Implementing the LSST Software Stack for DESGW processing and the Integration of Convolutional Neural Networks into the DESGW Pipeline

Alexander Navarro, Andrews University

ABSTRACT

The Dark Energy Survey Gravitational Wave (DESGW) group strives to understand the largescale structure of the universe and galaxies by looking for electromagnetic signatures in telescope images following gravitational wave detections. The DESGW group uses a processing pipeline to perform difference imaging to look for potential candidates. In order to perform these searches more effectively, we first explored using an alternative processing pipeline, the LSST Software Stack to analyze the telescope images. Because the LSST software stack is currently transitioning between its generation 2 and generation 3 system, we decided that while the software will be usable in the future once generation 3 is complete, at the moment its incompleteness makes it impractical to use. We then decided to work on improving the current pipeline by developing a module that integrates a Convolutional Neural Network (CNN) to test difference imaging products for bad subtractions due to misalignment.

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics
INTRODUCTION

The Dark Energy Survey Gravitational Wave Group (DESGW) is a subgroup of the larger Dark Energy Survey (DES) astronomical collaboration. One of the main objectives of the DES is to understand the current structure of the universe, especially with regard to how it relates to the expansion history of the universe, as driven by dark energy. DESGW focuses on one specific method of studying this, searching for and analyzing electromagnetic (EM) signatures following gravitational wave (GW) events as detected by the LIGO VIRGO KAGRA collaboration (LVK). Using the combination of the LVK GW and EM signals from telescopes like DECam [3], in addition to simply gathering more information about the composition and properties of neutron stars, an additional measure of the expansion of the universe can be obtained by using the light spectra to find the redshift, and therefore the expansion rate of the universe [2, 8, 14].

Since telescope observations produce immense amounts of data, a computational framework is necessary for the effective search and identification of candidate objects. The DESGW pipeline is based on the difference imaging pipeline used by the DES supernova analysis team [9], modified as described in Herner et al. (2020) [7]. These pipelines are built around difference imaging [6], which is a candidate detection method. Difference imaging is the process of taking template images from past observing runs of the telescope, subtracting them from the search image taken during the run which is in question, and then looking for any remaining artifacts. If a source is present in the search image that is not present in the template image, after the subtraction, the bright spot will remain, and can then be easily identified as a potential candidate.

While the current pipeline is complete and fully functional, that is not reason to avoid looking for improvements. There are two main problems with the DESGW pipeline that we decided to tackle. The first is the limited technical support and developmental resources for the pipeline, and the second is the need for human time in double checking the candidate lists from the output to remove several classes of false-positives. To address the first problem, the solution we pursued is the possible use of an alternative pipeline, the Legacy Survey of Space and Time (LSST) software stack. To address the second problem, we implemented a CNN developed by Shandonay et al. [15] in order to identify some of the false positives.

COMPONENTS OF THE GEN 3 LSST SOFTWARE STACK

With regard to the design of the LSST Stack, instead of creating a single set of software which is responsible for every aspect of data management and processing, the LSST team rather divided the task between different sub-programs that can interact.[11] In this way, the communication between components makes it necessary that the various pieces use flexible and
standardized methods for exchanging necessary data and information for the execution of the pipeline. This flexibility enables the stack as a whole to be more easily used by other research groups for their own individual purposes, as it makes it easier for groups to modify existing pieces of the stack, or to add their own components that may be necessary. To this end, the LSST Software stack is comprised of two software systems, each responsible for different tasks: the Butler system, and the main processing pipeline.

The main task of the Butler system is to take in, store, organize, and make available any information that may be needed by the processing pipeline or users, such as telescope images, star catalogs, and telescope specifications. To accomplish this task, the Butler system is further divided into three smaller systems: the Datastore, the Registry, and the Butler itself. The purpose of the Datastore is simply to hold onto data files and their identification keys. The Registry then deals with storing and managing metadata of the data files, as well as managing collections of files that are grouped together for the sake of processing, also called datasets. The Butler then serves as an interface between the user and the whole Butler system. The Butler is thus responsible for taking data files provided by the user, then formatting, or generally preparing, said files to be stored within the Datastore, and extracting desired metadata and image parameters to be stored in the Registry. The Butler is also responsible for providing data to the main processing pipeline. Whenever the pipeline needs a data file, telescope specifications, star catalog entries, or any other stored information for processing, the Butler must identify what specifically the pipeline is requesting, query the Registry for files with metadata matching what the pipeline is requesting, and then, once a set of data files is identified, it must query the Datastore to get the relevant data files, before finally returning the requested parameters or the data contained in the files to the processing pipeline.

The purpose of the processing pipeline is, as is implied by the name, to handle the transformation of unprocessed data files into information usable by scientists. The processing pipeline consists of a central pipeline toolkit, called pipe_base, which manages the execution of individual pipelines. A pipeline consists of a series of tasks, each of which performs a small piece of whatever process is being performed on the data. In a pipeline, some tasks may receive a specific piece of data, while it may need a piece of data that another task would produce from that data instead, and pipe_base allows the task to call other tasks, or sub-tasks, to perform
such necessary calculations for its own execution. [13] Thus, pipe_base manages task and sub-task execution, making sure that the pipeline executes as it was programmed to. Aside from pipe_base, the processing pipeline contains a variety of tools designed to ensure that the necessary tasks are all executed in the proper order, and that the various tasks successfully request and obtain the necessary data. Additionally, there are toolsets that provide scientists with the various tasks that are used to construct full pipelines.

IMPLEMENTATING THE LSST STACK FOR DESGW

In order to use the LSST stack for processing data in the DESGW group, both sections of the LSST stack, the Butler system, and the processing pipeline system, must be setup so as to be able to accept and work with DECam data. While the gen 2 system is largely compatible with DECam data, because of the eventual complete transition to gen 3, any progress made setting up gen 2 would later have to be redone again with gen 3, and thus would be meaningless. Thus, we focused on working with the in-development gen 3 system instead of working with the more complete gen 2 system.

After a gen 3 Butler was created, raw data files from DECam were ingested into the Butler. Previous work by Garrett [4] with gen 2 had successfully ingested already pre-processed science, mask, and weight files. The gen 3 Butler, however, had an incomplete ingester, and as such, through the command prompt, could only accept raw data files. In order to ingest more complex datasets, the gen 3 Butler is designed to be able to be set up, run, and used, as a Python object, using a specially made Butler class. Using class methods of the Butler object, and various Registry and Datastore data structures, it is theoretically possible to ingest any dataset. However, in order to ingest complex datasets using an instance of the Butler, one needs to be able to manually fill the Registry and Datastore parameters that are used in storing and organizing the datasets, which, in gen 2, had been done by ingestImagesDecam.py. While this project was in progress, a gen 3 ingester was being developed, but was incomplete, being completed only after the shift in project occurred. The ingester’s incomplete state meant that the Registry and Datastore metadata expected from preprocessed files had to be added manually. One attempted approach to bypass this problem is to, until gen 3 is completed, ingest files into a gen 2 Butler and Datastore, and then convert the gen 2 Butler into a gen 3 Butler. Other files, such as star catalogs, and telescope specifications, are more easily ingested into the gen 3 system.

Regarding the processing pipeline implementation, because of the larger community surrounding the LSST stack, as well as the fairly standard approach that is taken towards data processing for astronomy, consisting of the use of science images and template images to produce difference images, much of the processing pipeline that would be used by the DESGW group would be similar to other pipelines already developed by other groups using LSST. For example, the ap_pipe tool set
already contains tasks for coaddition (adding multiple template images to create a larger master template), warping exposures (mapping science images onto the master template) and difference imaging (subtracting warped science images from the template) [17]. Another tool, \texttt{pipe\_tasks}, also similarly contains various tasks used by pipelines, as well as several already setup pipelines for use in data processing, for example the Data Release Production Pipeline [17]. As such, in order to use the LSST processing pipeline, the creation of a new pipeline is not necessary, rather, the adaption and modification of an already established pipeline would likely be sufficient.

CURRENT CHALLENGES WITH THE LSST STACK

There are two primary problems preventing the use of the LSST software stack in place of the current legacy DESGW pipeline. First the current developmental nature of the generation 3 Butler and processing pipeline. Because the software is developmental, some of the tools, such as the aforementioned general data file ingestion tool, are either non-existent, or lacking many of the necessary features to be as flexible and widely usable as is necessary. This makes it challenging to complete many of the necessary tasks for the implementation of the LSST stack. As suggested before, one possibility to work around this was to do some of the work of ingestion using the gen 2 Butler, and then convert the Butler to gen 3. However, the DESGW group uses DECam data files that have been fed through alternative preprocessing pipeline, which the LSST stack was not primarily designed to work with. While the data could be ingested into a gen 2 Butler without much difficulty, the ingested files were incompatible with gen 3, missing various metadata parameters, or having said parameters formatted improperly for gen 3, thus making conversion from gen 2 to gen 3, maybe possible, but very difficult.

The second primary problem preventing the use of the LSST stack stems in large part also from the fact that the gen 3 software is in development, and that is the lack of in-depth documentation. Currently, multiple basic tutorials and guides exist for the gen 2 software, but few tutorials exist for gen 3. The higher-level documentation for gen 3, describing the components, classes, and tool kits contained within the LSST pipeline, while often containing lists of said classes and their respective methods and parameters, often lack a description of what is required for their use. For example, regarding the aforementioned attempt to ingest data using the Python instances of the Butler, it was mentioned how various metadata and image id parameters are expected by the Butler, Datastore, and Registry in order to ingest the data. One such parameter is the dimensions of a data file, which is a data structure that is used to describe some property or piece of information that can be used to help identify collections of data files that go together. While the LSST stack documentation describes the class methods and parameters of the dimensions structure, it does not include any information about what type of
dimensions are used and expected by the various components of the LSST stack, making it nearly impossible to create the necessary dimension objects to be put in the Registry, since it is difficult to find out what dimensions are even needed.

It is for these reasons that this project of integrating the LSST stack into the DESGW groups computational workflow was abandoned for the time being. While the use of the LSST stack would be advantageous in the future, at the moment, with gen 3 still in development, we decided it was better to ensure that the current pipeline is fully functional and ready for LIGO’s O4. For this reason, we refocused our attention on improving the current processing pipeline.

THE CURRENT DESGW PROCESSING PIPELINE

As stated previously, the DESGW pipeline is built around the difference imaging method of candidate identification. In order for this to be performed, the telescope images must be properly processed to be prepared for difference imaging. Then, once the difference imaging and any subsequent analysis is performed, the data produced by the analysis must be prepared for human review, so future observation plans can be well informed. Towards this end, the DESGW pipeline is composed of several main stages: pre-processing, difference imaging, candidate identification, and then post-processing.

The pre-processing stage, also referred to as the Single Epoch Processing Pipeline is a modified version of the Morganson et al. (2018) [12] pipeline. It primarily involves several image correction and calibration steps, followed by object cataloging of all observed stars or objects of note in the science images. Difference imaging, as described previously, involves the subtraction of image stamps from previous surveys from the new science stamps with the goal of identifying astronomical features not present in the survey images. Candidate identification, as implied by the name, involves identifying objects based on the subtraction images that could be responsible for the gravitational wave detection being investigated; this is currently performed with the autoScan algorithm [5]. Finally, post-processing involves performing photometry on the images obtained from difference imaging, estimating probabilities of the GW source being from candidate galaxies, as well as preparation of database and webpage information to be accessible to scientists for potential follow up observation.

MOTIVATION FOR USING A CNN FOR THE DESGW PIPELINE

Within the DESGW pipeline, the implementation of difference imaging as the primary method for candidate identification can lead to false-positive detections if the survey and science images are not properly aligned. In the past, these bad subtractions have been identified by hand, consuming human time, and slowing the availability of reliable candidate lists. In order to work towards reducing the number of images that need human identification because of bad subtractions, Shandonay et al. [15]
developed and trained a convolutional neural network (CNN) to score sets of search, template, and difference stamps based on whether the image truly contains a transient object with host galaxy. Shandonay et al. estimate that in practice, the CNN should obtain a purity of around 80%, meaning that at least 80% of candidates presented to the CNN should be identified correctly. Thus, by presenting these CNN scores as a part of the candidate lists, less human effort is required in identifying false positives, allowing candidate lists to be prepared for astronomers more quickly, and allowing astronomers to have more information in deciding which candidates are worth following up with further observations.

INTEGRATION OF THE CNN

The first implementation of the CNN to the DESGW pipeline had the necessary code implemented at the end of the difference imaging stage, at the end of the SEdiff.sh program. The motivation for placing the CNN at this location was because it was known that the stamp files (smaller image files that contain only a single candidate) that the CNN analyzes are readily available at this stage of the pipeline. Since the CNN requires a specific system environment with various Python packages available, the CNN was implemented by having the primary SEdiff.sh code run a secondary script, called PLCNN.sh, which would set up the required environment. PLCNN.sh would then execute the main Python code for running the CNN and preparing the data products, RUNNN.py. In order to run the CNN itself, RUNNN.py would call a function stored in runCNN.py, which was responsible for feeding the data into the CNN.

The CNN is designed to take several optional parameters, the PSF (Point Spread Function), FLUXCAL (Calibrated Flux), and FLUXCALERR (Calibrated Flux Error), in order to improve its performance. To use these values, we moved the CNN code into post-processing. Because of this move, we had to be more selective about where we would place the CNN in the pipeline in order to ensure that the stamps were readily available to the CNN. Since post-processing is mostly written in Python, while the difference imaging pipeline is mostly written in Shell script, the code had to be adapted to be better suited for this new location. To do this, the use of PLCNN.sh and RUNNN.py was abandoned, and the runCNN.py code was expanded to perform all the tasks that would be expected of it, from checking that the input data is complete, to generating the csv, which was previously performed by RUNNN.py. Additionally, this code’s flexible design allows it to be used outside of post-processing as a standalone tool.

RESULTS FROM USING THE CNN
While the new CNN code is ready to be added to the post-processing steps, it has not yet been added because of several database problems while running the pipeline. However, when the code was still a part of SEdiff.sh, it was successfully run as a part of that portion of the pipeline, confirming that the basic CNN code is able to take in the data products produced by the pipeline, and that it can be added to the pipeline and successfully produce data files. The CNN code was also run using previous exposures and their stamps for which it was already known that results of interest were contained within them, for example, on exposures corresponding to GW170817 [16], a gravitational wave event corresponding to a Binary Neutron star merger, for which the electromagnetic counterpart signal was detected, and received a high CNN score, meaning the CNN scored the kilo nova correctly.

Since the new runCNN.py code has yet to be added to the post-processing pipeline, several steps remain before the full addition of the pipeline can be considered complete. A couple additions need to be made to the pipeline to integrate runCNN.py into the pipeline, so that it can properly take in the necessary data, and so that its outputs can be available in both the DES databases as well as in the candidate

![Figure 2: Three sets of stamps showing the performance of the CNN compared to the autoScan algorithm. 10337380 is an example of a bad subtraction. 10331030 is an example of a possible of a possible transient + host galaxy, in this case, likely a supernova. 10331029 is the light signature associated with GW170817 kilonova. All three sets of stamps received a high autoScan score, but the bad subtraction was identified as such, as is indicated by its low CNN score.](image)
lists on public webpages. Most of these steps are relatively simple to implement, especially since the runCNN.py code was designed to be easy to stick into any python code, as well as the fact that the runCNN.py was written with a nearly full description of the arguments that the function expects, their expected data types. The steps taken by the function are also well documented, making the code understandable, so future debugging should be easier.

CONCLUSIONS AND FUTURE WORK

Regarding the first project outlined, pursuing the use of the LSST stack as a replacement for the legacy DESGW pipeline, the pipeline was deemed currently incomplete both as far as functionality and documentation. However, once completed, since the pipeline is designed to be flexible and customizable, as well as having a larger technical support network, it could serve as a replacement for the DESGW pipeline. For this reason, we believe that it would be advantageous for the DESGW to, at a later date, continue to pursue the possibility of adapting the LSST stack for DESGW use.

With the new CNN code, it is ready to be inserted into the post processing pipeline, and has been tested outside of the pipeline. Additionally, since the main steps for running the CNN did not change from the SEdiff.sh version to the post processing version, it is expected that the CNN code should be able to easily take in the stamps. Thus, the addition of runCNN.py to the pipeline should not be difficult, and once done, the availability of the CNN scores should reduce the human effort required for identifying false-positives, thus improving the speed at which data can be made available to astronomers.

ACKNOWLEDGEMENTS

I would especially like to thank Kenneth Herner for serving as my supervisor over the duration of these two projects and providing extensive assistance in both learning and troubleshooting the various pipelines and codes. Additionally, I would like to thank Nora Sherman and Alyssa Garcia for their help running and debugging the DESGW pipeline.

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI).

REFERENCES

LSST Software Stack. FERMILAB-PUB-20-407-SCD-WDRS.


