MIE324 Final Report

MISweeper: A Deep Learning Approach to Astronomical Time-Series Classification

Graham Hoyes - 1003109049
Daniil Orekhov - 1002302938

Word count: 1999, Penalty: 0%

December 2 2018

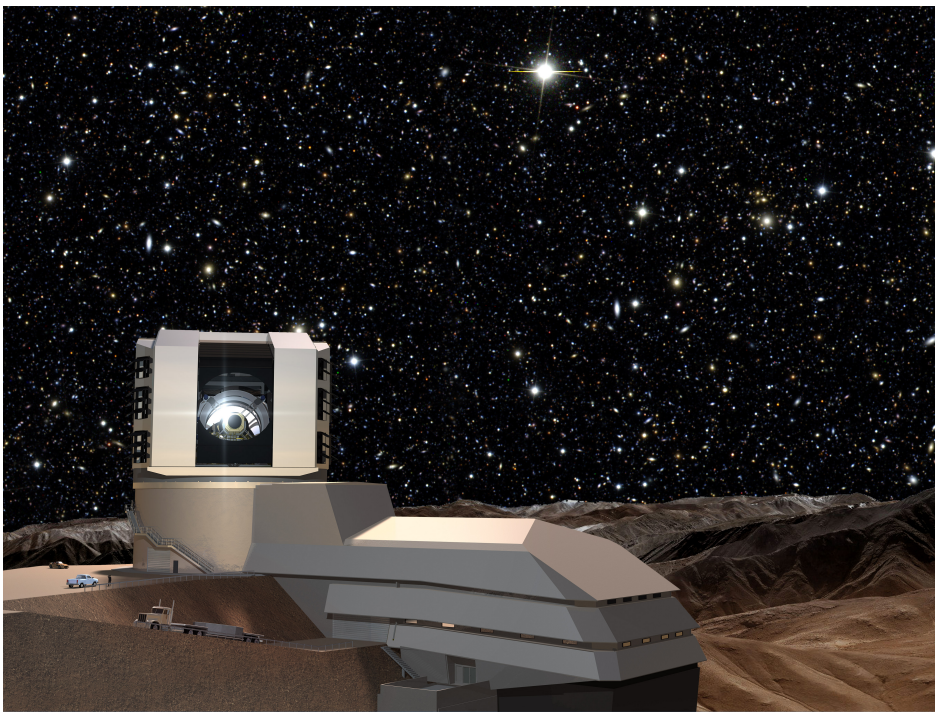


Figure 1: The LSST Telescope [1].

1 Introduction

The Large Synoptic Survey Telescope (LSST) is a telescope set to begin a 10-year survey of the sky starting in 2022 that will generate over 200 petabytes of image data (over 30 terabytes of data each night) [2]. The Photometric LSST Astronomical Time-series Classification Challenge (PLAsTiCC) is a data science challenge to classify astronomical time-series data coming from the Telescope. The Telescope captures in 6 passbands (frequency ranges), and subtracts images to obtain time-series data for astronomical data. There are two classes of objects: those with relatively constant photon flux that move spatially, and those whose position remains fixed but have a changing flux with time. The goal of this project is classifying the latter into 14 distinct classes, and an extra 15th class for previously undetected objects (which will not be present in the training set).

Without leveraging machine learning, it would be impossible to analyze the vast quantity of data from the telescope, and thus the project could not exist. By participating in this challenge, we are not only improving our grasp of machine intelligence techniques, but we are contributing to the worldwide scientific community.

2 Data and Processing

Raw data used is provided by the PLAsTiCC challenge. The LSST conducts observations in 6 different astronomical filters, called passbands (which are, respectively, the u, g, r, i, z, and y bands), each a capturing a different range of wavelengths. Since the LSST telescope goes online in 2019, data for this challenge is simulated based on expected results. Subsequent images (taken each night) are subtracted from each other, to create time-series "light curve" data, representing changes in photon flux over time [3]. A sample of such light curve data is shown in Figure 2 [4].

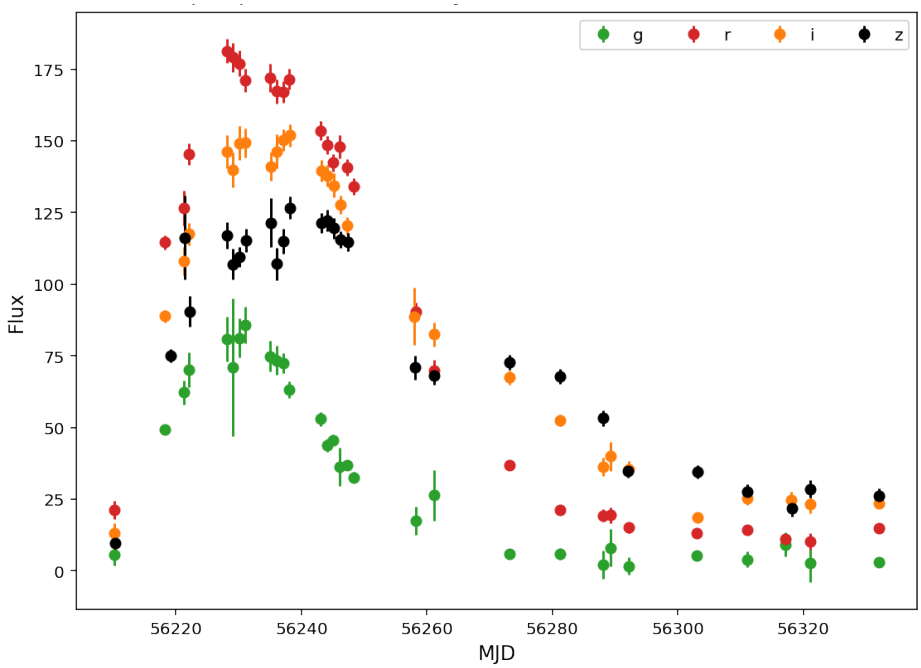


Figure 2: Sample light curve data in 4 passbands, plotted against time.

Raw data samples are provided on Kaggle as csv files. Each row of the training data corresponds to measurements of flux taken for a specific detected object, at a specific time (given in Modified Julian Date (MJD), defined as the number of days since midnight on November 17, 1858). Since different passbands are captured at different times, there are gaps between measurements in a given passband of between 30 minutes and 250 days.

Training data is annotated with the class label. Classes in the data for the competition are a subset of 14 classes the telescope will detect. In addition, the model should be able to recognize objects which do not fall into any of the classes (which would represent a previously undetected astronomical object), which are denoted with class 99.

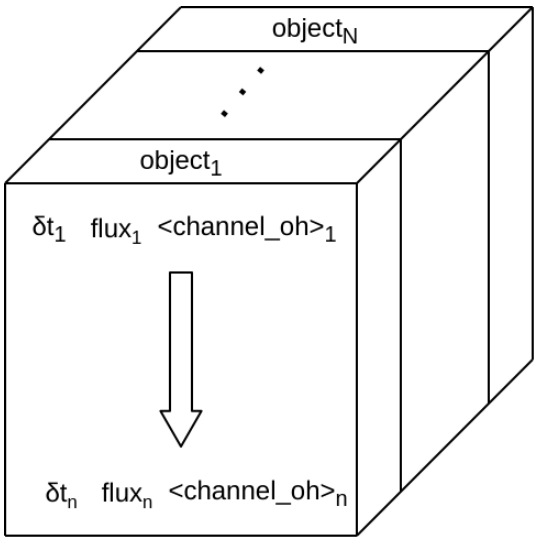


Figure 3: 3D tensor of data.

Our approach for classifying outliers is discussed in Section 4.

2.1 Preprocessing

The time-series data for each object records the measured flux, flux error, and whether the measurement's brightness is significantly different at the three-σ level relative to a reference image of the survey region, as a measurement of if the object was detected [3]. In addition, we compute the following two composite features [5]:

$$\text{flux square ratio} = \left(\frac{\text{flux}}{\text{flux error}}\right)^2$$

$$\text{flux by flux square ratio} = (\text{flux}) \times (\text{flux square ratio})$$

For each of the above features, we calculate the following statistics as aggregates across each object[1]:

Table 1: Aggregates calculated on the dataset

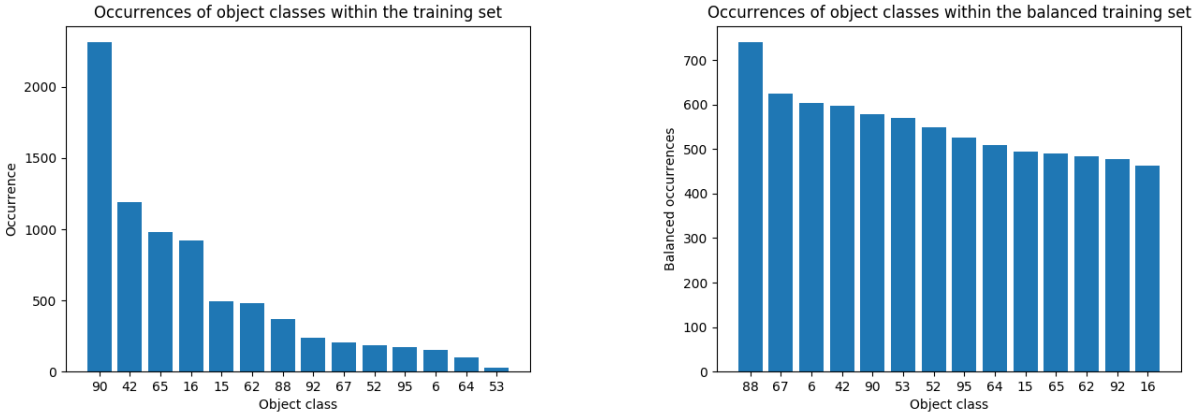
Feature	**Statistics**
MJD	Min, Max, Size
Passband	Min, Max, Mean, Median, Std
Flux	Min, Max, Mean, Median, Std, Skew
Flux error	Min, Max, Mean, Median, Std, Skew
Flux square ratio	Sum, Skew
Flux by flux square ratio	Sum, Skew

To pass into the model, time-series data is reshaped into the tensor in Figure 3. Datapoints are sorted by time (MJD), and time step since the previous point δt_i is recorded (with $\delta t_0 = 0$). Normalized flux and the one-hot encoded channel are also recorded. The tensor's height is the length of the longest time-series (352 samples for the training data). Objects with a shorter time-series are padded at the end with zero rows. The length of each time-series is saved in a separate vector, which is needed later in the model.

2.2 Data Balancing

As shown in Figure 4, the provided training set was severely unbalanced, with the most occurring class occurring ∼70x more than the least occurring class. When training on the unbalanced dataset, accuracy was not a reliable

[1]Note that size, calculated on MJD, is used to record the number of datapoints for each object, and could have been obtained on any of the features.



(a) Before balancing

(b) After balancing

Figure 4: Class occurrences in the training set before and after balancing

metric for model performance, since the model could achieve better than random (7% for a balanced dataset) by always picking the most occurring classes. Hence, F1 was used as the accuracy metric when training on the unbalanced dataset, and a weighted loss function was used, but model performance was still unsatisfactory.

Different techniques for balancing unbalanced datasets were investigated, the most promising of which was SMOTE, which generates new datapoints as averages of a datapoint and its k-nearest neighbors in feature space [6]. However, the time-series nature of the data made this technique infeasible. Instead, we undersampled over-represented classes and oversampled (taking the same objects multiple times) the under-represented classes, until each class had approximately the same number of samples (the target was set as the mean number of samples in the unbalanced dataset). The effect of balancing on class occurrences is shown in Figure 4 (b).

Because of the oversampled datapoints, simply splitting the data into a training and validation set would result in identical time-series being present in both sets. To fix this, data was split into training and validation before balancing. Passing the training labels into the `stratify` parameter of `train_test_split` ensured that both sets have the same proportions of classes and hence the desired train-validation ratio is preserved.

3 Software Structure

Since this project is a pure data science project, the expected functionality is limited to generating a data file with classification predictions based on data obtained from the telescope. Thus, the structure of the software for this project, developed using PyTorch, is a set of scripts that operate in two modes: training and inference as shown in Figure 5. These modes operate as follows:

- **Training Mode** - A file with the "raw" dataset for training is loaded and passed through a preprocessor script to convert data in a usable format, as described above. Then, the training script, based on defined model configuration, sets up a loader for the preprocessed data and all aspects of the training loop (such as optimizer and loss function). Once everything is prepared, the script begins to train the NN and saves the best model (see section 4.3).
- **Inference Mode** - Similar to the Training Mode, the file with the "raw" dataset for inference is loaded and passed through the same preprocessor as used for training. Then, the inference script loads the model with the highest cross-validation score and passes the preprocessed inference data through it to generate classification predictions, which are recorded.

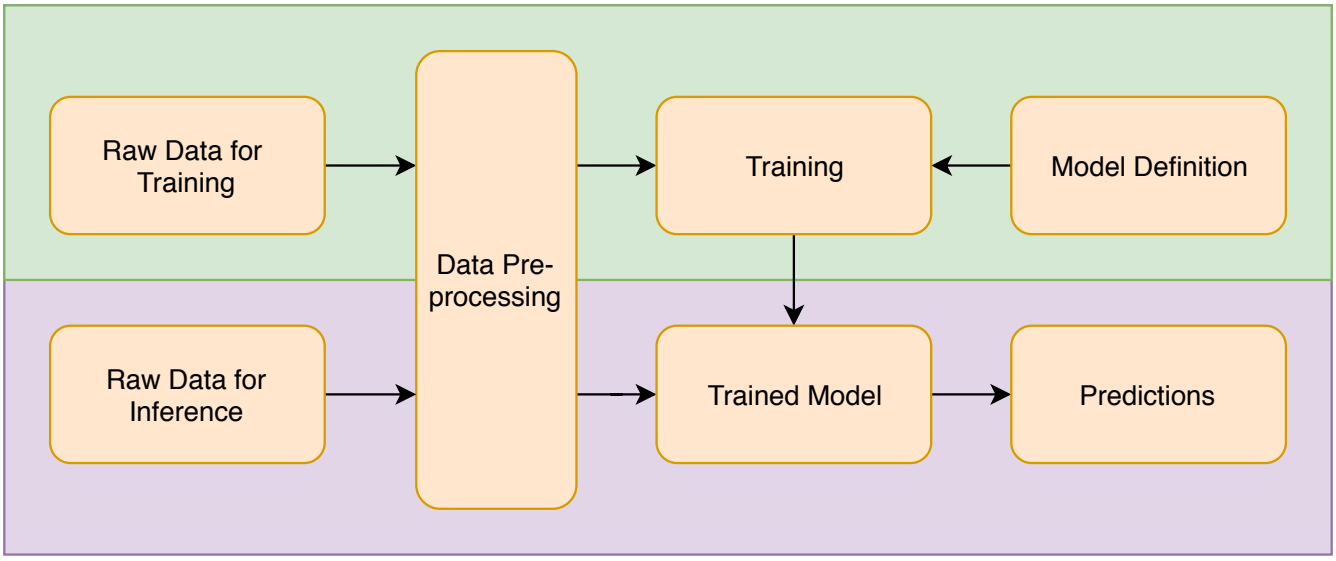


Figure 5: Software Structure: training mode (green) and inference mode (purple).

4 Model

4.1 Pack Padded Sequence

The architecture of NN used is shown in the Figure 6. Since the time-series tensors, shown in Figure 3, have varying sizes, they should be represented using `pack_padded_sequence`, which requires each batch of inputs to be sorted in order of decreasing lengths of the tensors. However, sorting the inputs does not preserve the consistency with the order of the labels of the batch. In order to solve this issue, sorting was reverted for the outputs of the model, which is shown in the top part of Figure 6.

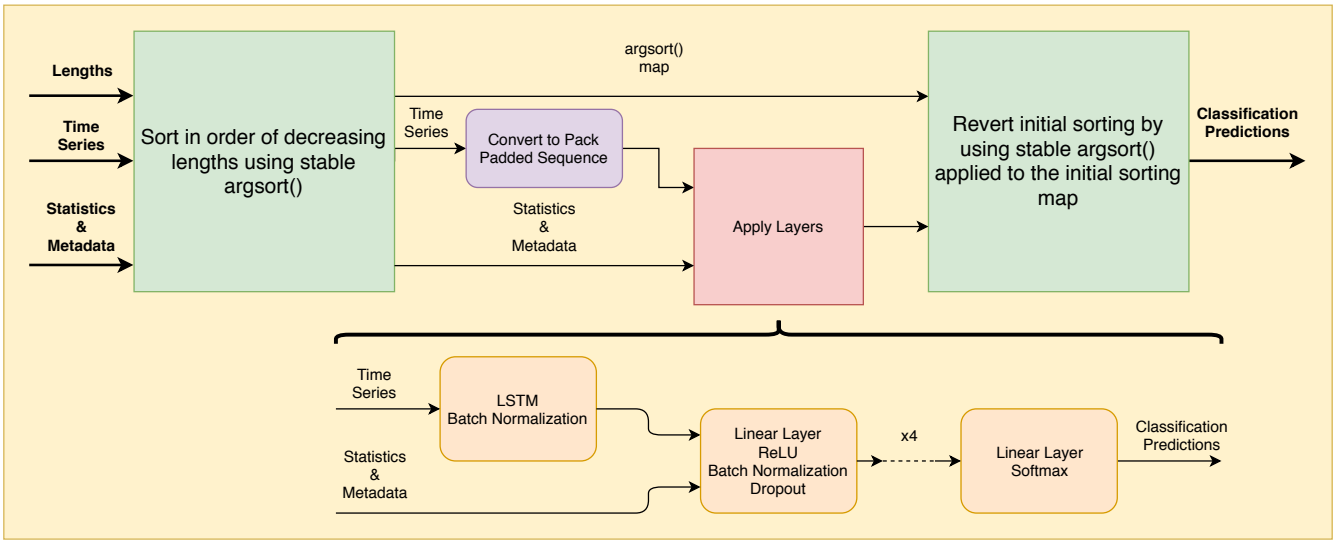


Figure 6: Sorting procedure and the NN structure used in the final model.

4.2 Neural Network

The actual NN structure used is shown in the bottom part of Figure 6. Once the time-series tensors are converted to `pack_padded_sequence`, the time-series data is passed through a set of LSTM layers, followed by batch normalization. Then, the output features from the recurrent layers are combined with the statistical features and metadata (as described in Section 2). The concatenated set of features is then passed through a number of sets of layers, each

Table 2: Parameters of the network structure

Layer	Input Size	Output Size	Notes
LSTM	8	100	num_layers = 2 dropout = 0.1
Linear #1	100+31	512	Dropout: p=0.25
Linear #2	512	256	
Linear #3	256	128	
Linear #4	128	64	Dropout: p=0.5
Linear #5	64	14	

with the same structure: a linear layer, a ReLU, batch normalization and ending with dropout. These sets of layers are repeated 4 times and, at the end, another linear layer is applied, followed by softmax function, which generates the classification probabilities. The parameters used to define the structure of the NN are listed in Table 2.

4.3 Training Loop

The Adam optimizer was used to improve the training time and eliminate the need to use learning rate decay. The loss function used was Weighted Cross Entropy loss, where weights were calculated based on class occurrences as described in Section 2.2. After data balancing, it was decided that using accuracy as measure of model performance during cross-validation is sufficient. Since the best models are saved during every cross-validation, training was done until manually stopped when accuracy remained constant. The batch size used in the final model is 1024 and learning rate is 0.01.

4.4 Predictions

Probabilities for each of the 14 known classes are obtained from the final output layer of the model using a softmax function. Let the probability that an object is in the ith class (by index) be $P(C_i) = x_i$. To generate the outlier class (class 99), we compute the probability that the object is not in any of the other classes [5]:

$$P(C_{99}) = \prod_{i=1}^{14}(1 - x_i)$$

After introducing class 99, the resulting 15 classes may sum to above 1 and so require re-normalization, however this is done by Kaggle automatically.

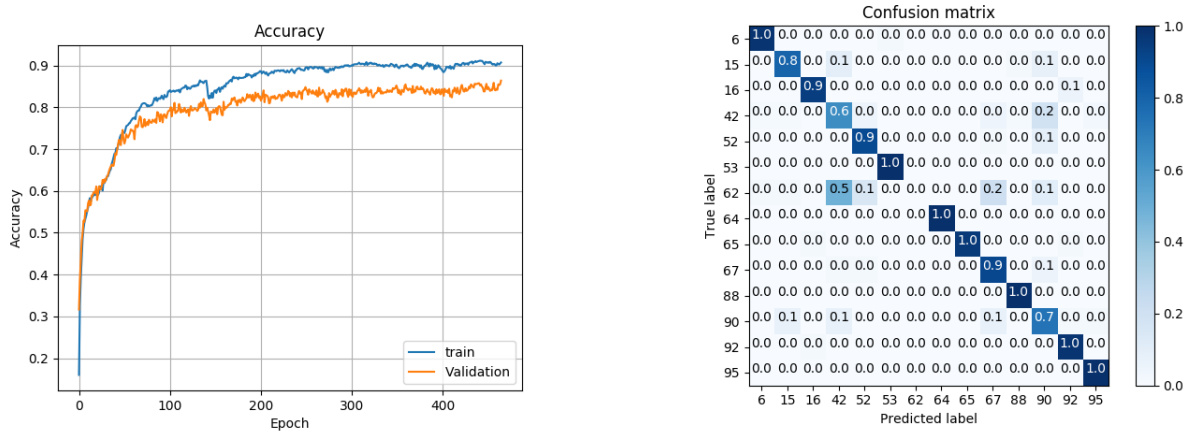
5 Results

Figure 7 shows the results of training the model in Figure 6 on a GPU, using 2 LSTM layers. Over 450 epochs of training, Figure 7 (a) shows that the model was not overfitting, as training and validation accuracy were both increasing until ∼90% and 85%, respectively. The decrease around epoch 150 was likely due to dropout in the model.

The confusion matrix in Figure 7 (b) shows that the model was performing well on most classes, with the exception of class 62 which is often mistaken for class 42. Figure 8 provides some insight as to why this is the case. Compared to Figure 2, which shows a clean curve with a measurable full-width at half-height (a statistic that can be used for analyzing such curves), class 62 represents a stellar burst event. Over the time scales of the training data, it may be impossible to determine the characteristic period of the bursts and hence it is difficult to classify them. Class 42, which is the second most represented class in the training set, is similarly a burst event and is speculated to be a type of supernova [7].

5.1 Competition Results

Our current competition score is a loss of 12.517, in position 836 out of 903. This low ranking is not what we would expect given the promising training results we have seen. We note that the size of the test set is over 500



(a) Training and validation accuracy

(b) Confusion Matrix

Figure 7: Accuracy and Confusion Matrix for the trained model.

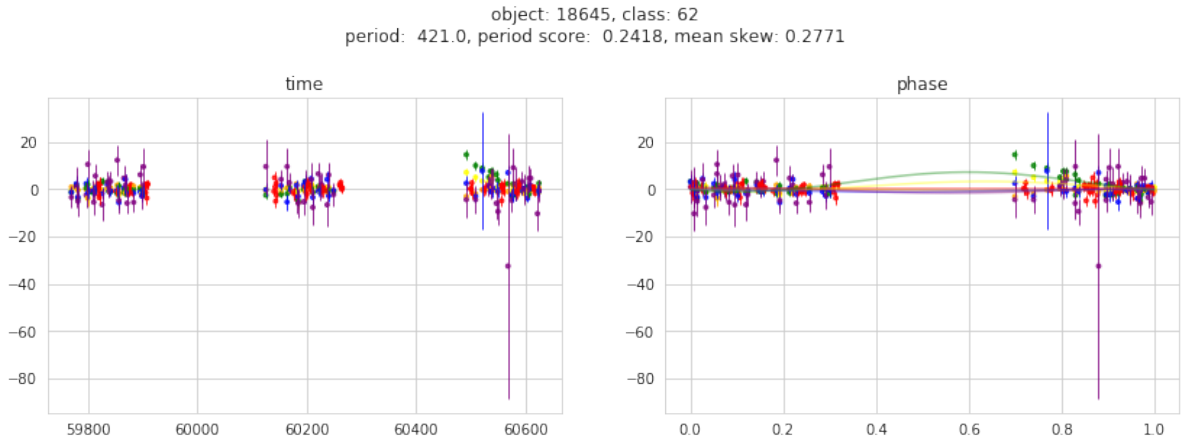


Figure 8: A particular light curve for Class 62 showing a burst event [7]

times larger than the training set, so our validation performance may not be truly reflective of our model's ability to learn. Further, a uniform probability for all classes achieves a loss of 2.708, indicating a likely error in our prediction generating process, but not necessarily our model.

Further, most submissions to the competition are not NNs, and focus on other machine learning techniques [8]. Nonetheless, we suspect that there may be an issue in our final results processing pipeline that we have not been able to detect, and will continue to make submissions to the competition until it's closed on December 17.

6 Ethical Issues

There are no realistic ethical issues of this project. Without using machine learning to process the large amounts of data coming from the LSST, it would not be feasible to build in the first place. Thus, using machine learning is not taking jobs away from data scientists - rather, it is allowing more jobs to be created by facilitating the telescope's existence.

7 Key Learning

Key learnings over the course of this project included properly managing a multi-person coding project. Git was critical for the success of the project, with over 100 commits made to our repository. Proper repository management

was important given that we were developing and training models locally and on cloud.

Thinking carefully about the data and its structure was helpful in developing the model and understanding results. Graphs such as those in Figure 4 alerted us to the need to balance data, whereas plotting raw data such as in Figures 2 and 8 let us understand why our model was behaving as it was.

The use of batch normalization in the model structure has shown to be highly beneficial to the model performance, since it did not allow the weights of the model to grow indefinitely, preventing the model from predicting a certain class for all objects, which was an issue we were facing.

Finally, we note the hazards of jumping straight to a deep learning approach, rather than other machine learning techniques. Our results do not perform nearly as well as non-deep learning approaches submitted to the competition, and many of the top scoring models were made by people with a background in Astrophysics, letting them take advantage of better feature engineering and statistical models.

References

[1] LSST. (2018). Lstt gallery, [Online]. Available: https://gallery.lsst.org/bp/#/folder/2334407/51647471.

[2] ——, (2018). About lsst, [Online]. Available: https://www.lsst.org/about.

[3] T. J. Allam, R. Hlozek, and C. M. Peters, "The photometric lsst astronomical time-series classification challenge (plasticc): Data set," *Cornell University Library*, 2018. [Online]. Available: https://arxiv.org/abs/1810.00001.

[4] M. Apers. (2018). The plasticc astronomy "starter kit", [Online]. Available: https://www.kaggle.com/michaelapers/the-plasticc-astronomy-starter-kit.

[5] Siddhartha. (2018). Simple neural net for time series classification, [Online]. Available: https://www.kaggle.com/meaninglesslives/simple-neural-net-for-time-series-classification.

[6] G. Lemaitre, F. Nogueira, and C. Aridas. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning, [Online]. Available: http://www.jmlr.org/papers/volume18/16-365/16-365.pdf.

[7] Mithrillion. (2018). All classes light curve characteristics (updated), [Online]. Available: https://www.kaggle.com/mithrillion/all-classes-light-curve-characteristics-updated.

[8] Kaggle. (2018). Plasticc kernels, [Online]. Available: https://www.kaggle.com/c/PLAsTiCC-2018/kernels.