Large Scale Analytics on Scientific Image Data

Parmita Mehta

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Magdalena Balazinska, Chair

Su-In Lee

Ariel Rokem

Hannaneh Hajishirzi

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Large Scale Analytics on Scientific Image Data

Parmita Mehta

Chair of the Supervisory Committee:
Professor Magdalena Balazinska
Computer Science and Engineering

Scientific discoveries are increasingly driven by analyzing large volumes of data. Advances in data collection and storage technologies, availability of cloud compute resources, and better algorithms and readily available open-source libraries are responsible in equal measure for this phenomenon. Large proportion of scientific data is in form of images as many scientific instruments such as telescopes, microscopes, satellites, x-rays, MRIs, etc. produce data in image formats. However, commercial systems have paid scant attention to scientific image analysis workloads and as a result scientists working with images spend a lot of effort building bespoke and often fragile support for such analyses.

In this dissertation, we first evaluate several popular systems on scientific image analysis workloads. We then perform an in-depth image analysis, which yields novel results in ophthalmology. Finally, we use our findings to propose a novel technique to ease some of the data management burden associated with scientific image analysis, specifically debugging of deep neural networks.

Specifically, first we assess existing big data systems and frameworks for suitability of scientific image analyses workloads. We evaluate five representative systems (SciDB, Myria, Spark, Dask, and TensorFlow) both qualitatively (ease of use) and quantitatively (scalability and performance) on two real-life image analysis use cases from astronomy and neuroscience. We find that each of them has shortcomings that complicate implementation or hurt performance.

Next, we propose a new, comprehensive, and more accurate ML-based approach for population-

level glaucoma screening. In this project we embed ourselves in the process of scientific discovery by analyzing a publicly available large dataset to further the state of art in ophthalmology. Our model is highly accurate (AUC 0.97) and interpretable. It validates biological features known to be related to the disease, such as age, intraocular pressure and optic disc morphology. Our model also points to previously unknown or disputed features, such as pulmonary capacity and retinal outer layers.

Finally, we utilize lessons from building interpretable deep learning models for automated glaucoma detection to propose a novel sampling technique for deep learning model diagnosis. Our experience demonstrated that scientists utilizing deep learning often spend majority of their time managing the data associated rather than focusing on science. Our sampling technique seeks to reduce the data management burden for scientist working on such analyses, making the process of deep learning model diagnosis simpler and more efficient.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Number Page

Acknowledgments

"In writing this book I have drawn so much on he work of other linguists that few if any of my ideas, except perhaps mistaken ones, are original with me."
– Seth Lindstromberg, English Prepositions Explained

" I am so glad that the undo thing is there! for I have done some bad things."
– Diya, Age 10, April 2020

This thesis would never be complete without the support of many people.

First I would like to thank my advisors, Magda Balazinska and Su-In Lee. I found fierce supporters, and unending optimists in both Magda and Su-In, for which I am very grateful. I want to thank DB lab faculty Dan Suciu and Alvin Cheung for providing valuable feedback. I would like to thank professors Hannaneh Hajishirzi, and Ione Fine for agreeing to be a part of my dissertation committee. I have been the doubly fortunate of being part of the DB lab and AIMS lab and I am thankful to my lab mates from both groups for listening to the often arcane research presentations and my laments about the various dead-ends that are natural outcome of any research. A large part of my PhD was collaborating with domain scientists, and I want to thank Prof. Andrew Connolly, Dr. Ariel Rokem, Dr. Aaron Lee for being very patient in teaching me about their respective fields (astronomy, neuroscience and ophthalmology) and answering my endless questions. I would like to thank my co-authors Sven Dorkenwald, Dongfang Zhao, Tomer Kaftan, Jacob Vanderplas, Yusra AlSayyad and Stephen Portillo, without them these papers would be nothing but doodles. My office mates, Ezgi, Deepali, Nick, Bindita, Beibin, Shima and Bindita for their support in figuring out how to get the printers and poster printers working, answering random questions on computer vision, and most importantly very timely notifications of when food was available in the kitchen downstairs.

I wanted to thank Allen school staff, without whom Allen school of computer science and engineering would cease to exist. I especially wanted to thank Sandy Kaplan and Elise deGoede. Sandy, for painstakingly editing my papers, teaching me the basics of technical writing and unflinching support and advice. As for Elsie, without her endless support and feedback I do not believe I could have completed my degree.

I am blessed to have a huge family who cheered me on. My parents, Ashok and Sharda Mehta without whom I literally would not exist. My kid Uma, who now knows more about image data and big data systems than they care for and for their patient instruction on the correct usage of commas, and the definitive article. My siblings Meenal and Bhaskar and, their spouses Pankaj and Divya. Family vacations and plans during the PhD years were oft marred by my conference and other self imposed deadline, which all of my they cheerfully put up with. I could not have done this without their support. I specially want to call out my sister, Meenal, who talked to me for endless hours, reminding me why I wanted to do this PhD, for while in throes of the program I needed frequent reminders. I want to thank my friends, Sangeeta, Sophie, Sally and Pushpa aunty all of whom put up with my endless disappearances.

Chapter 1

INTRODUCTION

"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."

– Lewis Carroll

"Scientific breakthroughs will be powered by advanced computing capabilities that help researchers manipulate and explore massive data sets."

– Jim Gray, The Fourth Paradigm

With advances in data collection and storage technologies, data analysis has been heralded as the fourth paradigm of science after empirical evidence, scientific theory, and computational science [61]. Scientific discoveries are increasingly driven by analyzing immense volumes of data. Many of these massive and varied data sets are publicly available. For example, UK Biobank [184] an international health resource that strives to improve prevention, diagnosis, and treatment of a wide range of life-threatening illnesses recruited 500,000 people between 40-69 years in 2006-2010; these participants underwent clinical assessment, provided blood, urine, and saliva samples for future analysis, and gave detailed information about themselves, agreeing to have their health followed over time. The size of this data is staggering: Magnetic Resonance Imaging (MRI) data alone contains over 200 TB [119]. This data is being used by over 13,000 researchers working in more than 1375 institutes in 68 countries across the globe [184]. Another example is the Large Synoptic Survey Telescope (LSST), a large-scale international initiative to build a new telescope

for surveying the visible sky [107], which plans to collect 60PB of images over 10 years. In previous astronomy surveys (e.g., the Sloan Digital Sky Survey [158]), an expert team of engineers processed collected images on dedicated servers, and results were distilled into textual catalogs for other astronomers to analyze. In contrast, LSST seeks to broaden access to the collected images for astronomers around the globe, enabling them to run analyses directly on the images.

These initiatives have become the norm. Many other programs are similarly making vast collections of data available to researchers [5, 87, 179]. In a broad range of scientific fields, an increasing portion of this data consists of images [82, 83].

1.1 Image Analytics

Image analytics involves extracting meaningful information from image data. Scientific instruments (such as telescopes, microscopes, satellites, x-rays, MRIs, etc.) record data in image form. Analyzing these scientific images requires performing tasks such as: (1) *cleaning images*, or removing background noise and instrument artifacts, (2) *finding areas of interest*, which involves automated segmenting or inferring areas of interest using a labeled image set, and (3) *combining images*, which requires stacking two-dimensional images to form a three-dimensional volume or stitching multiple over-lapping smaller images to generate a combined larger one. Once these and other tasks are completed, scientists can extract specific properties of interest and use the processed images to enhance further scientific discovery.

For instance, astronomers use telescope images to create a sky map. Their analysis requires loading images from Flexible Image Transport System (FITS) files; FITS is an open standard defining a digital file format useful for storage, transmission and processing of data. Scientists then process the images, which involves cleaning them by normalizing pixel values according to minimum and maximum intensities observed across the full sky; transforming the images by mapping their pixels to a new pixel grid; merging images across multiple observations to remove temporary artifacts; and rendering a sky map at the desired resolution with interpolation. Image analysis tasks include performing operations such as spatial joins, slicing, dicing, roll-ups, stencil operations (over sliding windows) and complex user-defined functions expressed in higher level

languages such as Python. This type of analysis is computationally intensive. As larger volumes of data are collected, scientists need to scale their image processing use cases.

However, systems for storing, managing, and analyzing 'big data' do not support image data and the associated scientific image analysis workflows, as we next describe.

1.2 Big Data Systems

Big data systems are characterized by their ability to store, manage and systematically extract information from, or otherwise deal with, data sets that are too large or complex to be dealt with by traditional data-processing application software [194]. A "big data system" describes any database management system or cluster computing library that provides parallel processing capabilities on large amounts of data as well as scalable and efficient ways to store and analyze such data via programming models that are easily used by domain scientists (e.g., astronomers, physicists, biologists). Most big data systems aim to support massive scale and high- speed data processing. Examples of popular big data systems include open-source systems such as Hadoop, Spark, and Cassandra [167, 68, 33].

Current big data systems are built and optimized for primitive data types, such as integers, floats and strings. Utilizing these existing systems for dense, high-dimensional data, such as images, poses critical performance and usability challenges. These challenges include ingesting, storing, and managing a smaller number of larger data items as opposed to a large number of smaller data items. Additionally, as noted, image analytics workloads require complex computations. These computations cannot be expressed using relational operators; they instead require complex, user-defined operations in high-level languages such as Python and R. Such use cases emphasize the need for effective systems to support the management and analysis of image data: systems that are efficient, scale effectively, and are easy to program without requiring deep systems expertise to deploy and tune.

Surprisingly, scant research from the data management community addresses building systems to support large-scale image *management and analytics*. Rasdaman [152] and SciDB [156] are two well-known DBMSs that specialize in the storage and processing of multidimensional array data

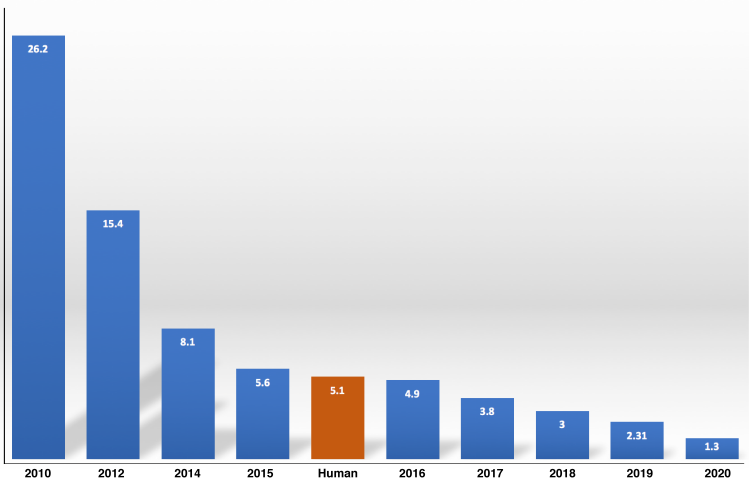


Figure 1.1: Top-5 error rate (percentage) on image classification on ImageNet data.

and are a natural choice for implementing image analytics. Besides these systems, most other work targets predominantly image storage and retrieval based on keyword or similarity searches [55, 32, 35] and does not support the image analysis use cases described above.

1.3 Deep Learning

Deep learning (DL) is an important tool scientists use to work with images. No discourse on large-scale image analytics is therefore complete without discussing its contributions. Here, we describe deep learning in the context of image analyses and data management support required to build, diagnose and explain deep learning image models.

Deep learning is a subset of machine learning where artificial neural networks, viz., algorithms inspired by the human brain, learn from large amounts of data. Compared to the traditional approach, i.e., selecting features *before* training a model, deep learning features are not manually given but rather learned post hoc and automatically from the input data, which is arranged in dense, high-dimensional numerical arrays (e.g., images, audio, and text). It is with this kind of data that deep learning has achieved paradigm-shifting performance. For instance, deep learning models outperform humans in classifying images from the ImageNet data set [84, 85], a com-

monly used benchmark for convolutional architectures, showing better-than-human performance (seeFigure 1.1).

Deep learning is a data-hungry workload [69]. The rule of thumb is that training should use $10X$ as many data samples as the number of dimensions of the input data [183]. For a $200 \times 200 \times 3$ image, this results in 1.2 million data samples. Additionally, machine learning is an iterative process. A machine learning (ML) practitioner must build and tune dozens of models before selecting one. Selection ultimately depends on measures such as accuracy being applied to a test data set that is new to the deep learning model. Selecting the *best* model means that the practitioner must run many experiments, each resulting in a candidate model. Each experiment may vary model hyper-parameters, which include: learning rate (step size of how much model weights are changed in each iteration), number of epochs (number of times that the learning algorithm will work through the entire training data set), or other strategies, such as different loss functions (optimization objective for learning), regularization strategies (explicitly designed to reduce the test error) or model architectures (number, width and type of layers). While such iterative work is true of any machine learning model, using high-dimensional input data takes longer to train deep learning models, which exacerbate the problem.

An enduring criticism of deep learning models is their black-box nature. This opacity presents a barrier to adoption of these models in applications such as medical diagnosis, where it is essential that models be easily explainable. Additionally, practitioners ask certain questions when a model does not produce desired results: "Why did the model incorrectly classify a data instance?" or "Why is the model's accuracy so low?" Conversely, when a model performs well, they want to know, "What did the model learn to achieve such high accuracy?"

Answers to these questions in the context of interpretability and diagnosis requires access to artifacts, such as model activations and gradients. Activation values, or *activations*, are learned representations of input data. *Gradients* are partial derivatives of target output (e.g., the true label of the input data) with respect to input data. These artifacts are essentially high-dimensional vectors of floating-point numbers that encapsulate how deep learning models represent input data (activations) and what parts of the input data they consider important (gradients). Pre-computing and

storing all artifacts required for model diagnosis is a naive solution, one that scales prohibitively as the product of input data size and the number of parameters of the deep learning model increases. For instance, consider a VGG-16 [190] model trained on CIFAR-10 [41]. CIFAR-10 is a moderate-sized data set with 60k images, $32 \times 32 \times 3$ pixels each; VGG-16 [190] is a deep learning model with 22 layers and $33,638,218$ learned parameters. Storing the activations for ten experiments of training CIFAR-10 data on a VGG-16 models results in *350GB of data.* While the total size of the artifacts for small data sets and models may be manageable, an overhead that is three orders of magnitude larger than the input data per model adds a significant burden of data management on the ML practitioner.

1.4 Contributions and Thesis Outline

This doctoral research and the thesis based on it make the following contributions to understanding of scientific image data processing and management:

1. We assess big data systems in terms of their ability to support image analytics workloads (Chapter 2).

2. We extend the state-of-the-art in ophthalmology by building an interpretable model for automated glaucoma detection (Chapter 3).

3. We develop new techniques for reducing the data management burden imposed by deep learning model diagnosis (Chapter 4).

Our first project investigates the efficacy of existing big data systems for large-scale image analytics. We assess five existing open-source big-data systems to evaluate their suitability for image processing tasks and outline the resulting *challenges and opportunities* for large-scale image analytics. The assessment criteria are both qualitative (ease of use) and quantitative (scalability and performance).

The second project introduces a new, comprehensive, and more accurate ML-based approach for population-level glaucoma screening. This approach is based on building multiple deep learning models on different retinal image modalities and combining them with clinical data to create *ensemble models* that are both highly accurate and explainable.

In the third and final project, we draw upon our experience building an automated disease detection model to propose a *new sampling technique* that reduces the data management burden, thereby making the process of building and diagnosing deep learning models simpler and more efficient. We discuss Related Work in Chapter 5 and conclude our inquiry in Chapter 6. We now describe each contribution in more detail.

1.4.1 Comparative Evaluation of Big-Data Systems for Large-Scale Image Analytics

In Chapter 2, we describe in detail our research that evaluates the suitability of large-scale data systems and frameworks for the scientific data analysis of images. Our goal was to assess where current big data systems perform well and where they fall short in their utility for image analytics. We investigated two real-world scientific image data processing use cases for this evaluation. The first use case takes as input diffusion MRI (dMRI) images of a human brain, which are cleaned, segmented and then used to build a model of brain connectivity. The second use case takes as input telescope images of the sky over multiple observations, which are cleaned, aligned and combined to create a comprehensive sky map.

The key questions we ask in this assessment are (1) How well do existing big data systems support the management and analysis requirements of real scientific image processing workloads? (2) Is it easy to implement large-scale image analytics using these systems? (3) How efficient are the resulting applications built atop such systems? (4) Do these systems require deep technical expertise to optimize?

We chose five big data systems for parallel data processing: a domain-specific DBMS for multidimensional array data (SciDB [156]); a general-purpose cluster computing library with persistence capabilities (Spark [167]); a traditional parallel general-purpose DBMS (Myria [70, 193]); and a general-purpose (Dask [154]) and domain-specific (TensorFlow [4]) parallel-programming li-

brary. We selected these systems because they have open-source implementations, can be deployed on commodity hardware, support complex analytics (such as linear algebra and user-defined functions), use the scientifically popular Python language for APIs [136], and have different internal architectures so we could evaluate the performance of different implementation paradigms.

Implementing both use cases on five systems was not always straight-forward and occasionally required completely re-writing the reference implementation provided by the scientists. The use cases themselves had important similarities: input data was in the form of multidimensional arrays encoded in domain-specific file formats (FITS [57], NIfTI [130], etc.); data processing involved slicing along different dimensions, aggregations, stencil (a.k.a. multidimensional window) operations, spatial joins and complex transformations expressed in Python.

Our analysis showed that leveraging the benefits of all systems requires deep technical expertise. For these systems to better support image analytics in domain sciences, they must simultaneously provide comprehensive support for multidimensional data and high performance for UDFs/UDAs written in popular languages (e.g., Python). Additionally, they must completely automate data and compute distribution across clusters and provide memory management in order to eliminate all possible sources of out-of-memory failures. Overall, we argue that current systems adequately support image analytics, but they also reveal new opportunities for further improvements and future research. The ability to use existing domain-specific code to perform sophisticated and difficult-to-rewrite operations, combined with the ability to automatically parallelize computation (by reasoning over multi-dimensional arrays) without having to manually create and process collections of image fragments, will make it easier for scientists from all fields to use these systems. It will also reduce the now effortful work of creating and maintaining bespoke and fragile solutions to support scientific image analysis.

As a precursor to this evaluation, we extended Myria with support for (1) large binary objects (BLOBs) as a native data type, and (2) user-defined functions (UDF) and aggregates (UDA) in a higher level language (e.g., Python) used in CIDR'17 [193]. The comparative evaluation study was published at VLDB'17 [116].

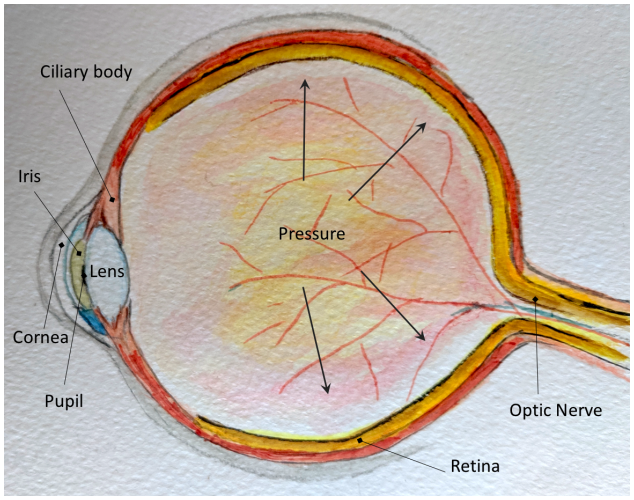


Figure 1.2: Illustration of an eye with glaucoma.

1.4.2 Automated Detection of Glaucoma in Multi-Modal Images

In Chapter 3 we present a new, comprehensive, and more accurate ML-based approach for population-level glaucoma screening. This project had two goals. First, we wanted to create a new and interpretable approach for detecting glaucoma using both clinical measurements as well as retinal images of multiple modalities. Second, we sought to understand the challenges inherent in building interpretable deep learning models by working on a real-life problem with domain experts to leverage their knowledge about useful systems and techniques for deep learning workloads.

Glaucoma, the leading cause of irreversible blindness worldwide, affects approximately 76 million people (2020) and is predicted to affect nearly 111.8 million by 2040 [178]. The risk of glaucoma increases with age [128]. As life expectancy continues to lengthen, glaucoma is expected to become a significant public health concern [178]. This disease is characterized by the progressive loss of retinal ganglion cells, which manifests as thinning of the retinal nerve fiber layer (RNFL) and characteristic changes in the optic nerve's head appearance [34]. In later stages of the disease, visual field defects develop which, if uncontrolled, can ultimately result in complete blindness.

Fig. 1.2 shows an eye affected by glaucoma. This specific case depicts primary open angle

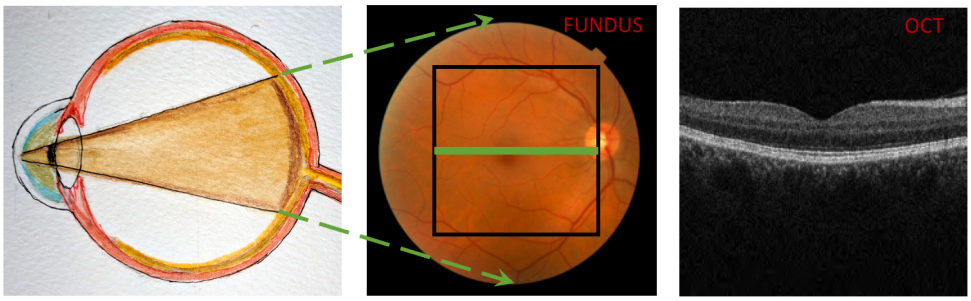


Figure 1.3: **CFP and OCT images**: From left, anatomical image of of the eye, CFP and OCT image. The black rectangle on the CFP shows the approximate area over which OCT images are captured, and the green line corresponds to the OCT image shown on the right.

glaucoma (POAG); POAG causes pressure to build in the eye and leads to damage of the optic nerve, resulting in thinning of the retinal nerve fiber layer and eventual loss of vision. To detect glaucoma, we used the UK Biobank [184] data set to build deep learning models based on retinal images. We used two categories of images: color fundus photos (CFPs) and Optical Coherence Tomography (OCT) images. In CFP, the camera uses the pupil to take an image of the back of the eye. Figure 1.3 shows an image of an eye on the left; at the back of the eye in the left image is the area depicted by the fundus image. The CFP image in the center. OCT images are shown on Figure 1.3. OCT, a non-invasive imaging test, uses light waves to take cross-section images of retina; the OCT device scans the retina in horizontal lines, traversing from the top of the eye to the bottom of the eye, creating several scans for a single retina. The black rectangle on the center fundus image shows the area scanned for OCT images. The green line shows a single OCT image on the right.

Our contributions from this project include: (1) *a multi-modal model* built upon a large data set that includes demographic, systemic and ocular data as well as raw image data taken from color fundus photos (CFPs) and macular Optical Coherence Tomography (OCT) scans, (2) *model interpretation* to identify and explain data features that lead to accurate model performance, and (3) *model validation* via comparison of model output with clinician interpretation of CFPs. We also validate the model on a cohort that was not diagnosed with glaucoma at the time of imaging

but eventually received a glaucoma diagnosis. Results show our model to be highly accurate (AUC 0.97) and interpretable. The model validates biological features known to be related to the disease, such as age, intraocular pressure, and optic disc morphology. It also highlights previously unknown or disputed features, such as pulmonary capacity and retinal outer layers.

This research was presented at the annual meeting of *Association for Research in Vision and Ophthalmology* [17] and is now under submission.

1.4.3 Sampling for Deep Learning Model Diagnosis

In Chapter 4 we focus on the data management challenges facing ML practitioners who build deep learning models. Specifically, we present a new sampling-based approach for deep learning model diagnosis. When diagnosing deep learning models, ML practitioners pose queries such as, "What are the top k maximally activated neurons for layer *conv2* for all incorrectly classified objects for $model_A$?" [90, 106, 92], and "What is the similarity between the logits of $class_a$ and the representation learned by the last convolution layer by $model_A$?" [113, 100]. Such analyses require activations for the *entire* model(s) over the *entire* input data set. If the training process is being examined, or multiple models are being compared, the activations for multiple checkpoints or models must be generated and stored.

As previously noted, the naive solution – pre-computing and storing all artifacts required for model diagnosis – scales as the product of size of input data and number of parameters of the deep learning model. Generating query results on-the-fly has a response time between tens of seconds to tens of minutes, which makes it difficult to efficiently perform diagnosis tasks, often preventing interactive diagnosis. Previous attempts at solving this problem either pre-generated all data required to provide interactive query times [91, 90, 106, 169] or used varied storage optimization techniques to manage the storage footprint [189, 117]. Both approaches require pre-generated artifacts. Some visualization tools pre-generate aggregates but severely limit the type of queries that can be posed, while others simply do the latter. Systems with storage optimizations [189] reduce data storage requirements using techniques such as de-duplication and quantization.

We use sampling as a means to reduce the quantity of data we need to store, manage and

query. Common sampling techniques that rely on uniform random sampling or stratified sampling yield poor results. Instead, we propose a novel technique for creating a sample for deep learning model diagnosis. Our key insight is for the deep learning model, along with its other objectives (such as classification), to learn a *lower-dimensional representation* of the data. Deep learning training transforms the input data, creating a new representation with each layer. Therefore, to diagnose the model, *we leverage this lower-dimensional representation of data rather than storing and analyzing the distribution of activation values to create a sample.* The sampling technique that we developed specifically targets model diagnosis queries, which include top-k queries as well as average values. It provides more accurate answers than uniform sampling, stratified sampling, and other sampling techniques from the literature.

Our technique selects a sample from the input (test and training) data set so that artifacts can be generated only for the sample. This approach not only reduces the storage footprint and speeds-up queries since we store less data, but it also speeds-up the overall diagnosis process by saving the time it would otherwise take to generate all artifacts for the entire data set. In Chapter 4, we describe our sampling technique in greater detail and evaluate its performance on two data sets compared to a variety of state-of-the-art alternatives. A short version of this paper was published at ICDE'20 [142].

In summary, this thesis focuses on large-scale image analytics for scientific image data. We assess existing big data systems on their ability to support image analytics and extend Myria to support such workloads. Our work on automated glaucoma detection built a deep learning based interpretable model to detect glaucoma. Finally, we drew on our experiences when building the glaucoma detection model to propose a novel sampling technique that reduces the data management complexity associated with building and diagnosing interpretable deep learning models for large image data sets.

Chapter 2

COMPARATIVE EVALUATION OF BIG-DATA SYSTEMS ON SCIENTIFIC IMAGE ANALYTICS WORKLOADS

"Sooner or later all things are numbers, yes?"
– Terry Pratchett, Monstrous Regiment

"It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts."
– Arthur Conan Doyle, A Scandal in Bohemia

In this chapter we describe in further detail the study evaluating the suitability of large-scale data systems and frameworks for scientific data analysis using two real-world scientific image data processing use cases. We evaluate five representative systems (SciDB, Myria, Spark, Dask, and TensorFlow) and find that each of them has shortcomings that complicate implementation or hurt performance. We evaluate all systems for both performance and ease-of-use. To evaluate these systems, we implement two representative end-to-end image analytics pipelines from astronomy and neuroscience. Each pipeline has a reference implementation in Python provided by domain scientists. We then attempt to re-implement them using the five big data systems described above and deploy the resulting implementation on commodity hardware in the Amazon Web Services cloud [18], to simulate the typical hardware and software setup in scientists' labs. We evaluate the

resulting implementations with the following goals in mind:

- Investigate if the given system can be used to implement the pipelines and, if so, how easy is it to do so (Section 2.3).
- Measure the execution time of the resulting pipelines in a cluster deployment (Section 2.4.1 and Section 2.4.2).
- Evaluate the system's ability to scale, both with the number of machines available in the cluster, and the size of the input data to process (Section 2.4.1).
- Assess tuning required by each system to correctly and efficiently execute each pipeline (Section 2.4.3).

Overall, we find that all systems have important limitations. While performance and scalability results are promising, there is much room for improvement in usability and efficiently supporting image analytics at scale.

2.1 Evaluated Systems

In this section we briefly describe the five evaluated systems and their design choices pertinent to image analytics. The source code for all of these systems is publicly available.

Dask [1] (v0.13.0) is a general-purpose parallel computing library implemented entirely in Python. We select Dask because the use cases we consider are written in Python. Dask represents parallel computation with task graphs. Dask supports parallel collections such as `Dask.array` and `Dask.dataframe`. Operations on these collections create a task graph implicitly. Custom (or existing) code can be parallelized via `Dask.delayed` statements, which delay function evaluations and insert them into a task graph. Individual task(s) can be submitted to the Dask scheduler directly. Submitted tasks return `Futures`. Further tasks can be submitted on `Futures`, sending computation to the worker where the `Future` is located. The Dask library includes a scheduler that dispatches tasks to worker processes across the cluster. Processes execute these tasks asyncronously. Dask's

scheduler determines where to execute the delayed computation, and serializes the required functions and data to the chosen machine before starting its execution. Dask uses a cost-model for scheduling and work stealing among workers. The cost-model considers worker load and user specified restrictions (e.g., if a task requires a worker with GPU) and data dependencies of the task. Computed results remain on the worker where the computation took place and are brought back to the local process by calling `result()`.

Myria [70, 193] is a relational, shared-nothing DBMS developed at the University of Washington. Myria uses PostgreSQL [150] as its node-local storage subsystem and includes its own query execution layer on top of it. Users write queries in MyriaL, an imperative-declarative hybrid language, with SQL-like declarative constructs and imperative statements such as loops. Myria query plans are represented as graphs of operators and may include cycles. During execution, operators pipeline data without materializing it to disk. Myria supports Python user-defined functions and a BLOB data type. The BLOB data type allows users to write queries that directly manipulate objects like NumPy arrays. We select Myria as a representative shared nothing parallel DBMS and also a system that we built. Our goal is to understand how it compares to other systems on image analysis workloads and what it requires to effectively support such tasks.

Spark [200] (v1.6.2) supports a dataflow programming paradigm. It is up to $100\times$ faster than Hadoop for in-memory workloads and up to $10\times$ faster for workloads that persist to disk [167]. We select Spark for its widespread adoption, its support for a large variety of use cases, the availability of a Python interface, and to evaluate the suitability of the MapReduce programming paradigm for large-scale image analytics. Spark's primary abstraction is a distributed, fault-tolerant collection of elements called Resilient Distributed Datasets (RDDs) [200], which are processed in parallel. RDDs can be created from files in HDFS or by transforming existing RDDs. RDDs are partitioned and Spark executes separate tasks for different RDD partitions. RDDs support two types of operations: transformations and actions. Transformations create a new dataset from an exisiting one, e.g., `map` is a transformation that passes each element through a function and returns a new RDD representing the results. Actions return a value after running a computation: e.g., `reduce` is an action that aggregates all the elements of the RDD using a function and returns the final result. All

transformations are lazy and are executed only when an action is called. Besides `map` and `reduce`, Spark's API supports relational algebra operations as well, such as distinct, union, join, grouping, etc.

SciDB [28] (v15.12) is a shared-nothing DBMS for storing and processing multidimensional arrays. SciDB is designed specifically to support image analytics tasks such as those we evaluate in this paper, and is an obvious system to include in the evaluation. In SciDB, data is stored as arrays divided into chunks distributed across nodes in a cluster. Users then query the stored data using the Array Query Language (AQL) or Array Functional Language (AFL) through the provided Python interface. SciDB supports user-defined functions in C++ and, recently, Python (with the latter executed in a separate Python process). In SciDB, query plans are represented as an operator tree, where operators, including user-defined ones, process data iteratively one chunk at a time.

TensorFlow [4] (v0.11) is a library from Google for numerical computation using dataflow graphs. Although TensorFlow is typically used for machine learning, it supports a wide range of functionality to express operations over N-dimensional tensors. Such operations are organized into dataflow graphs, where nodes represent computations, and edges are the flow of data expressed using tensors. TensorFlow optimizes these computation graphs and can execute them locally, in a cluster, on GPUs, or on mobile devices. We select TensorFlow to evaluate its suitability for large-scale image analytics given its general popularity.

2.2 Image Analytics Use Cases

Scientific image data is available in many modalities: X-ray, MRI, ultrasound, telescope, microscope, satellite, etc. We select two scientific image analytics use cases from neuroscience and astronomy, with real-world utility in two different domains. The choice of these use cases is influenced by access to domain experts in these fields, availability of large datasets, and interest from the domain experts in scaling their use cases to larger datasets with big data systems. Reference implementations for both use cases are provided by domain experts who are also coauthors on this paper, and were written for their prior work. Both reference implementations are in Python and utilize wrapped libraries written in C/C++. We present the details of the use cases in this section.

Both use cases are subdivided into steps. We use Step Ⓧ$_{\text{N}}$ and Step Ⓧ$_{\text{A}}$ to denote steps in the neuroscience and astronomy use cases respectively.

2.2.1 Neuroscience

Many sub-fields of neuroscience use image data to make inferences about the brain [88]. The use case we focus on analyzes Magnetic Resonance Imaging (MRI) data in human brains. Specifically, we focus on measurements of diffusion MRI (dMRI), an imaging technology that is sensitive to water diffusion in different directions within a human brain. These measurements are used to estimate large-scale brain connectivity, and to relate the properties of brain connections to brain health and cognitive functions [191].

Data: The input data comes from the Human Connectome Project [187]. We use data from the S900 release, which includes dMRI data from over 900 healthy adult subjects collected between 2012 and 2015. The dataset contains dMRI measurements obtained at a spatial resolution of $1.25\times1.25\times1.25\ mm^3$. Measurements were repeated 288 times in each subject, with different diffusion sensitization in each measurement. The data from each measurement, called a *volume*, is stored in a 3D ($145\times145\times174$) array of floating point numbers, with one value per three-dimensional pixel (a.k.a. voxel). Image data is stored in the standard NIfTI-1 file format used for neuroimaging data. Additional metadata about the measurement used during analysis is stored in text files that accompany each measurement. We use data from 25 subjects for this use case. Each subject's data is $\sim$1.4GB in compressed form, which expands to $\sim$4.2GB when uncompressed. The total amount of data from 25 subjects is thus approximately **105GB**.

Processing Pipeline: The benchmark contains three steps from a typical dMRI image analysis pipeline for each subject, as shown in Figure 2.1.

1. **Segmentation**: Step ①$_{\text{N}}$ constructs a 3D mask that segments each image volume into two parts: the brain and the background. As the brain comprises around two-thirds of the image volume, using the mask to filter out the background will speed up subsequent steps. Segmentation proceeds in three sub-steps. First, we select a subset of the volumes where no diffusion

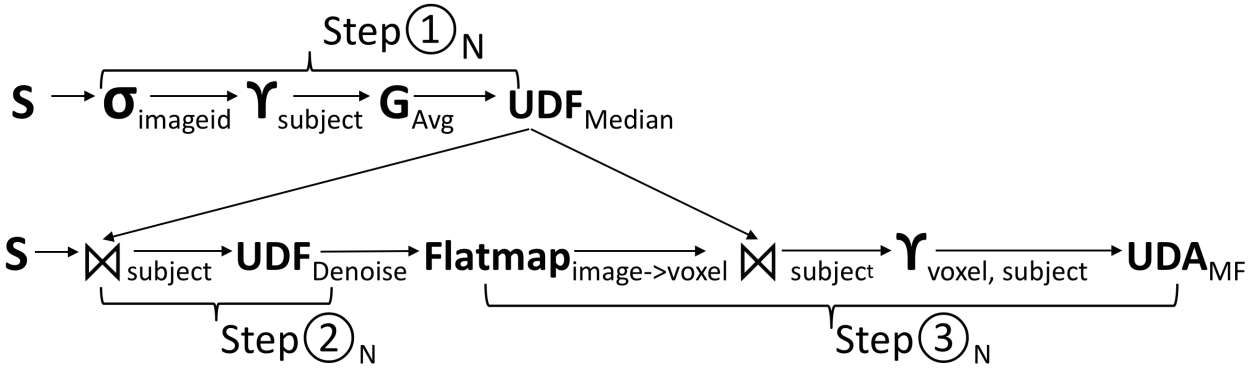


Figure 2.1: **Neuroscience use case**: Step ①$_N$ Segmentation, Step ②$_N$ Denoising, and Step ③$_N$ Model fitting.

sensitization has been applied. These images are used for segmentation as they have higher signal-to-noise ratio. Next, we compute a mean image from the selected volumes by averaging the value of each voxel. Finally, we apply the Otsu segmentation algorithm [138] to the mean volume to create a mask volume per subject.

2. **Denoising**: Denoising is needed to improve image quality and accuracy of the analysis results. This step (Step ②$_N$) can be performed on each volume independently. Denoising operates on a 3D sliding window of voxels using the non-local means algorithm [44] and uses the mask from Step ①$_N$ to denoise only parts of the image volume containing the brain.

3. **Model fitting**: Finally, Step ③$_N$ computes a physical model of diffusion. We use the diffusion tensor model (DTM) to summarize the directional diffusion profile within a voxel as a 3D Gaussian distribution [21]. Fitting the DTM is done independently for each voxel and can be parallelized. This step consists of a flatmap operation that takes a volume as input and outputs multiple voxel blocks. All 288 values for each voxel block are grouped together before fitting the DTM for each voxel. Given the 288 values in for each voxel, fitting the model requires estimating a 3×3 variance/covariance matrix. The model parameters are summarized as a scalar for each voxel called Fractional Anistropy (FA) that quantifies diffusivity differences across different directions.

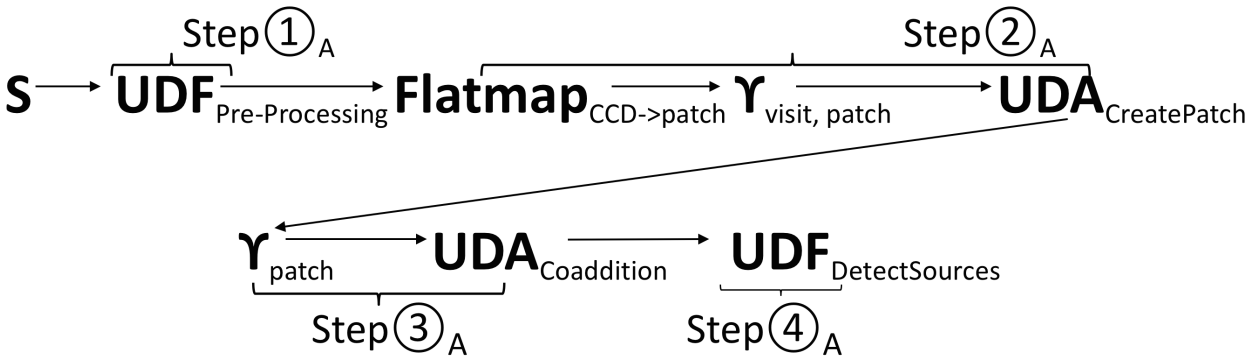


Figure 2.2: **Astronomy use case**: Step ①$_A$ Pre-Processing, Step ②$_A$ Patch Creation, Step ③$_A$ Co-addition, and Step ④$_A$ Source Detection.

The reference implementation is written in Python and Cython using Dipy [63].

2.2.2 Astronomy

As noted before, astronomy surveys are generating an increasing amount of image data. Our second use case is an abridged version of the LSST image processing pipeline [108], which includes analysis routines used by astronomers.

Data: We use data from the High Cadence Transient Survey [77] for this use case, as data from the LSST survey is not yet available. A telescope scans the sky through repeated *visits* to individual, possibly overlapping, locations. We use up to 24 visits that cover the same area of the sky in this evaluation. Each visit is divided into 60 *images*, with each consisting of an 80MB 2D image (4000×4072 pixels) with associated metadata. The total amount of data from all 24 visits is approximately **115GB**. The images are encoded using the FITS file format [57] with a header and data block. The data block has three 2D arrays, with each element containing flux, variance, and mask for every pixel.

Processing Pipeline Our benchmark contains four steps from the LSST processing pipeline as shown in Figure 2.2:

1. **Pre-Processing**: Step ①$_A$ pre-processes each image to remove background noise and ar-

tifacts caused by imaging instruments. These operations are implemented as convolutions with different kernels. This step can be parallelized across images.

2. **Patch Creation**: Step ②$_A$ re-grids the pre-processed images of the sky into regions called *patches*. Each image can be part of 1 to 6 patches requiring a flatmap operation. As pixels from multiple images can contribute to a single patch, this step groups the images associated with each patch and creates a new image object for each patch in each visit.

3. **Co-addition**: Step ③$_A$ groups the images associated with the same patch across different visits and stacks them by summing up the pixel (or flux) values. This is called co-addition and is performed to improve the signal to noise ratio of each image. Before summing up the pixel values, this step performs iterative outlier removal by computing the mean flux value for each pixel and setting any pixel that is three standard deviations away from the mean to null. Our reference implementation performs two such cleaning iterations.

4. **Source Detection**: Finally, Step ④$_A$ detects sources visible in each co-added image generated from Step ③$_A$ by estimating the background and detecting all pixel clusters with flux values above a given threshold.

The reference implementation is written in Python, and depends on several libraries implemented in C++, utilizing the LSST stack [107]. While the LSST stack can run on multiple nodes, the reference is a single node implementation.

2.3 Qualitative Evaluation

We evaluate the five big data systems along two dimensions. In this section, we discuss each system's ease of use and overall implementation complexity. We discuss performance and required physical tunings in Section 2.4. A subset of the computer science coauthors implemented each use case on each system based on the reference implementations provided by the domain expert coauthors. The team had previous experience with Myria, Spark, and SciDB but not Dask or TensorFlow.

2.3.1 Dask

Implementation: As described in Section 2.1, users specify their Dask computation using task graphs. However, unlike other big data systems with task graphs, users do not need to use a specific data abstraction (e.g., RDDs, relations, tensors, etc.). They can construct graphs directly with Python data structures. As an illustration, Figure 2.3 shows a code fragment from Step ①$_N$ of the neuroscience use case. We first construct a compute graph that downloads and filters each subject's data on Line 2. Note the use of `delayed` to construct a compute graph by postponing computation, and specifying that `downloadAndFilter` is to be called on each subject separately. At this point, only the compute graph is built but it has not been submitted, i.e., data has not been downloaded or filtered. Next, on Line 5 we request that Dask evaluate the graph via the call to `result` to compute the number of volumes in each subject's dataset. Calling `result` submits the graph and forces evaluation. We constructed the graph such that `downloadAndFilter` is called on individual subjects. Dask will parallelize the computation across the worker machines and will adjust each machine's load dynamically. We then build a new graph to compute the average image of the volumes by calling `mean`. We intend this computation to be parallelized across blocks of voxels, as indicated by the iterator construct on Line 9. Next, the individual averaged volumes are reassembled on Line 10, and calling `median_otsu` on Line 11 computes the mask. The rest of the neuroscience use case follows the same programming paradigm, and we implemented the astronomy use case similarly.

Qualitative Assessment: We find that Dask has the simplest installation and deployment. As Dask supports external Python libraries we reuse most of the reference implementation. Our Dask implementation has approximately 120 lines of code (LoC) for the neuroscience use case and 260 LoC for the astronomy one. When implementing compute graphs, however, a developer has to reason about when to insert barriers to evaluate the constructed graphs. Additionally, the developer must manually specify how data should be partitioned across different machines for each of the stages to facilitate parallelism (e.g., by image volume or blocks of voxels, as specified on Line 9 in Figure 2.3). While Dask's Python interface might be familiar to domain scientists, the explicit

```
for id in subjectIds:
  data[id].vols = delayed(downloadAndFilter)(id)

for id in subjectIds: # barrier
  data[id].numVols = len(data[id].vols.result())

for id in subjectIds:
  means = [delayed(mean)(block) for block in
           partitionVoxels(data[id].vols)]
  means = delayed(reassemble)(means)
  mask = delayed(median_otsu)(means)
```

Figure 2.3: Dask code for Step ①$_N$.

construction of compute graphs to parallelize computation is non-trivial. Additionally, as the Dask API supports multiple ways to parallelize code, choosing among them can impact the correctness and performance of the resulting implementation. For instance, having to choose between `futures` and `delayed` constructs to create a task graph implicitly and explicitly make Dask more flexibile but harder to use. Dask is also hard to debug due to its atypical behavior and instability. For instance, rather than failing a job after a large enough number of workers die, Dask respawns the killed processes but queues tasks executed on the killed worker processes to a `no-worker` queue. The `no-worker` state implies that tasks are ready to be computed, but no appropriate worker exists. As the running processes wait for the results from the requeued tasks this causes a deadlock. We also experienced several stability issues, some of which prevented us from running the astronomy use case initially, but were fixed in a later Dask version. We still had to stop and rerun the pipelines occasionally as they would freeze for unknown reasons.

2.3.2 Myria

Implementation: We use MyriaL to implement both pipelines, with calls to Python UDF/UDAs for all core image processing operations. In the neuroscience use case, we ingest the input data into the `Images` relation, with each tuple consisting of subject ID, image ID, and image volume (a

serialized NumPy array stored as a BLOB) attributes. We execute a query to compute the mask, which we broadcast across the cluster. The broadcast table is 0.3% the size of the `Images` relation. A second query then computes the rest starting from a broadcast join between the data and the mask. Figure 2.4 shows the code for denoising image volumes. Line 1 to Line 2 connect to Myria and register the denoise UDF. Line 3 then executes the query to join the `Images` relation with the `Mask` relation and denoise each image volume. We implement the astronomy use case similarly using MyriaL.

```
1 conn = MyriaConnection(url="...")
2 MyriaPythonFunction(Denoise, outType).register()
3 query = MyriaQuery.submit("""
4   T1 = SCAN(Images);
5   T2 = SCAN(Mask);
6   Joined = [SELECT T1.subjId, T1.imgId, T1.img, T2.mask
7             FROM T1, T2
8             WHERE T1.subjId = T2.subjId];
9   Denoised = [FROM Joined EMIT
10              Denoise(T1.img, T1.mask) as
11              img, T1.subjId, T1.imgId]; """ )
```

Figure 2.4: Myria code for Step ②$_{\text{N}}$.

Qualitative Assessment: Our MyriaL implementation leverages the reference Python code, facilitating the implementation of both use cases. We implement the neuroscience use case in 75 LoC and the astronomy one in 250. MyriaL's combination of declarative query constructs and imperative statements makes it easy to specify the analysis pipelines. However, for someone not familiar with SQL-like syntax this might be a challenge. Overall, implementing the use cases in Myria was easy but making the implementation efficient was non-trivial, as we discuss in Section 2.4.3.

2.3.3 Spark

Implementation: We use Spark's Python API to implement both use cases. Our implementation transforms the data into Spark's pair-RDDs, which are parallel collections of key-value pair

records. The key for each RDD is the identifier for an image fragment, and the value is a NumPy array with the image data. Our implementation then uses the predefined Spark operations (`map`, `flatmap`, and `groupby`) to split and regroup image data following the plan from Figure 2.1. To avoid joins, we make the mask, which is a single image per subject (18MB per subject, 0.3% of the image RDD) a broadcast variable available on all workers. We use the Python functions from the reference implementation to perform the actual computations on the values, passing them as anonymous functions to Spark's API. Figure 2.5 shows an abridged version of the code for the neuroscience use case. Line 2 denoises each image volume. Line 3 calls `repart` to partition each image volume into voxel groups, which are then grouped (line 4) and aggregated (line 5). Line 6 then fits a DTM for each voxel group. We implement the astronomy use case similarly.

```
1 modelsRDD = imgRDD
2   .map(lambda x:denoise(x,mask))
3   .flatMap(lambda x: repart(x, mask))
4   .groupBy(lambda x: (x[0][0],x[0][1]))
5   .map(regroup)
6   .map(fitmodel)
```

Figure 2.5: Spark code showing Step ②$_{\mathrm{N}}$ and Step ③$_{\mathrm{N}}$.

Qualitative Assessment: Spark can execute user-provided Python code as UDFs and its support for arbitrary python objects as keys made the implementation straightforward, with 114 and 238 LoC for the neuroscience and astronomy use cases respectively. The functional programming style used by Spark is succinct, but can pose a challenge if the developer is unfamiliar with functional programming. Spark's large community of developers and users, and its extensive documentation are a considerable advantage. Spark supports caching data in memory but does not store intermediate results by default. Unless data is specifically cached, a branch in the computation graph may result in re-computation of intermediate data. In our use cases, opportunities for data reuse are limited. Nevertheless, we found that caching the input data for the neuroscience use case yielded a consistent 7-8% runtime improvement across input data sizes. Caching did not improve the run-

time of the astronomy use case. As with Myria, the initial implementation was easy in Spark, but performance tuning was not always straightforward as we describe in Section 2.4.3.

```
from scidbpy import connect
sdb = connect('...')
data_sdb = sdb.from_array(data)
data_filtered = # Filter
  data_sdb.compress(sdb.from_array(gtab.b0s_mask), axis=3)
mean_b0_sdb = data_filtered.mean(index=3) # Mean
```

Figure 2.6: SciDB implementation of Step ①$_N$.

2.3.4 SciDB

Implementation: SciDB is designed for array analytics to be implemented in AQL/AFL, or optionally using SciDB's Python API. This is illustrated in Figure 2.6. When we began this project, SciDB lacked several functions necessary for the use cases. For example, high-dimensional convolutions are not implemented in AFQ/AFL, rendering the reimplementation of some steps impossible. SciDB recently added a `Stream()` interface, similar to Hadoop Streaming. This interface enables the execution of Python (or other language) UDFs, where each UDF call processes one array chunk. Using this feature, we implemented both use cases in their entirety. The SciDB implementation of the neuroscience and astronomy use cases required 155 LoC and 715 LoC of Python respectively. In addition, 200 LoC of AFL were written to rearrange chunks, and another 150 LoC in bash to set up the environment required by the application.

Qualitative Assessment: The recent `stream()` interface makes it possible to execute legacy Python code, but it requires that data be passed to the Python UDF and returned from that UDF as a string in TSV format (data ingest requires a CSV format). This means that we need to convert between TSV and scientific image formats. While this is relatively straightforward for the neuroscience use case, FITS files used in astronomy have multiple arrays per image and are more

challenging to transform between SciDB arrays and TSVs for UDFs. This transformation and subsequent parsing also causes performance issues as we discuss in Section 2.4.3. Setting up SciDB is also difficult as there is no support for standard cloud deployment tools. The integration with the LSST software stack for the astronomy use case is particularly challenging. Specifically, LSST's software stack requires dozens of dependent packages to be installed, along with setting up more than 100 environment variables within child processes that execute the UDFs. Another limitation of the `stream()` interface lies in its input and output ports: All the input can be read only through the standard input (i.e., stdin) and all the output can only be written to standard output (i.e., stdout). Therefore, the application crashes if it calls UDFs that have their own stdout messages. Overall, SciDB presents significant barriers in implementation and setup for a domain scientist.

2.3.5 TensorFlow

Implementation: TensorFlow's API operates on its own data structures (representations of multi-dimensional arrays, or tensors) and does not allow any of the standard external libraries to be used, such as NumPy. Hence, we had to completely rewrite the use cases. Given their complexity, we only implemented the neuroscience use case. Additionally, we implemented a somewhat simplified version of the final mask generation operation in Step ①$_N$. We rewrote Step ②$_N$ using convolutions, but without filtering with the mask as TensorFlow does not support element-wise data assignment. TensorFlow's support for distributed computation is currently limited. The developer must manually map computation and data to each worker as TensorFlow does not provide automatic static or dynamic work assignment. As the serialized compute graph cannot be larger than 2GB, we implemented the use case in steps, building a new compute graph for each step of the use case (as shown in Figure 2.1). We distribute the data for each step to the workers and execute it. We add a global barrier to wait for all workers to return their results before proceeding. The master node converts between NumPy arrays and tensors as needed. Figure 2.7 shows the code for mean computation in Step ①$_N$. The first loop assigns the input data with shape `sh` (Line 8) and the associated code (Line 9) to each worker. Then, we process the data in batches of size equal to the number of available workers on Line 19.

```
# steps contains the predefined mapping
# from data to workernodes
pl_inputs = []
work = []
# The first for loop defines the graph
for i_worker in range(len(steps[0])):
  with tf.device(steps[0][i_worker]):
    pl_inputs.append(tf.placeholder(shape=sh))
    work.append(tf.reduce_mean(pl_inputs[-1]))
mean_data = []
# Iterate over the predefined steps
for i in range(len(steps)):
  this_zs = zs[i*len(steps[0]):
               i*len(steps[0]) + len(steps[i])]
  # Define the input to be fed into the graph
  zip_d = zip(pl_inputs[:len(steps[i])],
              [part_arrs[z] for z in this_zs])
  # Executes the actual computation - incl. barrier
  mean_data += run(work[:len(steps[i])], zip_d)
```

Figure 2.7: TensorFlow code fragment showing compute graph construction and execution.

Qualitative Assessment: TensorFlow requires a complete rewrite of the use cases, which we find to be both difficult and time-consuming. TensorFlow also requires that users manually specify data distribution across machines and it automates only a small part of the cluster initialization procedure with Bazel [23]. The 2GB limit on graph size further complicates the implementation of use cases as we described above. In its current form, unless there is a specific reason, e.g., a specific algorithm only available in the TensorFlow library, or need to utilize GPUs or Android devices, the amount of effort required to re-write the computation in Tensorflow is high enough to render it not worthwhile. However, TensorFlow is under active development so this might change in future versions.

2.4 Quantitative Evaluation

We evaluate the performance of the implemented use cases and the system tuning necessary for successful and efficient execution. All experiments are performed on the AWS cloud using the r3.2xlarge instance type, with 8 vCPU,[1] 61GB of memory, and 160GB SSD storage.

2.4.1 End-to-End Performance

We first evaluate the performance of running the two use cases end-to-end. For each use case, we start the execution with data stored in Amazon S3. We execute all the steps and leave the final output in worker memories. We ask two questions: How does the performance compare across the systems? How well do the systems scale as we increase the size of the input data or the cluster? To answer these questions, we first fix the cluster size at 16 nodes and vary the input data size. For the neuroscience use case we vary the number of subjects from 1 to 25. For the astronomy use case, we vary the number of visits from 2 to 24. The largest input data sizes are then 105GB and 115GB, respectively as shown in Figure 2.8a and Figure 2.8b. The tables also show the sizes of the largest intermediate relations for the two use cases, which are 210GB and 288GB, respectively. In the second experiment, we use the largest input data size for each use case and vary the cluster size from 16 to 64 nodes to measure speedup.

Figure 2.8c and Figure 2.8f show the results as we vary the input data size. We implement the neuroscience use case in all five systems and the astronomy use case in all but TensorFlow.

For the neuroscience use case (Figure 2.8c), while Dask, Myria and Spark achieve comparable performance, SciDB and TensorFlow are much slower. It is not surprising that Dask, Spark, and Myria have similar runtimes as all three execute the same Python code from the reference implementation but wrapped in UDFs (or directly in the case of Dask). Interestingly, the fact that Spark and Myria incur the extra overhead of passing the data to and from external Python processes for each UDF invocation does not visibly affect performance. Dask is a little slower in the case of a single subject because the data is downloaded on a single node first before being spread across

[1] with Intel Xeon E5-2670 v2 (Ivy Bridge) processors.

the cluster through work stealing as discussed in Section 2.4.2. SciDB is slower primarily due to data reformatting at the input and output of each UDF and also at the beginning of the pipeline: (1) SciDB's data ingest requires the source data to be in CSV format, and in both use cases, we needed to convert the original formats to CSV before executing the pipeline. We discuss data ingest in more detail when we present Figure 2.9. (2) When passing data from SciDB to the UDFs through the `stream()` interface, the data gets flattened into a long 1-dimensional array and we need to reformat it into the multidimensional NumPy array required by the function. (3) The results returned from each UDF are arrays of strings, one per parallel UDF invocation and need to be parsed and merged to form voxel batches from image volumes. Additionally, we process one subject at a time in SciDB instead of merging data from all subjects into a single array as combining multiple subjects incurs more overhead. For example, when we combine two subjects into one array, the execution time doubled (not shown in the figure), relative to processing one subject at a time. TensorFlow is the slowest to execute the neuroscience end-to-end pipeline. We attribute this to several architectural restrictions. (1) All data needs to be downloaded and parsed on the master node and then dispatched to the workers, as opposed to being downloaded and processed directly by the workers as in the other systems. (2) Due to TensorFlow's limitation on the computation graph size (2GB), we construct multiple graphs for each step and put the results together on the master node. This adds significant overhead at each step and prevents TensorFlow from pipelining the computation. We further analyze per-step performance in the next section.

For the astronomy use case (Figure 2.8f), SciDB is an order of magnitude slower than the other engines. In Step $③_A$, we need to add patches from all visits. This means that we have to process all visits as a single array, leading to large overheads due to parsing and combining the output of the UDF in Step $②_A$. We also implemented Step $③_A$ in AQL in SciDB, which performed as fast as Spark, Myria and Dask, i.e., $\sim 10\times$ faster than the `stream()` implementation. As we were unable to implement the rest of the use case in AQL/AFL, we use the `stream()` timing in Figure 2.8.

In the astronomy use case, Dask does better with the smaller datasets but is unable to execute the largest data set to completion as it runs out of memory on 16 nodes. We discuss out of memory issues in Section 2.4.3. In terms of runtime, data for the astronomy use case is packaged into indi-

vidual files for each image rather than a single compressed file per subject as in the neuroscience use case, which results in finer-grained and more even data distribution.

Figure 2.8e and Figure 2.8h show the runtimes per subject and visit, respectively, i.e., the ratios of each pipeline runtime to that obtained for one subject or visit. As the data size increases, these ratios drop for all systems with the exception of SciDB in the neuroscience use case. The dropping ratios indicate that the systems become more efficient as they amortize start-up costs. Dask's efficiency increase is most pronounced, indicating that it had the largest start-up overhead. In the neuroscience use case, SciDB shows a constant ratio. This is due to the neuroscience use case implementation in SciDB where the cluster processes a single subject at a time.

Figure 2.8d and Figure 2.8g show the runtimes for all systems as we increase the cluster size and process the largest (>100GB) datasets. All systems show near linear speedup for both use cases. Myria achieves almost perfect linear speedup for both cases (5395s, 2863s, and 1406s for 16, 32 and 64 nodes, respectively). Linear speedup can imply poor single instance performance. To ensure that this is not the case, we tune single instance performance for Myria by increasing the number of workers until each node has high resource utilization (average CPU utilization $> 90\%$). Dask has better performance when data completely fits in memory, as in the neuroscience use case. For the astronomy use case, Dask runs out of memory on 16 nodes. For 32 nodes, the number of workers had to be reduced to one per node to allow Dask to finish processing. This results in lowered parallelism and higher runtime. In the 64 node experiment there was enough memory and every step could be pipelined for both Myria and Dask, resulting in faster runtimes. Spark divides each task into stages and does not start shuffling data until the previous map stage is complete. As the tuples are large in size, ($\sim$14MB for neuroscience and $\sim$80 MB for astronomy), the lack of overlap between shuffle and map stages results in Spark's slightly slower timings. This is a known limitation for Spark [163], and makes it slower than Dask and Myria for some of the steps in the context of image analytics as the shuffle cost for the larger image tuples is significant. Note that we tuned each of the systems to achieve the timings reported above. We discuss the impact of tuning in Section 2.4.3.

2.4.2 Individual Step Performance

Next we focus on individual steps and examine performance across systems. We provide these details for the neuroscience pipeline.

Data Ingest: Input data for both use cases is staged in Amazon S3. For the neuroscience use case, image data is presented as a single NIfTI file per subject which contains compressed image volumes. For the neuroscience use case, the reference implementation works on all of the image volumes associated with the subject concurrently. To parallelize the implementation within each subject, we split data for each subject into separate image volumes represented by NumPy arrays, which can be processed in parallel. Therefore, NIfTI files need to be preprocessed for Spark and Myria, de-compressed and saved as serialized NumPy Arrays. SciDB requires NIfTI files to be converted into CSV format. TensorFlow requires images to be in NumPy array format for conversion to Tensors. Pre-processing times are included in data ingest times, which are shown in Figure 2.9. As the figure shows, data ingest times vary greatly across systems (note the log scale on the Y-axis). Spark and Myria download pre-processed data in parallel on each of the workers from S3. Myria is given a CSV list of images in S3 as part of the load statement, and Spark is given the S3 bucket name. Even though Myria's data ingest writes files to disk, it is faster than Spark, which loads the data into memory. This is because Spark enumerates the files in the S3 bucket on master before downloading them in parallel and meta-data querying in S3 is known to be a slow operation. For Dask, we manually specify the number of subjects to download per node, as otherwise Dask's scheduler assigns a random number of subjects to each node which lead to memory exhaustion or excessive data shuffle between processing steps. Thus, Dask's data ingest time looks like a step function: when the number of subjects is fewer than the number of nodes (16), each node downloads one subject concurrently. With more than 16 subjects, some nodes download two subjects. For the TensorFlow implementation, all data is downloaded to the master node and partitions are sent to each worker node. This is slower than the parallel ingest available in other systems. For SciDB, we report two sets of timings in Figure 2.9. SciDB-1 shows the time to ingest NumPy arrays with the `from_array()` interface, and SciDB-2 shows the time to convert

NIfTI to CSV and ingest using the `aio_input` library. Because `aio_input()` reads in multiple files and parses them in parallel while SciDB's native Python API (i.e., scidb-py) processes input data in a serial manner, data ingest with the latter is an order of magnitude faster than the former and is on par with parallel ingest on Spark and Myria. Nevertheless, the NIfTI-to-CSV conversion overhead for SciDB is larger than the NIfTI-to-NumPy overhead for Spark and Myria, which makes SciDB ingest slower than Spark and Myria.

Segmentation (filtering): Segmentation is the first step in the neuroscience use case (i.e., Step $①_N$). We discuss the performance of two operations in this step: filtering the data to select a subset of the image volumes, and computing an average image volume for each subject. Figure 2.11a and Figure 2.11b show the runtimes for these two operations as we vary the input data size on the 16-node cluster. Myria and Dask are the fastest on the data filtering step. Myria pushes the selection to PostgreSQL, which efficiently scans the data and returns the matching records (without indices) on the `Images` relation. Dask is fast on this operation as all data is in memory and the operation is a simple filter. Spark is an order of magnitude slower than Dask, even though data is in memory for both systems. This is because the filter criteria is specified as an anonymous function in Spark, and data and function have to be passed from Java to the external Python process and back. SciDB is slower than other systems because the internal chunks are not aligned with the selection predicate. In addition to scanning chunks, SciDB must also extract subsets of these chunks and construct new chunks in the resulting arrays. In TensorFlow, the data (tensors) takes the form of 4D arrays. For each subject, the 4D array represents the 288 3D data volumes. The selection is on the volume ID, which is the fourth dimension of the input data. However, TensorFlow only supports filtering along the first dimension. We thus need to flatten the 4D array, apply the selection, and reshape the array back into a 4D structure. As reshaping is expensive compared with filtering, TensorFlow is orders of magnitude slower than the other engines on this step.

Segmentation (mean): Figure 2.11b shows the result for the mean image volume computations. SciDB is the fastest for mean computation on the small datasets as it is designed to process arrays in parallel at the granularity of chunks. In contrast, Myria and Spark group data by subject, which leads to low cluster utilization for small numbers of subjects. The three systems have similar

performance at larger scales. Dask is slower than the other three engines, especially for small datasets, due to startup and work stealing overheads. TensorFlow is very slow as the mean has to be computed in seperate graphs due to graph size limitations with data being sent to the master after each graph computation.

Denoising: Figure 2.11c shows the runtime for denoising (Step $\textcircled{2}_{\mathrm{N}}$). For this step, the bulk of the processing happens in the user-defined denoising function. Dask, Myria, Spark, and SciDB all run the same code from the reference implementation on similarly partitioned data, which leads to similar overall performance. As in the case of the end-to-end pipeline, Dask's higher start-up overhead results in slightly worse performance for smaller data sizes, but similar performance for larger datasets. SciDB's `stream()` interface performs slightly worse than Myria, Spark, and Dask. For every call to the UDF we have to convert between string streams and the UDF's expected formats, this known overhead for SciDB is small for this step as more time is spent in computation compared to other steps. The TensorFlow implementation cannot use the binary mask to reduce the computation for each data volume. This is because TensorFlow's operations can only be applied to whole tensors and cannot be masked. This limitation and compute graph-size limitations contribute to slower performance of TensorFlow.

Model fitting: Figure 2.11d shows the runtime for the final step in the neuroscience pipeline. There are two important difference between denoising and model fitting. First, model fitting is less computationally intensive compared to denoising, Second, it offers a larger degree of data parallelism. For denoising, parallelism is limited to an image volume per subject (288 image volumes per subject). This is because all of the pixels from a single image volume are required to denoise an image volume. For model fitting, each voxel in each image can be processed independently, leading to $145 \times 175 \times 145$ potential data partitions per subject that can be processes in parallel. We implement model fitting on voxel batches rather than individual voxels to balance the cost of scheduling and benefits of parallelism. We use similar voxel batch sizes for all systems to ensure fair comparison ($\sim$500 partitions per subject). A higher degree of parallelism (more partitions) and smaller partition sizes (leading to smaller data shuffling times) for this step reduce Myria's pipelining advantage and make Spark faster. Dask is slower in this step as larger numbers of paral-

lel computations lead to aggressive shuffling and job stealing, which dominate the processing time. We suspect that this is due to Dask's inaccurate estimate of the amount of data that needs to be shuffled among workers. This results in almost constant time for model fitting as the number of subjects increases. SciDB is the slowest. As the `stream()` interface does not support UDAs (only UDFs), the splitting and aggregation of the denoised images into voxel batches has to be done in AFL, which necessiates parsing the output from the previous step (i.e., denoising) from a stream of strings into SciDB arrays with the correct chunking schema, aggregating, and splitting into voxel batches. Finally, TensorFlow shows good performance on this step. Unlike denoising, which processe entire image volumes, model fitting executes on voxel batches, which can be filtered before the computation is performed. This and TensorFlow's efficient linear algebra implementation lead to a faster performance on this step compared to the other steps.

2.4.3 System Tuning

Finally, we evaluate the five systems on the complexity of the tuning required to achieve high performance.

Degree of Parallelism: This key parameter depends on three factors: (1) the number of machines in the cluster; (2) the number of workers that execute on each machine; and (3) the size of the data partitions that can be allocated to workers. We evaluated the impact of the cluster size in Section 2.4.1. Here, we evaluate the impact of (2) and (3).

For Myria, given a 16-node cluster, the degree of parallelism is entirely determined by the number of workers per node. As in traditional parallel DBMSs, Myria hash-partitions data across all workers by default. Figure 2.12 shows runtimes for different numbers of workers for the neuroscience use case. A larger number of workers yields a higher degree of parallelism but workers also compete for physical resources. For both use cases (astronomy not shown due to space constraints), four workers per node yields the best results. Changing the number of workers requires reshuffling or reingesting the data, which makes tuning this setting tedious and time-consuming.

Spark creates data partitions, which correspond to *tasks*, and each worker can execute as many tasks in parallel as available cores. The number of workers per node thus does not impact the degree

of parallelism, as long as the number of cores remains the same. The number of data partitions determines the number of tasks that can be scheduled, and the number of cores can be restricted in Spark to increase the memory available per core on each node. As Spark did not run out of memory for either use case we choose to utilize all the cores for the Spark implementation. We further discuss memory management later in this section.

Figure 2.13 shows the query runtimes for different numbers of input data partitions on Spark. On a 16-node cluster, the decrease in runtime was dramatic between 1 and 16 partitions, as an increasingly large fraction of the cluster became utilized. The runtime continues to improve until 128 data partitions, which is the total number of slots in the cluster (i.e., 16 nodes $\times$ 8 cores). Increasing the number of partitions from 16 to 97 resulted in 50% improvement. Further increases did not improve performance. Interestingly, if the number of data partitions is unspecified, Spark creates a partition for each HDFS block. The degree of parallelism then depends on the HDFS block size setting.

Dask allows specifying the number of workers per node and threads per worker. Through manual tuning, we find 8 to be the optimal number of workers per node when memory is abundant, as shown in Figure 2.14. Running multiple threads per process cause data corruption as the UDFs in Python are not all thread safe, so we use a single thread per process. Dask's work stealing automatically load balances across the machines, and work-stealing does not require any tuning.

In TensorFlow, we execute one process per machine. All steps include data conversions, which have to be performed on the master. These data conversions are the bottleneck, limiting opportunities for additional tuning. For all steps, we must partition the data such that graphs assigned to workers are below 2GB in size. Additionally, for the denoising step memory is the bottleneck requiring the assignment of only one image volume at a time per physical machine. The model fitting step can be parallelized at the granularity of individual voxels and we find that TensorFlow performs best with 128 partitions on a 16-worker configuration. In general, we observe that TensorFlow tends to perform better with smaller chunk sizes.

The SciDB documentation recommends [157] running one instance per 1-2 CPU cores. The chunk size, however, is more difficult to tune: we do not find a strong correlation between the

overall performance and common system configurations. We tune the chunk size for each operation by running the application with the same data but using different chunk sizes. For instance, in Step ③$_{\text{A}}$ we find that a chunk size of $[1000 \times 1000]$ leads to the best performance. A chunk size of $[500 \times 500]$, for example, is $3\times$ slower; while chunk sizes of $[1500 \times 1500]$ and $[2000 \times 2000]$ are 22% and 55% slower, respectively.

Memory Management: Image analytics workloads are memory intensive. Additionally, data allocation can be skewed across compute nodes. For example, the astronomy pipeline grows the data by $2.5\times$ on average during processing, but some workers experience data growth of $6\times$ due to skew. As a result, systems can easily run out of memory. Big data systems use different approaches to trade off query execution time with memory consumption. We evaluate some of these trade-offs in this section. In particular, we compare the performance of pipelined and materialized query execution. With pipelined execution, data flows directly from one operator to the next without going to disk and without synchronization barrier. Materialized query execution in contrast writes the output of each operation to disk and the next operation starts by reading input data from disk. For materialized query execution, we compare materializing intermediate results and processing subsets of the input data at a time. Figure 2.10 shows the results for the Myria system on the astronomy use case. As the figure shows, when data is small compared with the available memory, the fastest way to execute the query is to pipeline intermediate results from one operator to the next. This approach is 8-11% faster in this experiment than materialization, and 15-23% faster than executing multiple queries. As the ratio of data-to-memory grows, pipelining may cause a query to fail with an out-of-memory error. Intermediate result materialization then becomes the fastest query execution method. With even more data, the system must cut the data analysis into even smaller pieces for execution without failure.

In contrast to Myria, Spark automatically spills intermediate results to disk to avoid out-of-memory failures. Additionally, Spark can partition data into smaller fragments than the number of available nodes, and will process only as many fragments as there are available slots. Both techniques help to avoid out-of-memory failures with Spark. However, the lack of pipelined execution causes Spark to be slower than Myria when memory is plentiful (see also Figure 2.8g).

Dask supports spilling to disk, but the current implementation of this feature is not multi-process safe and thus not suitable for our use cases. To prevent running out of memory in Dask, we increased the memory-to-CPU ratio by reducing the number of workers on each node. For the 32-node cluster, we reduced the number of workers to one worker per node in the astronomy use case. On a 16-node cluster reducing the number of workers to one did not help and the use case could not run to completion. This was due to skew rather than insufficient memory: 2 workers in the astronomy use case ran out of memory and caused cascading failures.

2.5 Challenges and Opportunities

In this section, we summarize the lessons learned from three perspectives: system developers, users, and researchers:

Developers. Engine developers can improve both the architecture and implementation of their systems based on our observations, some of which are already known, but their importance is re-emphasized by this study. Most importantly, we find that all evaluated systems would benefit from automatically adjusting the degree of parallelism and gracefully spilling to disk, even when individual data partitions do not fit in memory to avoid all sources of out-of-memory failures.

Image analytics differs from other analytics in three respects: the large size of individual records (i.e., image fragments with metadata), heavy use of UDFs to execute complex, domain-specific operations, and the multidimensional nature of the data. Some systems, such as SciDB, have only limited support for UDFs/UDAs in languages other than C++. As we showed in Section 2.4, this significantly affects performance and ease of use. In contrast, only SciDB reasons about multidimensional array data. In all other systems, users must manually split images into fragments along different dimensions in preparation for their analysis, which is non-trivial.

Finally, most systems are optimized for large numbers of small records rather than small numbers of large records. Myria, for example, processes tuples in large batches by default. We had to change that default to reduce the number of tuples per batch and prevent out of memory failures.

We next discuss specific recommendations for each of the evaluated systems for running image analytical workloads.

Dask would further benefit from (1) a simpler API: e.g., reduce the number of ways to construct the compute graph (2) better debuggability as noted in Section 2.3.1; and (3) spilling to disk for multi-process workloads as noted in Section 2.4.3.

Myria would benefit from (1) automatically tuning the number of workers per machine and making it easier to change the number of workers as noted in Section 2.4.3; (2) adding support for local combiners before shuffles for user-defined aggregations: this would lead to fewer memory issues in case of skew. We were able to materialize intermediate results and split queries into multiple ones to achieve the same result, but it required better understanding of Myria and more effort.

Spark would benefit from (1) overlapping the shuffle phase with the map phase to increase performance when memory is sufficient and (2) making parallel data ingest from S3 more efficient.

SciDB would benefit from (1) binary data format support for the `aio_input()` interface; (2) support for more than TSV and stdin-stdout for the `stream()` interface; (3) more efficient methods for concatenating arrays; (4) support for advanced control over the child process such as setting environment variables; (5) simplified procedure for multi-node deployment; and (6) support for UDAs in the stream interface.

TensorFlow would benefit from (1) removing the restriction on graph size; (2) better tooling for cluster management and scheduling; (3) distributed data ingest; and (4) support for external libraries.

Users. For domain scientists wanting to utilize big data systems there are several considerations: (1) Re-write or re-use: can the computation be expressed in native SQL or AQL? If the computation is simple this may be the most performant solution. If not, systems such as Dask, Spark, and Myria can efficiently execute legacy Python (or other) scripts with minimal additional code provided by the user. (2) Data partitioning: turning a serial computation into a parallel one may pose the biggest challenge to domain users. A reference implementation may or may not indicate how computation can be parallelized. Understanding data dependency and the synchronization points in underlying computation is crucial to ensuing correctness and performance in a big data system.

Researchers. Our study raises a number of research questions. Image processing involves complex analytics, which include iterations and linear algebra operations that must be efficiently supported

in big data systems. However, users typically have legacy code that performs sophisticated and difficult to rewrite operations. Therefore, they need the ability to call existing libraries. They also need an easy mechanism to parallelize the computation: they should be able to reason about multi-dimensional array data directly rather than manually creating and processing collections of image fragments. It should be easy to mix and match UDF/UDA computations and pre-defined (e.g., relational) operations on complex data such as image fragments.

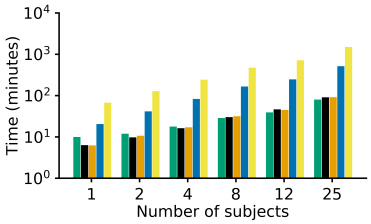
Our study also re-iterates the general need to efficiently support pipelines with UDF/UDAs both during query execution and query optimization. Image analytics implies large tuples and larger tuples put pressure on memory management techniques, systems' ability to shuffle data efficiently, and efficient methods to pass large records back and forth between core computation and UDFs/UDAs. This provides another research opportunity. Finally, making big data systems usable for scientists requires systems to be self tuning, which is already an active research area [76].

Subjects	1	2	4	8	12	25
Input	4.1	8.4	16.8	33.6	50.4	105
Largest inter.	8.4	16.8	33.6	67.2	100.8	210

(a) Neuroscience data sizes (GB)

Visits	2	4	8	12	24
Input	9.6	19.2	38.4	57.6	115.2
Largest inter.	24	48	96	144	288

(b) Astronomy data sizes (GB)

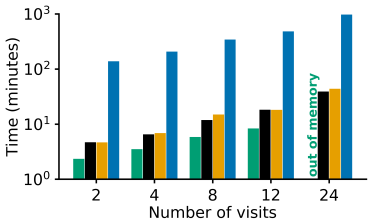


(c) Neuroscience: End-to-end runtime with varying data size

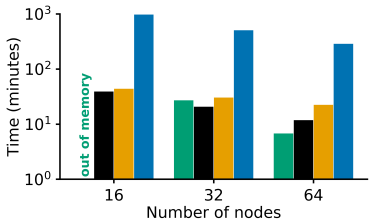
(d) Neuroscience: End-to-end runtime with varying cluster size

Subjects	1	2	4	8	12	25
SciDB	1	1.01	1.01	1.01	1.00	1.00
Spark	1	0.86	0.68	0.63	0.60	0.59
Myria	1	0.77	0.64	0.60	0.61	0.58
Dask	1	0.60	0.45	0.36	0.33	0.32
TensorFlow	1	0.95	0.90	0.88	0.89	0.89

(e) Neuroscience: Normalized runtime per subject



(f) Astronomy: End-to-end runtime with varying data size



(g) Astronomy: End-to-end runtime with varying cluster size

Visits	2	4	8	12	24
SciDB	1	0.74	0.62	0.58	0.58
Spark	1	0.74	0.80	0.65	0.78
Myria	1	0.70	0.64	0.65	0.69
Dask	1	0.75	0.62	0.59	–

(h) Astronomy: Normalized runtime per visit

Figure 2.8: **Overall performance**: results for end-to-end experiments for Neuroscience and Astronomy use cases.

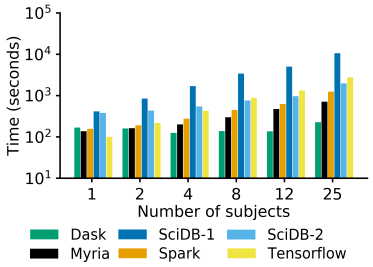


Figure 2.9: Data Ingest

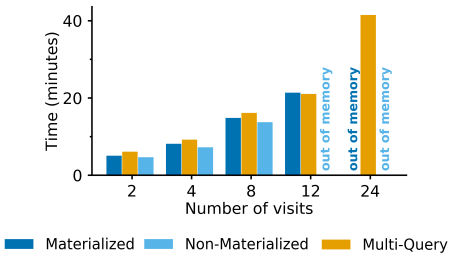


Figure 2.10: Variance in execution time, astronomy use case, Myria.

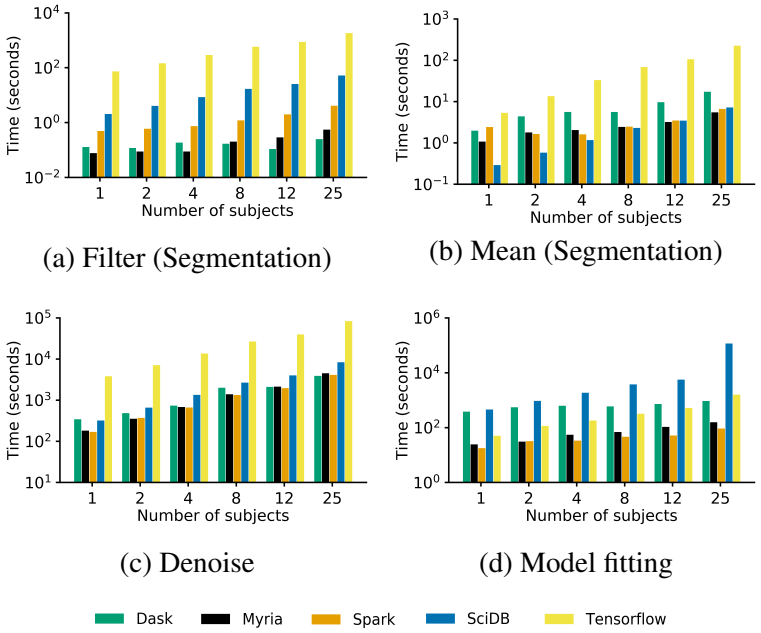


Figure 2.11: Individual step performance for Neuroscience use case (log scale on the y-axis). Experiments run on 16 nodes.

Figure 2.12: 25 subjects, 16 nodes.

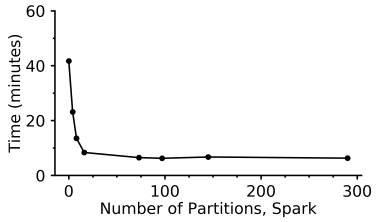


Figure 2.13: Single subject, 16 nodes.

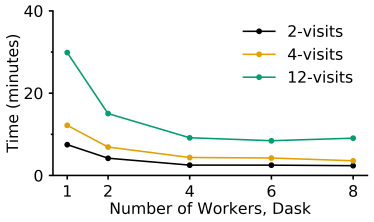


Figure 2.14: Multiple visits, 16 nodes.

Chapter 3

AUTOMATED DETECTION OF GLAUCOMA WITH INTERPRETABLE MACHINE LEARNING USING CLINICAL DATA AND MULTI-MODAL RETINAL IMAGES

"Any sufficiently advanced technology is indistinguishable from magic."
– Arthur C Clarke

"In the history of science and technology, the engineering artifacts have almost always preceded the theoretical understanding: the lens and the telescope preceded optics theory, the steam engine preceded thermodynamics, the airplane preceded flight aerodynamics, radio and data communication preceded information theory, the computer preceded computer science."
–Yann LeCun

Early detection and treatment of glaucoma is essential for minimizing risk of progressive vision loss and yet a number of challenges exist that prevent timely and accurate diagnosis. First, considerable expertise is required to perform the appropriate clinical exam and to interpret a number of specialized tests, such as visual field testing and retina and optic nerve imaging. The demand for this expertise is outpacing the supply of experts available to interpret tests and make diagnoses [59]. Second, glaucoma is often asymptomatic until the advanced stages of the disease. In the United States, approximately 50% of the estimated 3 million people with glaucoma are undiagnosed and in other parts of the world, estimates are as high as 90% [2, 162, 60, 48, 172]. New diagnostic

tools that improve the diagnostic efficiency of the existing clinical workforce are therefore vital for enabling earlier detection of the disease to facilitate early intervention [74].

In this chapter, we built a new, multi-modal, feature-agnostic model that includes clinical data, CFPs and macular OCT B-scans. Data for our model came from the UK Biobank, a multi-year, large-scale effort to gather medical information and data, with the goal of characterizing the environmental and genetic factors that influence health and disease [184]. About 65,000 UK Biobank participants underwent ophthalmological imaging procedures, which provided both macular OCT and CFP data that we matched with clinical diagnoses and with many other demographic, systemic and ocular variables. Specifically cardiovascular and pulmonary variables were chosen as markers of overall health. We used raw macular OCT and CFP data and did not rely on features extracted from these images. The use of machine learning, and particularly deep learning (DL), methods to analyze biomedical data has come under increased scrutiny because these methods can be difficult to interpret and interrogate [195, 20]; therefore, we applied machine learning interpretability methods to demystify and explain specific data features that led to accurate model performance [125]. Finally, we validated our model by comparing it to expert clinicians' interpretation of CFPs to provide an additional benchmark for the performance of our machine learning model relative to current clinical practice.

3.1 Results

We built multiple models using clinical data to establish a baseline for glaucoma detection. Glaucoma is related to many biological features, the most important of which is age [65]. Thus, we built our first baseline model (BM1) on basic demographic characteristics of the patient and control populations. BM1 included age, gender and ethnicity. Using these features, a boosted gradient tree based model predicted an occurrence of glaucoma well above chance (area under the ROC: 0.81, 95% CI 0.71- 0.90). In addition, we created two other models. The systemic data model (BM2) added cardiovascular and pulmonary variables – including Body Mass Index (BMI), Forced Vital Capacity (FVC), Peak Expiratory Flow (PEF), heart rate, diastolic and systolic blood pressure, and the presence of diabetes – to the demographic variables from BM1. We also included transient

factors, such as recent caffeine and nicotine intake, to account for any transient impact on blood pressure and heart rate. BM2 outperformed BM1 (0.87 AUC, 95% CI: 0.79-0.96). In the third model (BM3), we added ocular data to BM2, including IOP, corneal hysteresis, and corneal resistance factor. BM3 outperformed both of the other baseline models, with a test set AUC of 0.92, 95% CI: 0.87 - 0.96; see Figure 3.1 A and D.

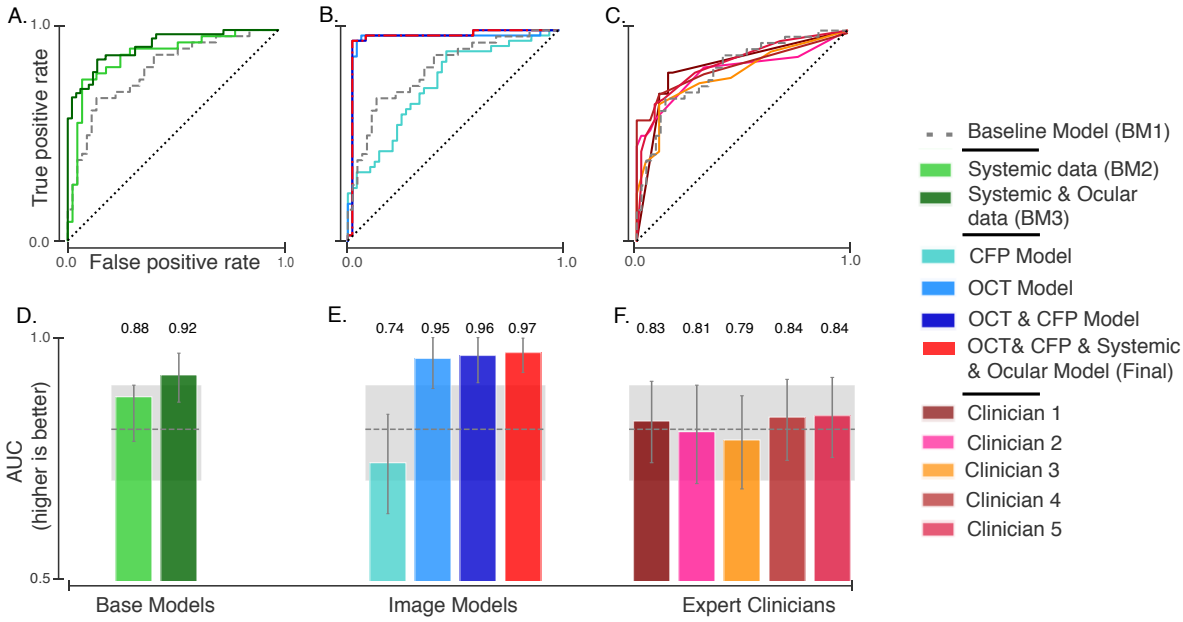


Figure 3.1: **Results of glaucoma detection models**. Receiver operating characteristic (ROC) curves are shown for (A) baseline models built with systemic and ocular data, (B) retinal imaging and final models, and (C) glaucoma expert ratings based on interpretation of CFPs. The corresponding area under the ROC curves (AUC) with (+/- 95% Confidence Interval) for models (D, E) and for clinician scores (F). The gray dashed line and shaded area denote the AUC and 95% CI for a base model (BM1) built on demographics (age, gender and ethnicity).

We used SHapley Additive exPlanations (SHAP) [110] to analyze the features that provide high predictive power in BM3. SHAP allocates optimal credit with local explanations using the classic Shapley values from game theory [155] and provides a quantitative estimate of the contribution of different features to the predictive power of a model. A higher absolute SHAP value indicates greater feature impact on the model prediction and greater feature importance. The five features

with the highest mean absolute SHAP values for BM3 were age, IOP, BMI, FVC and PEF. Figures 3.2 and 3.3 show the most important features in BM3, as evaluated through SHAP, and the interaction effect among the top features.

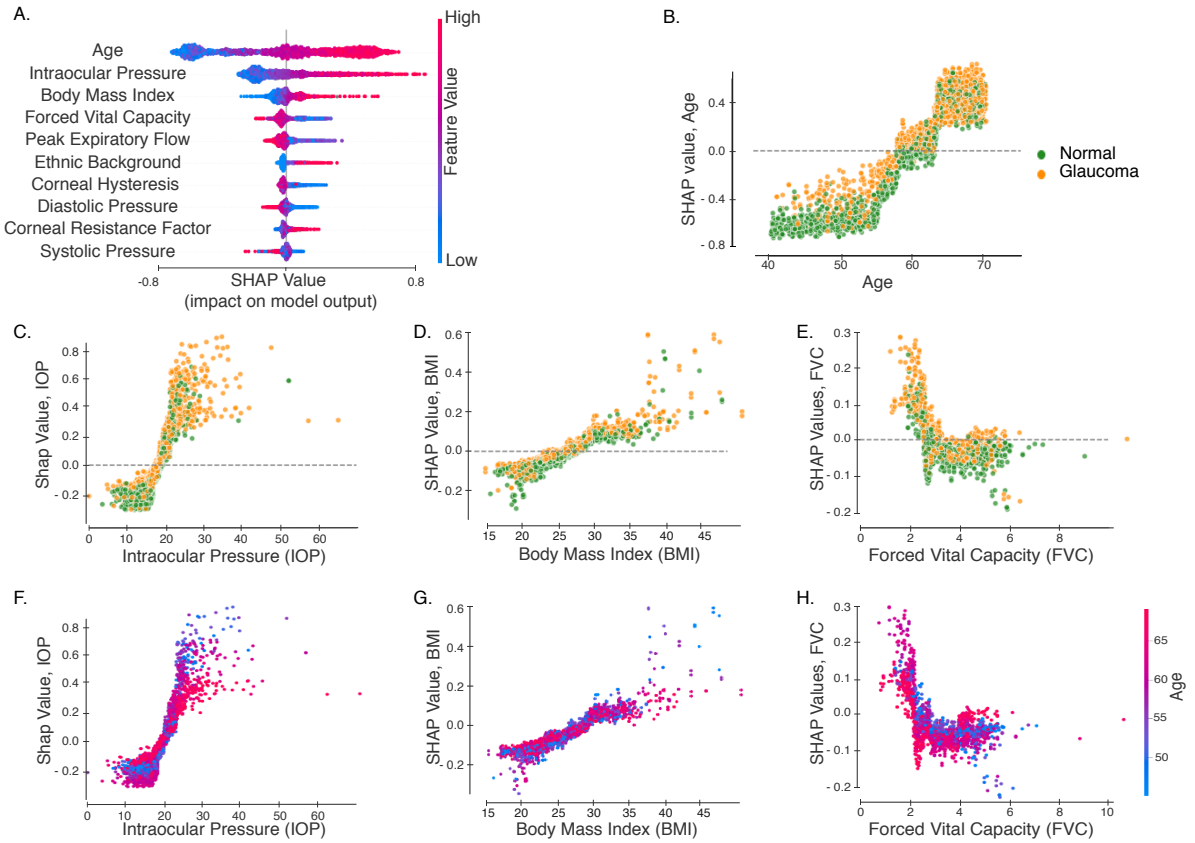


Figure 3.2: **Interpreting model built on demographic, systemic and ocular data**. A) The ten most important features from BM3 based on SHAP values. B-E) SHAP values vs feature values for age, IOP, BMI, and FVC respectively. Each point represents an individual subject and the color denotes whether or not they have been diagnosed with glaucoma. F-H) SHAP values vs feature values for features IOP, BMI, and FVC respectively, with each point colored based on age of subject.

We built a separate DL model on each retinal image modality. Glaucoma is characterized by structural changes in the optic disc and other parts of the retina. Visual examination of CFP and macular OCT images is therefore an important tool in current diagnostic practice [129]. Since our data set included both CFP and OCT images, we built separate DL models for each image

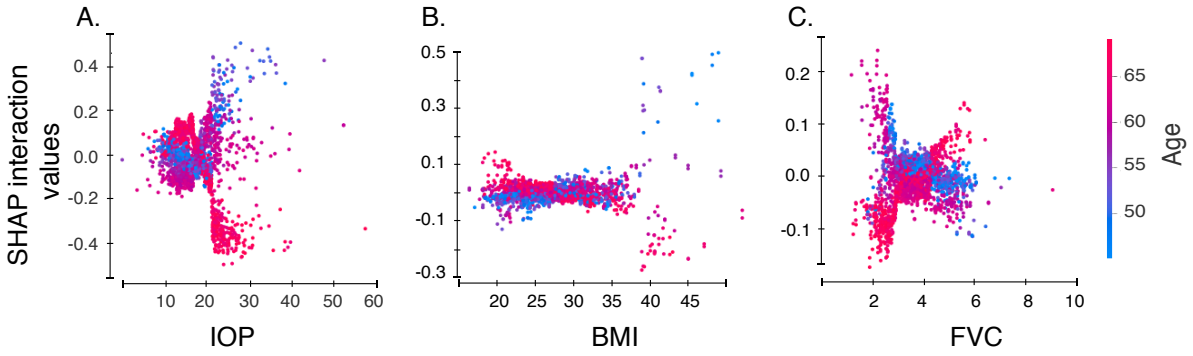


Figure 3.3: **SHAP interaction values in BM3**. Interactions between age and each of the other top variables in model BM3: **A** SHAP interaction values for age and IOP. **B** SHAP interaction values for age and BMI. **C** SHAP interaction values for.

modality (Section 3.2). The DL model built on CFP classified eyes diagnosed with glaucoma with modest accuracy (AUC: 0.74, 95% CI: 0.64-0.84; Figure 3.1 B and E). The DL model built on macular OCT images was more accurate than all of the baseline models and the model trained on CFP images (AUC: 0.95, 95% CI: 0.90- 1.0). When we combined information from both the DL models trained on CFP and OCT via an ensemble, the resulting model was marginally more accurate than the DL model build on macular OCT images alone (AUC: 0.963, 95% CI:0.91- 1.0). This suggests that most CFP information is redundant with information present in macular OCT images.

We used several methods to interpret the DL models. DL models are notoriously inscrutable. However, several methods for interrogating these models have recently emerged [175, 134, 135, 164, 171]. To assess the features that lead to high performance of the image-based models, we first assessed which scan of the macular OCT provided the most information. We fit individual models to each scan of the macular OCT. Recall that macular OCTs are volumetric images; in the UK Biobank data set, each macular OCT consists of 128 scans. We found that models using scans from the inferior and superior macula were more accurate than those using the central portion of the macula (Figure 3.4 A). Second, we built an ensemble model that used the results of the DL models of the individual macular OCT scans to predict glaucoma occurrence per retina. This

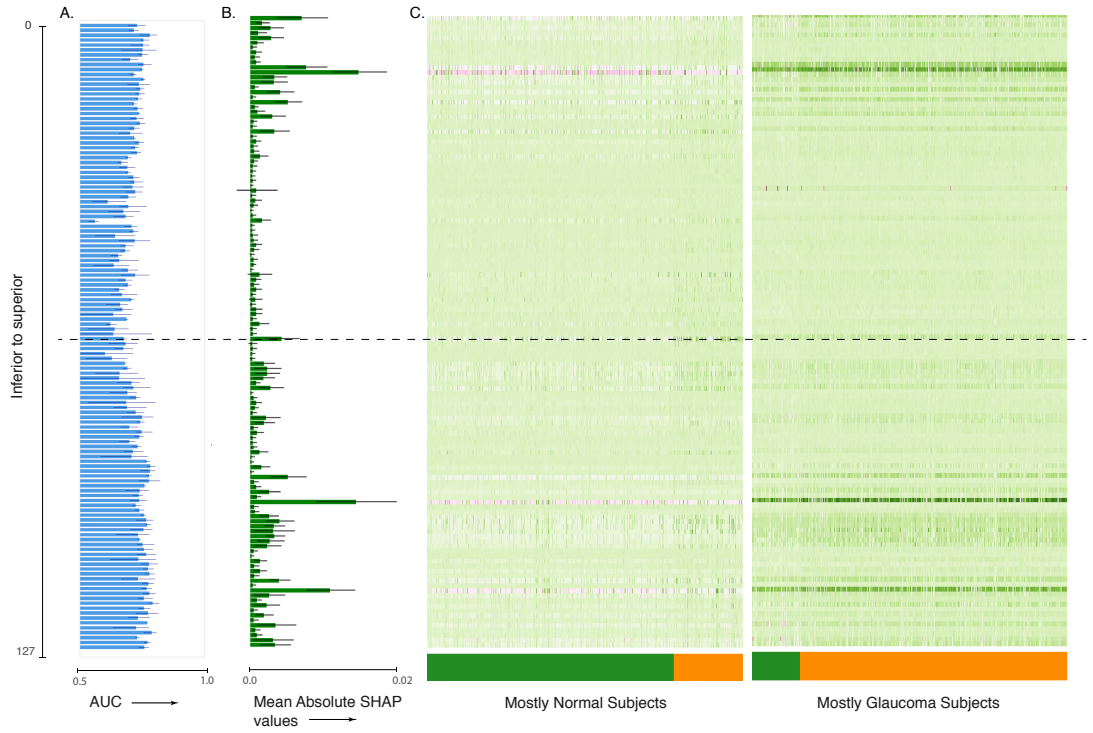


Figure 3.4: **Interpretation of the ensemble model built on macular OCT images**. **(A)** AUC for single image per retina models, **(B)** mean absolute SHAP values per retinal image for predicting glaucoma occurrence per retina, and **(C)** heat map of SHAP value per retinal image for predicting glaucoma occurrence per retina. The images are ordered from top to bottom and from superior to inferior retina. The dashed line indicates the central retinal image from the OCT volume.

model used each of the 128 macular OCT scans to make a prediction about the retina. Figure 3.4 B shows the feature importance attributed to each scan via SHAP; it shows that scans from the inferior retina were deemed more important by this model. Large patient and control populations are heterogeneous, and we do not generally expect that information will consistently come from one particular part of the retina. Nevertheless, when considering the SHAP values of each macular OCT scan, we found that the data set broke into two major clusters based on the SHAP values from different retinal parts (Figure 3.4 C). One cluster mostly contained retinas from healthy subjects and used scans from the inferior part of the retina as negative predictors of glaucoma. The second cluster mostly contained glaucomatous retinas, and SHAP values of these same scans from inferior

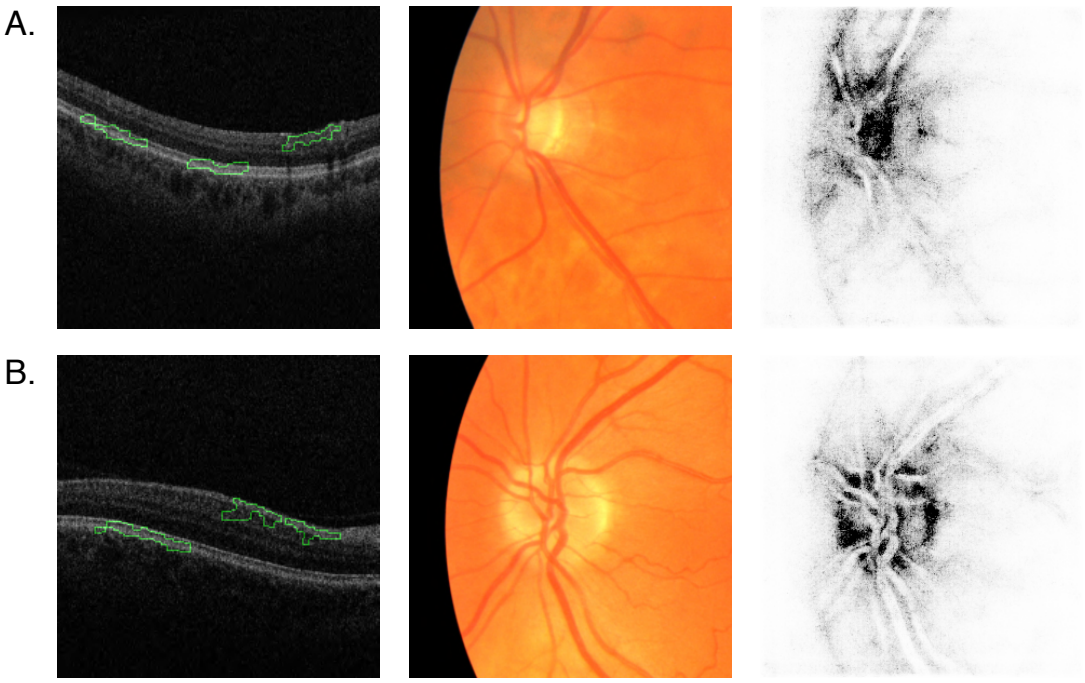


Figure 3.5: Saliency maps for macular OCT and CFP. Columns left to right: macular OCT image overlaid with saliency map, cropped CFP input to the neural network, CFP saliency map. Each macular OCT image is laid out with its temporal side to the left. **(A)** Retina of a subject with glaucoma diagnosis. **(B)** Retina of healthy subject. The green outline on the OCT saliency map indicates the areas the model deems most important. The darker pixels on the CFP saliency map indicate the areas the model deems most important.

and superior macula were used as positive predictors of glaucoma. This also explains why models fit only to scans from the inferior or the superior macula were more accurate.

In addition to the scan-by-scan analysis, image-based models can be evaluated pixel-by-pixel to determine the importance of specific image features to the DL models' decision making. Using integrated gradients [171], we generated saliency maps of the pixels responsible for DL model prediction. Figure 3.5 shows a macular OCT scan for an eye with glaucoma and a scan for a control eye along with the CFP images and CFP saliency maps for each eye. The CFP saliency maps typically highlight the optic nerve head in both normal and glaucomatous retinas. The saliency maps for OCT image typically highlight the nasal side of the RNFL and outer retinal layers.

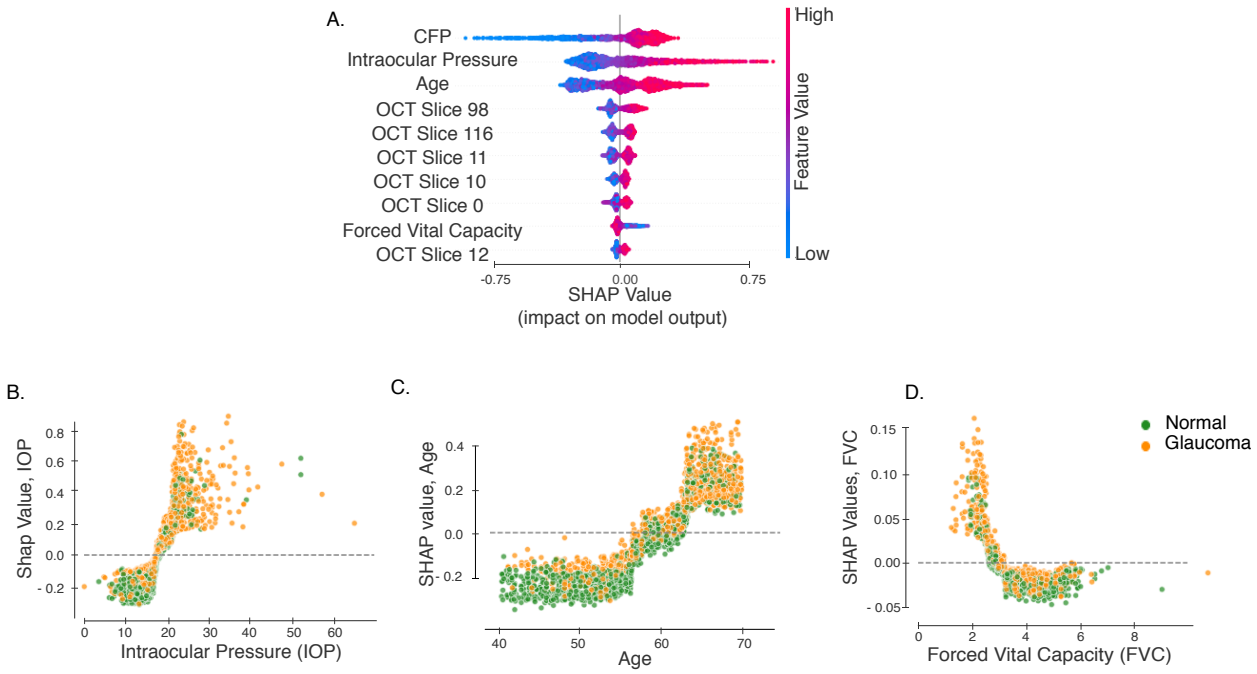


Figure 3.6: **Interpretation of the final model built on image, demographic, systemic and ocular data**. Interpretation for models built on medical and optometric features is based on SHAP values. **(A)** The 10 most important features from this model. SHAP values vs feature values for **(B)** Age, **(C)** IOP , **(D)** FVC and **(D)** BMI. Each point denotes a subject, and the color denotes whether the subject has been diagnosed with glaucoma.

We built the final model by combining both modalities of retinal imaging, demographic, systemic and ocular features. This model was an ensemble, which combined information from raw macular OCT B-scans and CFP images as well as all demographic, systemic and ocular data used in BM3. This final model had an AUC of 0.967, (95% CI: 0.93 - 1.0). Figure 3.6 shows the ten features with the highest mean absolute SHAP value over all observations in the data set. The most important features for this final model, as determined by their SHAP values, include age, IOP and FVC, in addition to the CFP and macular OCT scans from both inferior and superior macula. BMI is less significant than FVC in this final model. Further, IOP had a higher importance than age. This is a reversal in importance of features when compared to models built without information from retinal imaging. Unsurprisingly, this confirms that the CFP and OCT scans contain information

that supersedes in importance the information provided by BMI and age.

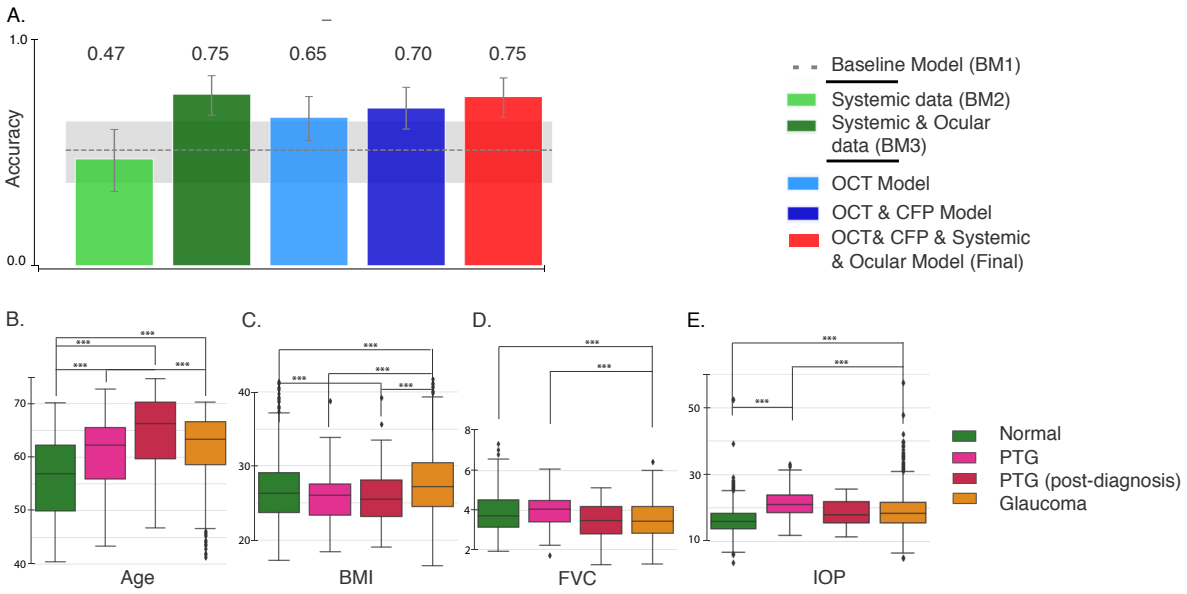


Figure 3.7: **Evaluation of various models on the PTG cohort**. (A) Accuracy of the models on the ”progress-to-glaucoma” (PTG) cohort. The gray dashed line and shaded area denote the AUC and 95% CI for a base model built on demographics alone (age, gender and ethnicity; BM1). The bottom row shows the distribution of Age(B), BMI(C), FVC(D) and IOP(E) for healthy, PTG, PTG (after glaucoma diagnosis) and glaucoma. ***P ¡ 0.0001.

We compared the performance of our model with ratings from glaucoma experts to provide a comparison to current clinical practice. To compare the performance of our final model to expert clinicians, five glaucoma experts evaluated CFPs of the test set. Initially, experts were also given access to OCT images for each subject. However, raw b-scans from macular OCTs are not an image modality that experts usually examine during regular clinical practice for glaucoma diagnosis. Since we did not have access to thickness maps, experts made the diagnoses using only the CFP data. (Figure 3.1 C and D). The highest AUC for the expert rating was 0.84, and the lowest was 0.79. The average pairwise kappa for the five experts was 0.75, indicating a good level of agreement between experts about the diagnosis.

We validated our model by evaluating it on patients that progress to glaucoma. The UK Biobank data set contained several subjects who lacked a glaucoma diagnosis on their first study visit but

received a diagnosis before a subsequent visit. These ”progress-to-glaucoma” (PTG) subjects provide a unique opportunity to evaluate our model, which was built on data from glaucomatous and healthy subjects. Detection of glaucoma in the PTG cohort was tested using all of our models (Figure 3.7 A). Both BM1 (based on age, gender and ethnicity) and BM2 (added systemic variables) were indistinguishable from chance performance (BM1: 51 % correct: 95% CI [36% - 64%]; BM2: 47% correct: 95% CI[33%-60%]). BM3, which included ocular variables, achieved substantially higher accuracy at 75% correct (95% CI: [67%-83%]). The model trained on macular OCT images achieved slightly lower accuracy at 65% correct (95% CI [55% - 74.5%]), and the model trained on combined CFP and OCT achieved an accuracy of 69% (95% CI [60.2% -78.6%]). The final model trained on OCT, CFP and all other available features achieved an accuracy of 75% correct (95% CI [65% - 83%]).

Since PTG subjects did not undergo glaucoma treatment, this evaluation provides additional insight into the biological features of the disease. For many of these features, including age and BMI, the PTG group lie between the normal and glaucoma groups (Figure 3.7 B to E). We identified two interesting deviations from this pattern. First, for the pulmonary capacity variables (FVC and PEF), the PTG group was indistinguishable from the healthy subjects in our sample, and both healthy and PTG subjects significantly differed from patients with glaucoma. This difference is statistically significant even when controlling for age (ADD). However, on a subsequent visit, after receiving a glaucoma diagnosis, the pulmonary capacity measurements of this group was indistinguishable from that of the glaucoma group. Second, the PTG group had a significantly higher IOP than the group diagnosed with glaucoma (Figure 3.7 D; Section 3.1.1. The post-diagnosis IOP measurements of the PTG group shows similar trend with lower IOP values.

3.1.1 Statistical analysis of group differences

Statistical analysis of differences between glaucoma, healthy, PTG and PTG post-diagnosis demonstrates that pulmonary capacity variables differ between the groups. This is true even when controlling for age: an ANCOVA of FVC values for the three groups, using age as a covariate found a statistically significant effect of group identity Table 3.1. To further confirm this, we followed

this ANCOVA with a pairwise Dunn's test (Tables: 3.2, 3.3, 3.4 3.5, 3.6). We found that FVC in PTG significantly differed from FVC in patients with glaucoma (Table 3.2). Likewise, FVC in normal vs. patients with glaucoma differed significantly, but we did not find a difference between normal and PTG. Values for PEF were similar: the ANCOVA using age as a covariate showed a statistically significant effect of group (Table 3.1). In addition, a pairwise Dunn's test confirmed that the IOP difference between glaucoma and PTG groups was statistically significant (Z=-10.74, p¡0.05). Post-diagnosis, the difference between PTG and glaucoma in FVC and PEF was no longer statistically significant. This was also true for IOP, but this is based on only a limited sample of PTG (n=21 retinas), for which there is a measurement of IOP post-diagnosis (Section 3.2).

3.2 Methods

Data access. Data was obtained through the UK Biobank health research program. Deidentified color fundus photos, OCT scans, and health data were downloaded from the UK Biobank repository and our study, which did not involve human subjects research, was exempt from IRB approval.
Data set and Cohort selection. The UK Biobank is an ongoing prospective study of human health, for which data has been collected from over half a million individuals [170]. Participants throughout the UK were recruited between 2006 and 2010 and were aged 40-69 years at the time of recruitment. The data set contains information from questionnaires, multi-modal imaging measurements, and a wide range of genotypic and phenotypic assessments. Data collection is ongoing, allowing for longitudinal assessments. We analyzed a subset of the UK Biobank participants based on a snapshot of the repository that was created in the fall of 2017. This subset consisted of data from 96,020 subjects, 65,000 of which had retinal imaging data. This data set consisted of between one to three visits for each of the subjects. Color Fundus Photographs (CFP) data was available for only the first visit for these subjects. Retinal OCT data was available for first and second visits. The participants were given questionnaires to report various eye conditions, to which they could report healthy or chose one or more the following eye conditions: glaucoma, cataract, macular degeneration, diabetic retinopathy and injury for each eye. We used the answers provided as the labels for each eye. We did not examine the images to determine or alter the labels thus associated

with the retinal image and clinical data.

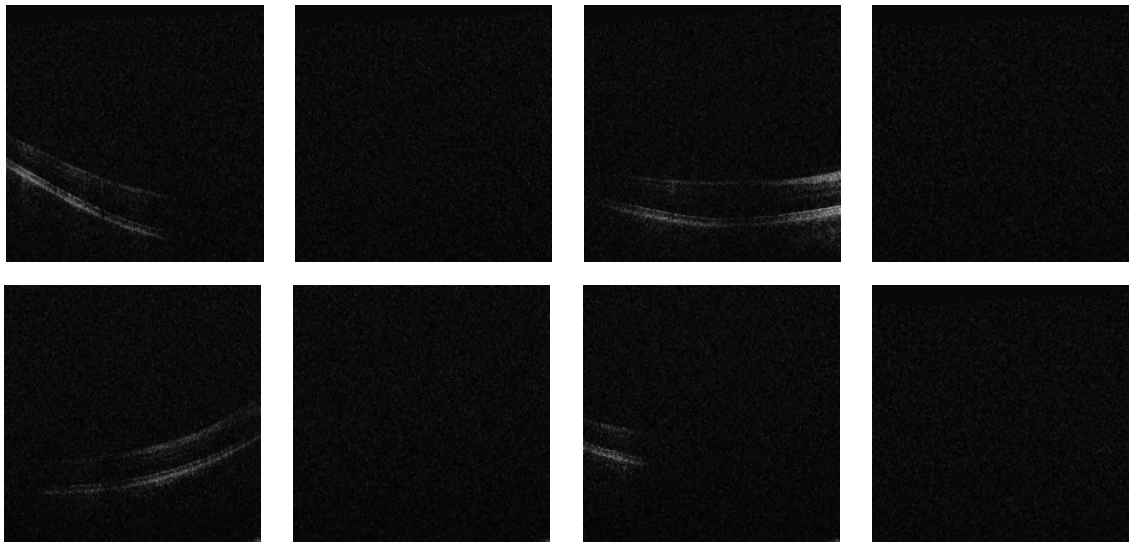


Figure 3.8: Sample of OCT volumes eliminated due to alignment errors, this shows first two OCT slices for four retinas where alignment failure occurred. All OCT slices were eliminated for such retinas.

Cohort selection: We selected a cohort from this data for the following three classes: A) subjects who in their first study visit report that they have been diagnosed with glaucoma and consistently report a glaucoma diagnosis in follow-up visits (glaucoma); B) subjects who in their first study visit report that they had no ocular conditions and consistently reported no ocular condition in follow-up visits (healthy); C) subjects who in their first visit report no ocular conditions, but in a subsequent visit report having received a diagnosis of glaucoma, labeled as the ”progress to glaucoma” group (PTG). Ocular measurements were only available for the first two visits. The ocular data includes retinal imaging (both CFPs and macular OCTs) as well as IOP, Corneal hysteresis and Corneal resistance factor. However, a subset of the PTG group (n=21 retinas) received glaucoma diagnosis between the first and second visit and we used this subset to conduct statistical analysis of IOP. Systemic and pulmonary variables were available for the entire PTG group both pre- and post-diagnosis, and we were able to analyze the impact of diagnosis on these variables for the entire PTG group.

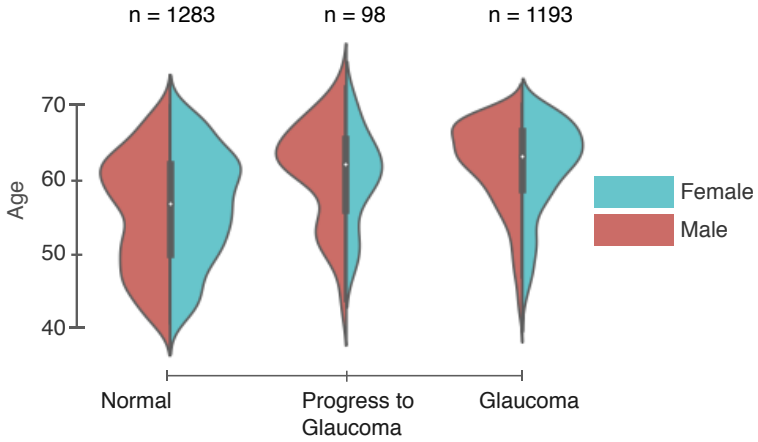


Figure 3.9: **Data distribution:** Age and gender distributions as well as number of subjects in the Normal (left), progress to glaucoma (middle) and glaucoma (right) subject groups.

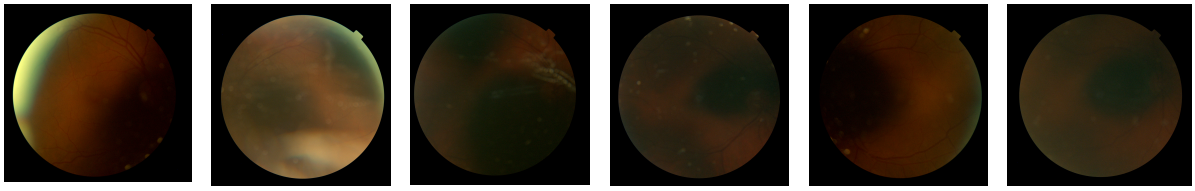


Figure 3.10: Sample of CFP eliminated from test set due to bad quality.

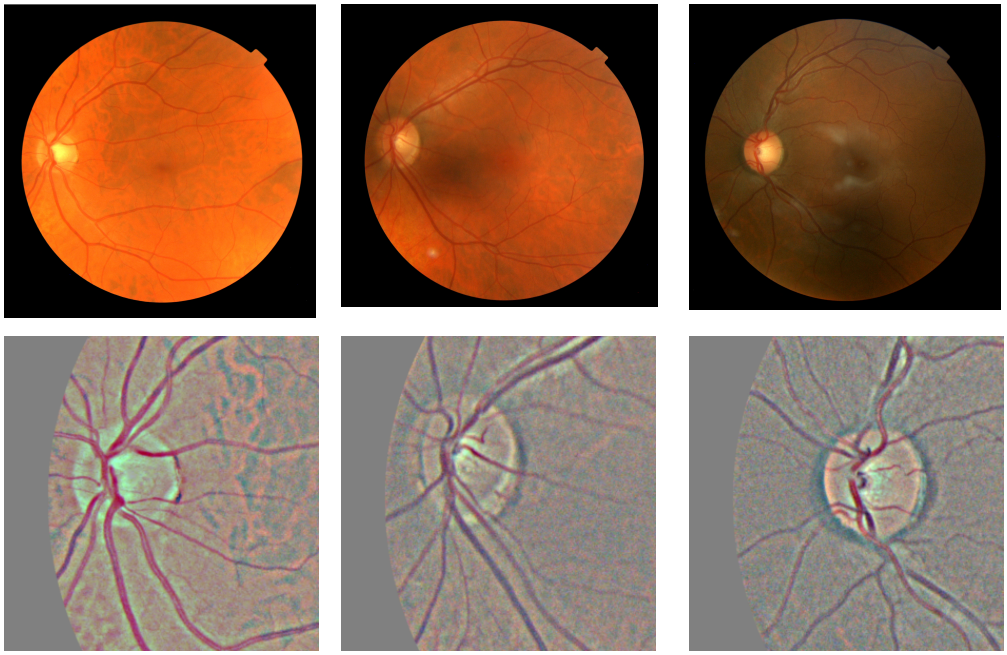


Figure 3.11: **Data processing for CFP images**. Top row shows original images, bottom row shows cropped and pre-processed images.

Exclusion criteria: We excluded all subjects who preferred not to answer questions about their ocular conditions, or did not know how to answer these questions. For *glaucoma* subjects, we excluded any subjects who listed any ocular conditions in addition to glaucoma, such as age-related macular degeneration, diabetic retinopathy, cataract, or injury. For the *healthy* subjects, we excluded any subjects whose visual acuity was recorded as worse than 20/30 vision in either eye. We also excluded any *healthy* subjects with any secondary health conditions (determined by primary diagnosis codes record in their hospital inpatient records). Finally, we excluded any retinal OCT scans from all three classes that could not be aligned using motion translation (x and/or y shift) Figure 3.8. shows a sample of excluded retinal OCT images. The final number of for the three groups was glaucoma ((s)ubjects=863,(e)yes=1193), healthy (s=771, e=1283), and PTG (s=55,e=98). Figure 3.9 shows the age and gender distribution of subjects in each of these groups. CFP images were available for only 56 of the 98 retinas in the PTG group (retinal OCT images were available for all PTG subjects).

Test set: At the outset, we randomly selected 100 eyes, 50 healthy and 50 with glaucoma. These were set aside as the test set on which we evaluated each of the models. We set an additional 170 eyes as a validation set for parameter tuning and model selection. The data was separated by subject such that both eyes of any subject belonged to either test, train or validation set. The test set was also rated by five glaucoma experts. Glaucoma experts used the CFPs for providing their scores. Glaucoma experts marked 13 CFPs from the test set as being such poor a quality to preclude any assessment. All comparisons of clinician and model performance excludes these 13 retinas Figure 3.10 shows a sample of excluded CFPs.

Evaluating expert performance. Five glaucoma-fellowship trained ophthalmologists were recruited for the study to evaluate CFP images from test set to provide an expert diagnoses. The glaucoma experts identified the eye in each CFP as either healthy or glaucoma, and rated the confidence in the diagnosis from 1 to 5. A higher number indicated higher confidence in their diagnosis. This resulted in a 10-point scale for the diagnosis. We used this 10 point scale to create ROC curves for each expert.

Machine learning models and training protocols. We built separate DL models for each imaging modality (retinal OCT and CFP). *Retinal OCT model*: the DL model built on the retinal OCT data took a single retinal OCT image as input and output a probability that the input image was from a subject with glaucoma. This model required individual **B-scans**. Each retinal OCT consisted of 128 B-scan images. This model was not provided any other additional information. This DL model was based on the densenet architecture [80], with four blocks with 6, 12, 48 and 32 layers each. We initialized model weights for this model with MSRA initialization [73]. Each retinal OCT B-scan is a gray scale 512 x 650 image. We flipped each right eye images left to right, we did this so that the optic nerve was on the same side for each scan. Additionally, we cropped each scan to an aspect ratio of 1:1 and down sampled to 224x224 pixels. We used a per pixel cross-entropy as the loss function with 0.1 label smoothing regularization [124]. We used Tensorflow [3] with Adam optimizer [97] and an initial learning rate of 1-e3 and epsilon of 0.1. We trained for 60 epochs(batch size 80) on one graphical processing unit (GPU)s. The hyper parameters for the training protocol were chosen by tuning on the validation data set. To improve the generalization ability of

our model, we augmented the data by applying affine, elastic and intensity transformation over the input images.

CFP model: the DL model on the CFP took a single CPF image as input and outputs a probability that the input image was from a subject with glaucoma. This model was built with transfer learning [139, 9]. We chose transfer learning as (a) we had 128X fewer CFP images, and (b) CFP are color images and transfer learning has been shown to be effective for detecting other pathology in fundus images [105]. We used the InceptionResnetV4 [174] model, pre-trained on ImageNet data [84]. We used the Adam optimizer with an initial learning rate of 1-e5. We trained the model for 20 epochs, with a batch size of 400. During training, we kept the weights in $2/3$ of the network (750 layers) frozen. We pre-processed each fundus image by flipping left CFP image so that optic nerve was on the same side of each image. We also subtracted local average color to reduce differences in lighting, and cropped the images to contain the area around the optical nerve (ADD). We augmented the CFP by applying affine, elastic and intensity transformations similar to the retinal OCT images.

Baseline models: modern gradient boosted decision trees often provide state-of-the-art performance on tabular style data sets where features are individually meaningful, as consistently demonstrated by open data science competitions [62]. We used gradient-boosted decision trees, implemented in XGBoost [36], to build all three baseline models (BM1, BM2, BM3) based on demographic features: age, gender, ethnicity; systemic features: Body Mass Index (BMI), Forced Vital Capacity (FVC), Peak Expiratory Flow (PEF), heart rate, diastolic and systolic blood pressure, presence of diabetes, recent caffeine and nicotine intake; and ocular features: Intraocular pressure (IOP), corneal hysteresis, and corneal resistance factor. The systemic features were chosen as markers of overall health. We used the following hyper parameters for training: learning rate of 0.001, early stopping; ℓ_2 regularization of 1.0, no ℓ_1 regularization, no column sampling during training, and bagging sub-sampling of 70%. Hyper parameters were chosen by tuning on the validation data set.

Final model: we combined clinical data with results from image-based models to build the final model. To combine data from image models we used the probability of glaucoma as estimated by

the respective image model as the feature value for each image. We combined these (128 OCT slices and one fundus) to a 129 element vector as the results of the image-based models. This vector was then combined with all of the features from BM3 for the final feature set. We used gradient-boosted decision trees to build this final model. The hyper parameters were chosen by tuning on the validation set and were as follows: learning rate 0.001, early stopping, bagging sub-sampling of 70%, ℓ_2 regularization of 1.0, no ℓ_1 regularization and no column sampling during training.

Interpretability Methods: for pixel-level importance in the image based DL models we used integrated gradients [171] and SmoothGrad [166] to determine salient pixels for the input images. For the tree-based models built using XGBoost, we used Tree explainer [109] to calculate the SHAP values. The SHAP values were used to determine feature importance and feature interaction.

Dependent Variable	dF	F Value	p value
FVC	1,2403	64.51	$1.49e-15$
PEF	1,2403	50.8	$1.35e-12$

Table 3.1: Analysis of covariance (ANCOVA) of pulmonary capacity variables (forced vital capacity (FVC), peak expiratory capacity(PEF)) for group identity amongst the three groups (Healthy, Glaucoma and PTG), controlling for age.

Groups	Kruskal-Wallis Z	P (unadjusted)	P (adjusted)
Healthy vs. Glaucoma	7.03	$2.12e-12$	$1.270017e-11$
Healthy vs. PTG	-1.63	$1.03e-01$	$6.17e-01$
Glaucoma vs. PTG	-4.36	$1.28e-05$	$7.66e-05$
Healthy vs.PTG (post-diagnosis)	2.48	$1.32e-02$	$7.93e-02$
Glaucoma vs.PTG (post-diagnosis)	-0.26	$7.99e-01$	1.0
PTG vs.PTG (post-diagnosis)	3.02	$2.51e-03$	$1.50e-02$

Table 3.2: Dunn's test comparing **FVC** for Glaucoma, healthy, PTG and PTG (post-diagnosis) groups, with Bonferroni correction for p-values.

Statistical analysis: We used bootstrapping [52] to determine confidence intervals for AUC and ac-

Groups	Kruskal-Wallis Z	P (unadjusted)	P (adjusted)
Healthy vs. Glaucoma	7.68	$1.54e-14$	$9.25e-14$
Healthy vs. PTG	-2.17	$3.02e-02$	$1.81e-01$
Glaucoma vs. PTG	-5.12	$3.01e-07$	$1.81e-06$
Healthy vs.PTG (post-diagnosis)	5.58	$2.38e-08$	$1.43e-07$
Glaucoma vs.PTG (post-diagnosis)	2.65	$8.12e-03$	$4.86e-02$
PTG vs.PTG (post-diagnosis)	5.70	$1.19e-08$	$7.17e-08$

Table 3.3: Dunn's test comparing **PEF** for Glaucoma, healthy, PTG and PTG (post-diagnosis) groups, with Bonferroni correction for p-values.

Groups	Kruskal-Wallis Z	P (unadjusted)	P (adjusted)
Healthy vs. Glaucoma	-19.26	$1.10e-82$	$6.61e-82$
Healthy vs. PTG	-5.90	$3.53e-09$	$2.12e-08$
Glaucoma vs. PTG	1.47	$1.41e-01$	$8.46e-01$
Healthy vs. PTG (post-diagnosis)	-11.51	$1.16e-30$	$6.96e-30$
Glaucoma vs. PTG (post-diagnosis)	-4.15	$3.33e-05$	$2.00e-04$
PTG vs. PTG (post-diagnosis)	-4.12	$3.73e-05$	$2.24e-0$

Table 3.4: Dunn's test comparing **Age** for Glaucoma, healthy, PTG and PTG (post-diagnosis) groups, with Bonferroni correction for p-values.

Groups	Kruskal-Wallis Z	P (unadjusted)	P (adjusted)
Healthy vs. Glaucoma	-6.09	$1.09e-09$	$6.57e-09$
Healthy vs. PTG	1.34	$1.80e-01$	1.0
Glaucoma vs. PTG	3.69	$2.26e-04$	$1.36e-03$
Healthy vs. PTG (post-diagnosis)	1.70	$8.84e-02$	$5.30e-01$
Glaucoma vs. PTG (post-diagnosis)	4.05	$5.07e-05$	$3.04e-04$
PTG vs. PTG (post-diagnosis)	0.27	$7.89e-01$	1.0

Table 3.5: Dunn's test comparing **BMI** for Glaucoma, healthy, PTG and PTG (post-diagnosis) groups groups, with Bonferroni correction for p-values.

curacy displayed in Figures 2 and 5. We performed analysis of variance(ANOVA) test to analyze the differences in pulmonary function features (FVC and PEF) among the three groups: healthy, glaucoma and PTG. We used the Dunn Test[51] with Bonferroni correction for pairwise compari-

son to determine differences between the three groups.

Groups	Kruskal-Wallis Z	P (unadjusted)	P (adjusted)
Healthy vs. Glaucoma	-14.31	$1.98e-46$	$1.19e-45$
Healthy vs. PTG	-10.74	$6.32e-27$	$3.79e-26$
Glaucoma vs. PTG	-5.20	$1.96e-07$	$1.18e-06$
Healthy vs. PTG (post-diagnosis)	-2.40	$1.64e-02$	$9.84e-02$
Glaucoma vs. PTG (post-diagnosis)	0.25	$7.99e-01$	1.0
PTG vs. PTG (post-diagnosis)	2.50	$1.23e-02$	$7.38e-02$

Table 3.6: Dunn's test comparing **IOP** for Glaucoma, healthy, PTG and PTG (post-diagnosis) groups, with Bonferroni correction for p-values.

3.3 Discussion

Automating glaucoma detection using imaging and clinical data may be an important and cost-effective strategy for providing population-level screening. In this study, we used machine learning to construct an interpretable machine learning model that combined clinical information with multi-modal retinal imaging to detect glaucoma. We created and compared several models based on clinical data to establish a baseline: BM1 used demographic data (age, gender, ethnicity), BM2 added systemic medical data (cardiovascular, pulmonary), and BM3 added ocular data (IOP, corneal hysteresis, corneal resistance factor). Our final model was an ensemble, which combined information from raw macular OCT B-scans and CFP images as well as all demographic, systemic and ocular data used in BM3. This final model had an AUC of 0.97.

In interpreting this final model, we found that CFP, age, IOP, macular OCT images from the inferior and superior macula, and FVC were the most important features (Figure 3.6). The significance placed upon age and IOP by our final model reiterate previously known risk factors for glaucoma. The positive SHAP values for IOP in our model rapidly increased above an IOP of approximately 20, consistent with the fact that ocular hypertension is a key risk factor for the disease [93, 146, 71, 15, 151]. Age and IOP switched places in their relative importance in our final model, which includes retinal imaging, in addition to BM3 features. This suggests that retinal imaging includes information that supersedes or is redundant with information linked to age. This finding is consistent with previous research, which demonstrated the ability of CFP to predict cardiovascular risk factors including age [149]. We also observe two discontinuities in the age vs. SHAP values for age (Figure 3.6B), at ages 57 and 65. At both of these ages the SHAP values for age increase at a higher rate than before. This could be both due to biological, as well as socio-economic factors (e.g., 65 is the age of retirement in the UK).

An important novel finding of our study was the correlation of pulmonary measures, especially decreased FVC, with glaucoma. There are several possible explanations for this finding. First, a recent study by Chua et al. found a correlation between glaucoma and atmospheric particulate matter [40]. Chua et al.'s study did not include pulmonary function tests such as FVC and was

correlational in nature but other studies have linked exposure to particulate matter with decreased FVC [38, 192, 72], suggesting common causes for reduced FVC and for glaucoma. Second, it may be that the treatment of glaucoma with topical beta blocker therapy has an impact on reducing FVC [144]. This idea receives further support from the findings in the PTG group, who do not have a diagnosis and have presumably not received any treatment. These individuals have FVC that is higher than the glaucoma group and is indistinguishable from the healthy group before a diagnosis is made. After a diagnosis is made, their FVC also decreases to levels indistinguishable from that of the glaucoma group. Thus, lower FVC values could indicate a result of glaucoma treatment.

Examination of the pixel-by-pixel importance of both retinal image modalities provided additional insight into what our model focused on when predicting glaucoma (Figure 3.5). For the CFPs, the model focused on the optic disc, a known source of information in the clinical diagnosis of glaucoma [14]. For the macular OCT B-scans, the model relied on previously validated retinal areas, including the inferior and superior macula [143]. In addition, the algorithm points to the nasal macular RNFL. The effect of glaucoma on RNFL integrity is well-understood, and RNFL thickness maps are often used clinically to diagnose glaucoma. However, the automated algorithms that are used clinically have a high segmentation error rate, resulting in variable thickness estimates, which may in turn lead to errors in diagnosis [114]. By avoiding reliance on extracted features such as thickness maps, our approach enabled the discovery of other possible biological features of glaucoma. For example, consistent with recent results in the same data set [96], the model also identified other (non-RNFL) parts of the inner retina as important (e.g., see Figure 3.5 B).

In addition to the RNFL and inner retina, the model relied on the outer layers of the retina for glaucoma diagnosis. The involvement of the retinal outer layers in glaucoma is controversial. In a typical analysis of OCT images that focuses on the thickness of different parts of the retinal layers, glaucoma effects are usually not found in outer layers [185, 42, 98], but an association between age, IOP and retinal pigment epithelium thickness is sometimes detected [99]. Some anatomical studies do not find any differences in the outer retinal layer between healthy and glaucomatous eyes [94]. Other studies, using psychophysical methods in human subjects with glaucoma [78, 132], using

histological methods in human eyes [140, 131] or examining animal models of glaucoma [29, 131] have shown the involvement of the retinal outer layer in glaucoma. In addition, Choi et al. [39] used high-resolution imaging techniques (ultrahigh-resolution Fourier-domain optical coherence tomography, UHR-FD-OCT, and adaptive optics, AO) to image glaucomatous retinas. They found a loss of retinal cone receptor cells that correspond to visual field loss. This loss of cones could cause subtle changes in the appearance of this part of the retina, that are not reflected in changes in thickness but are still captured by the DL model (e.g., changes in texture). The ability of DL models to use visual cues that are not apparent to the human eye has been previously demonstrated in another study in which retinal angiograms were generated from OCT images [104]. This finding is also consistent with a recent study that used unsegmented OCT scans and reported the involvement of outer retinal layers in a DL model that detects glaucoma [111, 181].

Our final model detected the occurrence of glaucoma with an accuracy of 75% on a cohort that had not yet been clinically diagnosed at the time of their testing (”progress-to-glaucoma”, PTG). This does not constitute early detection: even though the individuals were not clinically diagnosed, they may already have significant disease progression, since many patients are undiagnosed even in relatively late stages of the disease [172]. The median IOP value was higher for the PTG cohort than for the subjects diagnosed with glaucoma, possibly because treatments for glaucoma are designed to decrease IOP. The PTG group also tended to be younger than those diagnosed with glaucoma. Interestingly, FVC in the PTG group was higher than in the glaucoma group and was indistinguishable from healthy subjects. This finding helps explain why BM2, which relied heavily on PVC and PEF, performed relatively poorly on the PTG cohort, achieving an AUC of 47% (Figure 3.7 A). It also provides possible evidence against a causal relationship between FVC and glaucoma, as mentioned above. Furthermore, in a post-diagnosis visit, pulmonary factors (FVC and PEF) in these individuals were lower and indistinguishable from that of the patients with glaucoma, further supporting a possible treatment effect. This area warrants further investigation.

Before our model can be considered for use in a real-world setting, several limitations should be considered and addressed. First, we included only subjects without other ocular disorders. In the general population, glaucoma may coexist with other ocular comorbidities, and it is unclear

what effect this may have on the model's ability to detect glaucoma accurately. However, selecting subjects with only a glaucoma diagnosis and no other ocular morbidities instills confidence that the model we built is glaucoma-specific, delineates the boundaries between these groups, and identifies the features specific to glaucoma. Second, features of the optic disc are clinically important in diagnosing glaucoma. The limited quantity and poor quality of the CFPs in the UK Biobank data set likely contributed to the low AUC of both the CFP DL model and the expert clinician grading. Finally, a major limitation of our study was the veracity of ground truth labels used to train the model. Labels used were based on self-report. While we eliminated any subject who had inconsistent answers or declined to answer, the generally high rate of undiagnosed glaucoma and the potential for recollection error means that some participants may have been incorrectly labeled.

Our study combined information from multiple sources – including two different retinal imaging modalities (CFP and OCT), demographic data, and systemic and ocular measurement – to build a model that detects glaucoma. This approach yielded not only very accurate detection, but it also enabled us to isolate and interpret critical variables that helped us draw clinical insights into the pathogenesis of the disease.

Chapter 4

SAMPLING FOR DEEP LEARNING MODEL DIAGNOSIS

"Sampling, statisticians have told us, is a much more effective way of getting a good census."
– Rob Lowe

Deep learning (DL) models have enabled unprecedented breakthroughs in developing artificial intelligence systems for analyzing dense, high-dimensional data, such as text, audio, and images. Deep learning is a subset of machine learning methods that use multiple layers to progressively extract higher level features from raw input. Deep learning models have become an indispensable tool for a wide range of tasks, such as image classification, object recognition, speech analysis, machine translation, and more. The task of diagnosis for these purportedly black-box models requires additional artifacts, such as activations. These additional artifacts must be generated, stored, and queried for each DL model being debugged. The addition of these artifacts, which can be up to three orders of magnitude larger than the input data size for each model being diagnosed, turns the process of building, diagnosing, and selecting DL models into a large-scale data management challenge. Diagnosing DL models requires access to artifacts, such as model activations and gradients. Activation values, or *activations*, are learned representations of input data. *Gradients* are partial derivatives of the target output (e.g., the true label of the input data) with respect to the input data. At a high level, activations and gradients are high dimensional vectors with sizes that depend on input data dimensionality and DL model architecture. While activations depict what the deep learning model *sees*, gradients depict *areas of high model sensitivity*.

As previously noted, the naive solution of model diagnosis is pre-computing and storing all artifacts required for model diagnosis scales as the product of the size of input data and number of parameters of the deep learning model. Although the total number of artifacts for small datasets and models is manageable, an overhead that is three orders of magnitude larger than the input data per model is not scalable. On the other hand on the fly generation of these artifacts requires tens of second to several minutes. This makes it difficult to efficiently perform diagnosis tasks, often preventing interactive diagnosis. Thus, with hundreds of gigabytes of artifacts per model, building, diagnosing, and selecting a DL model becomes a large-scale data management challenge.

Sampling is a fast and flexible database technique for approximate query processing, it works well in high dimensions [8, 6, 19, 75] and is a potential candidate for this workload. Top-k maximally activated neurons and the distribution of maximally activated neurons form a large subset of the DL model diagnosis queries (see Section 4.2 in addition to the average values of neurons. e.g. *What are the top-10 maximally activated neurons for layer* $conv2$ *for all incorrectly classified objects for* $model_A$*?* The sample in this case would be a subset of data items from the test and training data sets. The top-k queries are an important area of database research. The best known general-purpose algorithm for identifying top-k items is the Threshold Algorithm [54], which operates on sorted multi-dimensional data required to compute the top-k elements. Approximate algorithms for top-k retrieval require building probabilistic models to fit the score distribution of the underlying data as proposed in [180]. However, we wish to avoid computing and storing activations for the entire dataset, which is three orders of magnitude larger than the input data set for each model being diagnosed.The key challenge therefore is in computing a good sample. As we show in this chapter uniform random sample and even a stratified sample do not perform well. Additionally, we seek to avoid computing and storing and analyzing the distribution of activation values for the entire dataset to create a sample. Instead, we leverage the lower dimensional representation of data learned to select a sample for DL model diagnosis (see Section 4.3 for details). Our approach not only reduces the storage footprint and speeds-up queries since we store less data, but it also speeds-up the overall diagnosis process by saving the time it would otherwise take to generate all artifacts for the entire dataset.

Specifically, our contributions include:

- Characterizing requirements of DL model diagnosis by studying debugging queries in the literature. We further develop a simple benchmark for this novel workload by generalizing individual queries used in model debugging papers into query sets that cover families of queries (Section 4.2).

- Presenting a new technique for creating samples for DL model diagnosis (Section 4.3).

- Evaluating our approach on two datasets and demonstrating its performance compared with a variety of state-of-the-art alternatives.

Our sampling technique can be used to debug any deep learning model where a lower dimensional representation of the input data is learned in a supervised, semi-supervised or unsupervised manner.

4.1 Preliminaries

We now summarize current approaches for DL model diagnosis and their associated data management challenges, which we address in this paper.

A DL model takes as input a vector $x = [x_1, \ldots, x_N], \in R^N$ and produces as output another vector $S(x) = [S_1(x), \ldots, S_C(x)]$, where C is the total number of output neurons. DL models are constructed in layers, intermediate layers are called *hidden layers*, and each hidden layer of the model is vector-valued. The dimensionality of these hidden layers determines the width of the model, and the number of hidden layers determines its depth. These layers often perform different operations such as convolutions, pooling, dropout, etc. - and are named accordingly. When the model is evaluated over an input data instance, such as an image, it produces a value for each of the C neurons. The raw values thus produced are activations, and derivatives of these values with respect to a target, such as class label, are gradients. Diagnosis of DL models relies on these artifacts. The ML community has a variety of techniques to diagnose these models, which we discuss below.

dataset name	Image size (KB)	No. model params	Ratio artifacts:input data
MNIST	0.78	107,786	53 ×
Galaxy Zoo2	1.3	1,095,842	2905.5 ×

Table 4.1: Data size and model sizes for standard ML dataset (MNIST) and scientific image dataset (Galaxy Zoo2).

4.1.1 Visualization

Manual visual inspection, is a popular diagnosis technique for DL models [64, 92, 91, 90, 106]. Standalone tools for visual inspection of DL models built on image data (Cnnvis [106]) and text data (Activis [90], LSTMvis [169]) have been proposed. Some visualization capability is also integrated with deep learning libraries (e.g. Tensorboard [177], etc.). These tools provide static and interactive visualizations of DL model activations and on occasion, gradients. They let ML practitioners view activation or gradient patterns for various layers as well as view aggregates (e.g., average activation) over sets of input data instances belonging to each class, which may be classified correctly or incorrectly. This lets ML practitioners identify specific neuron pattern anomalies and neuron groups and data instances that require further investigation.

Challenge: The size of the artifacts required for these visualizations depends on the size of the input data, and the complexity of the model. It can easily be 3 orders of magnitude larger than the input dataset as shown in Table 4.1. To support interactive visualization for arbitrary queries, these artifacts must be pre-computed since real-time computation is too slow to be interactive. To deal with the associated data explosion, tools such as Activis [90] limit the number of layers that can be visualized in the tool.

4.1.2 Examining learned representation

A DL model simultaneously learns a lower level representation of the data and a classifier (in the case of supervised learning). The learned representation (activations of neurons over an input dataset) is used for a variety of goals, such as understanding how a model discriminates between

different classes, comparing different model architectures or hyper-parameters, and examining how learning progresses over time by analyzing representations at various checkpoints in the learning process [113, 122, 101].

Challenge: One such analysis [113], takes as input the activations of neurons in two layers from the deep learning model, performs singular value decomposition(SVD) on them, projects activations into the subspace identified by SVD, and computes canonical correlation to find directions in these subspaces that are most aligned. These category of analyses require activations for the *entire* model(s) over the *entire* input dataset. If the training process is being examined, the activations for multiple checkpoints must be generated and stored. As above, the required artifacts, especially if diagnosing multiple models or multiple checkpoints, can result in a data explosion.

4.1.3 Feature visualization and saliency analysis

The feature visualization techniques answer questions about what a DL model or parts thereof are looking for by generating examples from the learned model [133]. *Feature visualization* uses derivatives to iteratively modify an input, such as random noise, with the goal of finding the input that maximally activates a particular neuron(s). *Saliency analysis* identifies parts of the input that have the largest effect on the output. This consists of a number of approaches that propagate gradients through the model to identify areas of highest activation and highest sensitivity [165, 201, 112, 160, 171].

Challenge: Even simple DL models consist of hundreds of thousands of neurons (e.g. Table 4.1). Finding the appropriate set of neurons to visualize can be beyond the powers of human cognition. Saliency analysis works on a per-input-data-item basis; ML practitioners would need specific input data points, such as images, for these methods. DL datasets consist of tens of thousands of instances, picking appropriate data instances from these large datasets is imprecise, especially if the dataset is new, large and contains unexplored scientific data.

4.1.4 Statistical analysis

Many datasets are annotated. Language models are annotated with parts of speech or linguistic features and image datasets are annotated with object information. For instance, Broden dataset [22], has pixel-level annotations that indicate the object to which each pixel belongs. These annotations are used to pose hypotheses and conduct statistical analyses between neuron(s) activations and annotation to evaluate these hypotheses [159]. We include statistical analysis as it is important technique for model diagnosis and interpretability.

Challenge: Statistical analyses require such annotations to formulate hypotheses. The two datasets we utilize do not have any annotations. Indeed, most scientific image datasets do not, which makes statistical analysis impossible,

4.2 Workload Characterization

We now develop a summary workload that captures the requirements of a large set of DL model diagnosis queries. Model diagnosis techniques, such as visualization and examination of learned representation, bring the number of neurons and data instances to be examined to a smaller and manageable number. This section focuses on queries from these two categories, as these queries helps make downstream analysis, such as feature visualization, attribution and saliency analysis tractable over massive datasets. We do not include queries from statistical analysis as it requires annotations on the dataset.

ML practitioners typically start model diagnosis with techniques utilized by visualization tools from Section 4.1. A common practice is to create data subsets that are incorrectly classified, generating aggregates (such as average activations, top-k highest activations etc.), and compare them to similar aggregates for data instances that were correctly classified for each class. ML practitioners start the analysis from sets [90, 106], such as all incorrectly labeled instances of $class_a$, rather than specific instances to find such patterns. This analysis helps them identify important patterns for the various subsets and reduce to a manageable number both data instances and parts of the model (layers and neurons) to be examined [90, 106]. Based on this analysis ML practitioner can now

QN.	Queries
Q1.	What is the average value for all neurons for layer $Conv2$ in $model_A$ across all classes? [90, 64, 113]
Q2.	What are the top k maximally activated neurons for layer $Conv2$ for all incorrectly classified objects for $model_A$? [90, 106, 92]
Q3.	What is the average neuron activation pattern for the last hidden layer in $model_A$ for incorrectly classified $class_a$ vs. correctly classified $class_a$? [90, 92, 166]
Q4.	Compute the similarity between the logits of $class_a$ and the representation learned by the last convolution layer by $model_A$? [113, 100]
Q5.	For images of $class_a$ classified as $class_b$, what are all of the maximally activated neurons in the last convolutional layer? [106, 90]
Q6.	Does $model_C$ learn a representation for $class_e$ faster than it learns the representation for $class_f$? [101, 113]
Q7.	How similar are the representations learnt by two different model architectures, $model_A$ and $model_B$, on the same dataset? [113, 100, 122]

Table 4.2: Representative queries for deep learning diagnosis workload.

start correlating input data and parts of the model, conducting attribution and saliency analyses. Similarly, the common practice when comparing two different models trained on the same data leverage techniques listed in examining learned representation from Section 4.1. For instance, ML practitioners generate neuron activation values for each data item for both models for each layer and then compare these to the logits for each class learned by the respective model to decipher each model's rate of learning to understand the impact of additional layers and their sizes and thus diagnose how complex the model must be to complete this task.

Table 4.2 lists representative queries from the literature used to diagnose DL models. We make two observations from this list of queries. First, DL model diagnosis queries require one of three quantities: the top-k maximally activated neurons, the distribution of maximally activated neurons or the average activation values. The focus on maximal and top-k values as opposed to minimal values is due to activation functions [7] used in DL models. Without such functions DL models are just complicated linear regression models. ReLU is the most commonly used activation function today [153]. It removes negative values and propagates positive values. Mathematically, ReLU is

defined as $max(0, val)$. Therefore, in the DL literature sample queries often focus on average or maximal values but not minimum values. Thus, to characterize an ML diagnosis workload instead of focusing on all aggregates we focus on three aggregates (1)Top-k maximally activated neurons, (2) Average activation values for neurons and (3) distribution of maximally activated neurons.

Second, each query in Table 4.2 is part of a family of queries. For instance, the answer to Q1 requires average values of all neurons for a specific layer ($conv2$) for all classes. A family of queries for Q1 would include average values of neurons for *any layer* and *any class* where data instances could be *correctly* or *incorrectly* classified. We can see that queries Q3 and Q1 belong to the same family. Similarly, Q2 belongs to a family of queries that require top-N neurons, across classes, layers, incorrect, and correct classification. Thus, to characterize this workload we utilize the entire family of queries. We call these families of queries *query sets*.

We now introduce some notation and define query sets formally.

A DL model M is a vector of N units or neurons. M is learned and tested over data D. Artifacts, such as activations A, are vectors of the same dimensionality as M, computed over data D. a_{id} is the activation value(s) of a set of i neurons, where $i \subseteq N$, on d^{th} data item(s), where $d \subseteq D$. A query computes a measure ϕ, such as the mean, top-K, count, or count of maximum values for a_{id}. A query set S is a set of queries that enumerate over all vectors of activation values a_{id} that correspond to all layers, all classes and both correct and incorrect classifications. Given the preceding notation we can define a DL model diagnosis query set:

Definition 1. *A query set $S(a_{id}, \phi)$ is a set of queries, where $i \subseteq N, d \subseteq D$, and ϕ is a measure.*

Instead of evaluating our techniques on specific queries from Table 4.2, we utilize the three query sets shown in Table 4.3 to characterize DL model diagnosis workload. These query sets include all queries of a specific family. We leverage these query sets to measure effectiveness of sampling techniques to ensure these techniques do well on the entire family of queries represented by the query set, not just on individual queries.

Query Q2 and all queries of this family are represented by query set S1, which computes the top-K maximally activated neurons. Queries Q1, Q3, and others of this family are represented by

Query Sets
S1. Set of top-K maximally activated neurons.
S2. Average activation values of neurons.
S3. Distribution of maximally activated neurons.

Table 4.3: Query sets for deep learning model diagnosis workload.

query set S2, which computes the average activation for neurons. Queries Q4, Q5, Q6 and Q7, and others of their family are jointly represented by query sets S2 and S3, because finding similarity depends on the average neuron values and the maximally activated neuron distribution.

Query sets can consist of any combination of neurons and data items. Instead of considering this immense set of combinations, we limit our evaluation to all combinations of layer, class and classification (correct or incorrect). Thus, to measure accuracy of a query set for a sample, we first compute the query results for each of these combinations (layer, class and classification). Next, we compute a metric comparing the results from the sample with the results for the same combination on the entire dataset. Our comparison utilizes metrics specific to each query set, e.g., precision for S1, cosine distance for S2 and, Jensen-Shannon distance for S3 (we describe these metrics and the rationale for picking them in more detail in Section 4.4). Finally, we calculate the over-all query set accuracy for each query set by averaging the value of the corresponding metric over the combinations.

4.3 Approach

To enable interactive model diagnosis, our approach creates a sample. We compute the results of a query set on this sample instead of the entire data. In this section we describe our approach and present other baseline techniques for selecting these samples.

The key insight we utilize to avoid generating and storing activation values is that DL models learn a lower dimension representation of the data, and a classifier. DL training transforms the input data, creating a new representation with each layer. Training criteria encourage training set

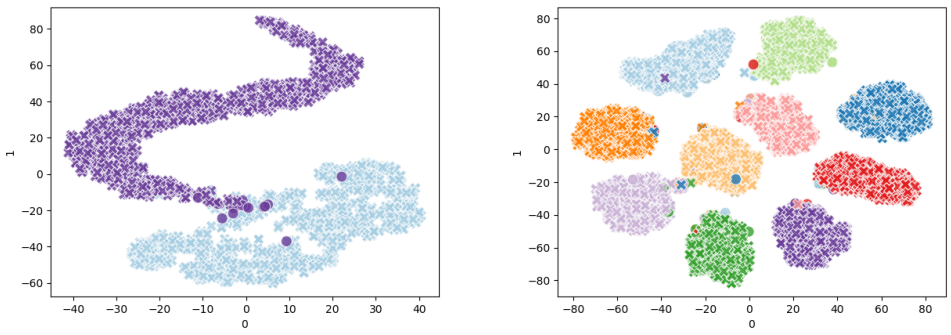


Figure 4.1: T-SNE representation of test data or Galaxy Zoo2 (left) and MNIST (right) from the last hidden layer. Each data point represents an input image from the test set. Data point colors represent the true labels.

neighbors, such as data points from the same class, to have similar representations. Leveraging this lower dimensional representation learned by the model has the dual benefit of reducing dimensionality of the data and focusing on the representation learned by the model. Since the objective of the workload is to diagnose this model, we hypothesize that leveraging the learned latent space to select a sample will be key to understanding what the model has learned. For model diagnosis we view the training, and test data points in the **latent space**, i.e., instead of viewing the data in the high dimensional original format of images, audio or text, we utilize this lower dimensional representation of the data learned by the model's last hidden layer to create samples.

Our goal is to diagnose the model, which implies that a subset of the queries will focus on what the model got wrong, as shown in Table 4.2. In a classification problem with multiple classes, the decision boundaries partitions the underlying vector space into multiple regions, one for each class. Decision boundaries are where the output label of a classifier is ambiguous, i.e., where errors and mis-classifications occur. The diagnosis of a DL model requires exploration of the **decision boundary** for a model [66, 197].

For instance, Figure 4.1 depicts this lower dimensional representation for two datasets we use for evaluation, MNIST [121] and Galaxy Zoo2 [67]. We use a dimensionality reduction technique, t-Distributed Stochastic Neighbor Embedding (t-SNE) [186], to reduce dimensions of this data to

visualize it in two dimensions. In Figure 4.1, each point represents an image from the test set, and colors indicate the true class labels. We make two observations from this visualization: (1) the data representations in latent space show separation for each class, and (2) most mis-classified instances are on the edges of data points groupings.

Thus, it is at the decision boundary, we find most of the incorrectly classified data points examples and the diverse correctly classified data items. Farthest from the boundary we find the correctly classified data instances, where their learned lower dimensional representation are very similar.

Our approach is based on utilizing the lower dimensional representation when selecting data items for our sample, and focusing on decision boundaries in the *latent space* when selecting the data points to include in our sample.

4.3.1 Baselines

In the AQP literature the two most popular sampling techniques are uniform sampling and stratified sampling. We consider both of these sampling techniques as baselines. Uniform sampling can directly be applied to our problem without changes. Stratified sampling partitions data into groups called strata and computes a separate sample for each stratum.

In database systems such as BlinkDB[8], strata are defined over a subset of columns that typically correspond to categorical valued attributes, e.g. `city`. For DL model diagnosis, the underlying data can be considered a relation, with each row representing a data item (e.g., one image) and each column a value of interest, such as the activation value for a neuron in the model. Each row can be extended with metadata, such as the predicted class and the class label. To implement stratified sampling for our use case we propose *stratified by CM* as a baseline technique. Because of the large dimensionality of the data, we cannot create a stratum for each possible value in each column, instead we use the classification result or the metadata associate with each data item to create the strata. Each data point is classified by the model as belonging to a class. This result is encapsulated in a *confusion matrix* (a.k.a. the error matrix), which tabulates the performance of a classification algorithm. For a binary classifier, the confusion matrix counts the number of true

positives, false positives, true negatives, and false negatives. For multiple labels, the confusion matrix generalizes this concept. Each row of the matrix represents a predicted class, while each column represents a true class. In this technique, we sample based on cells in the confusion matrix. We call this technique *stratified by confusion matrix (CM)*.

Additionally, since we utilize the latent space for creating the sample we also consider a baseline which selects a sample based on naively sampling from the latent space. This baseline covers the n-dimensional latent space by creating grid in this space and then sample from each each partition. Our goal for this baseline is to ensure our sample contains instances of data that lay in different regions of that latent space. However, the latent space we choose is high dimensional, e.g., for the MNIST dataset, the latent space is $84D$. Even if we divide each dimension into two buckets, we get a total of $2^{84} \sim 1.93e + 25$ buckets. Instead, we reduce the dimensionality of data in the latent space for this analysis using PCA. We call this naive technique *simple latent space* sampling. For this sampling technique, we collect equal number of instances at random from each underlying grid.

In addition, we use two other techniques from the literature as baselines. First, we use visualization aware sampling (VAS) for large scale data visualization, such as scatter and map-plots. VAS is based on the interchange algorithm [141], which selects tuples that minimize a visualization-inspired loss function. Visualization-inspired loss is based on three common visualization goals: regression, density estimation and clustering. The interchange algorithm creates a sample that maximizes visual fidelity of the data at arbitrary zoom levels.

Second, we use explicable boundary (EB) trees [196] to create a single sample from input data. This method constructs a boundary tree to approximate the complicated deep neural network models with high fidelity. EB trees provide a single sample for a dataset and a model which explains the boundary between each class learned by the DL model.

We describe details of sample selection from baselines in Section 4.4.3.

4.3.2 Clustering in Latent Space

An important part of our approach to selecting a sample for DL model diagnosis is to ensure that model decision boundaries are represented in the sample. As described earlier, model decision boundaries are where the model makes mistakes and when diagnosing the model, we want to focus on this area of the latent space. To determine boundaries in latent space, we cluster data in latent space and fit a model to estimate the parameters for each class in that space. We do this in both supervised and unsupervised manner. When fitting a supervised model, we use the class labels. In the unsupervised case, we use parameterized models so we utilize the number of unique classes present.

In both supervised and unsupervised cases the models fitted to the latent space provide us with the likelihood that and object belongs to a class or cluster. For binary classification to determine whether an object belongs to class A or class B, let $P(A|x_i)$ be the likelihood that a data instance x_i belongs to class A. In this case, the points on the decision boundary of class A and class B are those for which the ratio $\frac{P(A|x_i)}{P(B|x_i)}$ is ≈ 1. A lower value of likelihood ratio would imply that $P(B|x_i) > P(A|x_i)$ in which case x_i would be assigned to cluster or class B. The higher the likelihood that an object belongs to class A, the higher the ratio $\frac{P(A|xi)}{P(B|x_i)}$ will be.

For a multi-class classifier, where a data point x_i may belong to classes $\subset a, b, c, \ldots$, this ratio would be, $\frac{P(A|x_i)}{\sum_{z \subset b,c,d,\ldots} P(Z|x_i)}$, or

$$\frac{P(A|x_i)}{P(\neg A|x_i)}$$

Our sampling technique clusters the data in the latent space, then sorts data in each cluster or class by the ratio of likelihood of belonging to that particular class. This sorted list thus consists of exemplars on the higher end and outliers on the lower end of the list. We utilize a tuning parameter j to determine the proportion of exemplars and outliers in our sample. We select $j\%$ from the outliers and $1 - j\%$ from the exemplars. Algorithm 1 describes this approach in further detail.

For the unsupervised technique, we utilize a parameterized clustering technique, the Gaussian Mixture Model (GMM). These models offer a probabilistic way to represent normally distributed sub-populations within an overall population. We set the number of clusters in GMM to be equal

Algorithm 1: Algorithm for selecting sample.

Data: input data in latent space, f,k, j
// k num class labels, f is sample size
$Clusters \leftarrow None$
$sample \leftarrow None$
Clusters = ClusterAndSortData(data,k)
foreach $cluster_i$ *in Clusters* **do**
    $s1 \leftarrow data.head(f * j)$
    $s2 \leftarrow data.tail(f * (1 - j))$
    $sample \leftarrow s1 + s2 + sample$
end
return sample

to the number of unique classes in the dataset.

For the supervised technique, we use max-margin classifiers to classify the data in the latent space. Margin classifiers are a class of supervised classification algorithms that utilize distance from the decision boundary to bound the classifier's generalization of error. Support vector machine (SVM) [173] is an example of this category of classifiers, which learns boundaries based on labels so that the examples of the separate classes are divided by a clear gap that is as wide as possible. SVMs utilize kernel functions [95]; these help to projecting data to a higher dimensional space where points can be linearly separated. DL models do not have non-linear activation functions after the last hidden layer, so the latent representation from last the hidden layer should enable discovery of linear boundaries. Thus, we utilize a linear kernel for SVM [56], which has the dual advantage of being faster than non-linear kernels and less prone to over-fitting. Results of the classifier are turned into a probability distribution over classes by using Platt scaling [148, 198]. These probabilities are used to sort the data items in each cluster or predicted class and then select a sample.

4.4 Evaluation

In this section we empirically evaluate our sampling techniques which is based on the hypothesis that sampling evenly from the latent space is not sufficient; model decision boundaries are the most important region of this latent space for answering model diagnosis queries; and they must be well represented in a reliable sample.

We evaluated our sampling techniques from Section 4.3 on two different datasets. We first describe metrics we used to evaluate query sets defined in Section 4.2 and datasets and DL models we used for experimental evaluation. We then describe the experiments we conducted and results of these experiments in Section 4.4.3, finally we present the analysis of these results in Section 4.4.4.

4.4.1 Metrics

In this section we define metrics for evaluating the three query sets defined in Section 4.2. **Query set S1** retrieves the set of top-K maximally activated neurons. To measure how well our sample performs we use *precision* as the metric. Precision is the fraction of top-k results from the sample that belong to the true top-k result. Precision lies between $[0, 1]$. A precision value of 0 implies that the sample top-k does not contain any of the full data top-k neurons.

Query set S2 retrieves the average value of neurons. This is a high dimension vector of floating points, where dimension is the number of neurons in a layer. Additionally, this is a sparse vector, i.e., many neurons may have zero average activation because of non-linear activation like ReLU. We used cosine distance [43] to measure the distance between the average vector for the entire dataset and the average vector from sample due to the properties of high dimensionality and sparseness of the average neuron vectors, which lies between $[0, 1]$. Cosine distance between two vectors A and B is defined as:

$$1 - \frac{A.B}{\|A\|\|B\|}$$

Query set S3 retrieves the distribution of maximally activated neurons. As this is a true distribution an obvious metric would be Kullback-Leibler (KL) divergence [102]. However, we encounter two issues with using this metric. First, KL divergence is unbounded, which means it is not a

true metric, and it is difficult to assess how close two distributions were. Second, KL divergence is defined only on distributions with non-zero entries. This is not true for maximally activated neuron distribution, which may have neurons with zero counts. Thus, we used Jensen-Shannon divergence [89] instead, which is based on KL divergence. Jensen-shannon divergence is both symmetric and finite valued. Jensen-Shannon distance is squareroot of Jensen-Shannon divergence which is defined as:

$$\sqrt{\frac{D(p\|m) + D(q\|m)}{2}}$$

where p and q are the two distributions being measured, operator D represents KL divergence, and m is there mixture distribution. It is a true metric and lies in $[0, 1]$.

4.4.2 Datasets and Models

We evaluated our sampling techniques on two datasets, Galaxy Zoo2 [67] and MNIST [121]. For each dataset, we built and evaluated one deep learning model. The MNIST dataset consists of 28×28 pixel gray-scale images of handwritten numerical digits with a training and test set of 60K and 10K images, respectively. We trained the six layer neural network depicted in Figure 4.2. This model is a based on LeNet-5 [103] for classifying MNIST dataset with added batch-normalization after every layer.

Galaxy Zoo2 [67] is a public catalog of $\sim 240,000$ galaxies from the Sloan Digital Sky Survey [158] with classifications from citizen scientists. The Galaxy Zoo decision tree [47] lists the questions answered by citizen scientists. We took a subset of this dataset to classify images that appear edge-on vs face-on, i.e. which of the telescope images show the celestial object facing the telescope or 'face-on' vs. as seen from a side or 'edge-on' (question T01 in [47]). The training and test datasets consist of $54,333$ and 2118 images, respectively, each a 69×69 color image. We trained a model depicted in Figure 4.3, which is a variation of the model from [50]. In our variation of this model, we reduced the number of dropout layers and added batch normalization after every convolutional layer. We achieved 99% accuracy on the test set and an overall weighted F1 score of 0.99.

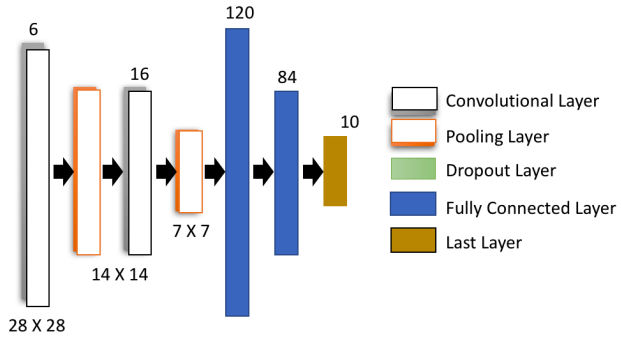


Figure 4.2: Deep learning model for MNIST dataset.

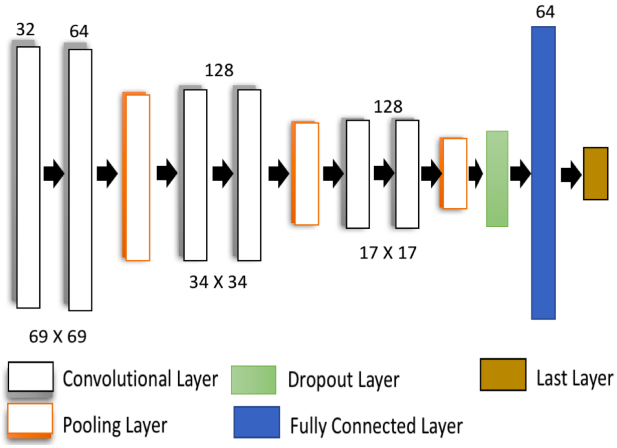


Figure 4.3: Deep learning model for galaxy Zoo2 dataset.

We trained the models and extracted activations from them using the TensorFlow [177] library. For both models, we used the representation from the last hidden layer to drive our sampling technique, and the last hidden layer was a fully connected (FC) layer. The MNIST data representation is from layer FC-2 (Figure 4.2) with 84 neurons. The Galaxy Zoo2 data representation is from layer FC-1 (Figure 4.3) with 64 neurons.

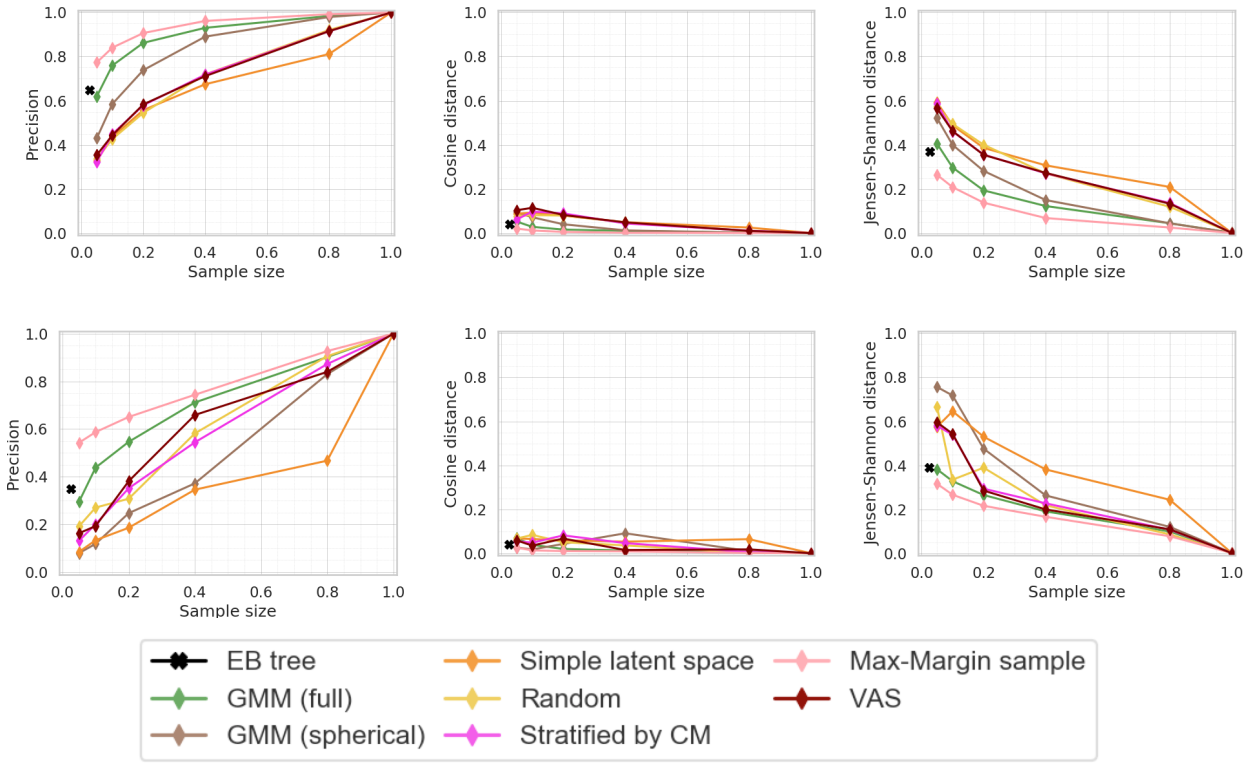


Figure 4.4: Performance of query set S1, S2 and, S3 for increasing sample size for various sampling strategies. Top row MNIST, bottom row Galazy Zoo2. From left to right columns, S1, S2, and S3. The X-axis shows the sample size as a fraction of the entire dataset.

4.4.3 Experiments

For our **first experiment**, we evaluated the three query sets on the two datasets using the metrics described above. For each dataset, we created samples of size 5%, 10%, 20%, 40% and 80% for the eight sampling techniques we are evaluating. Our rationale for choosing sampling techniques is described in Section 4.3, here, we provide a brief description of each technique:

(1) *Random* sampling draws a sample from the dataset uniformly at random without replacement. (2) *Stratified by CM* sampling contains a sample with data items drawn from each cell of the confusion matrix in proportion to the number of data items in the cell. For instance a 5% sample select uniformly at random, 5% of the data items from each cell in the CM. (3 and 4) *Visually aware sampling (VAS)* and *Explicable Boundary (EB) tree* sampling utilize the techniques

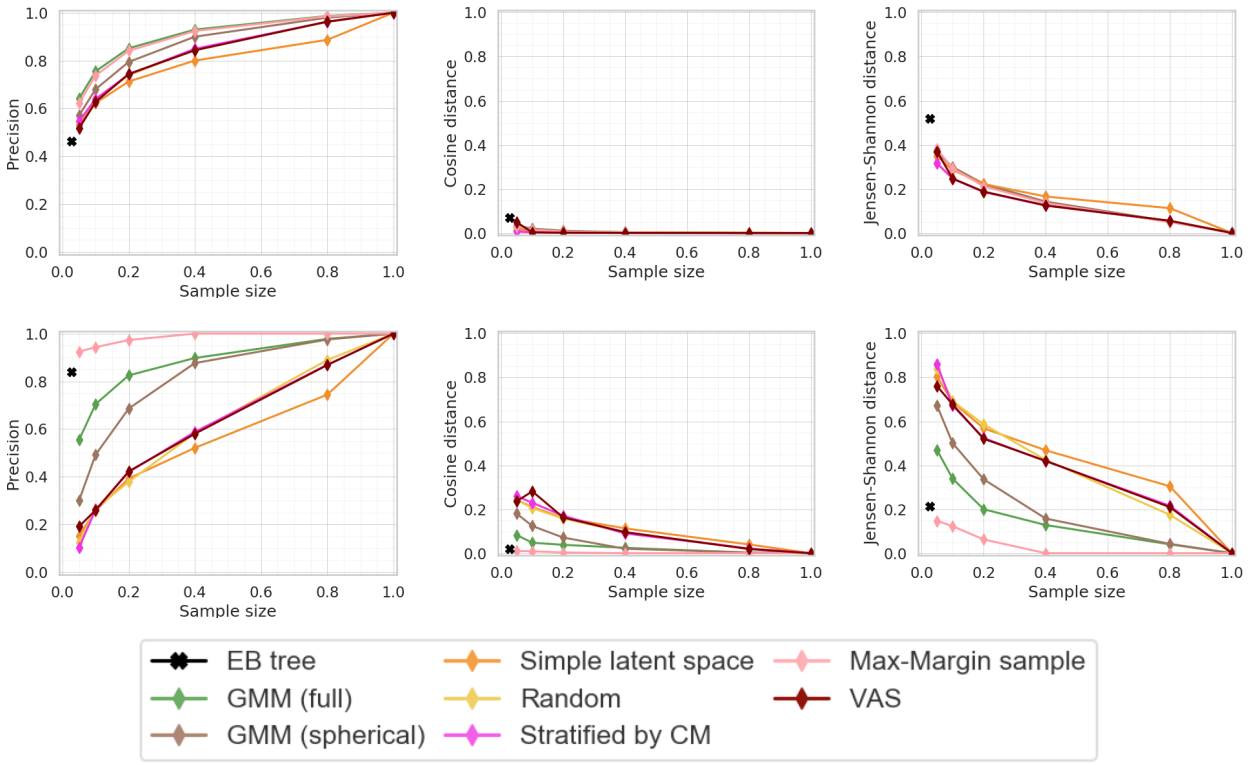


Figure 4.5: Performance of query set S1, S2 and, S3 for increasing sample size for various sampling strategies for MNIST dataset. Top row shows the results for correctly classified data items and the bottom row shows results for incorrectly classified items. From left to right columns, S1, S2, and S3. The X-axis shows the sample size as a fraction of the entire dataset.

presented in [141] and [197] respectively. (5) *Simple latent space* sampling divides the latent space into a grid and then samples equally from each cell in the grid. (6 and 7) *GMM* sampling fits a GMM to the data points in latent space. For each of the resulting clusters, data points belonging to each cluster are sorted by the likelihood ratio $\frac{P(A|x)}{P(\neg A|x)}$ of belonging to that cluster. The sample is then created by selecting data points from the two ends of this list for each cluster, with a tuning factor j determining what fraction is selected from either end. We have two GMM samples since we evaluated impact of two types of co-variance matrices, spherical and full. Finally, (8) *MaxMargin classification* sampling classifies the data points in the latent space with a max margin classifier, sorting points in each class by the ratio of their likelihood belonging to that class, and

choosing from the two ends of this list with a tuning factor of j, like the GMM samples.

For the first experiment, we fixed the tuning factor j to 0.7 for GMM and MaxMargin samples. We studied the impact of this tuning factor in the third experiment. The EB tree technique creates a single sample since there is a single boundary tree for a model and corresponding data. Figure 4.4 shows the results of this experiment for both datasets. As we increased sample size the query set results got increasingly more accurate until, at fraction 1.0 or on the full dataset, the metrics for all sampling techniques were coincident at 1.0 for S1 and 0 for S2 and S3.

For *simple latent space* sampling we reduced the dimensionality of latent space from 84 and 64 to five for both MNIST and Galaxy Zoo2 and then divided each dimension into 2 bins, resulting in 2^5 or 32 bins. We then sampled equally from each bin. This is the only technique where we sampled equally rather than sample in proportion to the number of items in the bin. We did this in order to evaluate the impact of sampling from the latent space. Interestingly, this technique did not do well on all three query sets. To minimize the impact of randomness, we selected each sample ten times and evaluated it and average results from these ten iterations. As we can see from Figure 4.4, the *simple latent space* sample behaved as well as the random sample. While this sample provided adequate results on S2, giving on average less than 10% error, its performance on S1 and S3 was not adequate. The knee seen for this sample (at 80% of the dataset) occurred because at this point the sample had the fewest number of data items compared to other samples: data were unevenly distributed in the latent space, and we sampled equally from each bin rather than in proportion to the size of the bin.

The *stratified by CM* sample performed much better than both the *random* and *simple latent space* samples for S1 and S3. *VAS* did as well as *stratified by CM*, this is of note because the VAS sample had no knowledge of classification of each data points and was trying to minimize a visualization-based loss function, which is trying to ensure that that the sample replicated the data density of the original distribution. All three clustering-based samples *GMM (full)*, *GMM (spherical)* and *MaxMargin classification* based samples did better than the baseline samples on all three query sets in most cases. *GMM (full)* did better than *GMM (spherical)* for both datasets. *GMM (full)* fit the data better, as expected, and thus did better on selecting exemplars and outliers

when compared to *GMM (spherical)*. The goodness of fit is dependent on the dataset. *GMM (spherical)* does better than *stratified by CM* for the MNIST dataset but worse for the Galaxy Zoo2 dataset. From the two dimensional representation of the data in latent space for the two datasets in Figure 4.1 we can see a separation between the ten clusters in MNIST, while the two clusters in the Galaxy Zoo2 dataset were not clearly separated. Additionally, for MNIST each cluster appeared to be somewhat symmetrical, but the two clusters for the Galaxy Zoo2 dataset did not have a clear separation, and one of the clusters is highly asymmetrical. *GMM (spherical)* with a isotropic co-variance matrix has difficulty fitting the Galaxy Zoo2 dataset. *GMM (full)* fit a more complex gaussian to each cluster, and this, in turn, provided a much better estimation of outliers vs exemplars. This difference can be seen in two datasets. While *GMM (full)* sample did better than *GMM (spherical)* sample for both datasets, the difference in performance was higher for when the underlying data distribution assumptions were not met for *GMM (spherical)*. *MaxMargin classifier* based sampling performed the best on all three query sets. Additionally, Figure 4.5 shows the results for all from this experiment on the three query sets separated by correctly(top row) and incorrectly(bottom row) classified items, here we can see that while *MaxMargin classifier* performs well on correctly classified data items, it performs exceptionally well on the incorrrectly classified data items. We delve into this further in Section 4.4.4.

Finally, *EB tree* technique provided a single sample since there was only one boundary for model. As expected, it did well picking the outliers and therefore performed well on both S1 and S3. For both datasets, EB-tree based sample was the smallest and performed second best on these two query sets. However, as the EB tree sample focused inordinately on the outliers, it did not perform as well on S2. On the well-separated latent space for the MNIST dataset the *EB tree* performed on par with other sampling techniques. However, for the Galaxy Zoo2 dataset, it did not perform as well. The *MaxMargin classification* based sampling performs better than EB-tree sample for all three queries for both datasets.

We analyze further analyze the results for this experiment in Section 4.4.4 and delve into why *MaxMargin classification* based sampling outperforms stratified by CM and other baselines.

In the second experiment, we examined the impact of varying the number of top-k neurons in

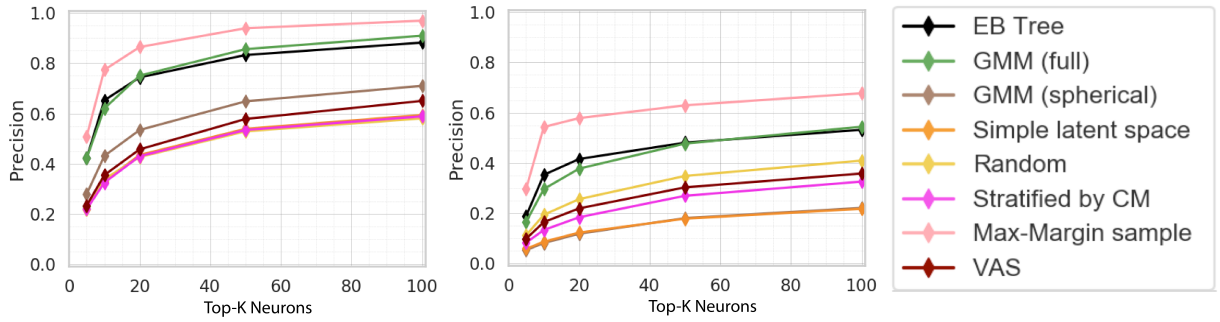


Figure 4.6: Precision for query set S1, number of top-k neurons (x-axis) in the 5% sample. Left panel MNIST, right panel Galaxy Zoo2.

S1 and measured the precision achieved by each of the eight sampling techniques. We show the results for a 5% sample, with k equal to 5, 10, 20, 50, 100. Results for this experiment are shown in Figure 4.6.

For the MNIST dataset, a 5% sample had a precision 0.98 for the top-100 neurons. However for the Galaxy Zoo2 dataset this number was much lower, at 0.70 for the top-100 neurons. This is due to two factors: (1) the test dataset, over which we evaluated this query for MNIST was 10k while for the Galaxy Zoo2 dataset it is 2k. A 5% sample was 500 data items for MNIST and 105 items for Galaxy Zoo2 dataset. (2) the model for MNIST had 107,786 parameters or neurons, and Galaxy Zoo2 had an order more parameters at 1,095,842. Thus, a 5% sample for the Galaxy Zoo2 dataset was both smaller and trying to capture a more complex model. This is confirmed by an additional experiment, where we increase the Galaxy Zoo2 sample size to 500 elements, we get 85% coverage on the top-100 neurons.

For both datasets *MaxMargin classification* sampling had the highest precision. *EB tree* was next for both datasets. This is not surprising because EB tree focuses on decision boundaries. This reinforces our hypothesis that decision boundaries need to be well represented for a sample to perform well on model diagnosis queries.

In the third experiment, we evaluated the impact of tuning factor j for three clustering samples *GMM (full), GMM (spherical), MaxMargin*. Tuning factor j is a number between 0 and 1 and is

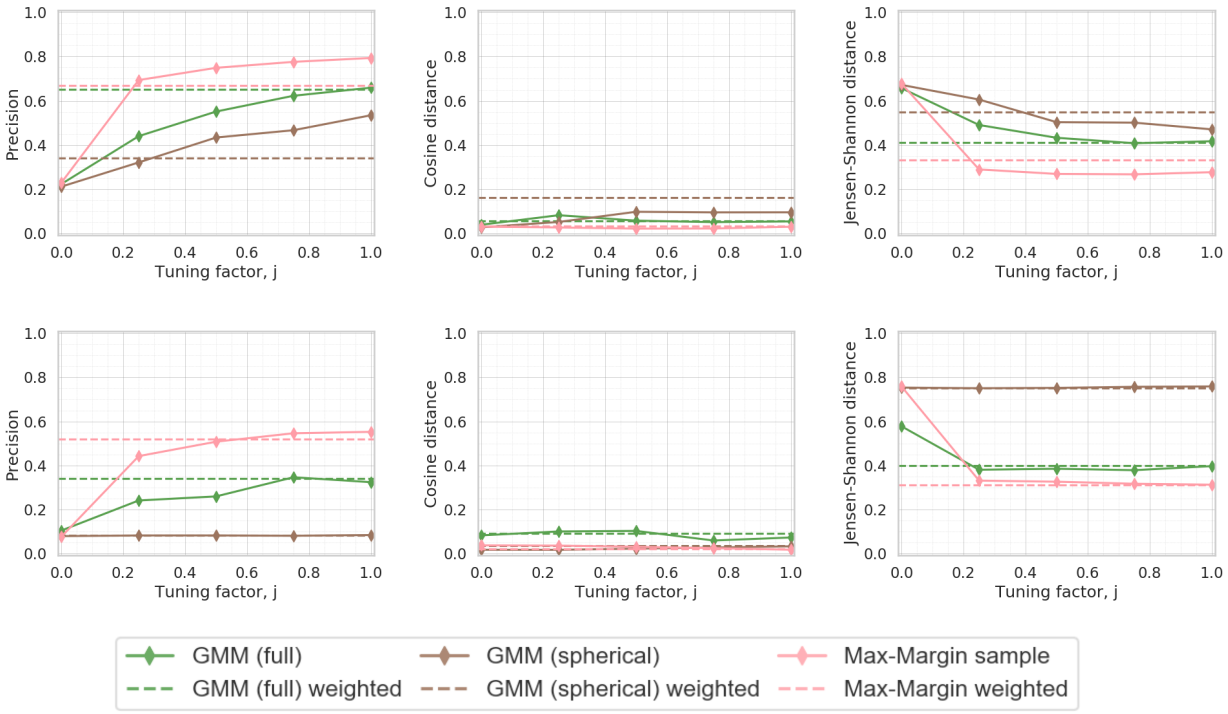


Figure 4.7: Impact of tuning factor j (x-axis) on metrics for MaxMargin and GMM based sampling strategies. From top row MNIST, bottom row Galazy Zoo2, from left to right columns S1, S2, and S3.

used to determine how many data points in the samples come from the lowest values of likelihood ratio or outliers. We evaluated the impact of this tuning factor on a 5% sample for all three sampling strategies. We evaluate query sets S1, S2 and S3 on tuning factor values of 0, 0.25, 0.50, 0.75 and 1.00. In all three sampling strategies, we picked data items in order from the sorted list for each cluster. Our sample is selected by selecting items from both ends of the sorted list and picking $frac * j$ items from the head or outlier end of the list and, $frac * (1 - j)$ from the exemplar end of the list. Thus, for the tuning factor value of 0, all data instances in the sample are picked from the exemplar end of the list and for a tuning factor value of 1 all data instances were picked from the outlier end of the list. In this experiment, we additionally created a weighted sample, where the weight was simply the reciprocal of the likelihood ratio. Likelihood ratio can be unbounded for exemplars, therefore for purposes of numerical stability we selected a threshold. To reduce

the impact of random selection, we selected a weighted sample ten times and reported the average value. Figure 4.7 shows the results of this experiment. For S1, when the dataset contained only exemplars at tuning factor 0, precision was the lowest for the sample. Precision grew as the value of tuning factor increased and plateaued at tuning factor ~ 0.7. The max top-10 precision for the MNIST dataset was 0.8 and galaxy Zoo2 is 0.57. This was the max value for top-10 that can be achieved on a 5% sample with the three sampling techniques for either dataset. For S2, the average activation value was not impacted as much by the tuning factor. The difference was small enough not to significantly impact the value of this metric. For S3, we saw results similar to S1. The highest values were at $j = 0$, because at this point there was the least amount of diversity in the data points; each cluster only contributed exemplar data points. As the number of outliers increased, the distance between the distribution became lower, the lowest point around $j \sim 0.5$, as the tuning factor increased further and the sample contains an increasing number of outliers this value became lower at a slower rate. Weighted sample values are indicated by dashed lines on the Figure 4.7. The weighted samples did not achieve the best value for any of the queries for either dataset. This indicates that picking the data items based on the likelihood ratio directly provides better results on rather than relying on selecting a sample weighted by the likelihood ratio.

Sampling Technique	**MNIST**	**GZoo2**
Entire Test set	9893(107)	2097(21)
Uniform	494(4)	105(1)
Latent space sample	487(9)	108(0)
Stratified by CM	494(11)	105(1)
Stratified Weighted	397(107)	85(21)
GMM (full) sample	464(48)	104(4)
GMM (sph) sample	500(11)	108(0)
Max margin sample	427(85)	89(18)
Visually Aware Sample	492(8)	92(4)
EB Tree sample	198(72)	44(8)

Table 4.4: Number of data points in samples for each sampling strategy with correctly classified (incorrectly classified) data points for 5% of the sample from the test set in both data sets.

4.4.4 *Performance Analysis*

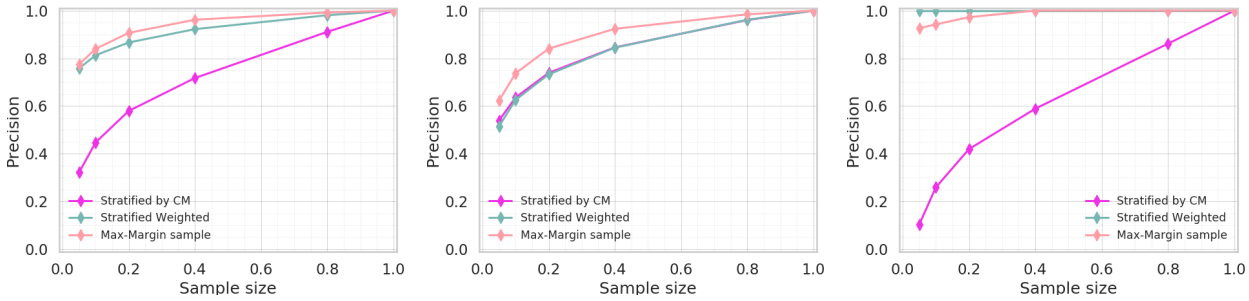


Figure 4.8: Precision for query set S1 for MNIST. From left: precision on the query set, metrics for a single query for correctly, and incorrectly classified data items.

As the three experiments in the previous section show, max-margin based sample performs better than the unsupervised clustering (GMM) based technique, which in turn performs better than the other baseline techniques. In this section we investigate why the *MaxMargin classification* based technique outperforms other sampling techniques.

Our first hypothesis is based on the observation that *MaxMargin classification* based sample consists of higher number of incorrectly classified data items when compared to all other sampling techniques. Table 4.4 shows the number of correctly and incorrectly classified data items in a 5% sample on for both datasets. A 5% sample, *MaxMargin classification* based sample has 79% and 85% of the incorrectly classified data items for the MNIST and GZoo2 datasets respectively. In comparison, *stratified by CM* sample has 10% of the incorrectly classified data items. We hypothesize that a sample with larger number of incorrectly classified data items outperforms samples with fewer incorrectly classified data items. To evaluate this hypothesis we create another sample *stratified weighted*, where we over-sample incorrectly classified data items. *Stratified weighted* sample is based on *stratified by CM* sample with one difference, instead of sampling uniformly from correctly and incorrectly classified data instances for each class, we pick 50% of the data items from incorrectly classified data instances and rest of the data items from the correctly classified data instances for each class. *Stratified weighted* and *MaxMargin classification* based samples

now have comparable number of incorrectly classified data items, while *stratified by cm* sample always exceeds the number of correctly classified data items when compared to *MaxMargin classification* based sample. Figure 4.8 illustrates the performance of *stratified weighted*, *stratified by CM* and *MaxMargin classification* based samples on query set S1 on MNIST dataset.

Examining results of this new sampling technique, we observe that *MaxMargin classification* based sample outperforms *stratified weighted* sample on the query set S1 (first column, Figure 4.8) despite having the same number of incorrectly classified data items. To understand this further we examine the performance of the three sampling techniques on a specific query for correctly and incorrectly classified data items (second and third column of Figure 4.8 respectively). Figure 4.8 shows results for correctly and incorrectly classified data items for MNIST dataset. This shows that while the *stratified weighted* and *MaxMargin classification* based samples have similar performance for incorrectly classified data items, for correctly classified data *MaxMargin classification* based sample outperforms *stratified weighted* sample. Additionally *MaxMargin classification* based sample outperforms *stratified by CM*, which has more correctly classified data items in the sample. This implies that compared to other sampling techniques *MaxMargin classification* based sample is able to capture the diversity of activation values for neurons, and overall outperforms both *stratified by CM* and *stratified weighted* samples. This is notable, because the *MaxMargin classification* sampling technique only utilizes the distance from the decision boundary to pick the sample rather than the actual classification result (whether the data item was correctly or incorrectly classified). Both of the models we examine are highly accurate, thus selecting 50% of the incorrect instances results in selecting all of the incorrectly classified data items. If the underlying models were not highly accurate, simply selecting a larger number of incorrectly classified data instances at random would perform similar to the graphs of correctly classified data items, where *MaxMargin classification* based sample outperforms the *stratified weighted* sample. It is possible that by carefully selecting a sample such that it consists of specific number of correct and incorrectly classified items for each class for models where the model accuracy was low, we could create a sample that performs at par with the *MaxMargin classification* based sample. However such a sample would need careful parameter selection for every DL model being diagnosed, the advantage of *MaxMar-*

gin classification based sample creation is that it enables the selection of sample automatically in a principled manner without tuning requirement for every DL model being diagnosed.

However, *MaxMargin classification* has a parameter, the tuning factor. We examine the impact of tuning factor on the performance on query sets in order to validate that selecting a larger proportion of the sample from the decision boundary results in superior performance of the *MaxMargin classification* based sample and determine if there is a suitable default value of this tunign factor. This analysis will enable selection of a default value of the tuning factor. We examine the impact of tuning factor on all three query sets. As previously described, *MaxMargin classification* based sample is created by selecting items from the two ends of the list of data items sorted by distance from the decision boundary. Tuning factor of 0.0 implies all of the items in the sample are from the largest distance to decision boundary end of the list and, tuning factor of 1.0 implies all of the items in the sample are from the smallest distance to the boundary end of the list. Finally, tuning factor 0.5 implies half of the items in the sample are farthest from the boundary and half from closest to the boundary. Figure 4.7 shows the impact of tuning factor on all three query sets on both datasets, and Figure 4.10 shows the impact of tuning factor on single query (on a single layer and single class) for correctly (top row) and incorrectly (bottom row) classified data items.

Figure 4.7 shows that sample based on items closest to the boundary (tuning factor:1.0), outperform the sample based on items farthest (tuning factor:0.0) from the boundary for all the query sets on both datasets. Notably, this is true for all three query sets. The detailed query results in Figure 4.10 show the impact of tuning factor on a single query (on one layer, for one class) from each of the three query sets on the MNIST dataset. Figure 4.10 shows that a tuning factor of 0.0 results in very poor performance on incorrectly classified data items (bottom row), and tuning factors 1.0 and 0.5 perform similarly for incorrectly classified data items. Additionally, all three queries for correctly classified data items (top row, Figure 4.10) also show lower performance of sample with tuning factor 0.0. For the correctly classified data items top-k query tuning factor of 1.0 provides the best performance. However, for the average neuron activation values query with tuning factor of 0.75 demonstrates a performance improvement between 22% -68% over tuning factor:0.0 and an improvement between 20% and 63.3% over tuning factor:1.0. Similarly, the maximally activated

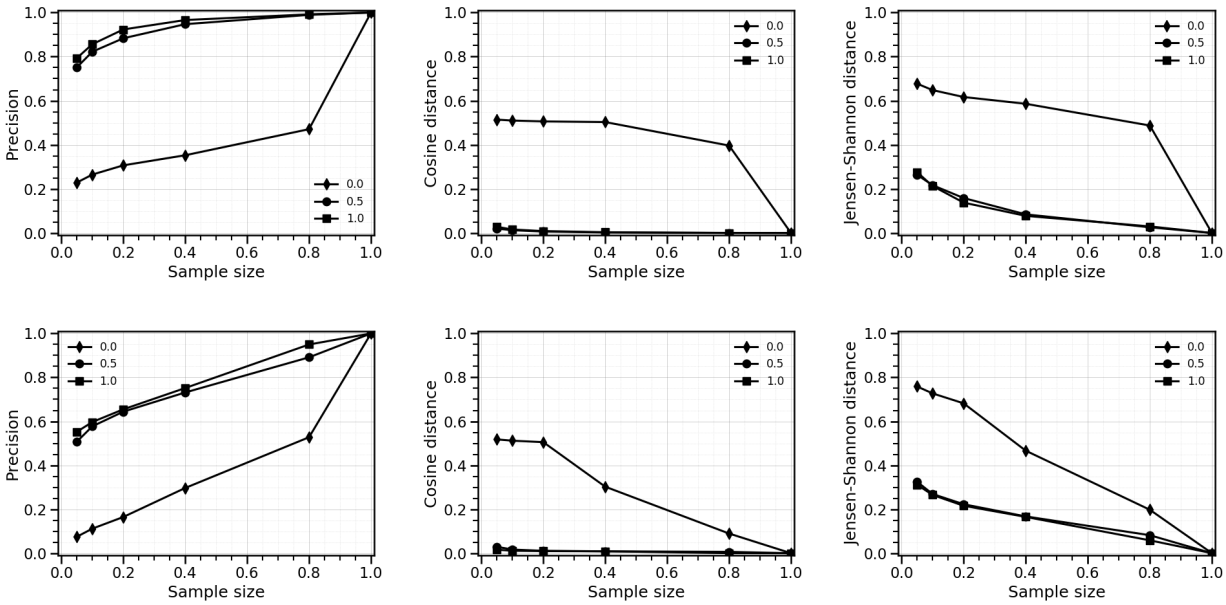


Figure 4.9: Impact of tuning factor j on metrics for Max-Margin sample for S1, S2 and S3, by increasing sample size for both datasets. MNIST:top row; Galaxy Zoo2: bottom row.

neuron distribution query with a tuning factor of 0.75 demonstrates a performance improvements between 2% to 45% over tuning factor:0 and, an improvement of between 5% to 14% over tuning factor:1.0.

This supports the hypothesis that selecting the sample from data items closest to the decision boundary captures the diversity of the neuron activation values better than selecting the sample with data items farthest from the decision boundary. We can also see that choice of tuning factor would be influenced by distribution of data items in the latent space. In the two dimensional (t-SNE) projection of the data in Figure 4.1 we can see that data items from each class are form compact clusters with clear separation between different clusters for MNIST when compared to galaxy zoo2 data. Thus, the sample farthest from the decision boundary would be a lot more homogeneous, i.e.have data items with similar important neurons for MNIST dataset rather than for the Galaxy Zoo2 dataset. Figure 4.7 supports this, the difference in the sample performance for tuning factor 0.0 and 1.0 is smaller for galaxy zoo2 dataset when compared to MNIST. Addition-

ally, the performance of samples with tuning factor 0.0 and 1.0 converge at lower sample sizes for the galaxy zoo2 dataset when compared to MNIST. An overall of tuning factor between 0.5 and 1.0 performs best, an 0.7 is a suitable default value.

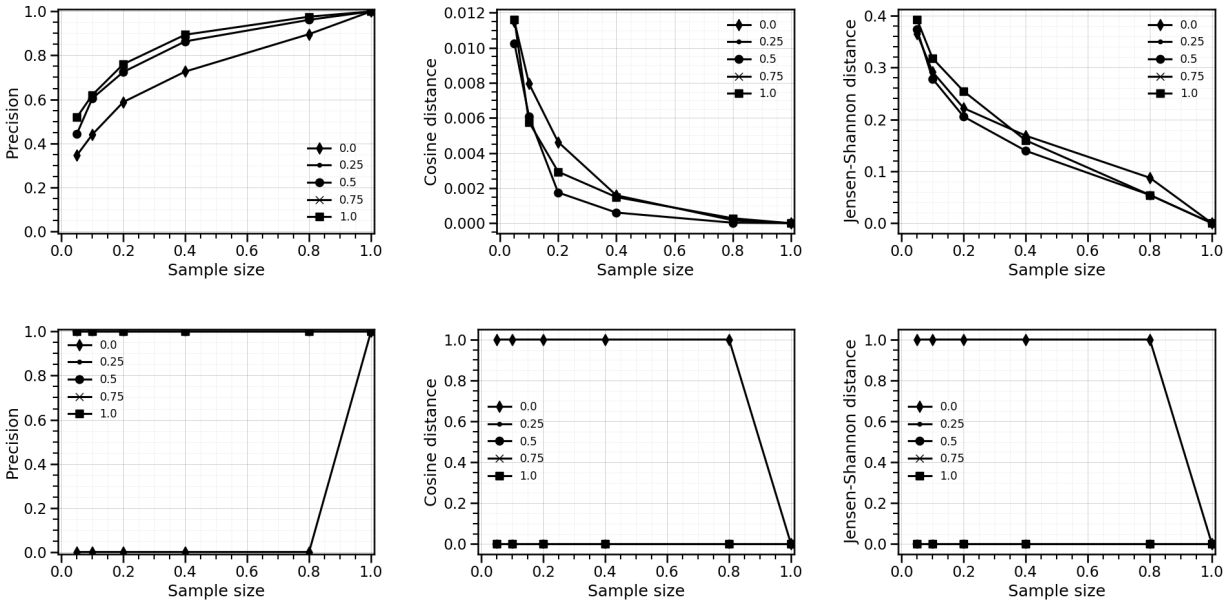


Figure 4.10: Impact of tuning factor j on metrics for Max-Margin sample for single query from query sets S1, S2 and S3 for on specific correct (top row) and incorrect (bottom row) classified data items on MNIST dataset.

4.4.5 Query timing and Sample Creation Overhead

In this section we evaluate the query execution time (from python) as well as the time to create the sample. We measure query execution time for a subset of queries from Table 4.2. Specifically we evaluate queries Q1 through Q5.

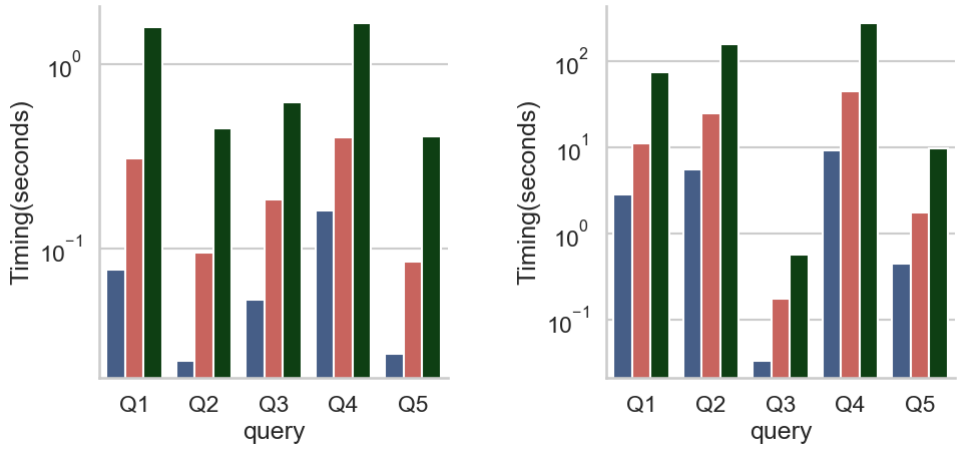
In Figure 4.11 we show the execution time in seconds for these queries on all both data sets for 5%, 20% sample, and the full training dataset. While the query runtime trends seem very disparate, the improvements in runtime for both models are consistent. In the 20% sample we see $3.36 - 6.8\times$ improvement in query runtime, and for the 5% sample we see $10 - 30\times$ improvement

in runtime. Q1 and Q2 have the longest runtimes on the full data set, as conv2, the layer for these two queries is the second convolutional layer in the models. The size of activations for the layer conv2, is $752.6MB$ for MNIST and $62GB$ for Galaxy Zoo2 on the full training data. While Q1 is looking for average activation for each neuron, Q2 is looking for average activation for incorrectly classified data points. To compute the results for Q2, we first join class label and predicted values with conv2 activations, filter to retrieve the incorrectly classified data points, and then compute the average for this group. For Q1, we need to load the activations for conv2 and compute average activations. Thus, when the data is large, join and filter operations required for Q2 make it take longer than Q1. This can be seen in the longer runtime for Q2 when compared Q1 for Galaxy Zoo2. Q4 computes the similarity between the logits. Logits are the un-normalized activations from the last layer in the model. For both data sets the time for the similarity computation is similar, ~ 0.05 seconds, rest of the time for the query is spend in loading activations. This time is proportional to size of the data, which is $752.6MB$ for MNIST and $8.GB$ for Galaxy Zoo2 for the entire training set.

Q5 is the fastest running query for MNIST and CIFAR-10 datasets. Q5 operates on the activations of last convolutional layer. Q3 operates on the activations last hidden layer which is *smaller* in size than the last convolutional layer, but Q3 takes longer to run than Q5 for these two datasets. This is because in addition to reading the data in Q3, we need to join the activations with class labels and predicted values, filter the result, and aggregate.

Q3 takes similar time for all three datasets. This is because the size of last hidden layer for the Galaxy Zoo2 data set is in similar range to MNIST as the size of this layer depends on the number of neurons and number of data points. Predictably, Q5 takes longer for Galaxy Zoo2 as the size of activations from the last convolutional layer is $10\times$ for MNIST. Above results demonstrate that running queries on a sample instead of the full data set results a savings of $3\times$ to $6.8\times$ for the 20% sample and between $10\times$ to $30\times$ on a 5% sample.

The queries above required all of the activations to be pre-generated. Thus, next we examine storage cost associated with storing and querying samples instead of the full data set. Unlike other sampling databases such as BlinkDB, our sampling technique offers storage savings in addition to



(a) MNIST (b) Galaxy Zoo2

Figure 4.11: Query timing for Q1 - Q5 from Table 4.2 for the 5% sample, 20% sample and entire dataset for both datasets (y-axis uses log scale).

query execution time savings. This is because creation of samples requires the activation values from the last hidden layer. Each of the sampling techniques requires as input the entire data sets representation in model space and as output returns the IDs of the data points in sample. Thus, creating a sample does not need all of the activations for all of the layers in the model to be generated. We only need to generate activation values for the last hidden layer. Figure 4.12 shows the storage saving for entire data from the three data sets. As expected the saving from a 5% sample are $\sim 20\times$, and from 20% sample are $\sim 5\times$. As seen in Section 4.4 depending upon the tolerance for error a 5% sample may be adequate, thus ML practitioners can expect upto $20\times$ reduction in storage footprint with our sampling technique.

Finally, we examine the time it took to create these samples for baselines as well as for our sampling techniques. Figure 4.13 depicts the time required to generate a 5% sample for all sampling techniques on the Galaxy Zoo2 test dataset; note the log scale on the y-axis. Results for the MNIST dataset were similar and are not shown. The Galaxy Zoo2 test set had 2118 points, each point is a vector of size $[1, 64]$. Generating the uniform sample was the fastest as expected.

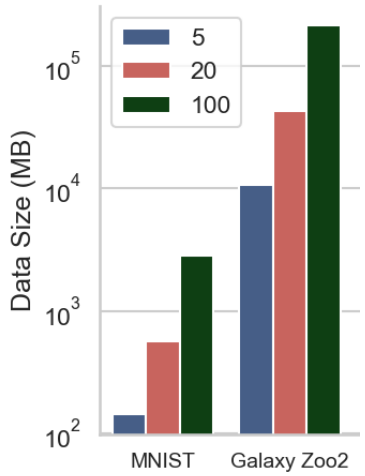


Figure 4.12: Data size for full, 20%, 5% samples for both data sets.

Generating, *stratified by CM* sample, and *GMM* samples, required similar time, taking less than a second. Generating *MaxMargin classification* based sample required 1.5 seconds. Both VAS and EB-tree samples took three orders of magnitude more time. VAS is created with the interchange algorithm [141], each point in the dataset has to be added and one point evicted, by comparing proximity of the added point with each element of the existing sample. This is $O(K_2N)$ where K is the sample size and N is number of points in the dataset. For large datasets as in cases of ML, the time to create this sample was unacceptably long. Boundary stitching algorithm [197] is $O(NK)$. This was faster than the VAS but still took longer than our sampling technique.

4.5 Summary

Deep learning models have become an indispensable tool for a wide range of image analysis. The task of diagnosis for deep learning models requires additional artifacts, such as activations. These additional artifacts must be generated, stored, and queried for each deep learning model being debugged. The addition of these artifacts, which can be up to three orders of magnitude larger than the input data size for each model being diagnosed, turns the process of building, diagnosing, and selecting DL models in to a large-scale data management challenge. Our sampling technique relies

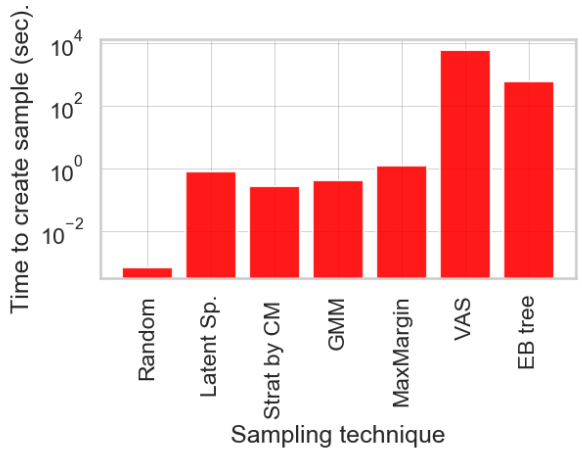


Figure 4.13: Time to create a 5% sample for Galaxy Zoo2 dataset for all sampling techniques.

on the lower dimensional representation of the data learned by the deep learning model and can be used to debug any deep learning model where a lower dimensional representation of the input data is learned in a supervised, semi-supervised or unsupervised manner. We demonstrate up to $20\times$ reduction in storage footprint and query time speed-up of up to $30\times$.

Chapter 5

RELATED WORK

Choice of sources can shield extreme bias behind a facade of objectivity.
–Noam Chomsky

In this chapter, we cover related work for the three previous chapters. We cover the literature in the following areas: image analytics benchmarks, automated glaucoma detection using machine learning, sampling, approximate query processing and deep learning model diagnosis.

5.1 Related Work on Comparative Evaluation of Big-Data Systems on Scientific Image Analytics Workloads

In comparative evaluation of Big-Data systems we evaluate the suitability of large-scale data systems and frameworks for scientific data analysis using two real-world scientific image data processing use cases. We evaluate five representative systems (SciDB, Myria, Spark, Dask, and TensorFlow) and find that each of them has shortcomings that complicate implementation or hurt performance. To evaluate these systems, we implement two representative end-to-end image analytics pipelines from astronomy and neuroscience. For the neuroscience use case we take as input diffusion MRI (dMRI) images of a human brain, which are cleaned, segmented and then used to build a model of brain connectivity. For the astronomy use case we take as input telescope images of the sky over multiple observations, which are cleaned, aligned and combined to create a comprehensive sky map.

Big-data systems

Several big data systems ([58, 86, 68, 152]) with similar capabilities to the ones we evaluated are available for large scale data analysis. We chose five big data systems for parallel data processing: a domain-specific DBMS for multidimensional array data (SciDB [156]); a general-purpose cluster computing library with persistence capabilities (Spark [167]); a traditional parallel general-purpose DBMS (Myria [70, 193]); and a general-purpose (Dask [154]) and domain-specific (TensorFlow [4]) parallel-programming library. We selected these systems because they have open-source implementations, can be deployed on commodity hardware, support complex analytics (such as linear algebra and user-defined functions), use the scientifically popular Python language for APIs [136], and have different internal architectures so we could evaluate the performance of different implementation paradigms. We considered Rasdaman [152], which is an array database with capabilities similar to SciDB, but were unable to make much progress as the community version does not support UDFs.

Image processing and DBMS benchmarks

Traditionally, image processing research has focused on effective indexing and querying of multimedia content [55, 32, 35]. These systems focus on utilizing image content to create indices using attributes like color, texture, shape of image objects, and regions and then specifying similarity measures for querying, joining, *etc*.

There have been many benchmarks proposed by the DBMS community over the years such as the Wisconsin Benchmark [25], Bucky [30], Linear Road [13], as well as the TPC benchmarks [182]. These benchmarks focus on traditional Business Intelligence computations, as epitomized by TPC-H and TPC-DS benchmarks, over structured data. The GenBase benchmark [176] takes this forward to focus on complex analytics besides data management tasks, but did not examine image data. Several recent papers [145, 115, 24, 163] evaluate the performance of Big Data systems, but the workload does not include image analysis. While prior work on raw files and scientific formats [10, 26] focuses on techniques for working with them directly, it does not offer

mechanisms to work with them in big data systems like the ones evaluated in this paper.

5.2 Related Work on Automated Glaucoma detection in multi-modal Images

In automated glaucoma detection, we present a new, comprehensive, and more accurate ML-based approach for population level glaucoma screening. In this work we build a multi-modal model on large data set that includes demographic systemic and ocular data as well as raw image data taken from color fundus photos (CFPs)macular Optical Coherence Tomography (OCT) scans. Although glaucoma is asymptomatic in its early stages, structural changes in the macula and RNFL precede the onset of clinically detectable vision loss [79]. Many studies have therefore attempted to automatically diagnose glaucoma using retinal imaging data. Most of these studies used either CFPs or features extracted from CFPs [27, 31, 161, 127, 37, 137, 147]. Other studies [126, 123] used features extracted from retinal B-scans obtained via OCT, a three-dimensional volumetric medical imaging technique used to image the retina. Macular OCT images are used to extract features such as thickness of the RNFL, ganglion cell-inner plexiform layer (GCIPL), or full macular thickness. Models evaluating changes in thickness of various retinal layers are promising since such changes, a direct result of tissue loss, are highly accurate disease predictors. However, thickness maps are derived automatically and, despite advances in OCT hardware and software, errors in segmenting retinal OCT images remain relatively common, with error estimates between 19.9% and 46.3% [114, 118, 16]. A study comparing a model built on raw macular OCT images with one built on thickness maps demonstrated that the former was significantly more accurate than the latter in detecting glaucoma [111].

5.3 Related Work on Sampling for Deep Learning Model Diagnosis

In sampling for deep learning model diagnosis we focus on the data management challenges facing ML practitioners who build deep learning models. Specifically, we present a new sampling-based approach for deep learning model diagnosis. The related work for sampling for model diagnosis is split in to three categories of research; (1) approximate query processing, (2) model diagnosis systems, and (3) model lifecycle management and tuning systems. We review work from each of

these categories below.

Approximate query processing (APQ) and top-K queries

([8, 6, 19, 75]) APQ is a well-studied area in databases and is an effective technique to deal with large-scale data. Algorithms for exact top-k queries are defined by the seminal work on the threshold algorithm (TA) [54], which require access to the indexed attribute(s) for a data set. Efficient processing of the top-k queries over samples is a challenging task [81]. Related work in this category includes top-k processing techniques that operate on deterministic data but report approximate answers in favor of performance. The approximate answers are usually associated with probabilistic guarantees; indicating how far they are from the exact answer. Algorithms presented in [180] are an approximate adaptation of TA where the approximate answers to the top-k query is associated with probabilistic guarantees. However, like TA this algorithm requires access to sorted attributes for the underlying data. Another approach to approximate top-k answers is considered in similarity search for multi-media databases [11]. This method uses a proximity measure to determine if a data region should be inspected. This utilizes the underlying data distribution rather than individual column value and in that sense is closer to our approach (i.e., instead of examining the underlying data, we utilize the latent space to create a sample).

Model diagnosis systems

([189, 117, 90, 12, 91]) Model tracker[12] is one of the earliest systems for model diagnosis. It diagnoses models by tracking its performance using statistical measures, such as accuracy, AUC, etc. and does not support model diagnosis for DL models. MLCube [91], one of the earlier visualization tool for model diagnosis visualizes data from pre-computed data cubes based on features from data and model results. The data-cubes utilized by this tool are based on less than 100 features and like Model tracker, it pre-dates the large scale of data that must be supported for DL model diagnosis. MISTIQUE [189] supports DL model diagnosis via examination of model activations, their primary approach is to reduce the storage footprint required by activations. MISTIQUE

shares our goals of reducing query runtime for model diagnosis, but it uses a different approach, quantization and de-duplication to reduce the storage. Modelhub [117] supports model diagnosis by storing learned models and training logs with an approach that reduces storage footprint. Modelhub focuses on different artifacts, learned models and training logs, which they store and retrieve efficiently by introducing a model versioning system and a domain-specific language for searching through model space, solving a very different problem. DeepBase [159] supports model interpretabiltiy and diagnosis by providing a declarative abstraction to express and execute the generation and comparison of these artifacts. DeepBase relies on the ability to encapsulate model interpretbility questions as hypothesis functions (e.g., parts of speech tags and image captions). DeepBase, ModelHub and MISTIQUE could benefit by leveraging our sampling techniques for their systems. Finally, a variety of visualization tools [90, 106, 169, 177, 199] utilize activations and gradients to interpret and diagnose DL models. All of these tools would benefit from our sampling techniques, as sampling would help reduce the scale of data required to support model diagnosis. Activis [90], for instance selectively pre-computes values for nodes of interest to save computation and storage. Sampling techniques such as ours will enable ML practitioners using tools such as Activis to avoid making such compromises.

Model lifecycle management and model tuning

([188, 120, 168, 53]) ModelDB [188] is a system for managing of ML models and pipelines. It provides versioning and metadata-based search and validation on models, simplifying the model building pipeline. MLflow [120] tracks experiments, packages the code to create reusable deployments and operationalizes the chosen models, addressing a very different aspect of model lifecycle management compared to ModelDB. However, neither of these systems help manage, store, or query any DL model diagnosis artifacts. While MLflow supports storing and tracking arbitrary artifacts in a framework and implementation agnostic manner, it does not utilize information such as representation learned by the models to help with the selection of appropriate model. In addition custom code has to be provided for generating and querying these artifacts in MLflow. These tools do not support model diagnosis or interpretability as a primary goal, if they were to adopt model

diagnosis as a goal our sampling technique could help with managing the size of data required.

Chapter 6

CONCLUSION AND FUTURE DIRECTIONS

Paradigm shifts arise when the dominant paradigm under which normal science operates is rendered incompatible with new phenomena, facilitating the adoption of a new theory or paradigm.
–Thomas Kuhn, The structure of scientific revolution

Scientific discoveries are driven by analyzing large volumes of data. Increasingly this data is in the form of images. Analytics over these images involves operations such as slicing, dicing, roll-ups, spatial joins and complex operations often expressed in higher language such as Python. Scientist utilize scripts written in Python, R, julia, bash scripts while the data is stored on local file system. Often this analysis is limited to what can fit in the local machine memory and systems support for large scale image analytics is still scarce.

In *Comparative evaluation of big data systems for large scale image analytics*, we evaluate existing big data systems for their suitability in supporting image analytics. In evaluating the big data systems we select two real life scientific use cases from neuroscience and astronomy and implement them in five (SciDB, Dask, Myria, Spark and tenserflow) existing large-scale data systems and frameworks. We evaluate these systems for ease of use, performance and scalability. All the experiments are run in public cloud environment with open source tools. We find the to better support image analytics in domain sciences, these systems need to simultaneously provide comprehensive support for multidimensional data and high performance for UDFs/UDAs written in popular languages (e.g., Python). Additionally, they need to completely automate data and compute distribution across a cluster and memory management to eliminate all possible sources

of out-of-memory failures. Our study raises a number of research questions. Image processing involves complex analytics, which include iterations and linear algebra operations that must be efficiently supported in big data systems. However, users typically have legacy code that performs sophisticated and difficult to rewrite operations. Therefore, they need the ability to call existing libraries. They also need an easy mechanism to parallelize the computation: they should be able to reason about multidimensional array data directly rather than manually creating and processing collections of image fragments. It should be easy to mix and match UDF/UDA computations and pre-defined (e.g., relational) operations on complex data such as image fragments.

Our study also re-iterates the general need to efficiently support pipelines with UDF/UDAs both during query execution and query optimization. Image analytics implies large tuples and larger tuples put pressure on memory management techniques, systems' ability to shuffle data efficiently, and efficient methods to pass large records back and forth between core computation and UDFs/UDAs. This provides another research opportunity. Finally, making big data systems usable for scientists requires systems to be self tuning, which is already an active research area [76].

Deep learning has enabled unprecedented breakthroughs in developing artificial intelligence systems which can analyze pixel data from images directly. Availability of large scale data, improvements in hardware like emergence of GPU based computing, availability of cloud compute resources, better algorithms and open-source deep learning libraries that are get easy to started with, have made deep learning powerful tool [45, 46] for data scientists and domain scientist working with image data.

In *Automated detection of glaucoma with interpretable machine learning using clinical data and multi-modal retinal images* we propose a new, comprehensive, and more accurate ML-based approach for population-level glaucoma screening. We use a publicly available data set [184] and build and evaluate multiple models on clinical and image data to detect glaucoma. Our final model is an ensemble tree based model that uses output from the image based deep learning models and clinical data. We compare the results of our model with assessments from five expert clinicians on a test set and on a cohort of subjects that have not yet been diagnosed with glaucoma (but go on

to develop glaucoma). These results show that our final model is highly accurate (AUC 0.97) and interpretable. Our model validates biological features known to be related to the glaucoma, such as age, intraocular pressure and optic disc morphology. It also points to previously unknown or disputed features,such as pulmonary capacity and retinal outer layers. Our approach of combining image and clinical data for the final results yielded not only very accurate detection, but it also enabled us to isolate and interpret critical variables that helped us draw clinical insights into the pathogenesis of the disease.

The process of developing the glaucoma detection model provided us with first hand experience of the data management challenges associated with developing and debugging interpretable deep learning models. Traditional methods of building machine learning models can be iterative and arduous. Scientists typically build from tens to hundreds of models before selecting one. This problem is exacerbated for deep learning models by large input data size and interpretability artifacts that need to be generated to build, diagnose, and interpret these models.

In *Sampling for deep learning model diagnosis* we propose a novel sampling technique to reduce the data management burden associated with building and diagnosing deep learning models. Existing sampling techniques, such as uniform random sampling and stratified sampling do not yield good results for deep learning model diagnosis. Our sampling technique relies on the insight that supervised deep learning models simultaneously learn a classifier and a low dimensional representation of the input data. The key idea underpinning our approach is to identify and target model decision boundaries to provide effective and efficient samples which can be utilized to debug and interpret these models. The sampling technique we present here focuses on sampling input data points, e.g. rows from the relation of data points. ML literature supports the notion of reducing the number of neurons for which activations need to be calculated [106, 113] and queried. We would like to explore this avenue in future work. The sampling technique described here works well with supervised learning models, i.e. deep learning models built with labeled data. In future work, we would like to explore our sampling technique and their efficacy for *unsupervised* DL models, such as generative models, auto regressive models, etc. [49] A large body of scientific data is unlabeled and requires unsupervised learning techniques, and extending our sampling technique

in this direction could be beneficial to the scientific community working on newer data sets.

The overall takeaway from this dissertation is that image analytics on scientific data is an important workload and presents unique data management challenges. Providing support for image data management and machine learning workloads will help domain scientist focus on scientific discovery.

BIBLIOGRAPHY

[1] http://dask.pydata.org.

[2] Glaucoma research foundation: Glaucoma facts and stats.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems. 2015.

[4] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. In *OSDI*, 2016.

[5] http://abcdstudy.org/.

[6] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD '99*.

[7] https://en.wikipedia.org/wiki/Activation_function.

[8] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys '13*, 2013.

[9] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. In *European Conference on Computer Vision*, pages 69–82. Springer, 2008.

[10] I. Alagiannis et al. Nodb: efficient query execution on raw data files. In *SIGMOD*, 2012.

[11] G. Amato, F. Rabitti, P. Savino, and P. Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM Transactions on Information Systems (TOIS)*, 21(2):192–227, 2003.

[12] S. Amershi, D. M. Chickering, S. M. Drucker, B. Lee, P. Y. Simard, and J. Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *CHI*, 2015.

[13] A. Arasu et al. Linear road: A stream data management benchmark. In *VLDB*, 2004.

[14] M. Armaly. Optic cup in normal and glaucomatous eyes. *Investigative Ophthalmology & Visual Science*, 9(6):425–429, 1970.

[15] M. F. Armaly, D. E. Krueger, L. Maunder, B. Becker, J. Hetherington, A. E. Kolker, R. Z. Levene, A. E. Maumenee, I. P. Pollack, and R. N. Shaffer. Biostatistical analysis of the collaborative glaucoma study: I. summary report of the risk factors for glaucomatous visual-field defects. *Archives of ophthalmology*, 98(12):2163–2171, 1980.

[16] S. Asrani, L. Essaid, B. D. Alder, and C. Santiago-Turla. Artifacts in spectral-domain optical coherence tomography measurements in glaucoma. *JAMA ophthalmology*, 132(4):396–402, 2014.

[17] https://www.arvo.org/.

[18] https://aws.amazon.com/.

[19] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD '03*, 2003.

[20] M. A. Badgeley, J. R. Zech, L. Oakden-Rayner, B. S. Glicksberg, M. Liu, W. Gale, M. V. McConnell, B. Percha, T. M. Snyder, and J. T. Dudley. Deep learning predicts hip fracture using confounding patient and healthcare variables. Nov. 2018.

[21] P. J. Basser et al. Estimation of the effective self-diffusion tensor from the NMR spin echo. *J. Magn. Reson. B*, 1994.

[22] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*, 2017.

[23] https://bazel.build/.

[24] M. Bertoni et al. Evaluating cloud frameworks on genomic applications. In *IEEE International Conference on Big Data*, 2015.

[25] D. Bitton et al. Benchmarking database systems A systematic approach. In *VLDB*, 1983.

[26] S. Blanas et al. Parallel data analysis directly on scientific file formats. In *SIGMOD*, 2014.

[27] R. Bock, J. Meier, L. G. Nyúl, J. Hornegger, and G. Michelson. Glaucoma risk index: automated glaucoma detection from color fundus images. *Medical image analysis*, 14(3):471–481, 2010.

[28] P. G. Brown. Overview of scidb: Large scale array storage, processing and analysis. In *SIGMOD*, 2010.

[29] E. R. Büchi. Cell death in the rat retina after a pressure-induced ischaemia-reperfusion insult: an electron microscopic study. i. ganglion cell layer and inner nuclear layer. *Experimental eye research*, 55(4):605–613, 1992.

[30] M. J. Carey et al. The BUCKY object-relational benchmark (experience paper). In *SIGMOD*, 1997.

[31] J. Carrillo, L. Bautista, J. Villamizar, J. Rueda, M. Sanchez, and D. rueda. Glaucoma detection using fundus images of the eye. In *2019 XXII Symposium on Image, Signal Processing and Artificial Vision (STSIVA)*, 2019.

[32] C. Carson et al. Blobworld: A system for region-based image indexing and retrieval. In *VISUAL*, 1999.

[33] A. Cassandra. Apache cassandra. *Website. Available online at http://planetcassandra.org/what-is-apache-cassandra*, 13, 2014.

[34] R. J. Casson, G. Chidlow, J. P. Wood, J. G. Crowston, and I. Goldberg. Definition of glaucoma: clinical and experimental concepts. *Clinical & experimental ophthalmology*, 40(4):341–349, 2012.

[35] S. Chaudhuri et al. Optimizing top-k selection queries over multimedia repositories. *IEEE Trans. Knowl. Data Eng.*, 2004.

[36] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[37] X. Chen, Y. Xu, D. W. K. Wong, T. Y. Wong, and J. Liu. Glaucoma detection based on deep convolutional neural network. In *2015 37th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*, pages 715–718. IEEE, 2015.

[38] L. G. Chestnut, J. Schwartz, D. A. Savitz, and C. M. Burchfiel. Pulmonary function and ambient particulate matter: epidemiological evidence from NHANES I. *Arch. Environ. Health*, 46(3):135–144, May 1991.

[39] S. S. Choi, R. J. Zawadzki, M. C. Lim, J. D. Brandt, J. L. Keltner, N. Doble, and J. S. Werner. Evidence of outer retinal changes in glaucoma patients as revealed by ultrahigh-resolution in vivo retinal imaging. *British journal of ophthalmology*, 95(1):131–141, 2011.

[40] S. Y. Chua, A. P. Khawaja, J. Morgan, N. Strouthidis, C. Reisman, A. D. Dick, P. T. Khaw, P. J. Patel, and P. J. Foster. The relationship between ambient atmospheric fine particulate matter (pm2. 5) and glaucoma in a large community cohort. *Investigative ophthalmology & visual science*, 60(14):4915–4923, 2019.

[41] https://www.cs.toronto.edu/~kriz/cifar.html.

[42] P. Cifuentes-Canorea, J. Ruiz-Medrano, R. Gutierrez-Bonet, P. Peña-Garcia, F. Saenz-Frances, J. Garcia-Feijoo, and J. M. Martinez-de-la Casa. Analysis of inner and outer retinal layers using spectral domain optical coherence tomography automated segmentation software in ocular hypertensive and glaucoma patients. *PloS one*, 13(4), 2018.

[43] https://en.wikipedia.org/wiki/Cosine_similarity.

[44] P. Coupe et al. An optimized blockwise nonlocal means denoising filter for 3-D magnetic resonance images. *IEEE Trans. Med. Imaging*, 27(4), Apr. 2008.

[45] https://www.economist.com/the-economist-explains/2016/07/15/why-artificial-intelligence-is-enjoying-a-renaissance.

[46] https://hbr.org/2017/07/whats-driving-the-machine-learning-explosion.

[47] https://data.galaxyzoo.org/gz_trees/gz_trees.html.

[48] M. Dirani, J. G. Crowston, P. S. Taylor, P. T. Moore, S. Rogers, M. L. Pezzullo, J. E. Keeffe, and H. R. Taylor. Economic impact of primary open-angle glaucoma in australia. *Clinical & experimental ophthalmology*, 39(7):623–632, 2011.

[49] https://deepmind.com/blog/unsupervised-learning/.

[50] H. Domínguez Sánchez, M. Huertas-Company, M. Bernardi, D. Tuccillo, and J. Fischer. Improving galaxy morphologies for sdss with deep learning. *Monthly Notices of the Royal Astronomical Society*, 2018.

[51] O. J. Dunn. Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252, 1964.

[52] B. Efron and R. Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75, 1986.

[53] C. B. et.al. Towards interactive curation and automatic tuning of ml pipelines. In *DEEM*, 2018.

[54] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

[55] C. Faloutsos et al. Efficient and effective querying by image content. *J. Intell. Inf. Syst.*, 1994.

[56] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 2008.

[57] http://fits.gsfc.nasa.gov.

[58] https://flink.apache.org/.

[59] B. Foot and C. MacEwen. Surveillance of sight loss due to delay in ophthalmic treatment or review: frequency, cause and outcome. *Eye*, 31(5):771, 2017.

[60] P. J. Foster, F. T. Oen, D. Machin, T.-P. Ng, J. G. Devereux, G. J. Johnson, P. T. Khaw, and S. K. Seah. The prevalence of glaucoma in chinese residents of singapore: a cross-sectional population survey of the tanjong pagar district. *Archives of ophthalmology*, 118(8):1105–1111, 2000.

[61] https://www.microsoft.com/en-us/research/publication/fourth-paradigm-data-intensive-scientific-discovery.

[62] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[63] E. Garyfallidis et al. Dipy, a library for the analysis of diffusion MRI data. *Front. Neuroinform.*, 8, 21 Feb. 2014.

[64] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[65] G. Guedes, J. C Tsai, and N. A Loewen. Glaucoma and aging. *Current aging science*, 4(2):110–117, 2011.

[66] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5), Aug. 2018.

[67] https://www.zooniverse.org/projects/zookeeper/galaxy-zoo.

[68] http://hadoop.apache.org/.

[69] A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[70] D. Halperin et al. Demonstration of the myria big data management service. In *SIGMOD*, 2014.

[71] W. M. Hart, M. Yablonski, M. A. Kass, and B. Becker. Multivariate analysis of the risk of glaucomatous visual field loss. *Archives of Ophthalmology*, 97(8):1455–1458, 1979.

[72] A. Havet, S. Hulo, D. Cuny, M. Riant, F. Occelli, N. Cherot-Kornobis, J. Giovannelli, R. Matran, P. Amouyel, J.-L. Edmé, and L. Dauchet. Residential exposure to outdoor air pollution and adult lung function, with focus on small airway obstruction. *Environ. Res.*, 183:109161, Jan. 2020.

[73] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[74] A. Heijl, M. C. Leske, B. Bengtsson, L. Hyman, B. Bengtsson, and M. Hussein. Reduction of intraocular pressure and glaucoma progression: results from the early manifest glaucoma trial. *Archives of ophthalmology*, 120(10):1268–1279, 2002.

[75] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD '97, Proceedings*, 1997.

[76] H. Herodotou et al. Starfish: A self-tuning system for big data analytics. In *CIDR*, 2011.

[77] http://astroinf.cmm.uchile.cl/category/projects/.

[78] K. Holopigian, W. Seiple, C. Mayron, R. Koty, and M. Lorenzo. Electrophysiological and psychophysical flicker sensitivity in patients with primary open-angle glaucoma and ocular hypertension. *Investigative ophthalmology & visual science*, 31(9):1863–1868, 1990.

[79] D. C. Hood and R. H. Kardon. A framework for comparing structural and functional measures of glaucomatous damage. *Progress in retinal and eye research*, 26(6):688–710, 2007.

[80] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2016.

[81] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[82] https://medium.com/data-collective/rapid-growth-in-available-data-c5e2705a2423.

[83] http://blog.d8a.com/post/9662265140/the-growth-of-image-data-mobile-and-web.

[84] http://www.image-net.org/.

[85] https://paperswithcode.com/sota/image-classification-on-imagenet.

[86] http://impala.apache.org/.

[87] T. L. Jernigan et al. The pediatric imaging, neurocognition, and genetics (ping) data repository. *Neuroimage*, 2016.

[88] N. Ji, , et al. Technologies for imaging neural activity in large volumes. *Nat. Neurosci.*, 2016.

[89] https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence.

[90] M. Kahng et al. Activis: Visual exploration of industry-scale deep neural network models. *IEEE TVCG*, 2018.

[91] M. Kahng, D. Fang, and D. H. P. Chau. Visual exploration of machine learning results using data cube analysis. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, page 1. ACM, 2016.

[92] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[93] M. A. Kass, D. K. Heuer, E. J. Higginbotham, C. A. Johnson, J. L. Keltner, J. P. Miller, R. K. Parrish, M. R. Wilson, and M. O. Gordon. The ocular hypertension treatment study: a randomized trial determines that topical ocular hypotensive medication delays or prevents the onset of primary open-angle glaucoma. *Archives of ophthalmology*, 120(6):701–713, 2002.

[94] K. R. Kendell, H. A. Quigley, L. A. Kerrigan, M. E. Pease, and E. N. Quigley. Primary open-angle glaucoma is not associated with photoreceptor loss. *Invest. Ophthalmol. Vis. Sci.*, 36(1):200–205, Jan. 1995.

[95] https://en.wikipedia.org/wiki/Kernel_method.

[96] A. P. Khawaja, S. Chua, P. G. Hysi, S. Georgoulas, H. Currant, T. W. Fitzgerald, E. Birney, F. Ko, Q. Yang, C. Reisman, D. F. Garway-Heath, C. J. Hammond, P. T. Khaw, P. J. Foster, P. J. Patel, N. Strouthidis, and UK Biobank Eye and Vision Consortium. Comparison of associations with different macular inner retinal thickness parameters in a large cohort: The UK biobank. *Ophthalmology*, 127(1):62–71, Jan. 2020.

[97] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[98] Y. Kita, A. Anraku, R. Kita, and I. Goldberg. The clinical utility of measuring the macular outer retinal thickness in patients with glaucoma. *European journal of ophthalmology*, 26(2):118–123, 2016.

[99] F. Ko, P. J. Foster, N. G. Strouthidis, Y. Shweikh, Q. Yang, C. A. Reisman, Z. A. Muthy, U. Chakravarthy, A. J. Lotery, P. A. Keane, A. Tufail, C. M. Grossi, P. J. Patel, and UK Biobank Eye & Vision Consortium. Associations with retinal pigment epithelium thickness measures in a large cohort: Results from the UK biobank. *Ophthalmology*, 124(1):105–117, Jan. 2017.

[100] S. Kornblith, M. Norouzi, H. Lee, and G. E. Hinton. Similarity of neural network representations revisited. In *ICML '19*, 2019.

[101] S. Kornblith, M. Norouzi, H. Lee, and G. E. Hinton. Similarity of neural network representations revisited. In *ICML '19*, 2019.

[102] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[103] Y. Lecun. Gradient-based learning applied to document recognition. 1998.

[104] C. S. Lee, A. J. Tyring, Y. Wu, S. Xiao, A. S. Rokem, N. P. DeRuyter, Q. Zhang, A. Tufail, R. K. Wang, and A. Y. Lee. Generating retinal flow maps from structural optical coherence tomography with artificial intelligence. *Scientific reports*, 9(1):1–11, 2019.

[105] X. Li, T. Pang, B. Xiong, W. Liu, P. Liang, and T. Wang. Convolutional neural networks based transfer learning for diabetic retinopathy fundus image classification. *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–11, 2017.

[106] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[107] https://www.lsst.org/.

[108] http://dm.lsst.org/.

[109] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. From local explanations to global understanding with explainable ai for trees.

[110] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *NIPs '17*. 2017.

[111] S. Maetschke, B. Antony, H. Ishikawa, G. Wollstein, J. Schuman, and R. Garnavi. A feature agnostic approach for glaucoma detection in oct volumes. *PloS one*, 14(7), 2019.

[112] A. Mahendran et al. Visualizing deep convolutional neural networks using natural pre-images. *IJCV '16*, 2016.

[113] R. Maithra et al. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NIPs*, 2017.

[114] S. L. Mansberger, S. A. Menda, B. A. Fortune, S. K. Gardiner, and S. Demirel. Automated segmentation errors when using optical coherence tomography to measure retinal nerve fiber layer thickness in glaucoma. *American journal of ophthalmology*, 174:1–8, 2017.

[115] O. Marcu et al. Spark versus flink: Understanding performance in big data analytics frameworks. In *IEEE (ICCC)*, 2016.

[116] P. Mehta, S. Dorkenwald, D. Zhao, T. Kaftan, A. Cheung, M. Balazinska, A. Rokem, A. Connolly, J. Vanderplas, and Y. AlSayyad. Comparative evaluation of big-data systems on scientific image analytics workloads. *PVLDB*, 2017.

[117] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Modelhub: Deep learning lifecycle management. *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017.

[118] A. Miki, M. Kumoi, S. Usui, T. Endo, R. Kawashima, T. Morimoto, K. Matsushita, T. Fujikado, and K. Nishida. Prevalence and associated factors of segmentation errors in the peripapillary retinal nerve fiber layer and macular ganglion cell complex in spectral-domain optical coherence tomography images. *Journal of glaucoma*, 26(11):995–1000, 2017.

[119] K. L. Miller et al. Multimodal population brain imaging in the uk biobank prospective epidemiological study. *Nature Neuroscience*, 2016.

[120] https://mlflow.org/.

[121] http://yann.lecun.com/exdb/mnist/.

[122] A. S. Morcos, M. Raghu, and S. Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS '18*, 2018.

[123] H. Muhammad, T. J. Fuchs, N. De Cuir, C. G. De Moraes, D. M. Blumberg, J. M. Liebmann, R. Ritch, and D. C. Hood. Hybrid deep learning on single wide-field optical coherence tomography scans accurately classifies glaucoma suspects. *Journal of glaucoma*, 26(12):1086, 2017.

[124] R. Müller, S. Kornblith, and G. E. Hinton. When does label smoothing help? In *Advances in Neural Information Processing Systems*, pages 4696–4705, 2019.

[125] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu. Definitions, methods, and applications in interpretable machine learning. *Proc. Natl. Acad. Sci. U. S. A.*, 116(44):22071–22080, Oct. 2019.

[126] J.-C. Mwanza, J. D. Oakley, D. L. Budenz, R. T. Chang, J. K. O'Rese, and W. J. Feuer. Macular ganglion cell–inner plexiform layer: automated detection and thickness reproducibility with spectral domain–optical coherence tomography in glaucoma. *Investigative ophthalmology & visual science*, 52(11):8323–8329, 2011.

[127] J. Nayak, R. Acharya, P. S. Bhat, N. Shetty, and T.-C. Lim. Automated diagnosis of glaucoma using digital fundus images. *Journal of medical systems*, 33(5):337, 2009.

[128] National eye institute.us age-specific prevalence rates for glaucoma by age and race/ethnicity.[accessed dec 2019. https://www.nei.nih.gov/learn-about-eye-health/resources-for-health-educators/eye-health-data-and-statistics/glaucoma-data-and-statistics.

[129] P. A. Newman-Casey, A. J. Verkade, G. Oren, and A. L. Robin. Gaps in glaucoma care: a systematic review of monoscopic disc photos to screen for glaucoma. *Expert review of ophthalmology*, 9(6):467–474, 2014.

[130] https://nifti.nimh.nih.gov/nifti-1.

[131] T. M. Nork, J. N. Ver Hoeve, G. L. Poulsen, R. W. Nickells, M. D. Davis, A. J. Weber, S. H. Sarks, H. L. Lemley, L. L. Millecchia, et al. Swelling and loss of photoreceptors in chronic human and experimental glaucomas. *Archives of ophthalmology*, 118(2):235–245, 2000.

[132] J. V. Odom, J. G. Feghali, J.-c. Jin, and G. W. Weinstein. Visual function deficits in glaucoma: electroretinogram pattern and luminance nonlinearities. *Archives of Ophthalmology*, 108(2):222–227, 1990.

[133] C. Olah et al. Feature visualization. *Distill*, 2017.

[134] C. Olah et al. The building blocks of interpretability. *Distill'18*, 2018.

[135] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[136] T. E. Oliphant. Python for scientific computing. *Computing in Science and Engg.*, 2007.

[137] J. I. Orlando, E. Prokofyeva, M. del Fresno, and M. B. Blaschko. Convolutional neural network transfer for automated glaucoma identification. In *12th International Symposium on Medical Information Processing and Analysis*, volume 10160, page 101600U. International Society for Optics and Photonics, 2017.

[138] N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 1975.

[139] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[140] S. Panda and J. B. Jonas. Decreased photoreceptor count in human eyes with secondary angle-closure glaucoma. *Investigative ophthalmology & visual science*, 33(8):2532–2536, 1992.

[141] Y. Park, M. J. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, 2016.

[142] M. B. A. C. Parmita Mehta, Stephen Portillo. Towards sampling for deep learning model diagnosis. In *ICDE'20*, 2020.

[143] S. B. Patel, N. Reddy, X. Lin, and J. T. Whitson. Optical coherence tomography retinal nerve fiber layer analysis in eyes with long axial lengths. *Clin. Ophthalmol.*, 12:827–832, May 2018.

[144] D. Pavia, J. Bateman, A. Lennard-Jones, J. Agnew, and S. Clarke. Effect of selective and non-selective beta blockade on pulmonary function and tracheobronchial mucociliary clearance in healthy subjects. *Thorax*, 41(4):301–305, 1986.

[145] A. Pavlo et al. A comparison of approaches to large-scale data analysis. In *SIGMOD*, 2009.

[146] E. Perkins. The bedford glaucoma survey. i. long-term follow-up of borderline cases. *The British journal of ophthalmology*, 57(3):179, 1973.

[147] S. Phene, R. C. Dunn, N. Hammel, Y. Liu, J. Krause, N. Kitade, M. Schaekermann, R. Sayres, D. J. Wu, A. Bora, C. Semturs, A. Misra, A. E. Huang, A. Spitze, F. A. Medeiros, A. Y. Maa, M. Gandhi, G. S. Corrado, L. Peng, and D. R. Webster. Deep learning and glaucoma specialists: The relative importance of optic disc features to predict glaucoma referral in fundus photographs. *Ophthalmology*, 126(12):1627–1639, Dec. 2019.

[148] J. Platt. Probabilistic outputs for svms and comparisons to regularized likehood methods, advances in large margin classifiers, 1999.

[149] R. Poplin, A. Varadarajan, K. Blumer, Y. Liu, M. McConnell, G. Corrado, L. Peng, and D. Webster. Predicting cardiovascular risk factors from retinal fundus photographs using deep learning. arxiv 2017. *arXiv preprint arXiv:1708.09843*.

[150] https://www.postgresql.org/.

[151] H. A. Quigley, C. Enger, J. Katz, A. Sommer, R. Scott, and D. Gilbert. Risk factors for the development of glaucomatous visual field loss in ocular hypertension. *Archives of ophthalmology*, 112(5):644–649, 1994.

[152] http://www.rasdaman.org/.

[153] https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning.

[154] M. Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th Python in Science Conference*, 2015.

[155] A. E. Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[156] http://www.scidb.org/.

[157] http://forum.paradigm4.com/t/persistenting-data-to-remote-nodes/1408/8.

[158] http://skyserver.sdss.org/dr7.

[159] T. Sellam, K. Lin, I. Y. Huang, M. Yang, C. Vondrick, and E. Wu. Deepbase: Deep inspection of neural networks. In *SIGMOD '19, Amsterdam*, 2019.

[160] R. R. Selvaraju et al. Grad-cam: Visual explanations from deep networks via gradient-based localization. *ICCV'17*, 2018.

[161] A. Septiarini, D. M. Khairina, A. H. Kridalaksana, and H. Hamdani. Automatic glaucoma detection method applying a statistical approach to fundus images. *Healthc. Inform. Res.*, Jan. 2018.

[162] S. Y. Shen, T. Y. Wong, P. J. Foster, J.-L. Loo, M. Rosman, S.-C. Loon, W. L. Wong, S.-M. Saw, and T. Aung. The prevalence and types of glaucoma in malay people: the singapore malay eye study. *Investigative ophthalmology & visual science*, 49(9):3846–3851, 2008.

[163] J. Shi et al. Clash of the titans: Mapreduce vs. spark for large scale data analytics. *PVLDB*, 2015.

[164] A. Shrikumar et al. Learning important features through propagating activation differences. In *ICML '17*, 2017.

[165] K. Simonyan et al. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR'13*, 2013.

[166] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[167] http://spark.apache.org/.

[168] E. Sparks et al. Automating model search for large scale machine learning. In *SoCC'15*, 2015.

[169] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Trans. Vis. Comput. Graph.*, 2018.

[170] C. Sudlow, J. Gallacher, N. Allen, V. Beral, P. Burton, J. Danesh, P. Downey, P. Elliott, J. Green, M. Landray, B. Liu, P. Matthews, G. Ong, J. Pell, A. Silman, A. Young, T. Sprosen, T. Peakman, and R. Collins. Uk biobank: An open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLOS Medicine*, 12(3):1–10, 03 2015.

[171] M. Sundararajan et al. Axiomatic attribution for deep networks. In *ICML*, 2017.

[172] R. Susanna, C. G. De Moraes, G. A. Cioffi, and R. Ritch. Why do people (still) go blind from glaucoma? *Translational vision science & technology*, 4(2):1–1, 2015.

[173] https://en.wikipedia.org/wiki/Support-vector_machine.

[174] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[175] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[176] R. Taft et al. Genbase: a complex analytics genomics benchmark. In *SIGMOD*, 2014.

[177] http://www.tensorflow.org/.

[178] Y.-C. Tham, X. Li, T. Y. Wong, H. A. Quigley, T. Aung, and C.-Y. Cheng. Global prevalence of glaucoma and projections of glaucoma burden through 2040: a systematic review and meta-analysis. *Ophthalmology*, 121(11):2081–2090, 2014.

[179] The Dark Energy Survey Collaboration. The Dark Energy Survey. *ArXiv Astrophysics e-prints*, Oct. 2005.

[180] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 648–659. VLDB Endowment, 2004.

[181] A. C. Thompson, A. A. Jammal, S. I. Berchuck, E. B. Mariottoni, and F. A. Medeiros. Assessment of a Segmentation-Free Deep Learning Algorithm for Diagnosing Glaucoma From Optical Coherence Tomography Scans. *JAMA Ophthalmology*, 02 2020.

[182] http://www.tpc.org.

[183] https://www.youtube.com/watch?v=Dc0sr0kdBVI.

[184] Uk biobank. https://www.ukbiobank.ac.uk/. Online; accessed September,2019.

[185] J. D. Unterlauft, M. Rehak, M. R. Böhm, and F. G. Rauscher. Analyzing the impact of glaucoma on the macular architecture using spectral-domain optical coherence tomography. *PloS one*, 13(12), 2018.

[186] L. van der Maaten and G. E. Hinton. Visualizing data using t-sne. In *Journal of Machine Learning Research 9, Nov*, 2008.

[187] D. C. Van Essen et al. The WU-Minn human connectome project: an overview. *Neuroimage*, 15 Oct. 2013.

[188] M. Vartak. Modeldb: A system for machine learning model management. In *CIDR'17*, 2017.

[189] M. Vartak, J. M. F. da Trindade, S. Madden, and M. Zaharia. Mistique: A system to store and query model intermediates for model diagnosis. In *SIGMOD Conference*, 2018.

[190] http://www.robots.ox.ac.uk/~vgg/research/very_deep.

[191] B. A. Wandell. Clarifying human white matter. *Annu. Rev. Neurosci.*, 1 Apr. 2016.

[192] H.-J. Wang, Q. Li, Y. Guo, J.-Y. Song, Z. Wang, and J. Ma. Geographic variation in chinese children' forced vital capacity and its association with long-term exposure to local PM10: a national cross-sectional study. *Environ. Sci. Pollut. Res. Int.*, 24(28):22442–22449, Oct. 2017.

[193] J. Wang et al. The myria big data management and analytics system and cloud services. In *CIDR*, 2017.

[194] https://en.wikipedia.org/wiki/Big_data.

[195] J. K. Winkler, C. Fink, F. Toberer, A. Enk, T. Deinlein, R. Hofmann-Wellenhof, L. Thomas, A. Lallas, A. Blum, W. Stolz, and H. A. Haenssle. Association between surgical skin markings in dermoscopic images and diagnostic performance of a deep learning convolutional neural network for melanoma recognition. *JAMA Dermatol.*, Aug. 2019.

[196] H. Wu, C. Wang, J. Yin, K. Lu, and L. Zhu. Interpreting shared deep learning models via explicable boundary trees. *ArXiv*, abs/1709.03730, 2017.

[197] H. Wu, C. Wang, J. Yin, K. Lu, and L. Zhu. Sharing deep neural network models with interpretation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, 2018.

[198] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5(Aug):975–1005, 2004.

[199] J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *CoRR*, 2015.

[200] M. Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

[201] M. Zeiler et al. Visualizing and understanding convolutional networks. In *ECCV '14*, 2014.