). Shifter at the time of writing did not have mature support in any scientific workflow language.

Nextflow supports both Docker and Singularity container technologies. It is the only major bioinformatics workflow language at the time of writing with mature Singularity support. Containers can be defined at either the process level or a single container can be defined for a given workflow. Nextflow will automatically pull the defined container if it is available on a public

repository. Additional commandline options can be defined for a container, including the enabling of GPU pass through for Singularity containers. (www.nextflow.io)

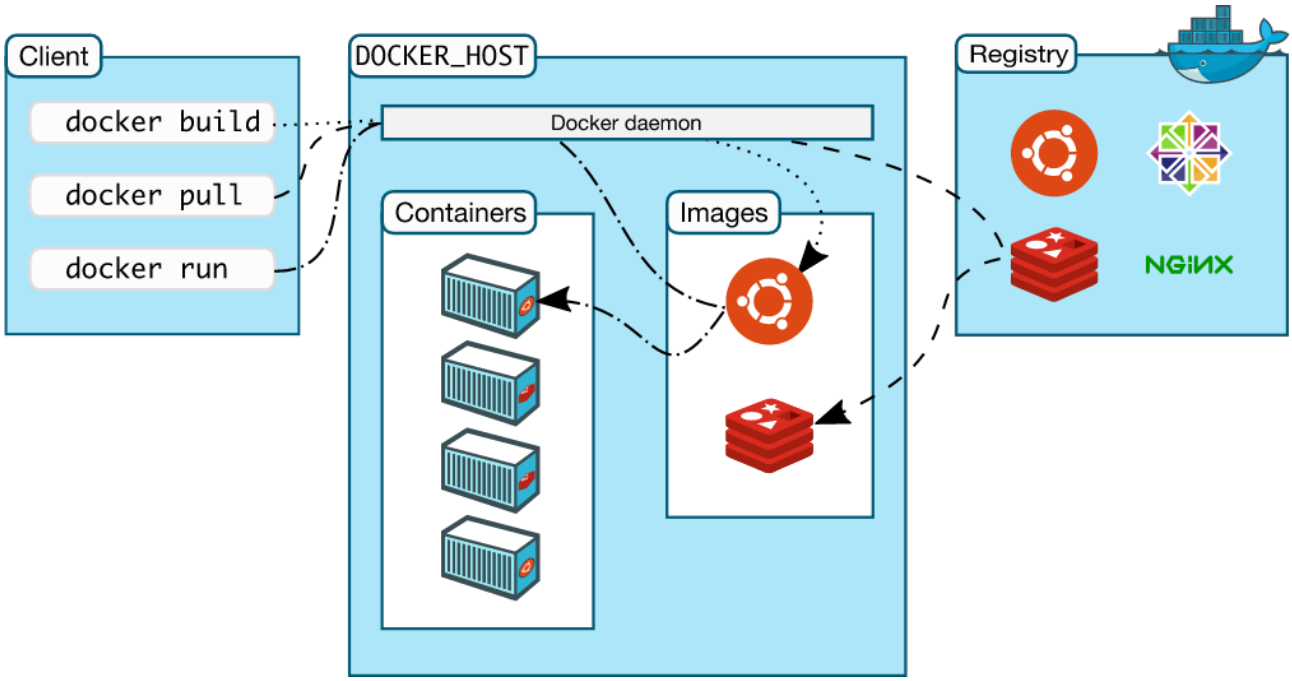


Figure 3: Illustration of the relationship between image and container. Each time the Docker daemon calls a new container, a base immutable image is used to create the image. Additionally, these images may be retrieved from publicly available repositories such as Docker Hub when the 'docker pull' command is used. (Obtained from https://docs.docker.com/engine/docker-overview/#docker-architecture)

2 Methods

2.1 Data acquisition

The data was acquired from BioProject (Accession: GSE75688) which contains 563 single cell RNA sequences from breast cancer cells. The data was acquired using an Illumina HiSeq 2500 sequencer by the Samsung Genome Institute. The cancer cells belonged to four subtypes estrogen receptor positive (ER+), human epidermal growth factor receptor 2 positive (HER2+), HER2+ and ER+, and triple-negative breast cancer (TNBC). The data set also included lymph node metastasis, these were discarded, and were not used for the classifier development. Additionally sample BC09 appears to have been resequenced, but no information was provided so this sample was also excluded. The remaining samples BC01-BC08, BC10, and BC11 consisted of 381 single cell RNA-seq read pairs. Bulk tumor samples and pools were also not utilized for this work. The distribution of sample types is presented in **Figure 4** for both individual classes as well as TNBC vs. all other classes. Samples were automatically downloaded through the Sequence Read Archive Toolkit (SRA Toolkit). A Docker container was built with the SRA Toolkit included. This was utilized as a step in the workflow to download read files and split them into read pairs. Reads were therefore downloaded on demand, and never stored beyond the processing time. The GRCh38, release 88, of the human transcriptome was used as the reference sequence. The workflow ultimately generated a variant call format (VCF) file containing information of sequence variants detected. All intermediate files were discarded after run completion.

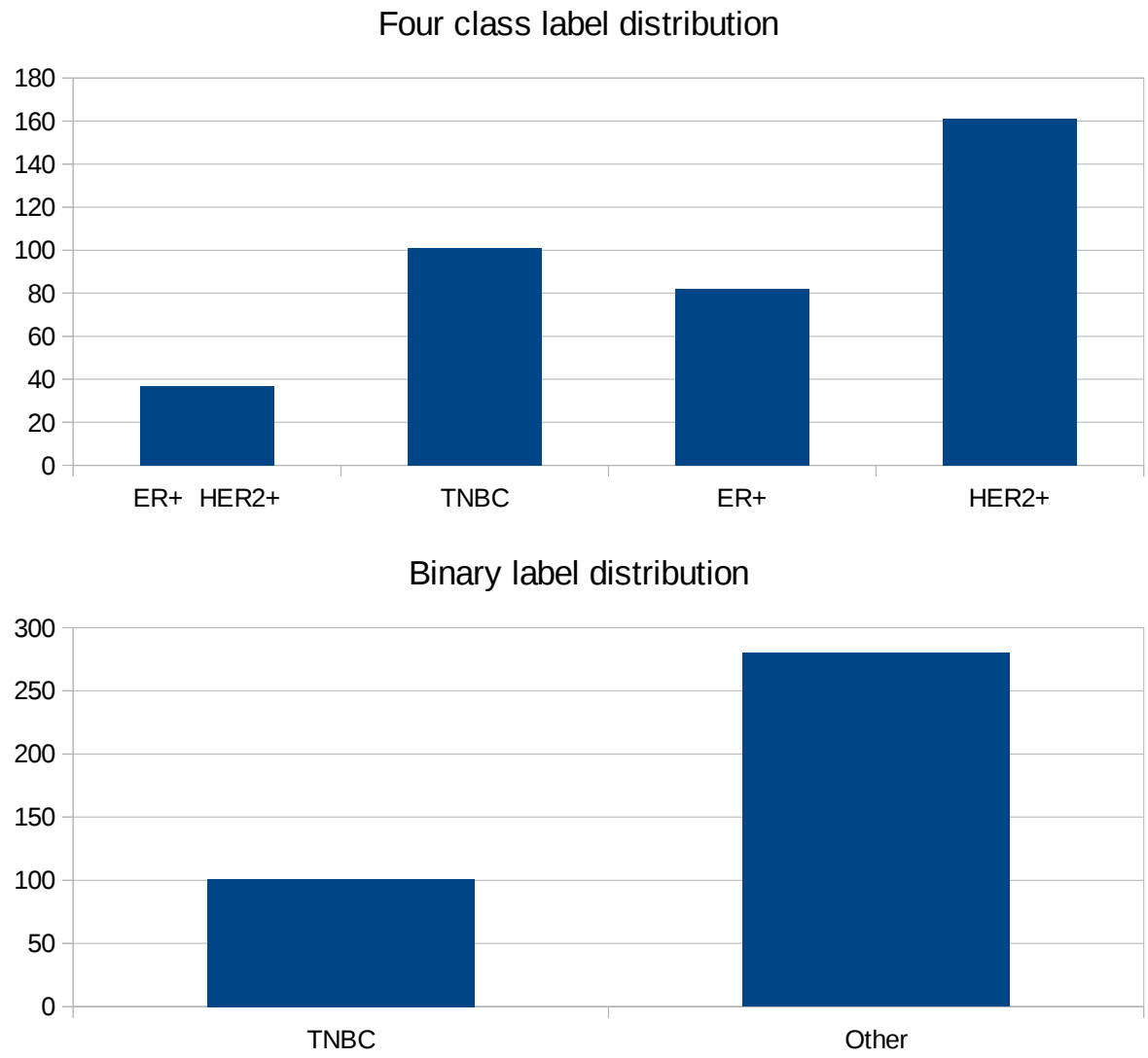


Figure 4: (top) The distribution of labels among the four single cell classification labels. (**Bottom)** The distribution of labels when the classifications are reduced to binary classes of triple negative breast cancer against all other classes.

2.2 HPC and Cloud resources

Nextflow and the containerization of tools enable running of the workflow locally, on AWS, and on any supported cluster. **Appendix VI** lists the computational configurations which were utilized for each portion of this work. Initial development work was performed on a local machine with a consumer grade NVIDIA GPU. Production runs were done on AWS instances. Instances were provisioned using the Nextflow cloud creation option. A configuration file containing the instance type, AMI, region, spot pricing, and instance storage size was used to provision the desired instances. Spot instances were utilized to lower the computing costs. Instances were created from a modified Ubuntu 16.04 LTS AMI (private ami-f379948), the configuration of which is outlined in **Appendix I**. The workflow is hosted on GitHub and can be pulled automatically by Nextflow. Once

the instances were running Nextflow executed the workflow. The final VCF file output was stored in a S3 AWS bucket. An EFS share was used to store the work directory for intermediate files, and also the Singularity cached images.

The BarraCUDA aligner can be used with up to two GPUs for paired end alignments. The available p2 AWS instances have either 1, 8, or 16 NVIDIA GPUs. The GPU instances with a single NVIDIA K80 were selected so that the additional cards did not go unused.

2.3 Containerization of tools

As previously discussed containers provide a way to distribute software which will run in a specified environment, also removing the need to address versions or dependencies. Each tool utilized in the workflow was built into a publicly available container which are presented in **Appendix II**. Tools and dependencies were specified in a Dockerfile and built locally before being pushed to Docker Hub. The base image utilized in the built Docker containers was the latest Ubuntu image provided through Docker. The BarraCUDA and nvBowtie containers were created using the most recent NVIDIA provided CUDA 8.0 Docker image as it provides the CUDA libraries necessary for compiling these tools in the image. The Singularity file was uploaded to GitHub, and then the image was built by Singularity Hub after specifying the GitHub project to use. The Singularity bootstrap file is shown in **Figure 5**. The bootstrap file specifies the base image to use, and then the commands to execute once the image has been created from this base image. The steps for the BarraCUDA image creation involve installing tools which enable the compiling of source code, and then download of the BarraCUDA source and compilation using the 'make' file. The base image provided from NVIDIA also contains the NVCC compiler necessary for compiling CUDA source code. Once compiled the BarraCUDA binary is added to the path of the image so that it is executable when a container is created. The machine learning container used the latest TensorFlow GPU container available on Docker Hub.

During single instance execution all containers were pulled using Nextflow's built in Singularity support. A known issue writing to EFS prevented the automatic pull for runs with multiple instances in a cluster. A BASH script was also provided that pulled each image to the head node before placing them in Nextflow's Singularity cache directory. This only had to be done once as long as the EFS was maintained. Docker containers were automatically converted for use as Singularity containers. Additionally the Singularity option for enabling the GPU pass through was defined in the Nextflow configuration file. Currently the Singularity options are used for all containers so the GPU was made available even for those containers which did not utilize it.

```
BootStrap: docker
From: nvidia/cuda:8.0-devel

%post
    apt-get update
    apt-get -y install wget build-essential zlib1g-dev
    wget https://vorboss.dl.sourceforge.net/project/seqbarracuda/Source%20Code/Version%200.7.0/barracuda_0.7.107h.tar.gz
    tar xvf barracuda_0.7.107h.tar.gz
    cd barracuda
    make all
    mv bin/barracuda /usr/local/bin/barracuda
```

Figure 5: The BarraCUDA Singularity bootstrap file used to generate the Singularity Hub image. This figure shows the usage of the base NVIDIA image used, and the steps necessary to compile BarraCUDA in the image.

2.4 RNA-seq workflow

The RNA-seq workflow presented here is based upon a similar analysis decsribed by Laura Oikkonen and Stefano Lise (2). A directed acyclic graph (DAG) is presented in **Figure 6** showing the workflow steps. The workflow takes in a text file with an SRA sample identifier on each line. Initially the reference transcriptome is downloaded and unzipped.

Both the nvBowtie and BarraCUDA workflow were used to process the samples in separate runs for comparison of results. The references were initially indexed using either BarraCUDA or nvBWT depending on which workflow was being used. Next the samples specified in the input file were downloaded from the Sequence Read Archive (SRA) using the sratoolkit. The sratoolkit split the reads into the two paired end FASTA files. Once the paired end files were present they were aligned with either BarraCUDA or nvBowtie. In the BarraCUDA workflow, after alignment the BarraCUDA 'sampe' process takes the pairs of aligned reads and converts these into a SAM file. Next Opossum was used to remove reads which did not pass QC or were duplicates. Opossum was run with the defaults set for most parameters, the only modified parameter was the SoftClipsExist which was explicitly modified to False for BarraCUDA and True for nvBowtie. The minimum default mapping quality was used at a depth of 40. The BAM file output from the Opossum step was then indexed used Samtools. Finally, Platypus was used to perform variant calling on the processed data. The input BAM file from Opossum, the reference transcriptome, and the output file were specified in the Nextflow script. All other parameters were left as default for the analysis. After Platypus was done running a VCF file was saved in a S3 bucket on AWS.

Nextflow automatically handles the parallelization of tasks which can be executed at the same time. Steps such as the sample download were run in parallel. The Alignment step was limited to one parallel process at a given time as multiple parallel alignments would lead to the GPU driver crashing as it ran out of memory.

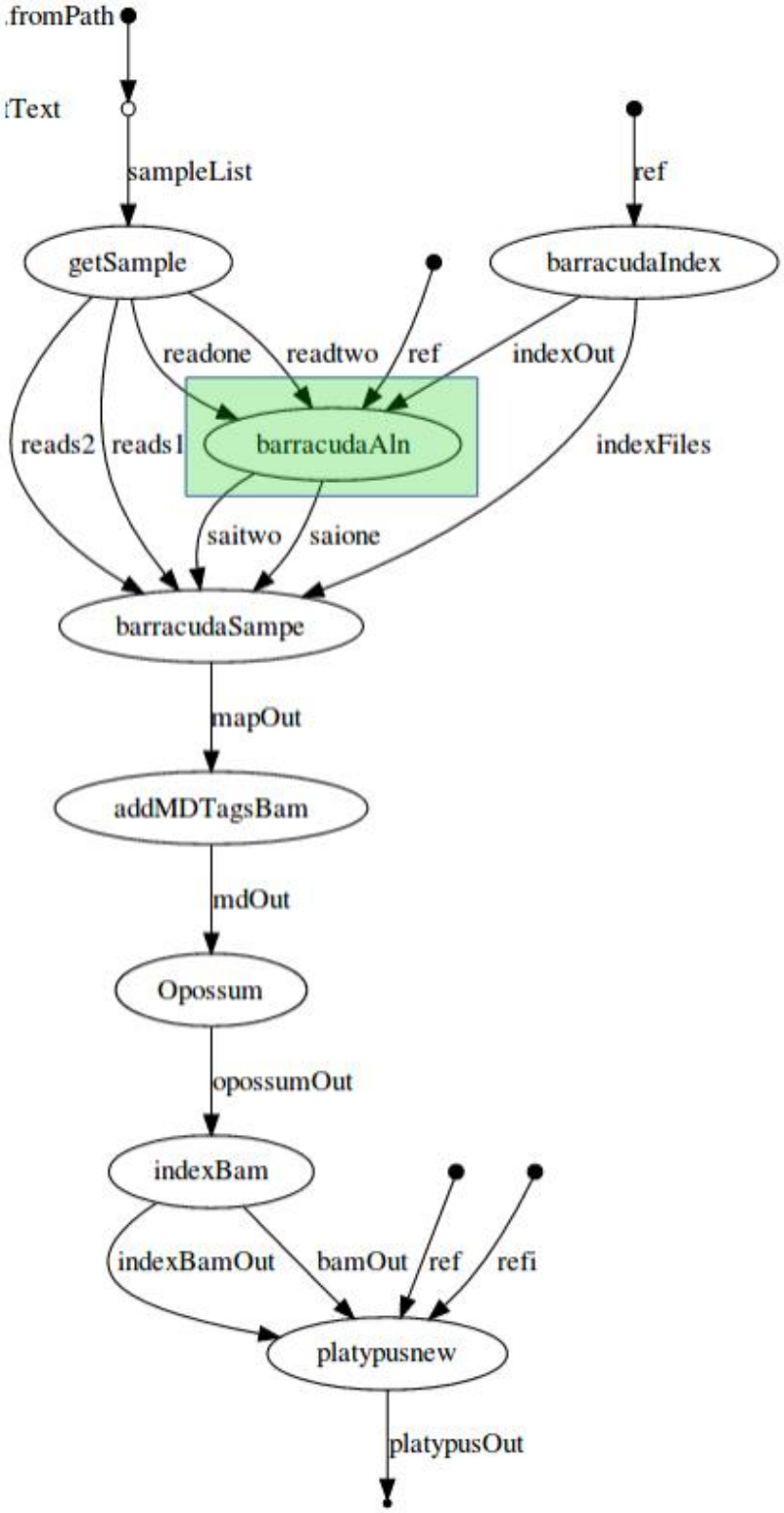


Figure 6.
Directed acyclic graph of the BarraCUDA sample processing workflow. Each oval represents a process carried out in Nextflow. The solid points represent the various inputs, and the arrows indicate the flow of inputs and outputs connecting the processes. The green boxed ‘barracudaAln’ process indicates the step executed in a Singularity GPU container.

2.5 Machine learning preprocessing

A basic one hot encoding Python script (https://github.com/KevinSayers/OneHotVCF) was developed to encode the data for the machine learning model. The VCF parsing utilizes the PyVCF package (https://github.com/jamescasbon/PyVCF). The encoder iterates over the VCF files in two passes. The first pass loads each VCF and makes a list of all variant positions across all VCFs. The second pass then iterates over each VCF determining which subset of the variant positions is present. The result is a one dimension array of length equal to all variant positions for a given experiment. Each position in the array is then encoded as either a 0 or 1 depending on if the position is missing or present respectively in the given VCF file.

The OneHotVCF tool also takes as an input a CSV file with the VCF files and the classification of that sample. The one hot encoded array is then appended to the respective line for the given VCF file. The final output is therefore a CSV file with each line representing a given single cell sample, the classification, and the one dimensional array.

A filtering step was also employed to reduce the sparsity of data. A minimum cutoff was established that the variant position needed to occur in at least 5% of the samples. The workflow steps are shown in **Figure 7**.

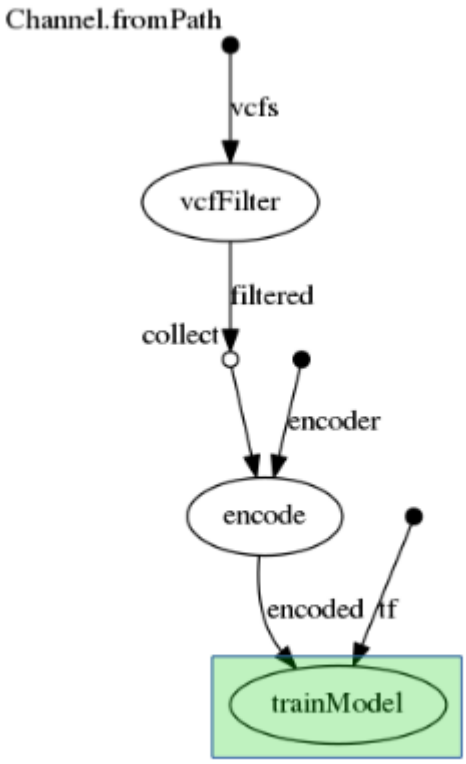


Figure 7. The preprocessing steps carried out to prepare the data for training the machine learning model. The green boxed 'trainModel' step is the TensorFlow step which utilizes the Singularity GPU container.

2.6 Machine learning

Python and Tensorflow were used to build a basic machine learning classifier for the one hot encoded VCF data. A separate Nextflow script was created to run the classification step, the model can be specified in the parameters. The data was loaded and then split into training and test data,

with 90% of the samples being used for training, and the other 10% were used for testing the trained model. Two models were developed to test the learning potential from the data, an SVM and a softmax regression. The SVM model was built to perform a binary classification of TNBC or other. Two versions of the neural network model were created, the first performed a binary classification the second attempted to classify each of the four classes of sample. The softmax regression model was originally based off of the MNIST example provided by TensorFlow (https://www.tensorflow.org/get_started/mnist/beginners). The SVM model was a modified version of TensorFlow examples provided on GitHub under the MIT open source license (https://github.com/nfmcclure/tensorflow_cookbook/tree/master/04_Support_Vector_Machines/02_Working_with_Linear_SVMs).

The first model , a softmax regression, developed to classify the RNA data from a single cell was a binary classification as either triple negative or not. Next the model was modified such that each of the four classifications, ER+, HER2+, ER+ and HER2+, and triple negative were used. The rest of the model was the same between the two except where the labels were used in which the dimensions were either 2x1 or 4x1. The training was run for twenty thousand iterations. The weights were initialized in an array of size determined by the number of one hot encoded positions, and the second dimension was based on if the model was the binary or quaternary classifier. The array for the weights was initialized using the 'truncated_normal' function in Tensorflow. This picks the values from a normal distribution within a specified standard deviation. The bias values were stored in a one dimension array with each value being initialized as a nonzero decimal value. The softmax function output a probability for each of the considered classes (**Figure 8**).

The SVM model was based on a standard linear regression function. It used L2 normalization, each value was squared to improve separation of the classes. The weights for the linear regression were stored in array equal to the size of inputs. The labels were store in a one dimensional array with 1 representing TNBC, and 0 representing any other class.

Both models were tested with both the GradientDescentOptimizer and AdamOptimizer functions in in TensorFlow. This was done as different optimizers can change the efficiency at which the problem is learned and alter the accuracy.

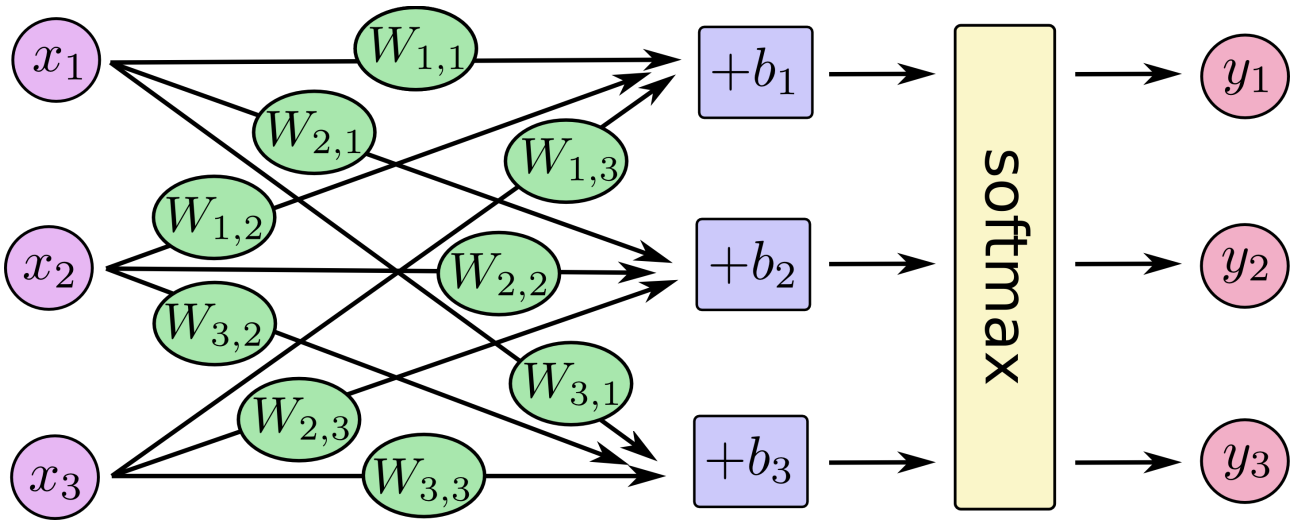


Figure 8: The softmax regression model. The inputs *x* are the encoded variant positions, the weights represented by the green ovals with W labels are the weights. The weights and biases are then run through a softmax layer which outputs probabilities of each class.

2.7 Benchmarking

Workflows for indexing the reference and aligning one set of paired-end reads were developed for nvBowtie, BarraCUDA, and Bowtie2. These were independet of the RNA-seq

workflow and included only the two relevant steps. The nvBowtie and Bowtie2 workflows allowed a closer comparison of CPU vs GPU processing time as nvBowtie is intended to be a near identical implementation of Bowtie2. BarraCUDA and and nvBowtie were also compared as to analyze the GPU aligners against each other. The nvBowtie workflow utilized nvBWT for the indexing of the reference this also used the GPU, whereas BarraCUDA used the CPU for reference indexing. The BarraCUDA workflow included the ‘sampe’ step to generate the SAM file from the coordinates. Bowtie2 and nvBowtie output a SAM file from the alignment step. The Bowtie2 benchmarks were done on the same instance type, both the indexing and alignment steps were configured to use all four available CPU cores.

Comparison of the GPU containerization was also performed. The workflows were modified removing the container for each process. The Nextflow config file was left unchanged so Singularity was still enabled, but nothing was executed in a container. The compiled versions of nvBowtie and BarraCUDA were placed in the ‘bin’ folder so Nextflow would treat them as part of the path.

The benchmarks were performed on a single p2.xlarge node, this included a single NVIDIA K80 GPU card. Initially all pipelines were run once so as the necessary containers were all cached locally. The reference and reads were also downloaded and cached locally. These steps removed any influence on the benchmarks from networking. Each of the alignment workflows was then run five times, and timings were recorded using the Nextflow ‘-with-trace’ option. This gave independent timings for each of the executed steps.

The index step benchmark was done using the 342mb reference transcriptome file. The Nextflow ‘-with-trace’ option outputs to a txt file, results of consecutive runs were concatenated to a cumulative file. The alignment was monitored in the same way for paired-end alignment of FASTQ files containing 5005984 reads of 100bp. The files are 1.4Gb in size each. A sample, SRR5023465, was randomly chosen to act as the benchmark.

The SAM file outputs from each of the aligners were also compared to determine the differences in mapping between the three tools. After the SAM file was generated, they were converted to BAMs and sorted. The sorted BAM files were then process with ‘samtools flagstat’ to check the various mapping statistics.

3 Results

3.1 Benchmarking

The raw timings for the benchmarks are presented in **Appendix III** which shows the timings for each run. The benchmarks comparing nvBowtie to Bowtie2 showed a substantial advantage for nvBowtie. The indexing of the reference transcriptome file was over 16x faster than the CPU implementation. Likewise, the alignment of read pairs to the reference transcriptome was over 3.5x faster than the CPU version.

The analysis of the reads mapping showed a surprising difference. BarraCUDA and Bowtie2 had a similar percentage of reads mapped, 50.54% and 46.11% respectively, whereas the nvBowtie results showed 80.88%. Upon further analysis the numbers for read1 and read2 were also different, and the two values were not equivalent as they should be.

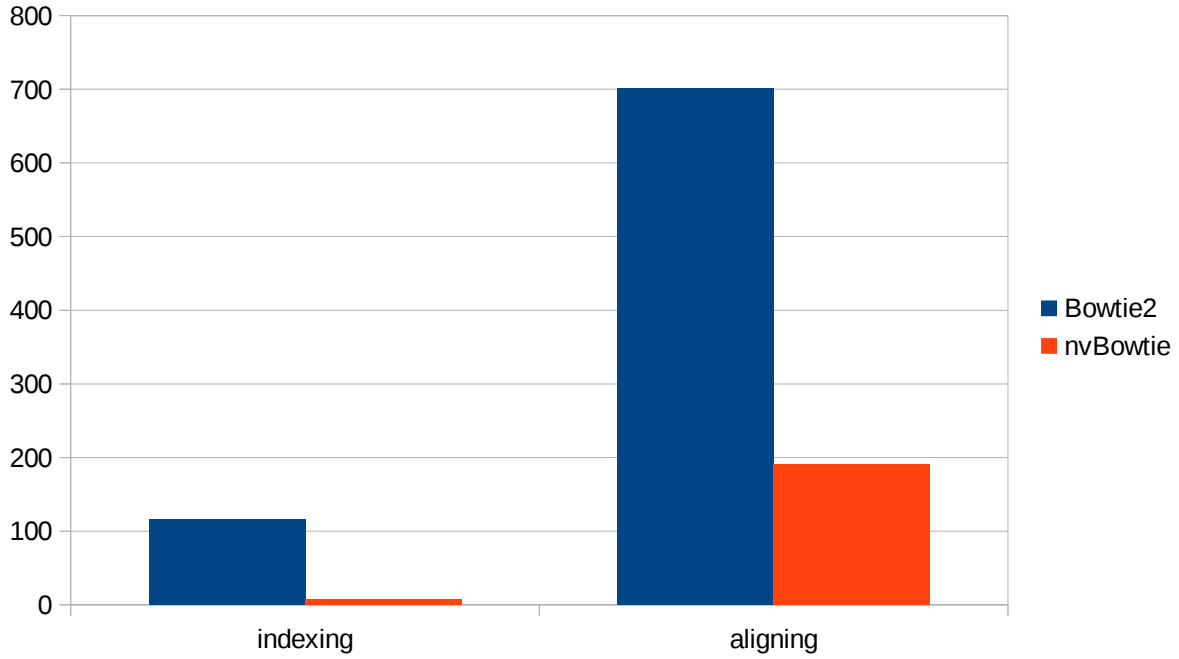


Figure 9: A comparison of the clock time to index and align (Stand in figure at the moment will improve).

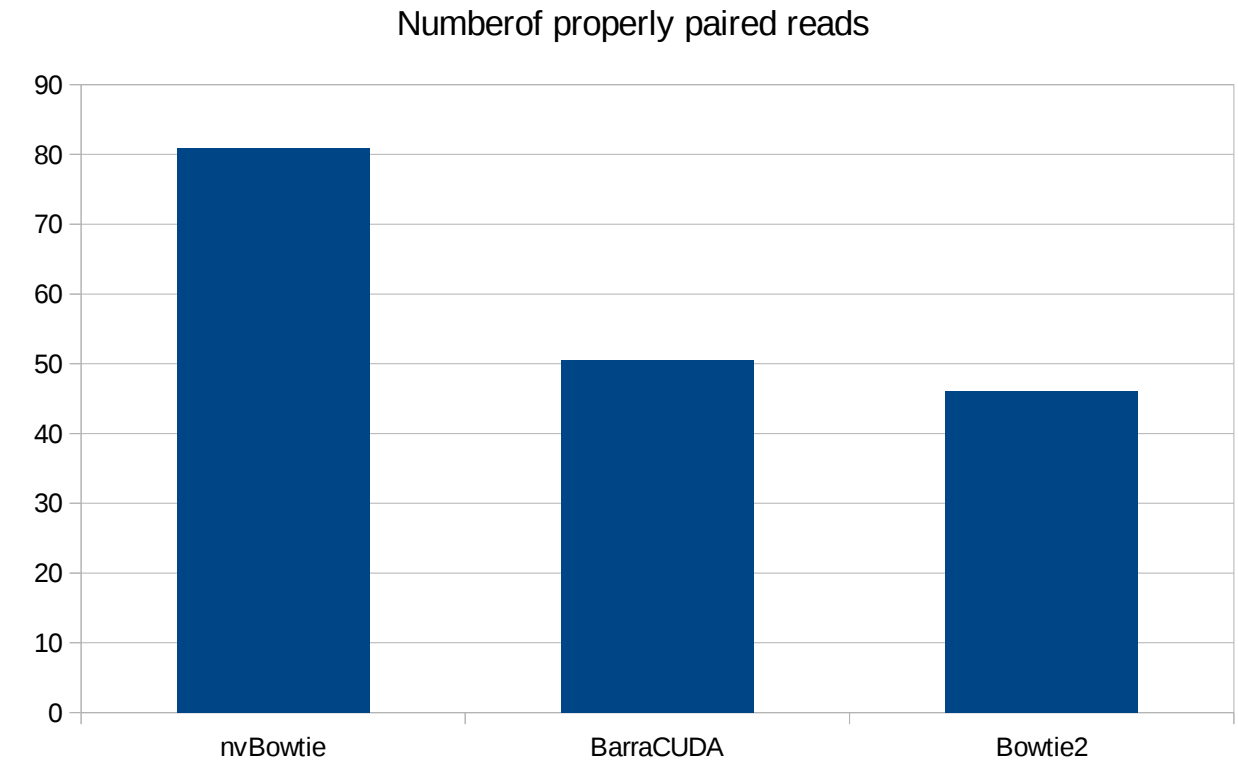


Figure 10: The number of percentage of reads properly aligned as determined using the 'samtools flagstat'

The benchmarking workflows were also run without the containers. For these runs the compiled BarraCUDA tools were placed in the 'bin' folder in workflow directory. Nextflow automatically adds files in this folder to the path. The timing between using the tools in the container, and without were virtually identical.

3.2 scRNA-seq workflow

Despite nvBowtie being significantly faster, the BarraCUDA workflow was used to analyze the results as the benchmarking results indicated there may be some oddities from nvBowtie. All 381 samples were processed using AWS GPU computing instances. The use of spot instances enabled the entire analyses to be performed for under $75 of AWS credits including the EC2 instances, S3 and EFS storage volumes. The time to process a single sample through the pipeline from start to end took between 20 and 25 minutes. The development and processing of samples involved roughly 155 hours of AWS compute time.

The AWS instances were initially provisioned as part of a VPC to act as a cluster in the cloud. Initially issues were caused by Singularity not being able to pull images to an AWS EFS share. This was resolved by making a BASH script to download the images locally on the head node, and then copy them to the EFS. Though this resolved the image pulling issue, another unresolved Singularity issue would cause the workflow to crash. The root cause of this was still undetermined at the time of writing. Instead of using a VPC, multiple individual instances were provisioned using the Nextflow cloud creation commands. Each instance was then given a separate sample list to process.

The machine learning development was done offline from the workflow so the data was initially stored in an S3 bucket. Once all samples had been processed through the scRNA-seq variant calling pipeline the data was downloaded locally for preprocessing.

An exploratory analysis was done to see if any clusters of variants were observable. A binary comparison between the TNBC classification and the others is presented in **Figure 11**. There did not appear to be any significant clustering observed in the plot.

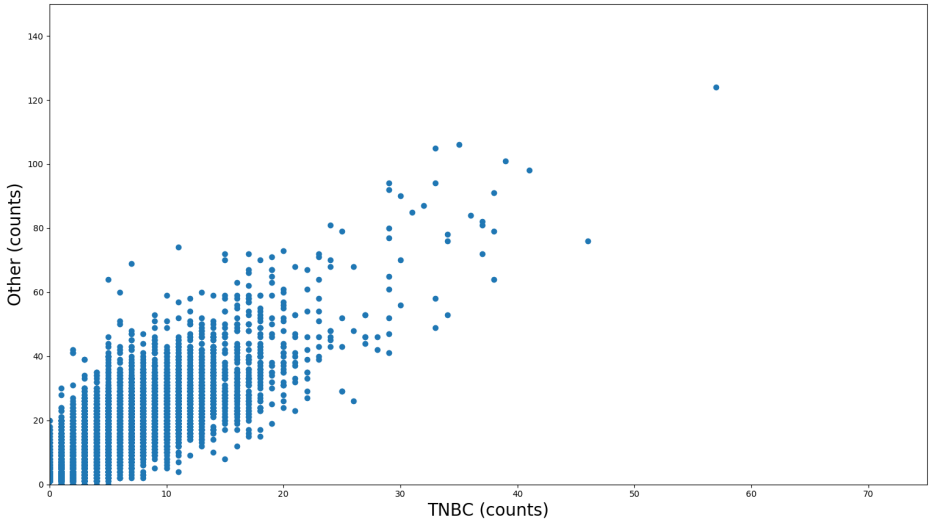


Figure 11: For each of the variant positions the number of times it occurred in a TNBC sample was summed, and plotted against the sum of occurrences for other sample types. No clear clustering was readily apparent.

3.3 Machine learning

The machine learning models were run for twenty thousand iterations each. During each iteration of the model the accuracy against the training labels was calculated, the loss function value, and the accuracy of the model on the test data. These were plotted in individual plots to show how the model was performing.

The SVM binary classification metrics are presented in **Figure 12**. The training accuracy starts around 53% and increases before plateauing at 100% accuracy. Likewise the loss function decreases at a slightly slower rate but eventually approaches 0. The test label accuracy fluctuates more as was expected. Interestingly even at iterations above 12k the accuracy alternates plus or minus a couple percentage points. The optimizer functions had a significant impact on the learning curves, with different optimizer curves presented in **Appendix IV** for comparison.

The softmax regression classification metrics are presented in **Figure 13**. This model though it achieved 100% training accuracy, and the loss function approached 0 did not achieve high accuracy when run with the test data. The four class neural network presented in **Figure 14** performed even more poorly, never achieving accuracy above 70% on the test data.

The TesnorFlow GPU containers were used exclusively for the machine learning. It is possible to use the TensorFlow containers compiled for usage with the CPU. The model and data being used here most likely did not significantly benefit from the use of a GPU. Larger datasets and more complex deep learning models could see a substantial performance improvement from using one or more GPUs with TensorFlow. When using the TensorFlow containers from the official Docker Hub the host OS cannot have a more recent version of CUDA than 7.5.

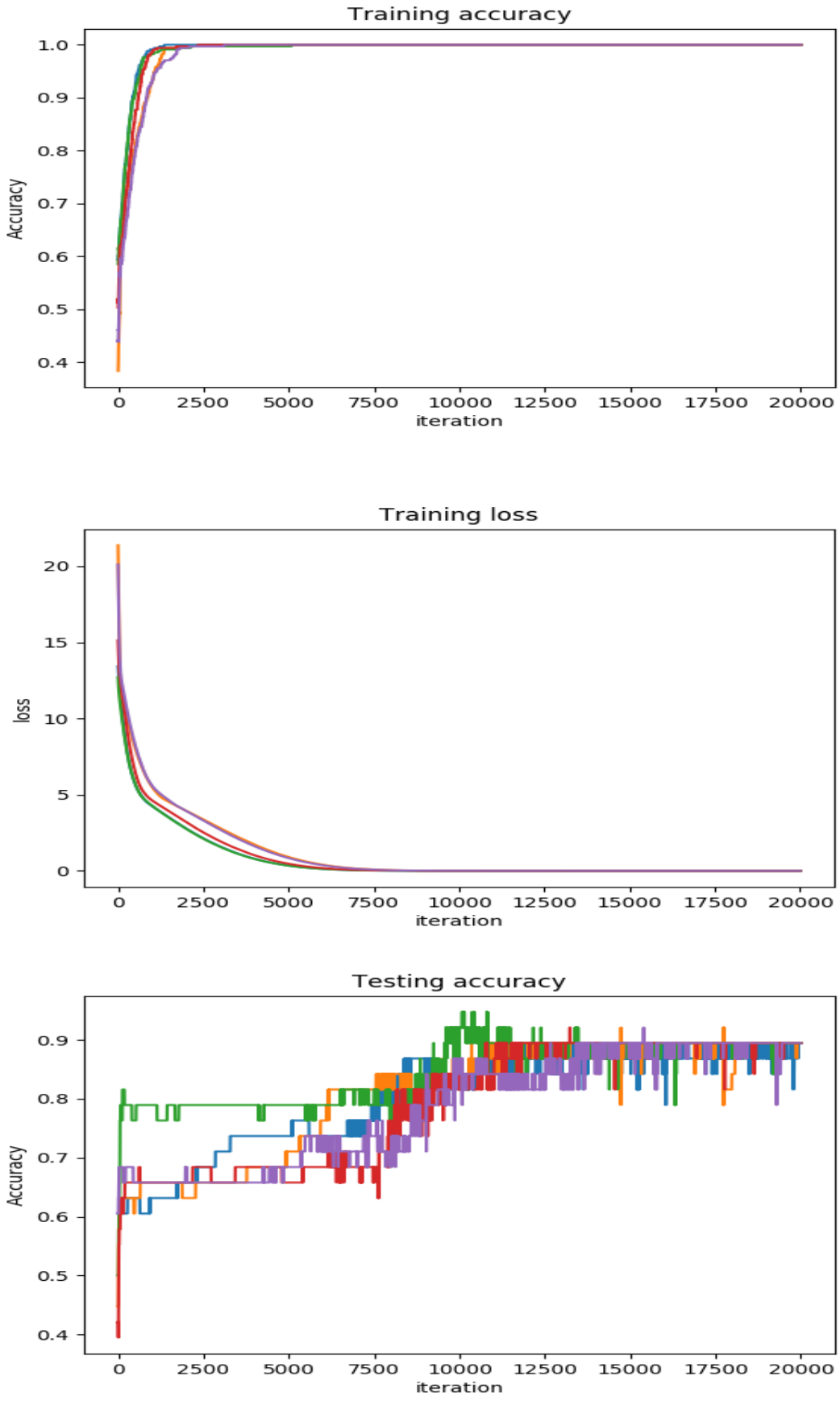


Figure 12: The binary SVM curves for training accuracy (top left), loss (top right), and accuracy on the test data. Data generated from 20k iterations. The model was run five consecutive times without modification.

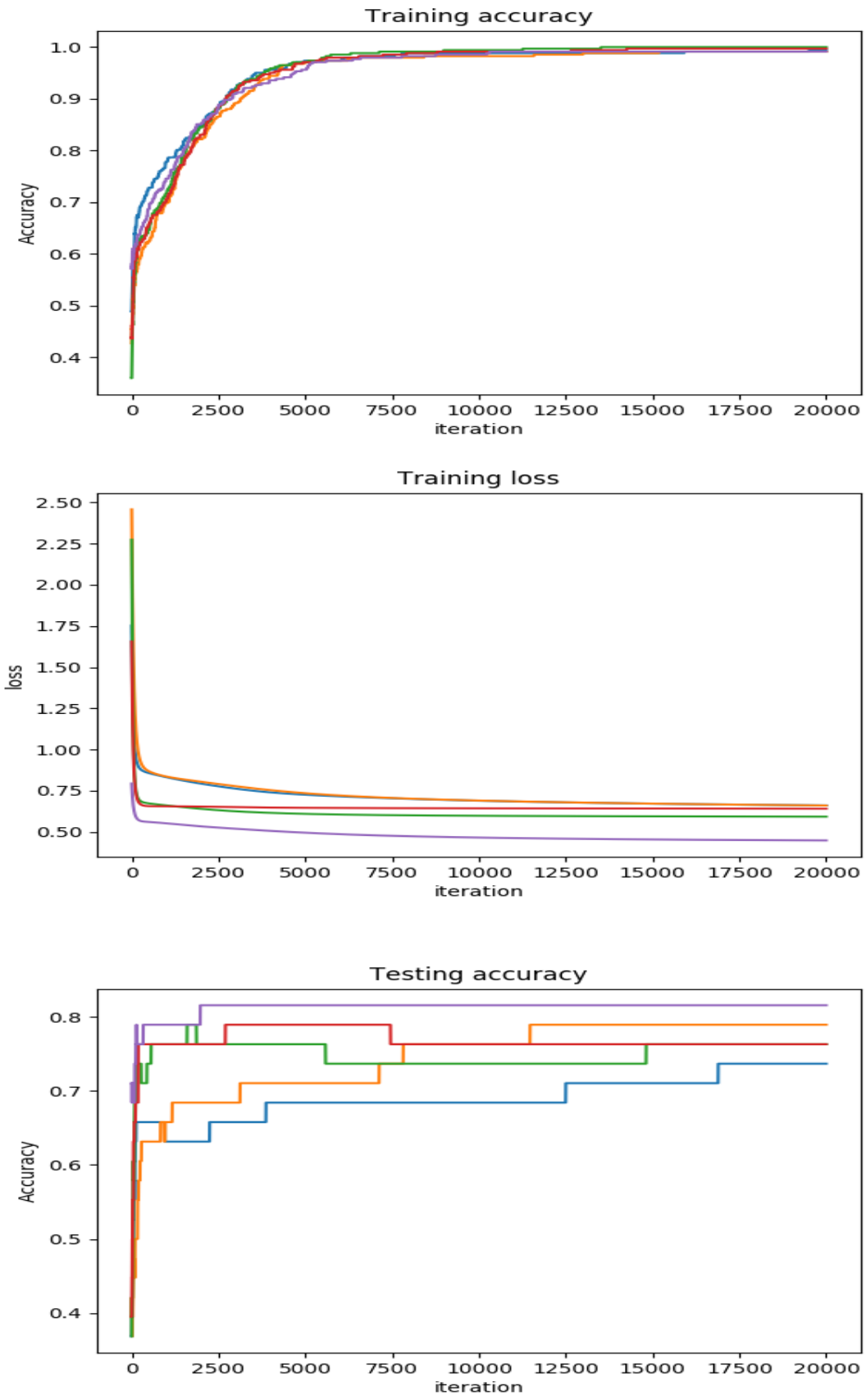


Figure 13: A comparison of the accuracy of the softmax regression (top), the loss (middle), and the testing accuracy (bottom) for binary classification across 5 independent runs.

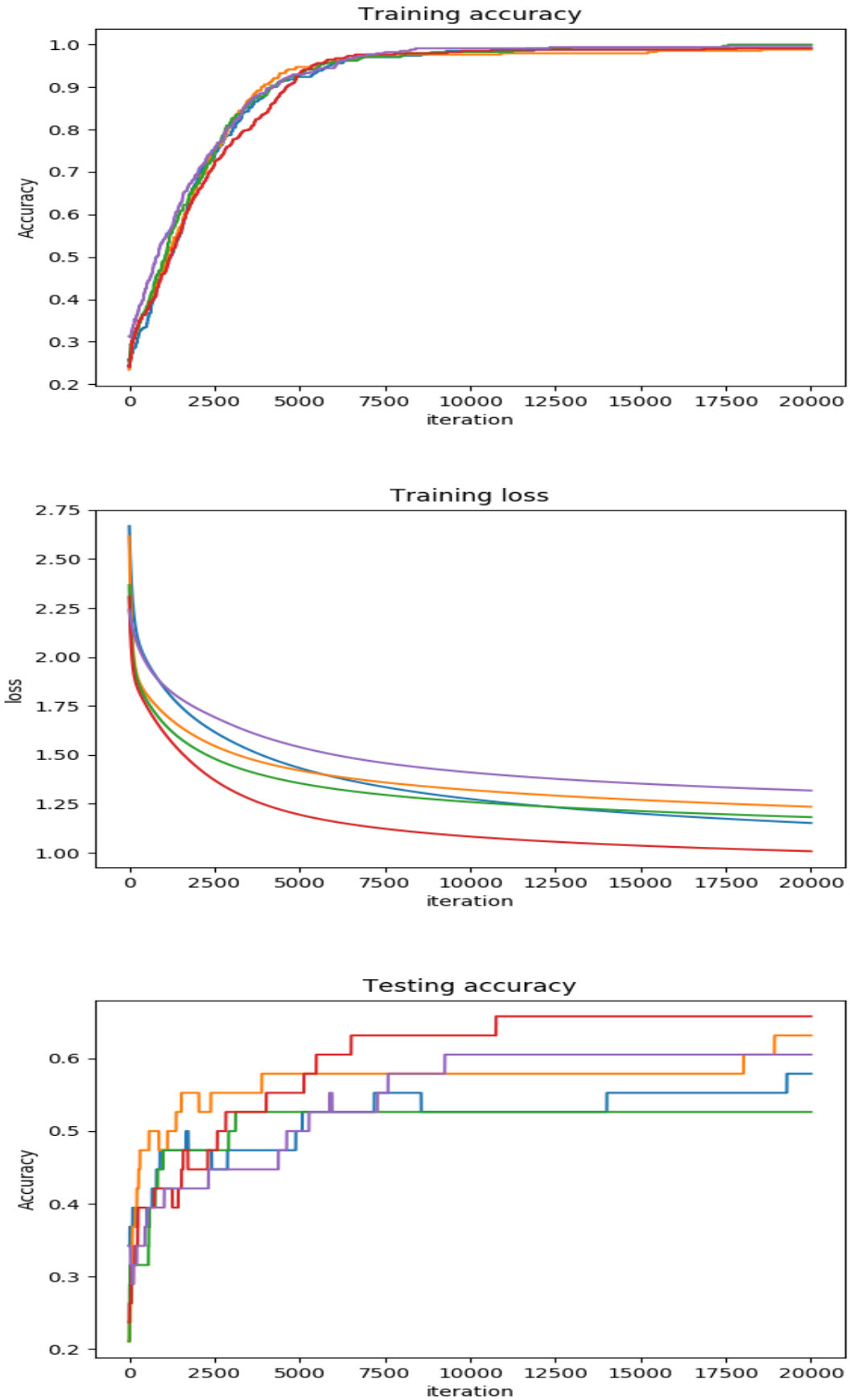


Figure 14: The four class softmax regression classifier metrics for training accuracy (top), loss (middle), and testing accuracy (bottom).

4 Discussion

Nextflow was demonstrated to be well suited for working with a mix of Docker and Singularity containers through Nextflow's built-in Singularity support. It can be easily configured to pass necessary command line options to Singularity to perform the GPU pass through. Singularity is even able to pull GPU containers from Docker Hub and execute these as was demonstrated by the TensorFlow runs which used a Docker container. Although other works have demonstrated the use of GPU containers for a single tool, the use of multiple heterogeneous containers appears to be unique at the time of writing.

The benchmarks demonstrated the advantages of using GPU short read aligners, providing a way to significantly speedup the alignment step. It was also determined that the containers did not result in any significant performance loss when compared to running the tools without a container. Likewise, a machine learning classifier was deployable in a Singularity container, although the model itself could be improved.

This work has resulted in a number of containers and tools being made publicly available. The containers for both nvBowtie and BarraCUDA are available on Singularity Hub, and can be easily incorporated into Nextflow workflows or used independently. The OneHotVCF Python package was developed to provide a way to encode variants for machine learning, and is available on GitHub. The workflow itself has been published to GitHub and can be run directly from any computer with Nextflow, Singularity, and NVIDIA CUDA drivers installed. The GPU containers are relatively large due requiring the CUDA libraries to compile the software packages as part of Singularity Hub. Once the tools have been compiled the CUDA drivers are not necessary as the required libraries are just mounted from the host operating system. For images created locally this can be avoided by compiling the GPU tools on the host and then placing just the executable in the container. It may also be possible to uninstall the CUDA libraries after compiling, though this was not tested. The containers were demonstrated to be highly portable, as the configuration of the containers was done on a local machine with a consumer grade GPU. They were then deployed on AWS cloud instances with professional NVIDIA K80 GPUs.

An underlying challenge of this work is that the cell populations are heterogeneous. The specificity of the machine learning classifier may benefit from additional labels or an unsupervised model which could cluster the variants based on similarities and not exclusively the labels. The authors of the paper that generated this data analyzed the cells using gene expression and copy number. They found various cellular subtypes, including six subtypes of the TNBC cells (Chung et al., 2017). The sample set was chosen, before the paper was published describing the authors' findings, because it initially appeared to have four distinct classes. A larger sample set may also improve the accuracy of the machine learning models. The four class softmax regression performed substantially worse than the binary. This could be in part due to training on smaller numbers of samples for each class. The two models used here are primarily for linearly separable data. Additional models, such as a neural network, may be able to separate a more complex relationship in the data.

Although the processing parameters and machine learning model could be improved, this work demonstrates the possibility of using GPU Singularity containers within a heterogeneous Nextflow workflow. This has benefits for bioinformatics tools which can utilize the parallelization of GPUs and also for machine learning steps which could be incorporated within a workflow. The Singularity GPU option is a relatively new feature with limited documentation at the time of writing. This work seems to be the first in the field of bioinformatics utilizing this container feature.

As bioinformatics workflows are required to process more data, GPGPU tools may become even more widespread. The ability to easily incorporate these into Nextflow is a unique feature that may enable further novel research. Additionally, this may extend the utility of Nextflow as a

workflow manager to domains within bioinformatics which rely heavily on GPU software such as molecular modeling.

5 Future works

This work has shown that Nextflow and Singularity are a viable way to develop GPU enabled workflows for deployment to the cloud. Currently Nextflow provides a way to provision only a single type of AWS instance. A significant improvement to the platform would be a way to provision a heterogeneous cluster. Each process block could then define an instance type for execution as different steps of a bioinformatics workflow can benefit from different hardware configurations. This could also potentially be achieved using AWS Batch as Nextflow is developing support for this service. Another modification to Nextflow could be that each process could define container runtime parameters, such as the GPU flag used. No effort was made to compare launching of containers with and without the GPU flag, but it may be worth investigating.

The aligners used here were not splice aware as they were designed primarily for aligning DNA reads to a reference genome. The transcriptome file avoids this issue as the introns have already been removed and only the coding segments remain. This does limit the mapping to only those previously discovered and annotated coding regions. RNA-seq mappers such as STAR and Tophat2 can align reads to a reference genome. These tools did not have GPU implementations at the time of writing. Tophat2 uses bowtie2 as the underlying aligner, and it was briefly explored if nvBowtie could replace bowtie2. Despite having identical commandline options, Tophat2 performs a version check which was not compatible with nvBowtie.

The work here utilized BarraCUDA and nvBowtie for indexing and alignment of the RNA reads. Additional short read aligners exist which may be valuable to containerize and compare. These aligners had compile or execution issues that are outside of the scope of this work. A more complete comparison of aligners, both GPU and CPU based, would highlight any limitations of BarraCUDA. Also Platypus was the only variant caller used, a comparison with GATK or other variant callers could also be of interest.

Bibliography

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *None*, *1*(212), 19. https://doi.org/10.1038/nn.3331

2. Benedicic, L., Cruz, F. A., Madonna, A., & Mariotti, K. (2017). Portable, high-performance containers for HPC. *arXiv*. Retrieved from http://arxiv.org/abs/1704.03383

3. Bergstra, J., Breuleux, O., Bastien, F. F., Lamblin, P., Pascanu, R., Desjardins, G., … Bengio, Y. (2010). Theano: a CPU and GPU math compiler in Python. *Proceedings of the Python for Scientific Computing Conference (SciPy)*, (Scipy), 1–7. Retrieved from http://www-etud.iro.umontreal.ca/~wardefar/publications/theano_scipy2010.pdf

4. Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, *49*(1), 71–79. https://doi.org/10.1145/2723872.2723882

5. Calmels, J., & Abecassis, F. (2017). Using Containers for GPU-Accelerated Applications. Retrieved from http://on-demand.gputechconf.com/gtc/2017/presentation/s7177-abecassis-calmels-containers-gpu-accelerated-applications.pdf

6. Chung, W., Eum, H. H., Lee, H.-O., Lee, K.-M., Lee, H.-B., Kim, K.-T., … Park, W. (2017). Single-cell RNA-seq enables comprehensive tumour and immune cell profiling in primary breast cancer. *Nature Communications*, *8*(May), 15081. https://doi.org/10.1038/ncomms15081

7. Conesa, A., Madrigal, P., Tarazona, S., Gomez-Cabrero, D., Cervera, A., McPherson, A., … Mortazavi, A. (2016). A survey of best practices for RNA-seq data analysis. *Genome Biology*, *17*, 13. https://doi.org/10.1186/s13059-016-0881-8

8. Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., & Notredame, C. (2017). Nextflow enables reproducible computational workflows. *Nature Biotechnology*, *35*(4), 316–319. https://doi.org/10.1038/nbt.3820

9. Di Tommaso, P., Palumbo, E., Chatzou, M., Prieto, P., Heuer, M. L., & Notredame, C. (2015). The impact of Docker containers on the performance of genomic pipelines. *PeerJ*, *3*, e1273. https://doi.org/10.7717/peerj.1273

10. Dobin, A., Davis, C. A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., … Gingeras, T. R. (2013). STAR: Ultrafast universal RNA-seq aligner. *Bioinformatics*, *29*(1), 15–21. https://doi.org/10.1093/bioinformatics/bts635

11. Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). Cloud Computing and Grid Computing 360-degree compared. In *Grid Computing Environments Workshop, GCE 2008*. https://doi.org/10.1109/GCE.2008.4738445

12. Hamp, T., & Rost, B. (2015). More challenges for machine-learning protein interactions. *Bioinformatics*, *31*(10), 1521–1525. https://doi.org/10.1093/bioinformatics/btu857

13. Kashyap, H., Ahmed, H. A., Hoque, N., Roy, S., & Bhattacharyya, D. K. (2015). Big Data Analytics in Bioinformatics: A Machine Learning Perspective, *13*(9), 1–20.

14. Kim, D., Pertea, G., Trapnell, C., Pimentel, H., Kelley, R., & Salzberg, S. L. (2013). TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, *14*(4), R36. https://doi.org/10.1186/gb-2013-14-4-r36

15. Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, 1–15. https://doi.org/http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503

16. Klus, P., Lam, S., Lyberg, D., Cheung, M. S., Pullan, G., McFarlane, I., … Lam, B. Y. (2012). BarraCUDA - a fast short read sequence aligner using graphics processing units. *BMC Research Notes*, *5*(October 2015), 27. https://doi.org/10.1186/1756-0500-5-27

17. Kolodziejczyk, A. A., Kim, J. K., Svensson, V., Marioni, J. C., & Teichmann, S. A. (2015). The Technology and Biology of Single-Cell RNA Sequencing. *Molecular Cell*. https://doi.org/10.1016/j.molcel.2015.04.005

18. Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PLoS ONE*, *12*(5). https://doi.org/10.1371/journal.pone.0177459

19. Langdon, W. B., Lam, B. Y. H., Petke, J., & Harman, M. (2015). Improving CUDA DNA Analysis Software with Genetic Programming. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15* (pp. 1063–1070). https://doi.org/10.1145/2739480.2754652

20. Leprevost, F. da V., Grüning, B. A., Alves Aflitos, S., Röst, H. L., Uszkoreit, J., Barsnes, H., … Perez-Riverol, Y. (2017). BioContainers: An open-source and community-driven framework for software standardization. *Bioinformatics*, (March), 1–3. https://doi.org/10.1093/bioinformatics/btx192

21. Libbrecht, M. W., & Noble, W. S. (2015). Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, *16*(6), 321–332. https://doi.org/10.1038/nrg3920

22. Liu, C. M., Wong, T., Wu, E., Luo, R., Yiu, S. M., Li, Y., … Lam, T. W. (2012). SOAP3: Ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics*, *28*(6), 878–879. https://doi.org/10.1093/bioinformatics/bts061

23. Love, M. I., Huber, W., & Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, *15*(12), 550. https://doi.org/10.1186/s13059-014-0550-8

24. Manavski, S. a, & Valle, G. (2008). CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics*, *9 Suppl 2*, S10. https://doi.org/10.1186/1471-2105-9-S2-S10

25. Markovtsev, V. (2017). Using Docker & CoreOS For GPU Based Deep Learning. Retrieved from https://blog.sourced.tech/post/docker_coreos_gpu_deep_learning/

26. Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment | Linux Journal. Retrieved August 28, 2017, from http://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment?page=0,1

27. Nickolls, J., & Dally, W. J. (2010). The GPU computing era. *IEEE Micro*, *30*(2), 56–69. https://doi.org/10.1109/MM.2010.41

28. Nobile, M. S., Cazzaniga, P., Tangherloni, A., & Besozzi, D. (2016). Graphics processing units in bioinformatics, computational biology and systems biology. *Briefings in Bioinformatics*, (May), bbw058. https://doi.org/10.1093/bib/bbw058

29. Noble, W. S. (2006). What is a support vector machine? *Nature Biotechnology*, *24*(12), 1565–1567. https://doi.org/10.1038/nbt1206-1565

30. Oikkonen, L., & Lise, S. (2017). Making the most of RNA-seq: Pre-processing sequencing data with Opossum for reliable SNP variant detection. *Wellcome Open Research*, *2*, 6. https://doi.org/10.12688/wellcomeopenres.10501.1

31. Quinn, E. M., Cormican, P., Kenny, E. M., Hill, M., Anney, R., Gill, M., … Morris, D. W. (2013). Development of Strategies for SNP Detection in RNA-Seq Data: Application to Lymphoblastoid Cell Lines and Evaluation Using 1000 Genomes Data. *PLoS ONE*, *8*(3). https://doi.org/10.1371/journal.pone.0058815

32. Rimmer, A., Phan, H., Mathieson, I., Iqbal, Z., Twigg, S. R. F., Wilkie, A. O. M., … Lunter, G. (2014). Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications. *Nature Genetics*, *46*(November 2013), 1–9. https://doi.org/10.1038/ng.3036

33. Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv*, 1–14. https://doi.org/10.1111/j.0006-341X.1999.00591.x

34. Saliba, A. E., Westermann, A. J., Gorski, S. A., & Vogel, J. (2014). Single-cell RNA-seq: Advances and future challenges. *Nucleic Acids Research*. https://doi.org/10.1093/nar/gku555

35. Schatz, M. C., Trapnell, C., Delcher, A. L., & Varshney, A. (2007). High-throughput sequence alignment using Graphics Processing Units. *BMC Bioinformatics*, *8*(1), 474. https://doi.org/10.1186/1471-2105-8-474

36. Schmidt, B. (2010). Bioinformatics: high performance parallel computer architectures. *CRC Press*.

37. Stephens, Z. D., Lee, S. Y., Faghri, F., Campbell, R. H., Zhai, C., Efron, M. J., … Robinson, G. E. (2015). Big data: Astronomical or genomical? *PLoS Biology*, *13*(7), 1–11. https://doi.org/10.1371/journal.pbio.1002195

38. Swan, A. L., Mobasheri, A., Allaway, D., Liddell, S., & Bacardit, J. (2013). Application of Machine Learning to Proteomics Data: Classification and Biomarker Identification in Postgenomics Biology. *OMICS: A Journal of Integrative Biology*, *17*(12), 595–610. https://doi.org/10.1089/omi.2013.0017

39. Vapnik, V. N. (1999). An overview of statistical learning theory. *{IEEE} Transactions on Neural Networks*, *10*(5), 988–999. https://doi.org/10.1109/72.788640

40. Vervier, K., Mahé, P., Tournoud, M., Veyrieras, J. B., & Vert, J. P. (2016). Large-scale machine learning for metagenomics sequence classification. *Bioinformatics*, *32*(7), 1023–1032. https://doi.org/10.1093/bioinformatics/btv683

41. Vouzis, P. D., & Sahinidis, N. V. (2011). GPU-BLAST: using graphics processors to accelerate protein sequence alignment. *Bioinformatics (Oxford, England)*, *27*, 182–188. https://doi.org/10.1093/bioinformatics/btq644

42. Wang, Z., Gerstein, M., & Snyder, M. (2009). RNA-Seq: a revolutionary tool for transcriptomics. *Nature Reviews. Genetics*, *10*(1), 57–63. https://doi.org/10.1038/nrg2484

43. Wu, E., & Liu, Y. (2008). Emerging technology about GPGPU. In *IEEE Asia-Pacific Conference on Circuits and Systems, Proceedings, APCCAS* (pp. 618–622). https://doi.org/10.1109/APCCAS.2008.4746099

44. Zhao, S. (2014). Assessment of the impact of using a reference transcriptome in mapping short RNA-Seq reads. *PLoS ONE*, *9*(7). https://doi.org/10.1371/journal.pone.0101374

Appendix I: AWS AMI setup

The steps to replicate the private AMI used

1. The AMI ami-b3eccbd5 provided by the University of Surrey for deep learning was used as the base AMI. This AMI already has the CUDA8 libraries installed
2. An instance was started with this AMI
3. The Singularity source code (v 2.3.1) was downloaded and installed following the instructions provided here: http://singularity.lbl.gov/install-linux
4. Nextflow was installed using the command provided on the site (v0.25.5)
5. Next in the AWS console select the instance, Actions → image → Create Image
6. It was then named and created

Appendix II: Containers and repositories

The containers used in this work are listed below along with the repository address that can be used to pull them using either Singularity or Docker (excluding nvBowtie and BarraCUDA).

Containerized tool	Container type	Repository
Opossum	Docker	Docker://sayerskt/opossum/
Platypus	Docker	Docker://sayerskt/platypus/
SRA Toolkit	Docker	Docker://sayerskt/sratoolkit/
Samtools	Docker	Docker://sayerskt/samtools/
BarraCUDA	Singularity	Shub://KevinSayers/BarraCUDA_Singularity
NvBowtie	Singularity	Shub://KevinSayers/nvBowtie_Singularity
TensorFlow	Docker	docker://tensorflow/tensorflow:latest-gpu

Appendix III: Benchmark results

name	status	exit	duration	realtime	%cpu
barracudaIndex	COMPLETED	0	5m 1s	5m 1s	100.00%
barracudaAln	COMPLETED	0	2m 41s	2m 41s	98.40%
barracudaSampe	COMPLETED	0	2m 42s	2m 42s	343.00%
name	status	exit	duration	realtime	%cpu
nvBWTIndex	COMPLETED	0	1m 27s	1m 27s	100.00%
nvBowtieAlign	COMPLETED	0	3m 9s	3m 9s	187.00%
name	status	exit	duration	realtime	%cpu
barracudaIndex	COMPLETED	0	5m 4s	5m 3s	100.00%
barracudaAln	COMPLETED	0	2m 42s	2m 42s	97.20%
barracudaSampe	COMPLETED	0	2m 41s	2m 41s	310.00%
name	status	exit	duration	realtime	%cpu
nvBWTIndex	COMPLETED	0	1m 27s	1m 27s	99.60%
nvBowtieAlign	COMPLETED	0	3m 10s	3m 10s	184.00%
name	status	exit	duration	realtime	%cpu
barracudaIndex	COMPLETED	0	5m	5m	107.00%
barracudaAln	COMPLETED	0	2m 42s	2m 42s	97.40%
barracudaSampe	COMPLETED	0	2m 42s	2m 41s	342.00%
name	status	exit	duration	realtime	%cpu
nvBWTIndex	COMPLETED	0	1m 27s	1m 27s	99.50%
nvBowtieAlign	COMPLETED	0	3m 9s	3m 9s	189.00%
name	status	exit	duration	realtime	%cpu
barracudaIndex	COMPLETED	0	5m 3s	5m 3s	100.00%
barracudaAln	COMPLETED	0	2m 42s	2m 41s	98.50%
barracudaSampe	COMPLETED	0	2m 41s	2m 41s	313.00%
name	status	exit	duration	realtime	%cpu
nvBWTIndex	COMPLETED	0	1m 28s	1m 28s	99.40%
nvBowtieAlign	COMPLETED	0	3m 11s	3m 11s	187.00%
name	status	exit	duration	realtime	%cpu
barracudaIndex	COMPLETED	0	5m 2s	5m 2s	106.00%
barracudaAln	COMPLETED	0	2m 42s	2m 42s	97.60%
barracudaSampe	COMPLETED	0	2m 50s	2m 50s	313.00%
name	status	exit	duration	realtime	%cpu
nvBWTIndex	COMPLETED	0	1m 27s	1m 27s	99.90%
nvBowtieAlign	COMPLETED	0	3m 11s	3m 11s	187.00%

nvBowtie stats

10011968 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 duplicates
5629476 + 0 mapped (56.23%:-nan%)
10011968 + 0 paired in sequencing
5005984 + 0 read1
5005984 + 0 read2
5060542 + 0 properly paired (50.54%:-nan%)
5176434 + 0 with itself and mate mapped
453042 + 0 singletons (4.53%:-nan%)
161257 + 0 with mate mapped to a different chr
88309 + 0 with mate mapped to a different chr (mapQ>=5)

BarraCUDA stats

10011968 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 duplicates

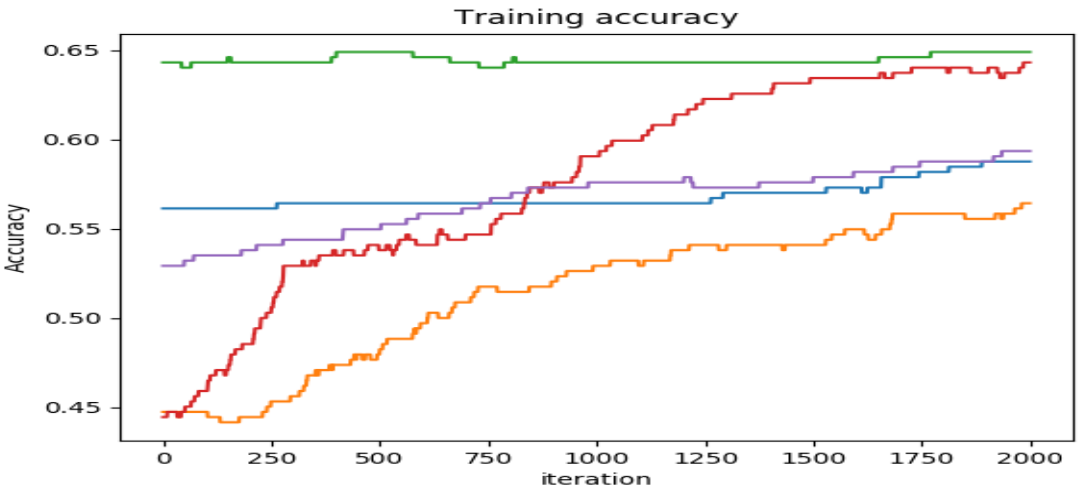
5629476 + 0 mapped (56.23%:-nan%)
10011968 + 0 paired in sequencing
5005984 + 0 read1
5005984 + 0 read2
5060542 + 0 properly paired (50.54%:-nan%)
5176434 + 0 with itself and mate mapped
453042 + 0 singletons (4.53%:-nan%)
161257 + 0 with mate mapped to a different chr
88309 + 0 with mate mapped to a different chr (mapQ>=5)

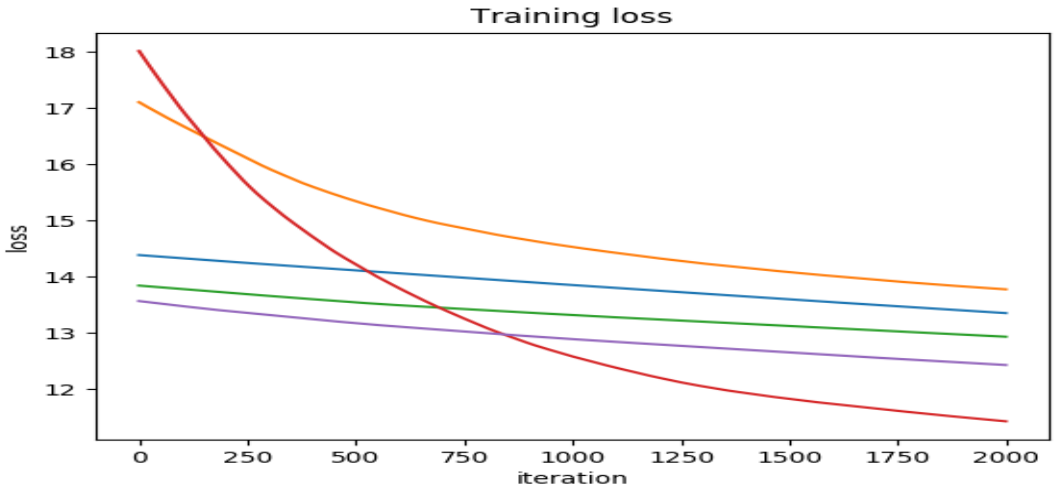
Bowtie2 stats
10011968 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 duplicates
6175625 + 0 mapped (61.68%:-nan%)
10011968 + 0 paired in sequencing
5005984 + 0 read1
5005984 + 0 read2
4616444 + 0 properly paired (46.11%:-nan%)
5729004 + 0 with itself and mate mapped
446621 + 0 singletons (4.46%:-nan%)
571616 + 0 with mate mapped to a different chr
81050 + 0 with mate mapped to a different chr (mapQ>=5)

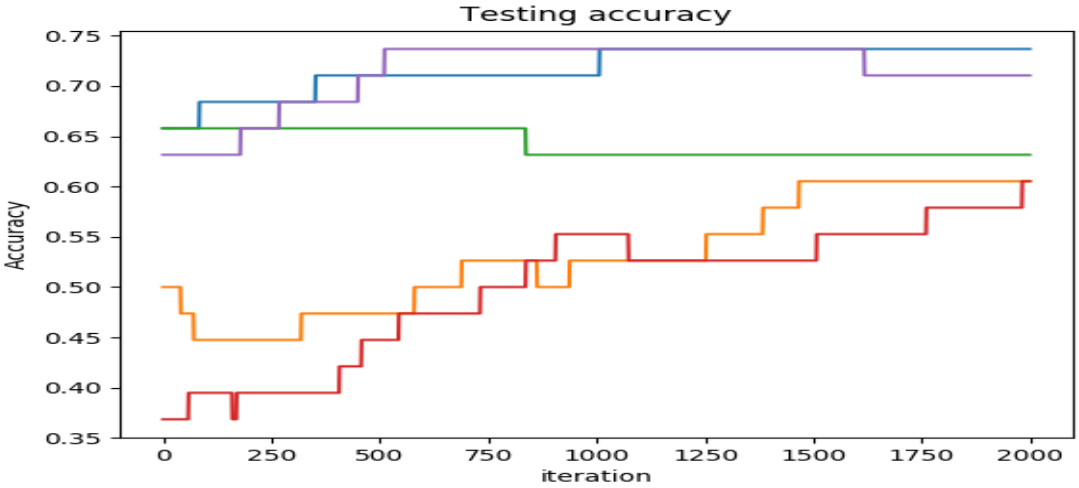
Appendix IV Optimizer comparison

Two optimization functions were compared. The AdamOptimizer (Kingma & Ba, 2015) and the standard GradientDescentOptimizer (Ruder, 2016) This was done primarily as a way to understand the differences in learning achieved by altering the optimizer. The plots are shown below. Adam was ultimately selected as the training accuracy seemed to perform more consistently on the data.

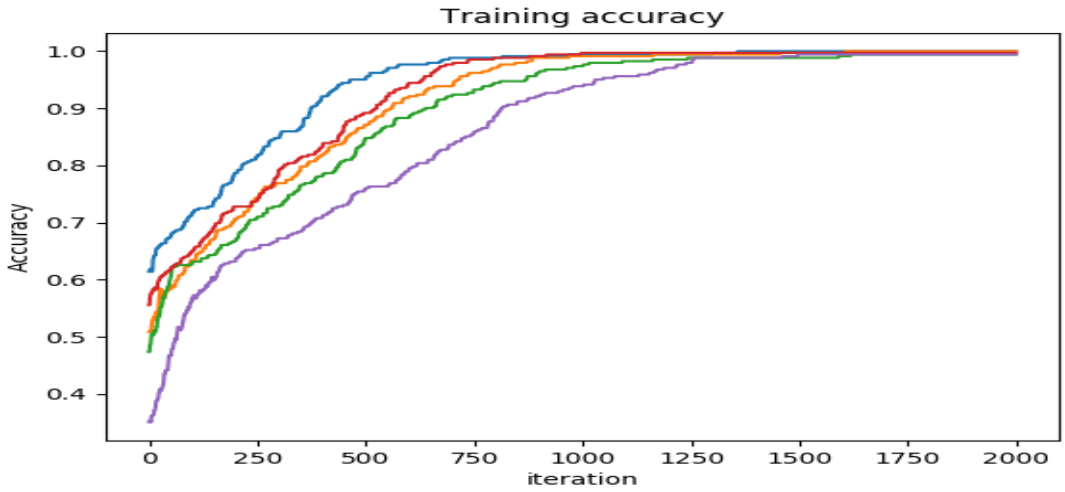
Gradient Descent Optimization:

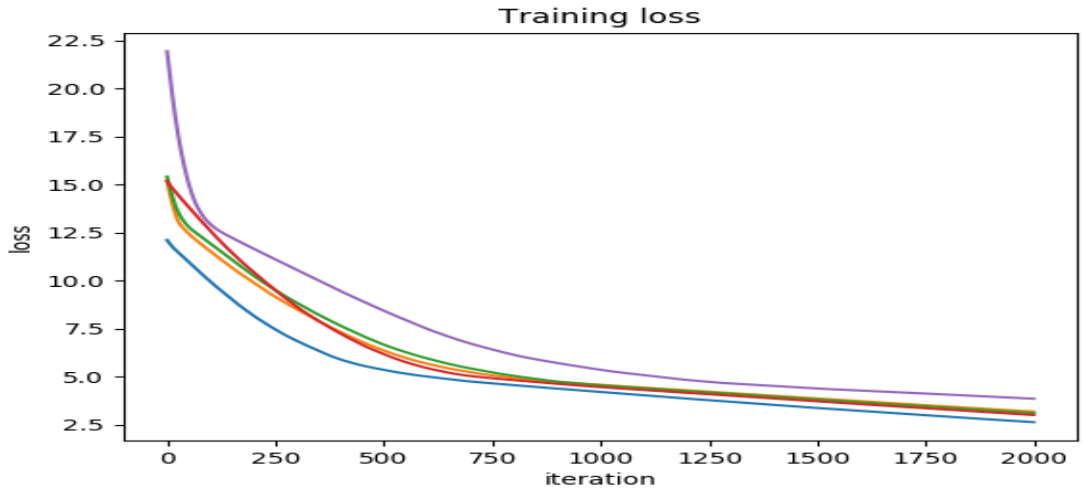


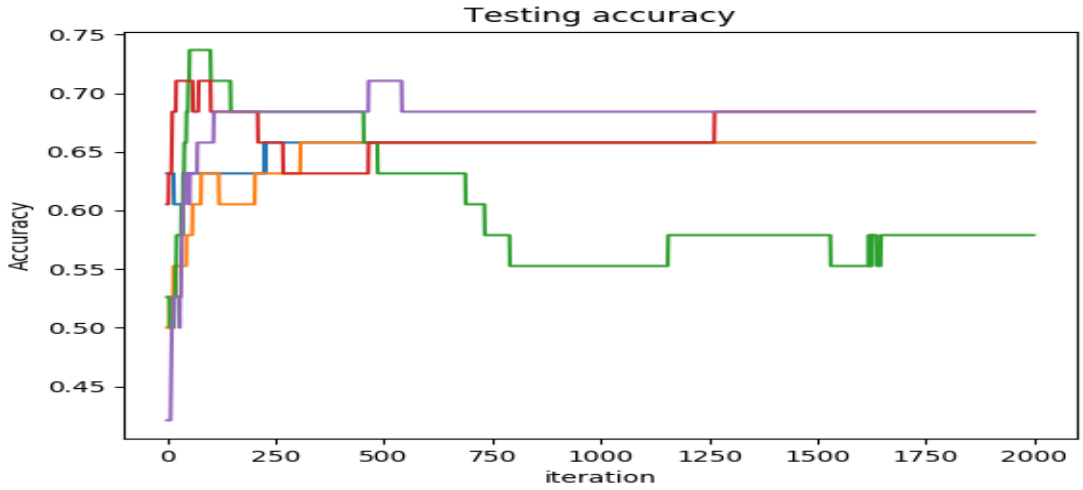




Adam Optimizer




Training loss
loss
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
250
500
750
1000
1250
1500
1750
2000
iteration


Testing accuracy
Accuracy
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0
250
500
750
1000
1250
1500
1750
2000
iteration

Appendix V Example run of SRAGPU-nf

Below are a list of necessary commands to execute the workflow developed here from a local machine. It is assumed the AWS credentials have already been configured in the host OS.

- Git clone https://github.com/KevinSayers/SRAGPU-nf
    - This will pull the project from github, this allows the provisioning of instances using Nextflow.
- nextflow cloud create gpuinstance1 -c 1
    - This provisions a single instance the default configuration is with a private AMI. This must be adjusted using the AMI creation steps in **Appendix I**
- ssh -i key username@ec2-52-211-84-127.eu-west1.compute.amazonaws.com
    - Once the cloud has been created an ssh command will be displayed with the correct information to the head node
- nextflow run main.nf
    - This will run the Nextflow sample processing workflow. Containers will automatically be pulled at each process execution
    - This step will take roughly 20-25 minutes per sample
- nextflow run preprocess.nf
    - Will take the VCF output folder and run the preprocessing steps on it. This will ultimately yield a textfile of onehot encoded positions.
- Nextflow run classify.nf
    - This will run the TensorFlow models on the encoded file
    - It outputs the train, loss, and testing plots presented in the main body of this work.

Appendix VI hardware

- AWS instances
  - p2.xlarge
  - 4 vCPUs
  - 61GB RAM
  - 800GB storage (default)
  - NVIDIA Tesla K80 GPU
- local development
  - Intel core i7 quad core
  - 16GB
  - 512GB storage
  - NVIDIA GeForce 960M