Machine Learning Engineer Nanodegree

Capstone Project

Daniel Bank

December 3rd, 2018

Overview

Photometric LSST Astronomical Time Series Classification Challenge

The Large Synoptic Survey Telescope (LSST), currently under construction in the Atacama desert of Chile, is the most awesome telescope that mankind will have built thus far in history. It features an 8.4m mirror, 3.2 gigapixel camera, and 6.5m effective aperture [1]. Every night, it will gather between 20 and 40 TB of image data of the southern sky. While sifting through that much data to identify cosmological objects would be impractical for humans, it is a perfect use case for machine learning.

In preparation for the massive amounts of observations that will come from the LSST, an open data competition has been launched on Kaggle called the Photometric LSST Astronomical Time Series Classification Challenge (PLAsTiCC). The goal of the challenge is to correctly classify time-varying cosmological objects in simulated astronomical time-series data similar to the data that the LSST will generate.

I first became aware of this challenge while browsing the Kaggle competitions. My personal interest in astronomy stems partly from my background as a Physics major but more so from son, Arthur, who loves everything about the night sky. He has taught me to really appreciate the beauty of the cosmos.

Problem Statement

PLAsTiCC includes a 60 MB training dataset of labeled data. That is to say, the sources in the

training data have known classifications that our model can be trained on. Our trained model must then for every object i in the test dataset predict the probabilities of it belonging to class j. For any object, these probabilities should sum to 1.

$$0 \leq P_{ij} \leq 1 \qquad \forall i, j$$

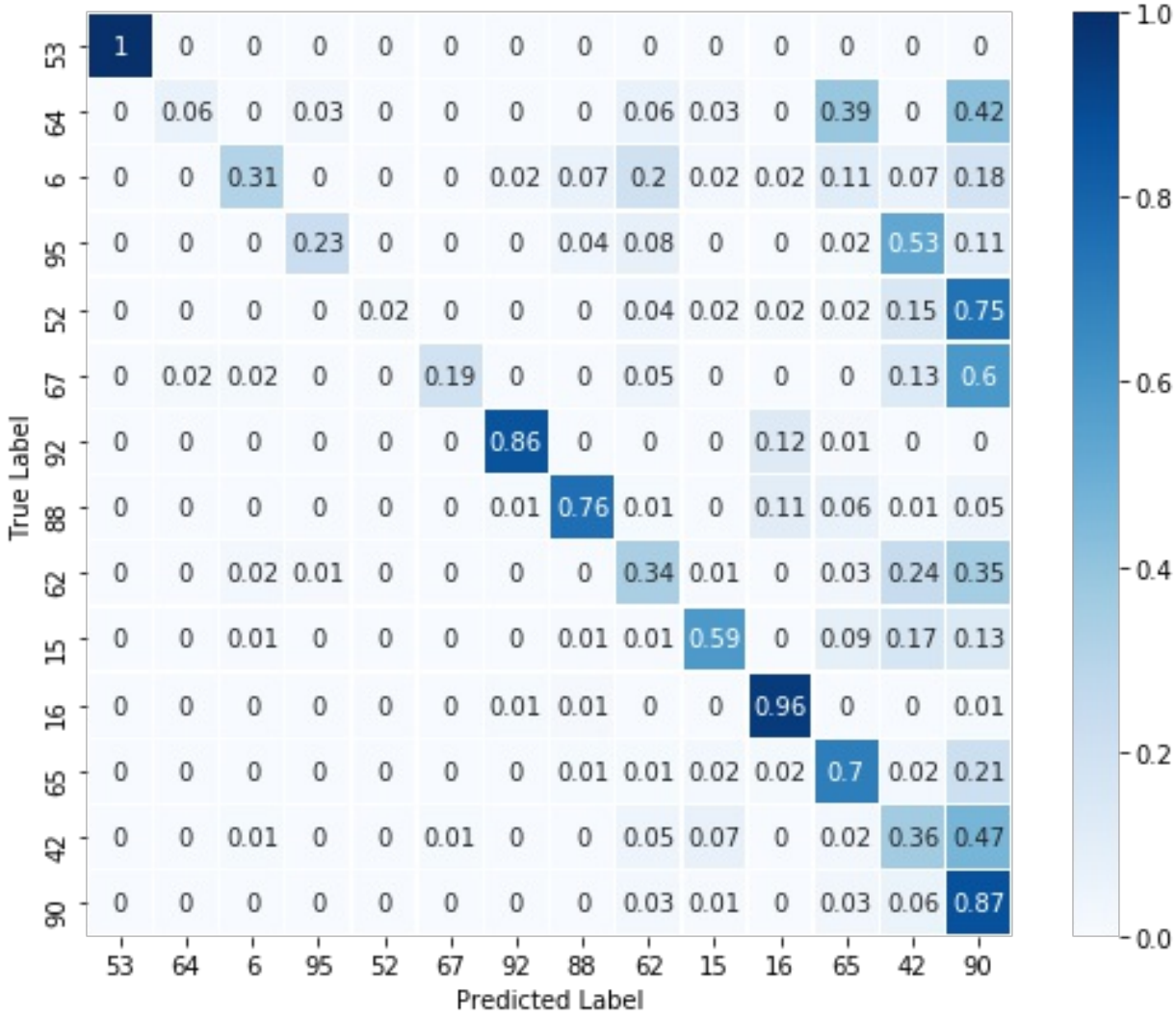
$$\sum_{j} P_{ij} = 1 \qquad \forall i$$

The largest `Pij` for object i is what our model will finally classify it as. These predicted classifications can then be compared to the actual classifications for the objects in the test dataset to obtain an accuracy for our model.

Our objective is to beat the accuracy of the PLAsTiCC Astronomy Classification Demo Classifier.

Evaluation Metrics

The PLAsTiCC team has provided a demo classifier for the challenge [2]. The performance of the demo classifier on the test data can be visualized in the following confusion matrix:



In this matrix, we can see the model's predicted labels versus the true labels for the test set. Ideally, this confusion matrix would be the Identity matrix, with all values along the diagonal being 1 and everything else being 0. That would mean that the model predicted the correct label 100% of the time.

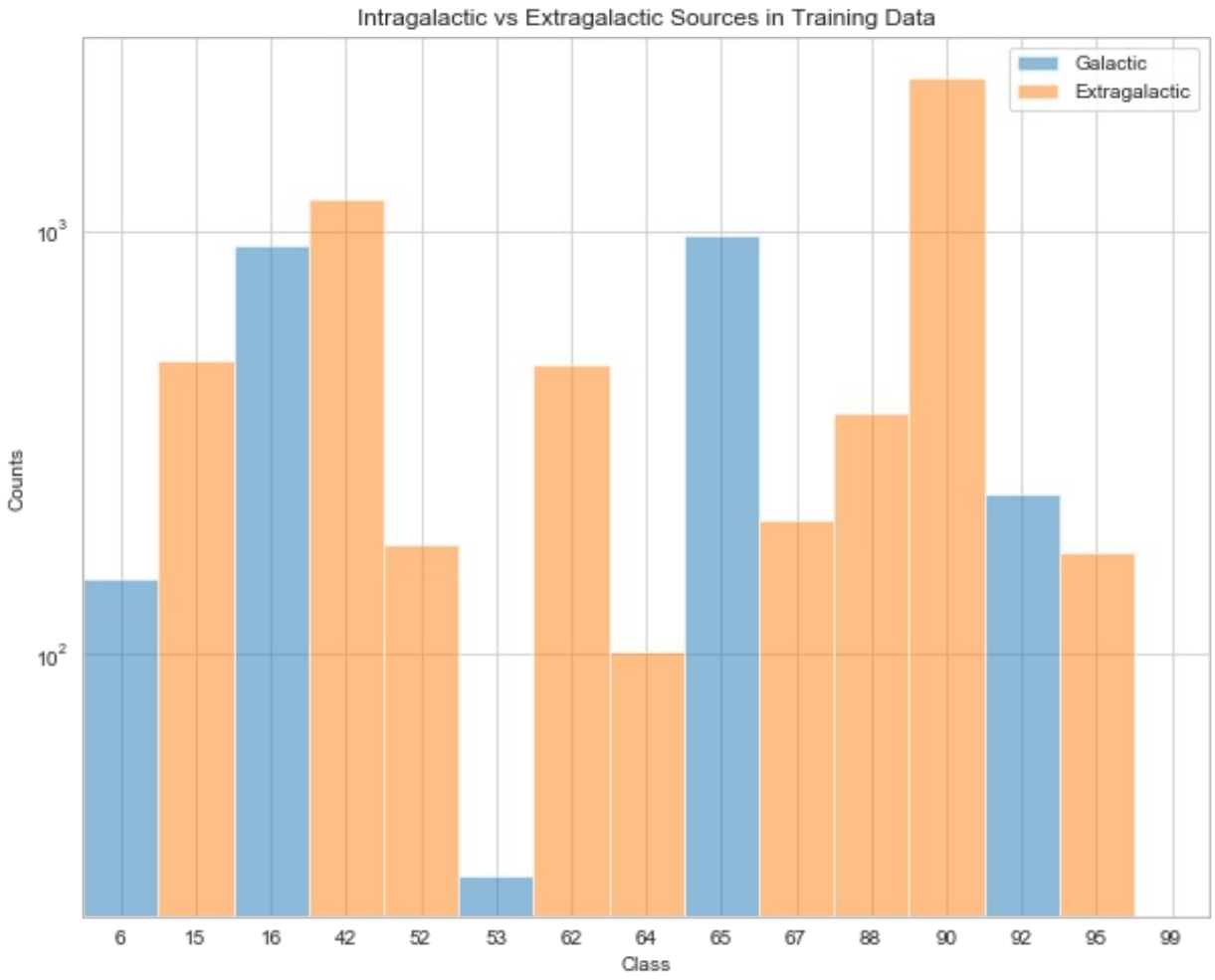
When designing my classifier, I will generate a similar confusion matrix so that my results can be objectively compared to the demo classifier. My objective is to have an average prediction accuracy per label that is higher than the demo classifier's accuracy of 0.518 (see calculation below):

```
(1.00 + 0.06 + 0.31 + 0.23 + 0.02 +
 0.19 + 0.86 + 0.76 + 0.34 + 0.59 +
 0.96 + 0.70 + 0.36 + 0.87) / 14
= 0.518
```

Analysis

Data Exploration

One interesting observation is that classes in the training data set divide evenly into intragalactic and extragalactic categories [3]. There are no classes that are present in both categories. This suggests that we can partition the dataset along these lines and train two separate CNN models to predict using the subset of classes for each category.

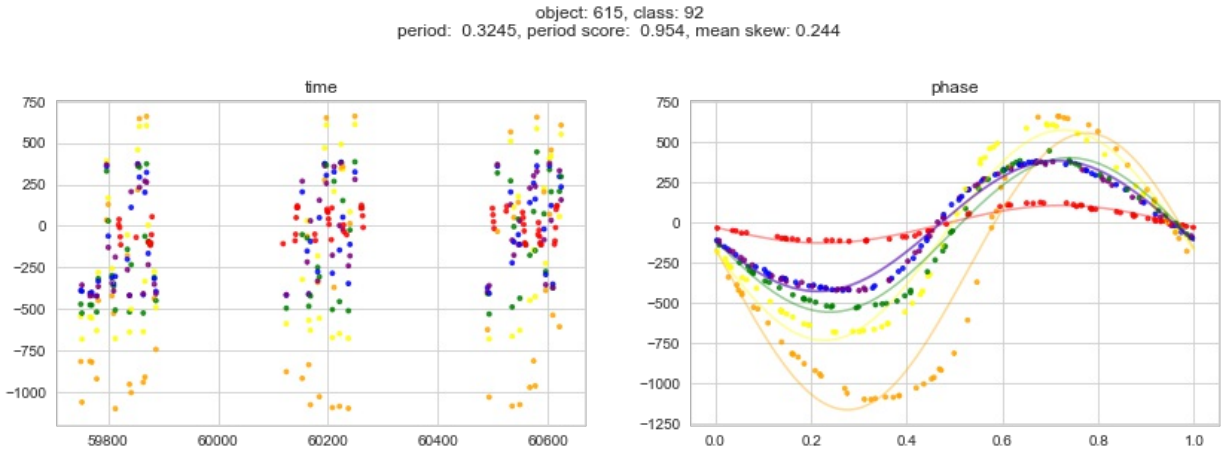


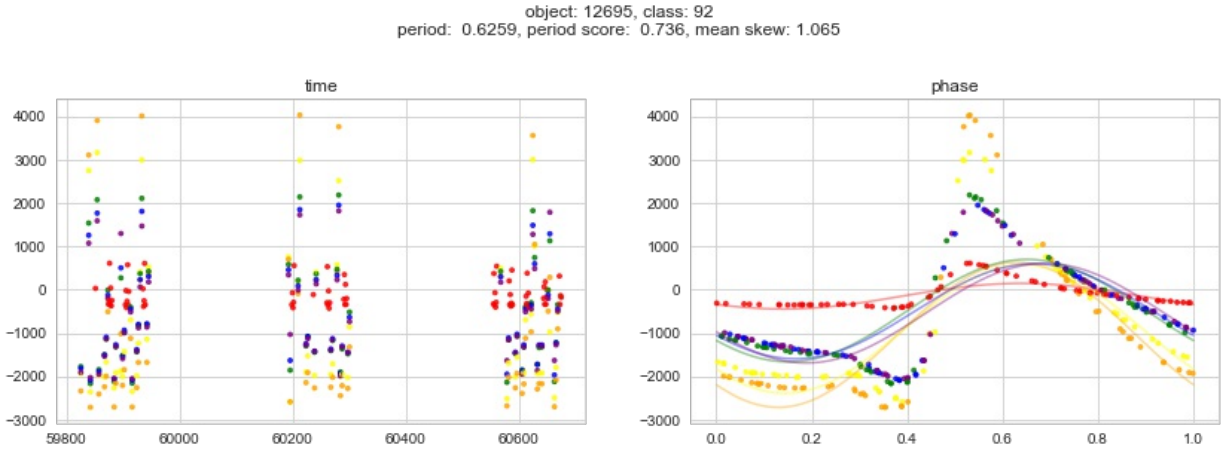
Another point of interest is that Class 99 (the unknown classification) is not present at all in the training dataset. This raises a serious problem: How to predict this mystery class when it is not represented at all in the training data? While predicting Class 99 is not necessary for the limited scope of this report (our test data is split from the training dataset using `sklearn.model_selection.StratifiedKFold`), it nonetheless gives us an appreciation for the real difficulty of this competition. The problem is further compounded by another challenging aspect of the competition, that the training dataset is a small subset of the test dataset and also a poor representation of it. This design decision was made to imitate the real-world

challenges that astronomers face when trying to classify cosmological objects.

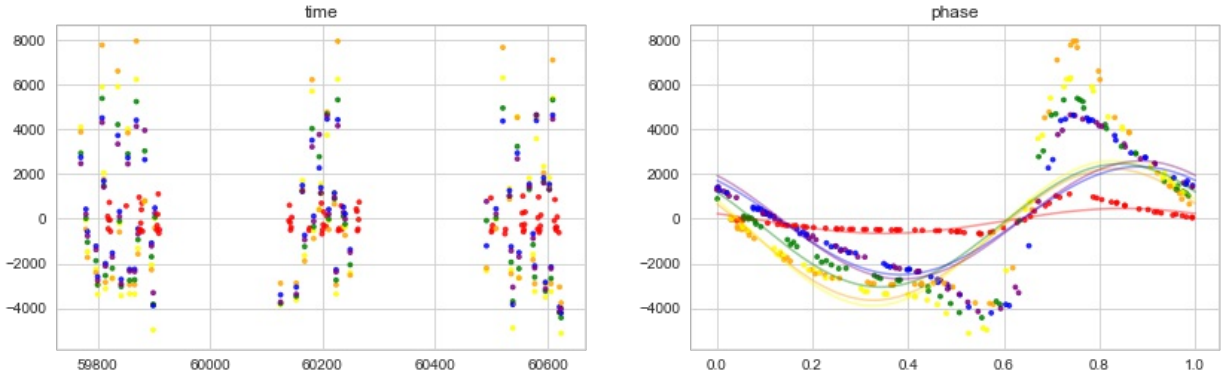
Exploratory Visualization

It is helpful to look at the example passband data for the various classes to get an understanding of what the data looks like and understand some of the distinguishing features between classes. As there are 14 classes represented in the training dataset, we will only examine two of the more interesting classes: Class 92 and Class 42. Class 92 are highly periodic while Class 42 exhibit what appear to be "burst" events with diminishing activity afterwards. The author of the code that produced this visualization, Mithrillion, theorizes that Class 92 are regular variable stars and Class 42 are supernovae events [4].

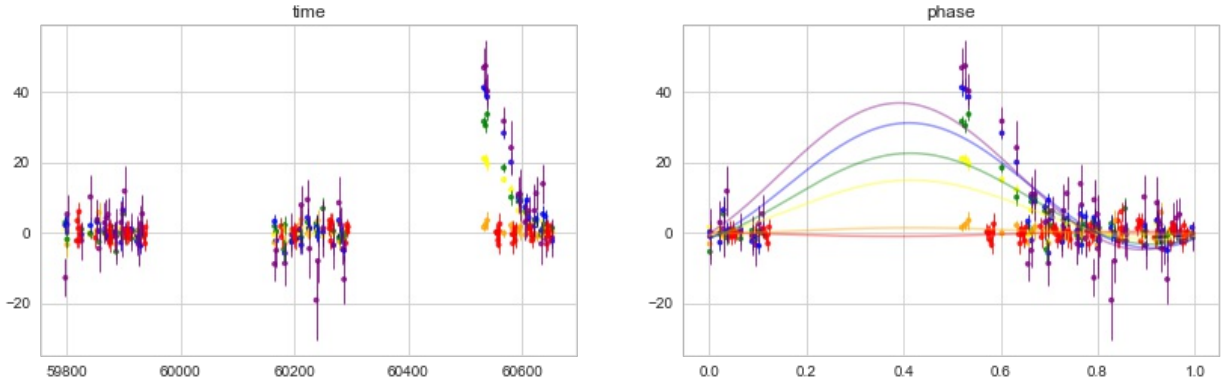




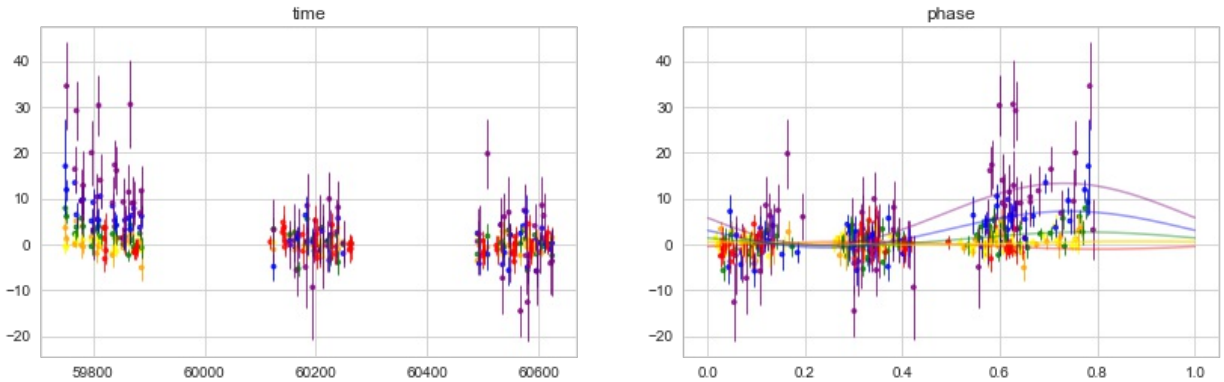
object: 26161, class: 92
period: 0.5577, period score: 0.7778, mean skew: 0.5086

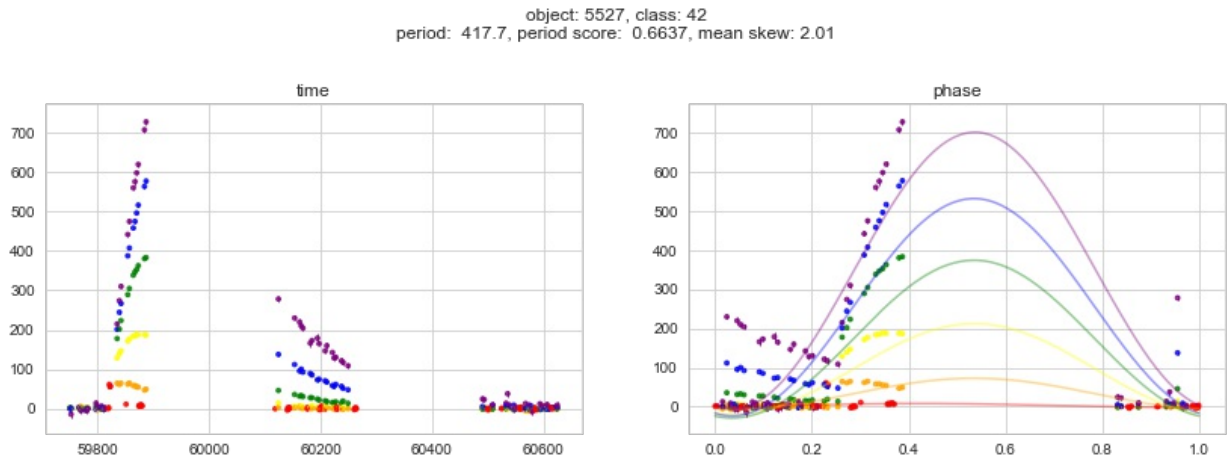


object: 730, class: 42
period: 424.7, period score: 0.5918, mean skew: 1.633



object: 1632, class: 42
period: 0.998, period score: 0.2723, mean skew: 0.2874





Benchmark

The demo classifier provided by the PLAsTiCC team was designed as follows:

1. The raw tabular data is converted into a form suitable for analysis. Using the **cesium** package for Python, **Timeseries** objects are constructed for each light curve.
2. The data is split into a training and test set.
3. The dimensionality of the training data is reduced using Principle Component Analysis (PCA). The result of this step are features which are linear combinations of the passband data. With just eight (8) features, we can account for almost all of the explained variance.
4. The reduced training dataset is fit with a Random Forest Classifier and predictions are obtained for the test dataset.

PLAsTiCC Prediction Pipeline

Using Higepon's CNN based model as a starting point [5], we design a prediction pipeline as follows:

1. Data Preprocessing: time series features for each object are scaled to unit variance.
2. Partitioning Intragalactic and Extragalactic Sources: the entire dataset is split into two segments based upon whether or not the spectroscopic redshift equals 0 (intragalactic) or not (extragalactic).
3. Data Transformation: the time series data for each object is transformed into a two dimensional monotone image which can be provided as input to a CNN.
4. CNN Model Implementation: Two separate CNN models are created and trained on the intragalactic and extragalactic datasets.

Data Preprocessing

The data for this challenge has been provided in a curated form by the PLAsTiCC Team. As such most data preprocessing techniques, like defaulting missing data values or one-hot encoding text categories, are not required for our pipeline. We do make use of `sklearn.preprocessing.StandardScaler` to standardize the time-series features (`mjd`, `flux`, and `flux_err`). Standardizing means removing the mean and scaling the feature to unit variance [6]. This is helpful for CNN training.

Partitioning Intragalactic and Extragalactic Sources

As seen in the Data Exploration section, we can narrow number of possible classifications to 5 out of 14 possibilities for galactic sources and 9 out of 14 possibilities for extragalactic sources. It makes sense to partition our data along these lines and train two separate CNN's for each category. In this manner, we can reduce the number of erroneous classifications for each model.

The determination for whether a source is intragalactic or extragalactic is done using the `hostgal_specz` field in the training metadata. This field represents the spectroscopic redshift of the source and it will (effectively) equal 0 for intragalactic sources.

Data Transformation

CNN's are commonly used for classifying visual images which are two-dimensional, so we should convert our data into a form that is appropriate for this model. Using the time series data for each object, we construct a monotone image with width of size `len(flux) + len(flux_err) + len(detected)` and height of size `# of passbands`. The "images" for all objects are used for training the CNN and making predictions.

The general steps for doing the 2-D transformation are to:

- Group the time-series data (flux, flux_err, detected) by passband per object
- Take the Matrix Tranpose (m x n) => (n x m)
- Extract object labels for the passband == 0 row for each object

CNN Model Implementation

Our CNN architecture makes use of the following Keras layers:

- `Conv1D`: a layer where the linear algebra of weights and inputs occurs.
- `Activation`: a layer which normalizes outputs to a given range. In our model, we use a ReLU (Rectified Linear Unit) activation function which clamps negative values to 0 and leaves positive values unaffected.
- `MaxPooling`: a layer which collects the outputs of the previous layer into a single neuron.
- `BatchNormalization`: a layer that normalizes the activations of the previous layer.
- `Dropout`: a layer that randomly sets a fraction of the inputs to 0 to permit all neurons to train, thereby reducing overfitting.
- `Lambda`: a layer which returns performs a desired function. In our architecture, the function is to average it's inputs which serves as a Global Average Pooling layer. GAP layers are structural regularizers that prevent overfitting of the overall structure [7].
- `Dense`: the final layer is a Dense layer that maps the inputs into the respective probabilities for each classification (shaped 5 for intragalactic, shaped 9 for extragalactic) using a *softmax* activation function.

The CNN architecture itself went through several iterations:

Initially, I started with eight `Conv1D` layers with a filter size (output space) of 64. My idea here was that a smaller filter sizes would train quickly and the limitations of size could be mitigated by adding more layers. Running over 100 epochs, this model achieved an accuracy of 0.506. I scaled the number of epochs to 10,000 and the model accuracy went down slightly to .503. This result combined with the Model Accuracy graphs suggested that the model was overfitting.

As a next step, I decided to add in `Dropout` layers with a rate of 0.2. The intention of this change was to mitigate the overfitting. The resulting model achieved .509 accuracy, a negligible improvement.

Next, I decided to use three `Conv1D` layers with filter sizes 128, 256, and 512, respectively. I left the `Dropout` layers but experimented with rates of 0.1, 0.3, and 0.7. With this model, the accuracy after 100 epochs was .470. Disappointing.

Although increasing the filter size seemed to be reducing accuracy, I tried one more experiment with two `Conv1D` layers both filter sizes of 1024 and two `Dropout` layers with rates of 0.2. This model achieved an accuracy of .478. At this point, I finally abandoned the larger filter sizes and returned to a model with a filter size of 256. In retrospect, this seemed most appropriate because the input shape had a size of 216 rows by 6 columns.

My final change was to increase the kernel size of the convolutional layers to 160 and include only one `MaxPooling` and one `BatchNormalization` layer just prior to the `Lambda` (Global Average Pooling) layer. I also included one `Dropout` layer with a rate of 0.5. After 1000 epochs, this achieved the best accuracy I had observed which will be discussed in the Justification section below.

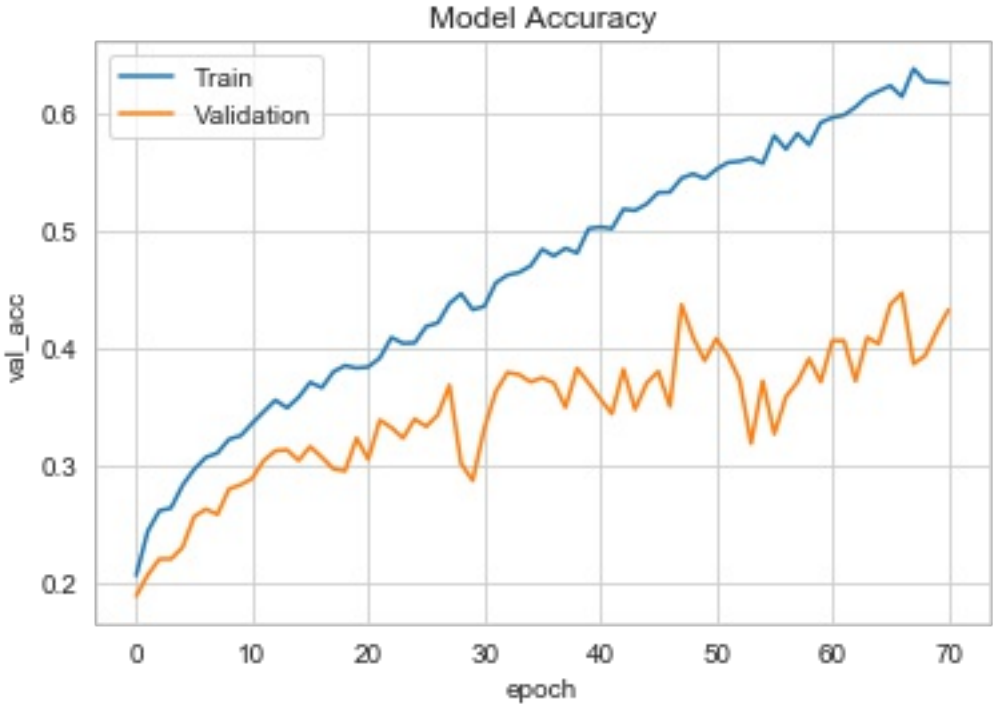
Results

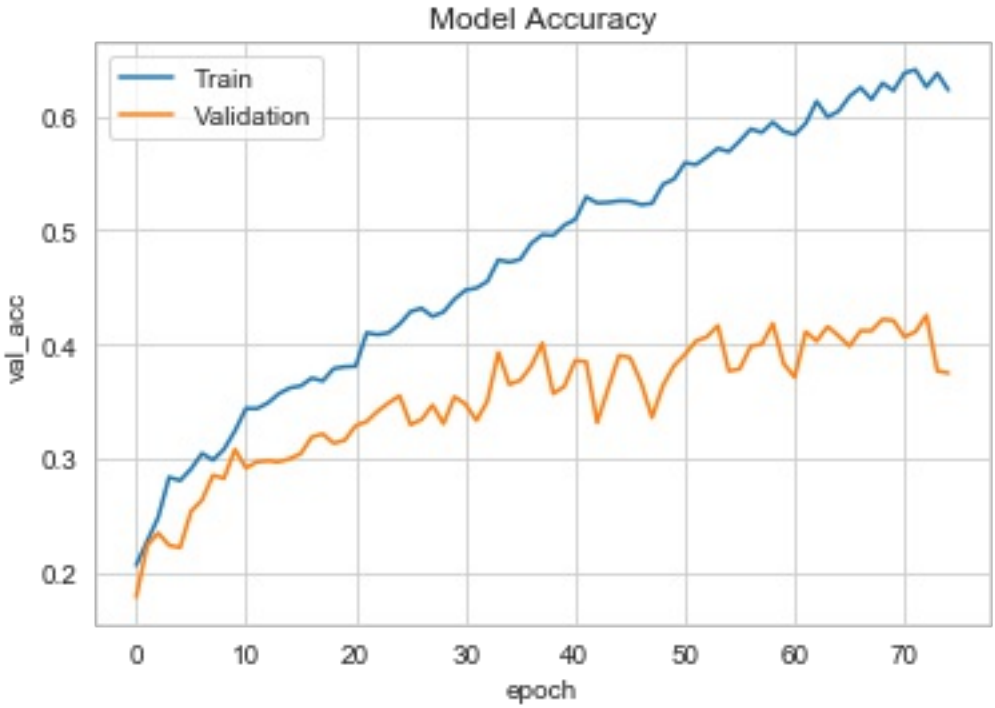
Model Evaluation

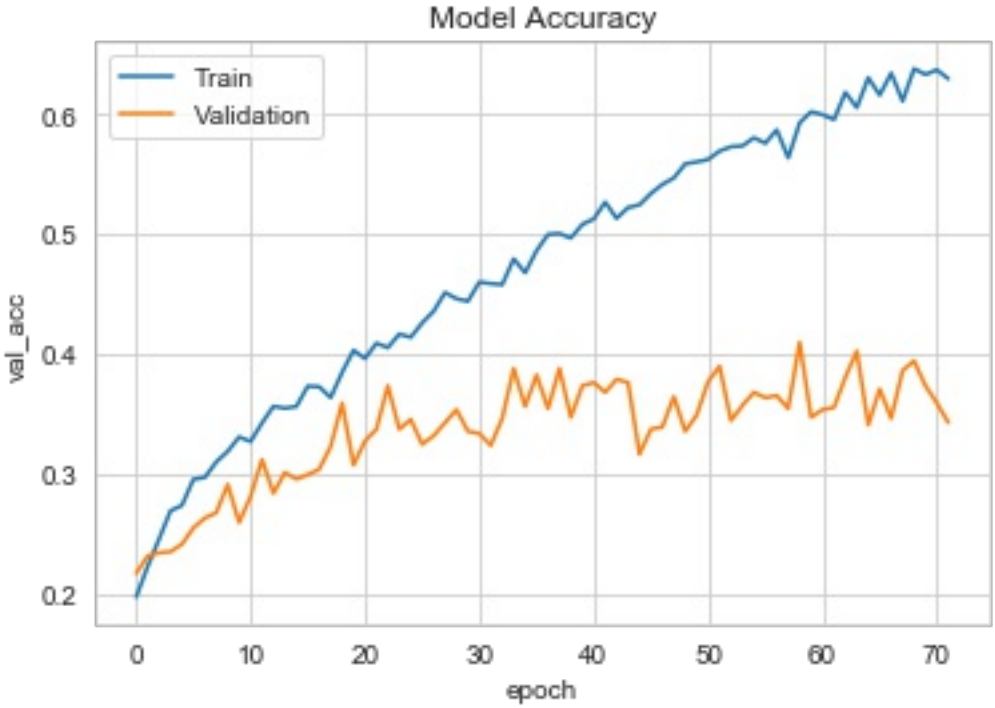
The summary of the architecture for one of the CNN's is shown below. Here the input is a (216 x 6) matrix where 216 is `len(flux) + len(flux_err) + len(detected)` and 6 is the number of passbands. The final model has a total of 249,353 parameters with 246,016 of those being in the convolutional layers, 1024 in batch normalization layers, and 2313 in the dense layer that produces the predictions. Of these parameters, 512 are non-trainable. The model trains in 2,128 seconds (35.46 minutes).

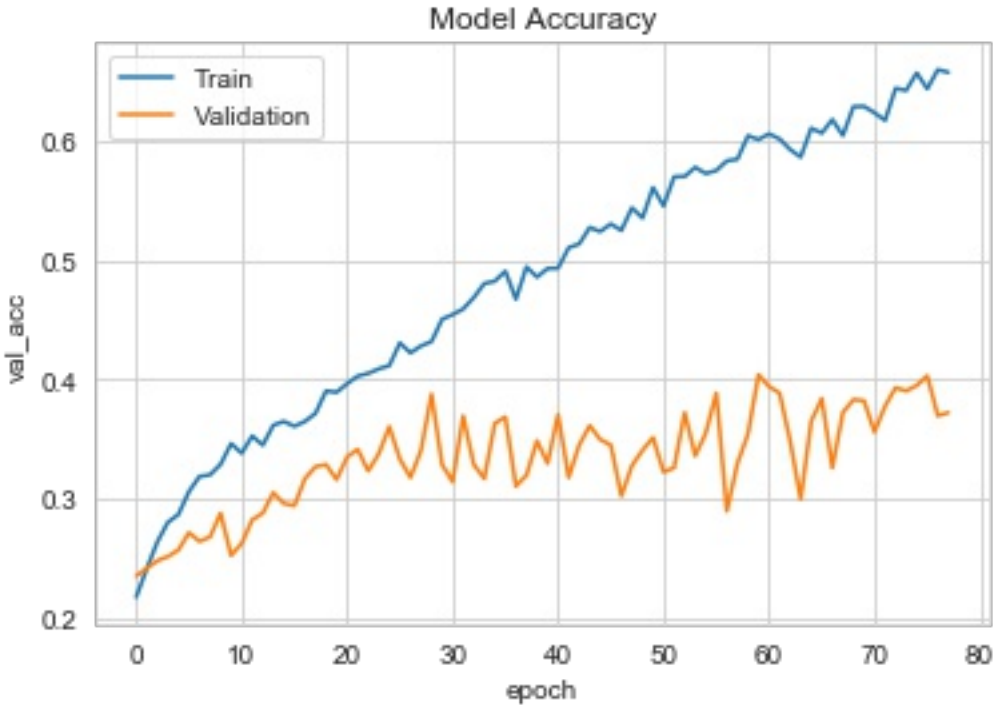
Model Validation

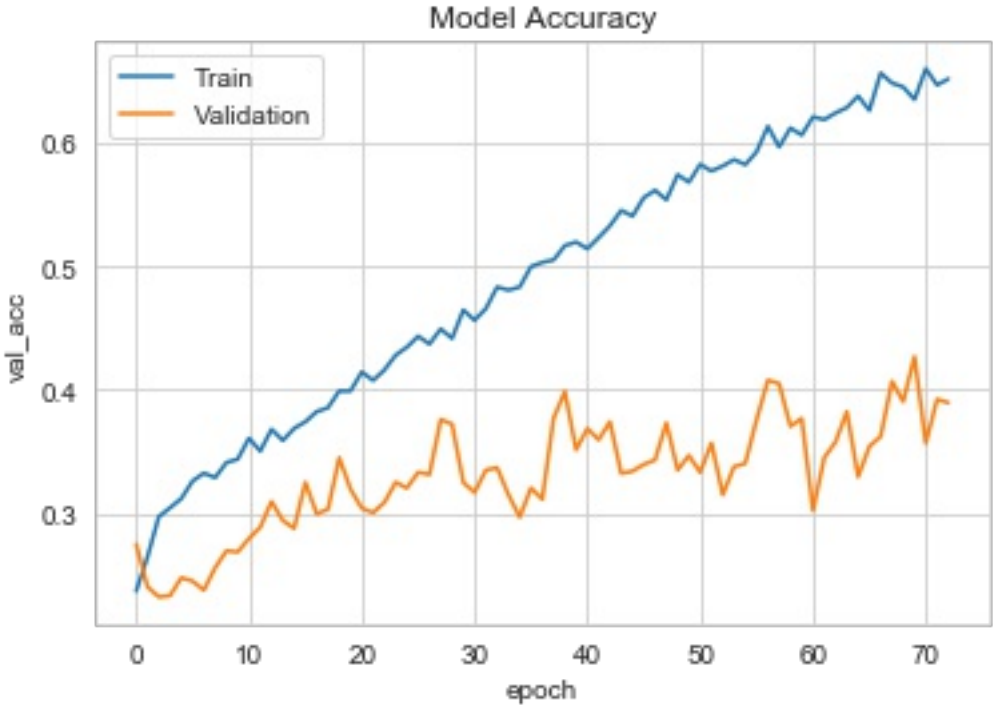
Model accuracy has been graphed versus epoch for both the intragalactic and extragalactic datasets for each of the five folds from the `StratifiedKFold` step. The intragalactic dataset has better training and validation results, consistently achieving above 0.7 accuracy. The extragalactic dataset only reaches about about 0.35 accuracy. In all the graphs, we see evidence of overfitting. The validation score increases until reaching a plateau point around which it oscillates, sometimes with quite large swings in accuracy.

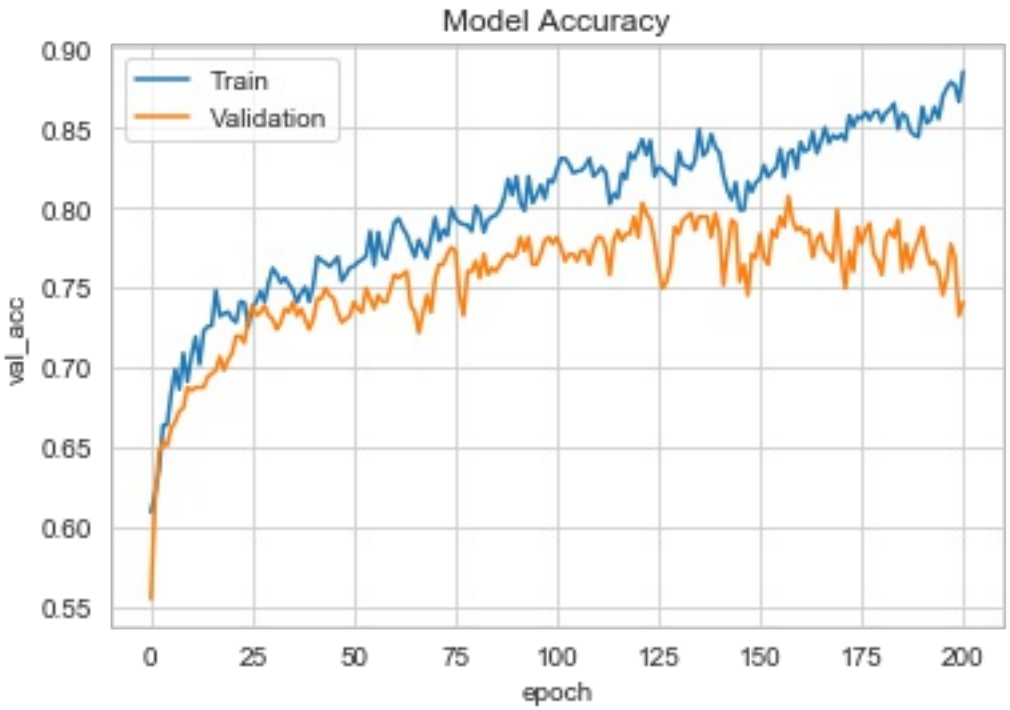


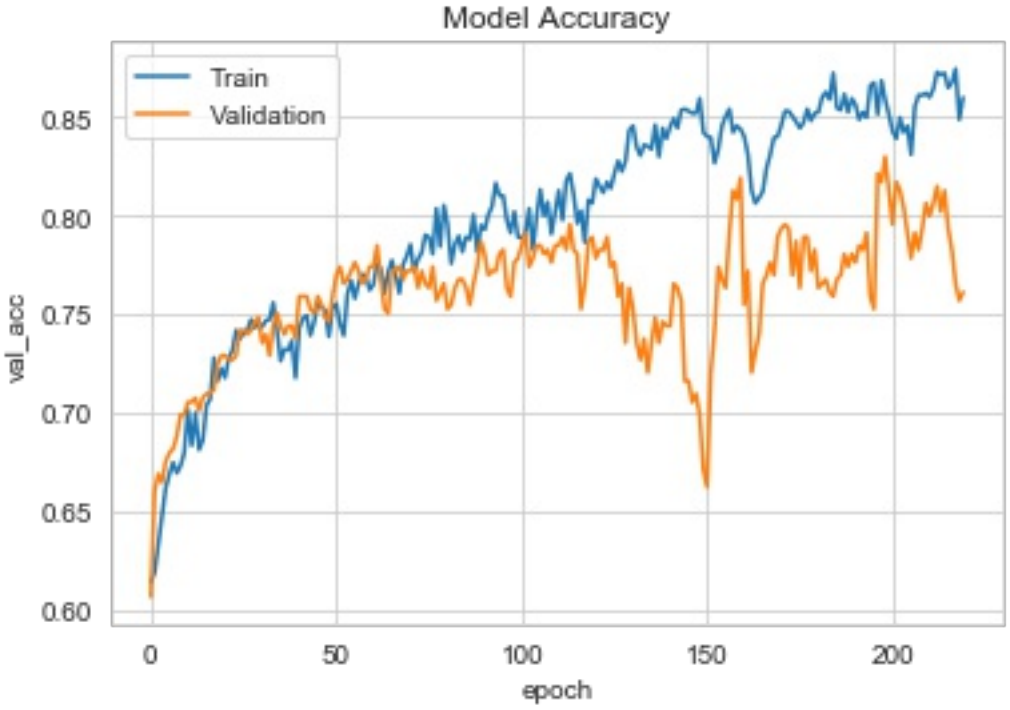

Model Accuracy
Train
Validation
val_acc
epoch
0.2
0.3
0.4
0.5
0.6
0
10
20
30
40
50
60
70

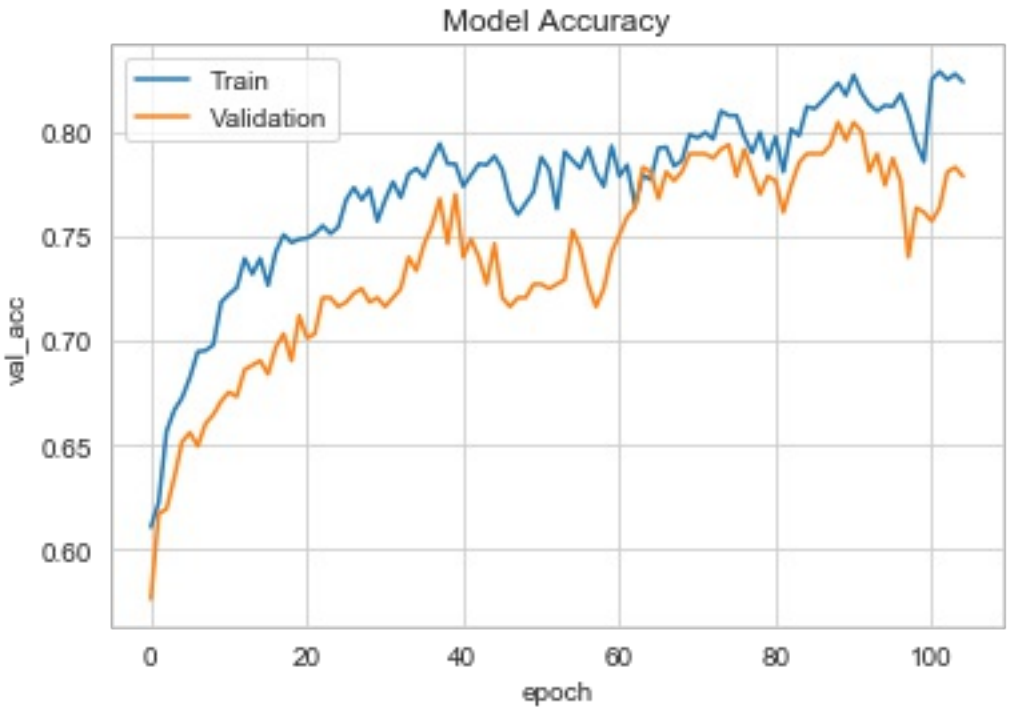

Model Accuracy
Train
Validation
val_acc
epoch
0.2
0.3
0.4
0.5
0.6
0
10
20
30
40
50
60
70

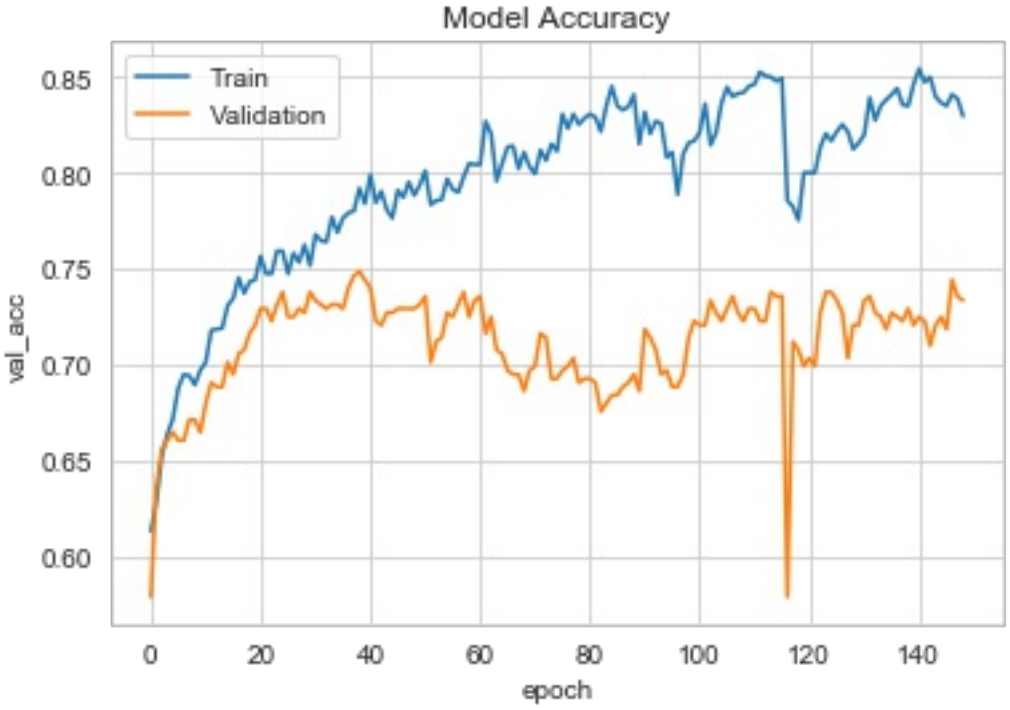

Model Accuracy
Train
Validation
val_acc
epoch
0.2
0.3
0.4
0.5
0.6
0
10
20
30
40
50
60
70
80

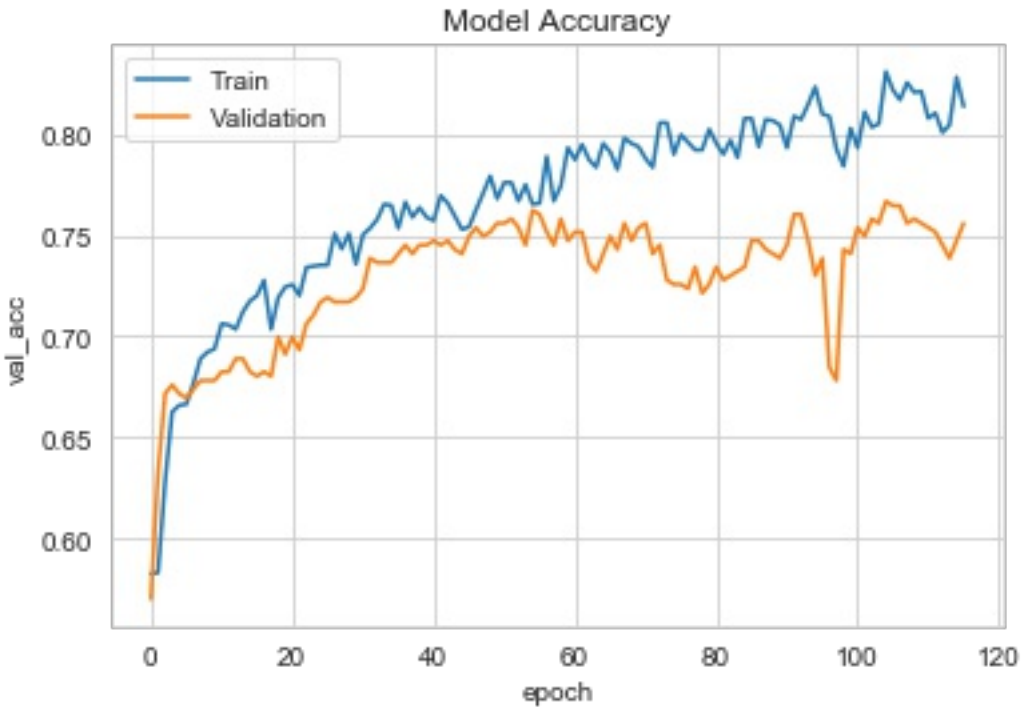

Model Accuracy
Train
Validation
val_acc
epoch
0.3
0.4
0.5
0.6
0
10
20
30
40
50
60
70


Model Accuracy
Train
Validation
val_acc
epoch
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
25
50
75
100
125
150
175
200


Model Accuracy
Train
Validation
val_acc
epoch
0.85
0.80
0.75
0.70
0.65
0.60
0
50
100
150
200


Model Accuracy
Train
Validation
val_acc
0.80
0.75
0.70
0.65
0.60
0
20
40
60
80
100
epoch


Model Accuracy
Train
Validation
val_acc
0.85
0.80
0.75
0.70
0.65
0.60
0
20
40
60
80
100
120
140
epoch



Justification

Our CNN-based model yields the following confusion matrix with an average prediction accuracy per label of 0.546. This is only 5.4% better than the demo classifer's score of 0.518, which isn't meaningfully enough to declare it a better solution. Furthermore, it required 35 minutes of training compared to the demo classifier notebook which ran in under 5 minutes. As such, I would not consider it to have adequately solved the problem.

```
(0.90 + 0.82 + 0.62 + 0.38 + 0.15 +
 0.39 + 0.86 + 0.72 + 0.32 + 0.62 +
 0.73 + 0.79 + 0.09 + 0.26) / 14
= 0.546
```

Conclusion

Reflection

The PLAsTiCC competition has been challenging for me on many levels. Even with the example kernels available on Kaggle, it was difficult to get started because the data was structured in a more complicated manner than anything I had previously worked with. Specifically, we have object metadata in one CSV file and time series data for the objects in another. Whereas with the object metadata, the relationship between object to row is one to one, with the time-series data it is one to many. Transforming the time-series data into a workable form required grouping it and merging it with the metadata. I found that these steps of the programming were the most arduous.

Optimizing the CNN architecture was another challenge because it was time-consuming. Whereas it was fun to anticipate the model's accuracy after a long training run, it was also

frequently disheartening. This was especially true when there were bugs and typo's in the code that aborted a training run, which I didn't realize it until returning from a coffee break.

Lastly, although outside of the scope of this report, I gained an appreciation for the true complexity of the competition itself. Firstly, we are given a training set which is a tiny subset of the test data and is also a bad representation of it. This design choice was made to simulate the challenge astronomers will face when they begin analyzing the LSST dataset which will dwarf any currently available telescopic data. Secondly, the test data includes a new class not present in the training data: Class 99. This design choice was made to represent the new cosmological sources that astronomers expect to find when they begin analyzing the LSST data. Taken together, these two points pose an incredibly difficult problem for us competitors: How can we create a model that identifies this mystery class in the test data when we have such a limited training dataset which doesn't include it in the first place?

The approaches I have seen for solving this have been quite creative. One way is to assign a constant prediction (e.g. 0.2) to Class 99 so that the model must have at least that much confidence in a different class in order to pick it. However, this is reported to too heavily bias Class 99. Another approach is to assign Class 99 a probability of `1 - max(other class probabilities)`. This again means that the model must have at least 50% confidence in another class in order to pick it. This approach also is reported to over-bias Class 99. Finally, one last approach is to assign Class 99 a probability equal to the product of the other class probabilities. While I did not have a chance to approach this problem in the scope of the report, it is nonetheless interesting to see the varying ways people are approaching it.

Improvement

I was disappointed with the result of the CNN-based model. I expected that it would perform significantly better than the demo classifier, however the achieved results were statistically similar. Moreover the CNN-based model required significantly more training time. It was especially surprising that dividing the data between intragalactic and extragalactic sources did not improve results considering that this reduced the number of possible classifications each object could have.

We might be able to improve the model's accuracy by refining the CNN archiecture, which based on my own experimentations yielded accuracies between 0.471 and 0.546. However, I am not convinced that this is a good route to investigate. Instead, I would experiment with engineering more features or using a different type of neural network altogether. Specifically a Long Short Term Memory (LSTM) Network could be utilized as it is more specialized for classifying time-series data [8]. The challenge here is that the length of the time series data for the various objects is not uniform, which is a requirement for RNN's. To workaround this we could pad the data with 0's to make it uniform, but this could cause other unintended consequences.

Closing Remarks

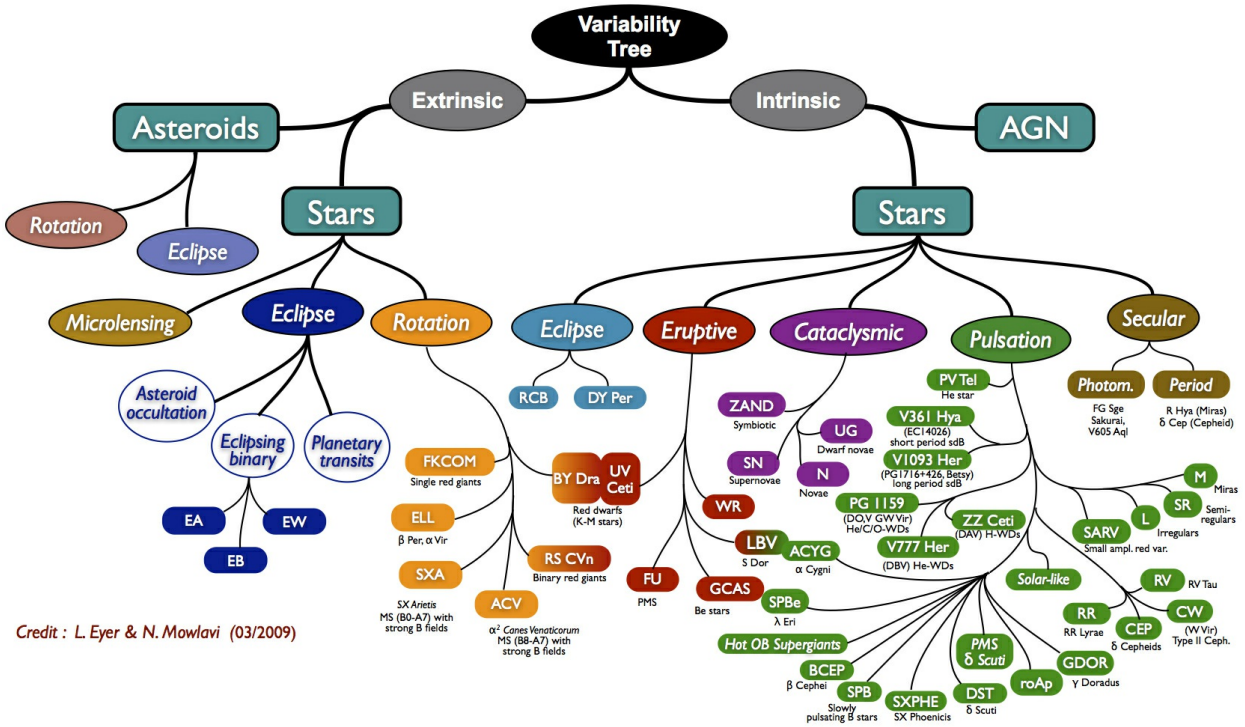


Figure 5: The hierarchy of variable and transient sources. Credit: Laurent Eyer and Nami Mowlawi, Université de Genève

The tree diagram above illustrates every type of light source seen in the night sky that mankind currently knows about. This tree is predicted to grow when the LSST comes online and begins collecting 20-40TB of light curve data every night. The task of sifting through this enormous amount of data and finding those new types of light sources is impractical for humans but a perfect use case for machine learning. Nonetheless, it is a huge technical challenge. Finding those interesting sources whose light curve is different than the known classifications is like finding a needle in a haystack. In the nomenclature of PLAsTiCC, astronomers are trying to detect Class 99 in a huge test dataset with models trained on on a small training set that doesn't include it.

In this report, we focused on a test dataset split from the training dataset and attempted to develop a CNN-based classier that was more accurate than the PLAsTiCC demo classifier. Our achieved accuracy was 0.546 which was 5.4% better than the demo classifier's accuracy of 0.518. Unfortunately our classifier took over 7 times as long to train for that minimal improvement.

References

[1] - PLAsTiCC Astronomy Starter Kit

[2] - The PLAsTiCC Astronomy Classification Demo

[3] - Naive Benchmark - Galactic vs Extragalactic

[4] - All Classes Light Curve Characteristics

[5] - Higepon's CNN-based Model for PLAsTiCC

[6] - sklearn.preprocessing.StandardScaler

[7] - Network in Network

[8] - Long short term memory