Visualization for Online Image Sequence Classification of Astronomical Events

Nicholas Ruta

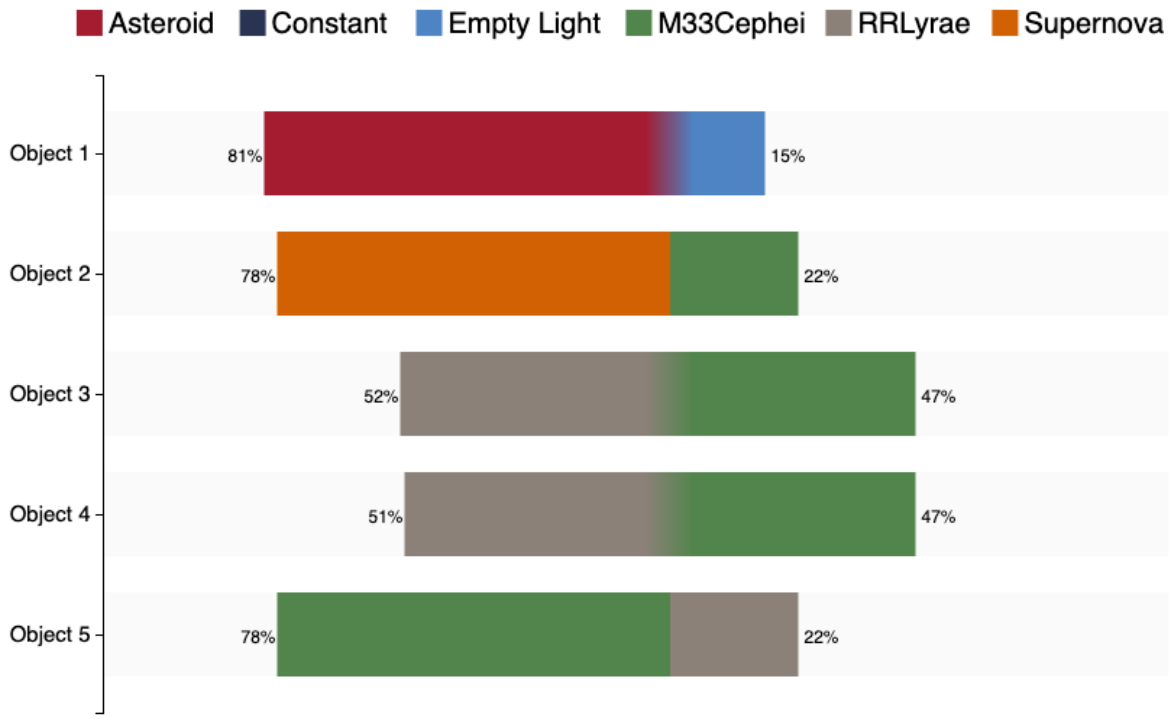

Fig. 1. The Online Classification Dashboard shows a legend at the top containing the six types of celestial objects being classified. The main visualization shows a color-encoded bar that provides the top two candidates for each of the five currently observed objects. An online demo is available at: https://nickruta.github.io/online-classification-visualization/

Abstract—Advances in imaging technology capability have led to a significant increase in the availability of astronomical data. Particularly, time series analysis in astronomy involves approximately 40 petabytes of data (provided by Pan-STARRS) and is expected to approach 100 petabytes (compliments of LSST) around 2021. Scientists interested in tracking the brightness of stars need an application that can process this astronomical "big data" and return results within a few seconds. This project aims to build a visualization dashboard that can provide classification of many celestial objects from a series of nightly images as soon as possible using a Long Short-term memory (LSTM) architecture. Specifically, our model will (1) take in a variable number of images of the object and report a probabilistic classification after each image, (2) outperform naive models (that don't incorporate sequential information) using the same data and (3) be monitored online through a visualization dashboard which allows the analyst to determine when each object has been classified as soon as possible. Our results show we are able to make predictions sooner and more accurately when compared to conventional convolutional neural networks and other baselines.

Index Terms—image sequence data, online classification, data visualization, recurrent neural network.

1 Introduction

This project aims to provide classification of an object as soon as possible from a series of nightly images of that object. Traditional approaches to classify these celestial objects involves starting with sequences of images, and features extracted from them [8], to calculate the total amount of light arriving from the source to the camera as a function of time. This involves a large amount of data preprocessing which creates time-series data (called light curves in astronomy). These calculations are more likely to induce errors during the classification process. Also, the transient nature of such images makes these tasks computationally intensive. We will highlight recent developments in astronomical computation and top methods that have been explored in the past for celestial object classification. We will outline our proposed approach and present results of our final model compared to a series of baseline models.

In order to avoid the higher error rate associated with the preprocessing stage of time-series based calculations, we utilized a deep learning approach that works on the raw image data. Using the common parlance of Recurrent Neural Networks (RNN), we will refer to the set of nightly images of the sky as a video and the individual images as frames. We will develop models that take in an arbitrary number of

• *Nicholas Ruta is with Harvard University. E-mail: nruta@g.harvard.edu.*

frames for a given object and return either a best-guess classification, in our baseline models, or an approximate probability vector over the possible object classes in our final neural network. We will use these probability vectors as the data source for our visualization dashboard.

Our baseline models are (1) several variations of Support Vector Machines trained on the first N frames of an object's video and (2) a logistic regression model trained on the first N frames of an object's video. To be clear, in each case there will be 48 separate models, each trained on an ever-larger input vector. Next, we implemented static Convolutional Neural Networks (CNN) that learn convolutional filters but do not have any sort of correlation between previous frames and the next frame.

The project's final model will be a specific type of RNN, the Long Short-term memory (LSTM), trained on the sequence of frames. This architecture follows the following steps:

1. a given frame as input is condensed via a CNN to a low-dimensional vector

2. the vector is combined with an existing vector in the model's memory, with the result replacing the existing vector

3. the updated memory vector is passed through a feedforward neural network (FNN) to produce an estimated probability vector over the possible object classes.

The LSTM architecture will work well for online classification since it can absorb an arbitrary number of frames and produce a vector of probabilities at each step. This means that only one model is needed for any length of video. Because the input data to the LSTM is an image, we used a CNN to map the frame data down to a low-dimensional vector. We optimized this LSTM for speed, accuracy and overfitting by tuning network options for batch normalization and dropout. The details are explained below in the Modeling Approach section.

Since human beings tend towards vision as the most important channel for receiving information from their environment, we provide our results using a visualization dashboard. This allows the analyst to assess the confidence of predictions made based on the probability that observed objects belong to the trained model's classification types to determine if they have been adequately processed. This allows objects to be classified as fast as possible, allowing computational resources to be focused on observations that have not yet been classified.

2 Related Work

Traditional methods to classify astronomical objects are based on data pre-processing of a sequence of images and involved feature extraction [8]. This is done by calculating the total amount of light arriving from the source to the camera as a function of time, generating a time series (called a light curve [7]). In practice, these light curves are sparse and heteroscedastic [6] which required techniques such as interpolation that can be inefficient. Most recently, neural networks have become the most popular option for astronomical classification problems due to their ability to take in images directly rather than requiring additional error-prone calculations.

Previous work has been done to use CNNs that classify astronomical objects using a single epoch [4]. Our dataset includes objects that are extremely similar at a single time step. As a result, it will be important for us to incorporate information from previous states. In this way, a single CNN will not be sufficient. In order to capture information in a sequence of observations, the long context window of a RNN can be used [5]. Since we want to provide online classification, one time step at a time, the RNN architecture is a good choice. It allows us to make predictions of objects given a frame at a time and strengthens its accuracy as it is updated with each longer video. Furthermore, cutting edge RNN designs such as LSTMs offer a mathematical improvement by avoiding the vanishing gradient problem during the process of passing across time steps [3]. This is important in our use case since the strong similarity between certain classes of objects at a single time step can lead to misclassification.

An important form of regularization in neural networks is dropout. The goal of dropout is to prevent co-adaptation of the outputs, hence neurons do not depend on other neurons being present in the network [1]. This technique is used during training of the network to obtain most robust weights throughout the network layers. It is not present during model evaluation where all neuron nodes must be active.

Probably the most recent and relevant work involves creating an LSTM for image sequence classification of astronomical events [2]. Their choice of baseline model, a random forest, was shown to underperform when compared to their final LSTM architecture for test accuracy. This informed us on the initial neural network that we designed and also gave us evidence that spending additional time to train the computationally expensive random forest model would not be needed. Although they suggest future designs could incorporate online classification feedback for faster object observation periods, our final implementation not only implements online classification but also demonstrates a novel approach to visualizing the results dynamically.

3 Data Resources

The dataset used for this project consists of 72,000 simulated histories of celestial objects. Each of the records contains 48 21 by 21 grayscale images, with each of the 48 images collected on a different night of imaging a particular part of the sky. Each record is labeled as one of 6 types of astronomical object. They are Asteroids, Constant stars, EmptyLight, M33Cephei, RRLyrae, and Supernovae.

3.1 Exploratory Data Analysis (EDA)

Our primary focus during EDA was on exploring the image data in order to develop heuristic rules for how to discern the differences between the 6 types of objects. For example, Asteroids and Supernovae (shown in fig. 3 below) generally look like white noise with possibly one frame of a circular object, while Empty Light objects tend to be oblong and consistently above any background noise. Supernovae, intuitively, are often extremely bright for a small number of frames, but may be distant or close to the level of background noise.

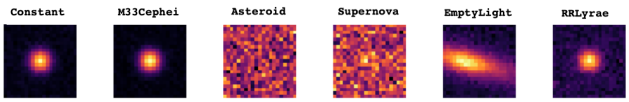

Fig. 2. Showing the mean intensity for the six astronomical object classification types in the dataset.

Looking at fig. 2 above, we see that several types look very similar at first glance. For example, Constant, RRLyrae and M33Cephei are almost indistinguishable from each other based on their mean intensity. Asteroids and supernova are also very similar. These similarities will be noted during the online classification stage of the project as particularly difficult to correctly distinguish.

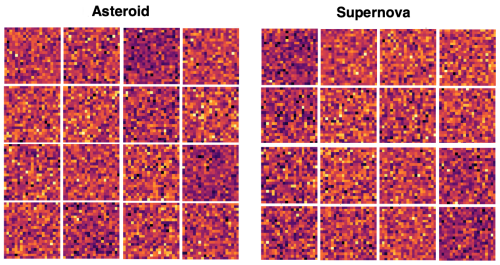

Fig. 3. Asteroid and Supernova sequences generally look like white noise.

Below, in fig. 4, we noted that in different time-stamps, the intensity in both the central region and the surrounding area changes. This reflects the periodic pulsating associated with these types of stars. Fig.

4 also shows that the difference between a sequence of M33Cephei is only subtly different when compared to that of a Constant. We will need to exploit the temporal distribution stored in the different epochs of each to improve our predictions. This will help our online classification to complete as soon as possible.

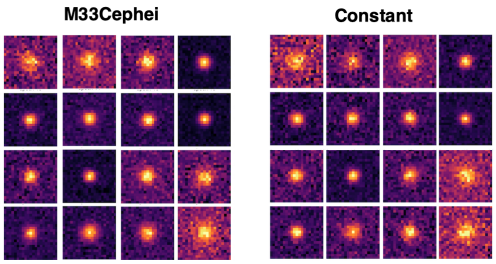

Fig. 4. The full sequences of M33Cephei and Constant samples show that while both stars pulsate periodically, the Constant maintains it's core size slightly more than the M33Cephei.

Next, we wanted to check for the presence of outliers in each object class. We plotted the mean intensity by object class for each observation's entire sequence seen below in fig. 5. We can see that the frames generated by each object class follow more or less the same brightness distribution. However, the extreme tail of the distribution depends heavily on the object's class, with Asteroids maxing out at a lower amount when compared to RRLyrae or M33Cephei, for example. These results mean we need to carefully transform and normalize the data due to the presence of extreme outliers.

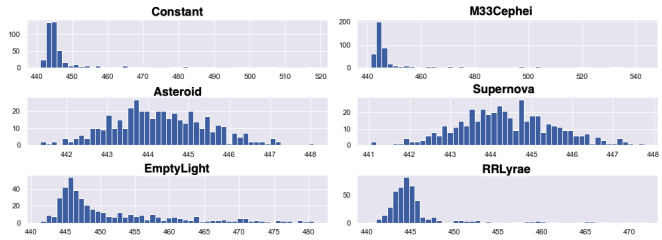

Fig. 5. Plotting the mean intensity by Object Class for the entire sequence.

4 Modeling

We began our modeling approach by fitting several baseline models using our training dataset. These will serve as benchmarks that can be compared to our more complex neural network-based models. We then created two convolutional neural networks that predict on the full sequence (48 images). Finally, we created our recurrent neural network (LSTM) that predicts on an arbitrary number of frames in the sequence to provide online classifications. It also strengthens our test accuracy by incorporating the correlation between previous frames and the current frame during training.

4.1 Baseline Models

To identify how well conventional classifiers can perform on our dataset, we built baseline models on subsets of the data using Support Vector Machines and Logistic Regression since these methods are available at-scale using scikit-learn's python implementations. We observed slower training times and lower accuracy for Logistic Regression and therefore drew our attention to SVMs. We trained kernelized SVMs using both RBF and Linear kernels on a random subset of 7,500 training images. We used 5-fold cross validation on the training set to tune the model parameters. Without using regularization, the best training set accuracy was 96%. Using the appropriate regularization lowered the training set accuracy to 36%. This eliminated most overfitting and slight improved the test accuracy. Ultimately, we used a stochastic gradient descent linear SVM which reached 35% test accuracy when trained on the entire training set. This eliminated overfitting and slightly improved test accuracy. We stopped at this point, feeling that a considerable improvement would not be reached, since training new versions of the model took approx. 30 minutes.

4.2 Convolutional Neural Network Models

For the first CNN, we treated all images, including those within the same sequence, as separate samples. This design choice is not ideal since we are discarding all temporal information about each object's history. We refer to this as the 2-dimensional version of our CNN. We found that a better approach is to include the time component, particularly the epoch number, as a third dimension in addition to the width and height of the pixels. This means that the epoch's dimension can take the place of an image channel. This 3-dimensional version of our CNN incorporates the time component to learn filters that are relevant across time steps and appear anywhere in the video. This is important given the heteroscedastic nature of our observations. For both of our CNN approaches, a single CNN architecture was designed. It consists of multiple convolutions that use maxpooling which are finally flattened with an output layer that uses softmax to provide probabilities for each class. The architecture is provided below in fig. 6.

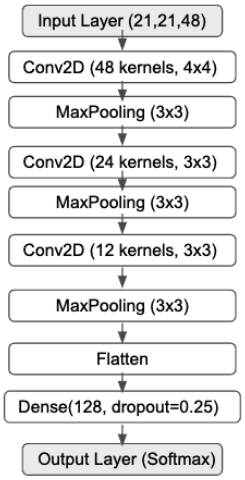

Fig. 6. Showing our CNN architecture.

Although the 3d CNN showed an improvement over our 2d version, we were still not adequately capturing the temporal information between epochs. In order to utilize the sequential nature of our observations, we next created a recurrent neural network design.

4.3 Final Recurrent Neural Network (LSTM) Model

We explored several variations of RNNs based on our literature review. We determined that a stacked model [9] is the most intuitive design based on the sequential image data that we are working with. The CNN-LSTM architecture we chose learns features from each frame of the video using convolutions that are fed to a 128-entry LSTM which is able to update its internal state as new frames are received. With this setup, we can combine the advantage of convolutional recurrent layers. This updated state is finally decoded by a dense network and a 6-class probability vector is provided by a softmax layer as output. The architecture is provided below in fig. 7. We see that it starts with the same CNN described in the previous section and appends the LSTM and output layer to create our final recurrent neural network model.

We tuned our final LSTM by testing different hyperparameters. We used dropout and batch normalization layers to avoid overfitting. Fig.

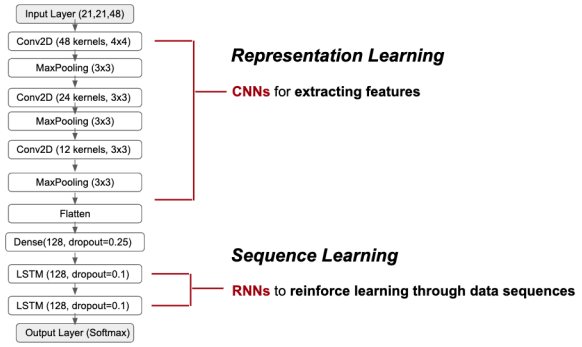

Fig. 7. Showing our LSTM architecture. It combines the representation learning of CNNs with the sequential learning of RNNs to give us the highest test accuracy.

8 below shows the tested dropout rates and their impact on the model's performance.

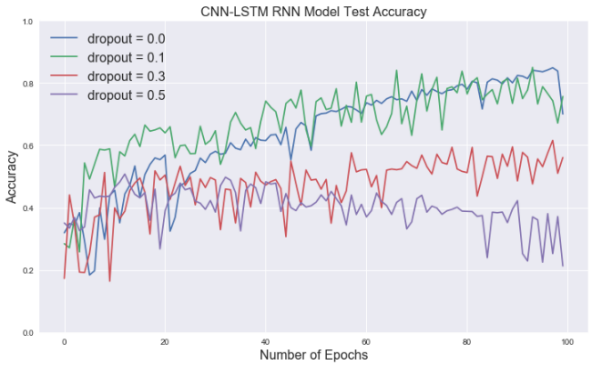

Fig. 8. Showing the test accuracy of our CNN-LSTM RNN for different dropout rates over the first 100 epochs.

As shown in fig. 8 above, including a 10% dropout rate yields a slightly higher accuracy than not using dropout at all. Dropout rates higher than 10% resulted in worse model accuracy which suggests that higher levels of dropout mislead the model by removing too much information about each observation's sequence.

In order to improve the convergence and running time of our model, we used feature normalization. This was useful since our dataset consisted of features that were not on close scale across all observation types. Further, this allowed us to avoid the vanishing gradient problem since features with larger values, RRLyrae as described earlier, can stymie back-propagation by creating derivatives that are either extremely small or large. This results in gradient descent updates that are too far off. After testing different approaches, we normalized with each image frame as opposed to normalizing across the entire video. We normalized to the range of zero to one and standardized by subtracting the mean and diving by the standard deviation.

Finally, in an attempt to further improve our model, we reviewed various transfer learning options. We considered the Inception, AlexNet, LeNet-5, VGG-16 and VGG-19 networks. While LeNet-5 is typically used for digit recognition and did not seem to apply to our dataset [10], AlexNet was traditionally used to classify high-resolution images. The VGG-16 and VGG-19 networks [10] consist of sequential convolutional layers followed by max-pooling layers and fully connected layers at the end. Inception networks use multiple types of convolutional layers at each network layer with multiple kernel sizes to aim for sparsity in the network layers [10]. Although these image-based networks work well in practice, we did not want to lose the ability to diagnose errors [10] by adding these larger networks to our model.

4.4 Modeling Results

With all of our models described, we now turn our attention to their results. In general, we see that the deep learning methods significantly outperformed our SVM baseline model. While the SVM achieved a test accuracy rate of 34%, our final CNN-LSTM model was the most accurate at 93% for object classification. Based on these results, we will use our final CNN-LSTM model to create the online classification visualization described in the next section.

Model	Train Accuracy	Test Accuracy
SVM	0.364	0.341
CNN: 2d	0.904	0.892
CNN: 3d	0.914	0.911
CNN-LSTM	0.922	0.930

5 Online Classification Visualization

For online classification, we utilized the stateful option in keras LSTM layers, which enables us to feed in images one epoch at a time and receive dynamic predictions based on all of the accumulated images of a sequence. By doing this, we are able to make predictions on the fly, with any number of epochs of images. For online classification, we are not able to make the same classification as we would with the full video.

Our technical implementation involves creating an online classification model that loads our pre-trained CNN-LSTM model. Then, for each new image that is received, we instantiate a new classification object. We call an update function that provides a new prediction, based on the current point in the video and reports a prediction as a vector of probabilities (seen below in fig. 9) for each class. Once a new object is being observed, we reset the state of our classifier by removing previous information such as the epoch number and previous predictions made for the last observed object.

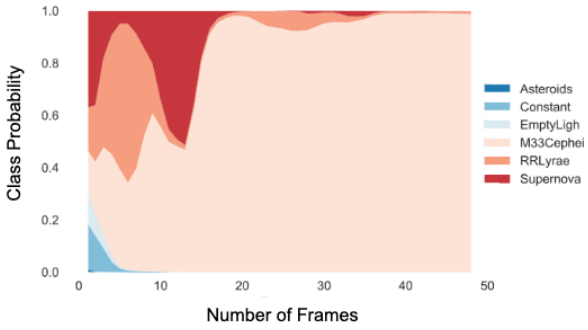

Fig. 9. Showing an example of our online classifier's history of assigned probabilities for each object type over the entire video.

5.1 Motivating Tension

We want to explore the results of our online classification model for each object currently being observed. In order to do this, we have to visualize the probability of type classification for each class of celestial object. If we think about only the top 2 candidates for the object's type, we could imagine that in some instances the model is not clear and thinks that either of the top 2 are strong candidates. At other times, the model has a clear winner for the classification type. When the model is confident, the analyst can move on to a new object and release the currently observed from the online classifier's resources. When dealing with many objects being monitored online, it can be computationally expensive to continue past this point of confidence with objects that no longer need to be examined.

We can think of the model's confidence across all of its class predictions as the tension between the class predictions. We define the tension between two class's predictions as:

$$\frac{|P_1 - P_2|}{\sqrt{\sigma_1^2 + \sigma_2^2}}$$

Where P_1 and P_2 are the probability distributions for each class, $|P_1 P_2|$ is the spread between those distributions and $\sqrt{\sigma_1^2 + \sigma_2^2}$ is the distribution fatness.

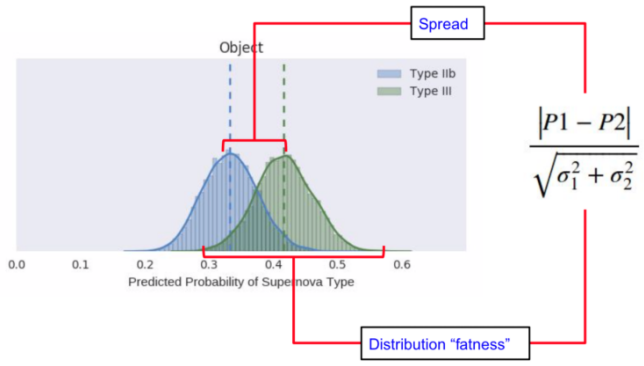

Fig. 10. Showing tension based on the spread and distribution fatness for the top two class types of an object's classification process.

Defining tension this way gives us a metric by which we can determine whether the model is confident enough to stop during online classification of a sequence of images or if it should continue to collect frames in order to strengthen its predictions.

5.2 Online Visualization Dashboard

With our online classifier outputting probabilities at each frame in the video and the tension between the top two class type candidates defined, we now turn our attention to producing an Online Visualization Dashboard. We color encode a bar for each object being observed where the top two probabilities are represented. At the point where each of the two probabilities meet on the bar, we demonstrate the tension between them as either a straight line or a fuzzy line depending on the level of tension. This allows an analyst to quickly view a dashboard of many observations to determine which can be discarded since they have been confidently classified. Fig. 11 below shows the design. The color-encoded classes are placed at the top. Each object is placed as a row with a bar that represents the top two classes and their current probabilities. The tension marker is placed between them to express the model's confidence based on the sequence up to the current frame. It is updated each time a new frame is received for an object.

We provide a working demo of this Online Visualization Dashboard at: https://nickruta.github.io/online-classification-visualization/. You can reload the page to see the demo simulate online observation of 5 celestial objects. These 5 objects are real data points provided to us by the HITS survey [1].

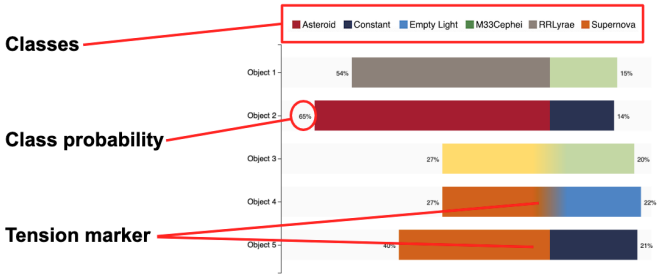

Fig. 11. Visualization Dashboard design showing (1) the color-encoded classes, (2) the current probability of the top 2 classes for each observed object and (3) the tension between the top 2 classes.

6 Conclusion

This project provided the required steps to build an online classification dashboard based on deep learning for image sequences. The first part was to identify the correct neural network architecture that would provide adequate accuracy and speed for online predictions. We found that when compared to the 35% test accuracy of our SVM baseline model, our CNN-LSTM final model was able to achieve 93% test accuracy for object classification after tuning key hyperparameters such as dropout and pre-processing using batch normalization.

We then defined a metric, referred to as tension, by which we could determine the confidence of our model's predictions so that analysts could decide when objects can be discarded from the online classifier's pipeline. Finally, we provided a visualization design for the dashboard to express the top 2 candidates for classification type and mark their tension visually for quick interpretation of dynamic results.

7 Future Work

Future work on the chosen model could include a comparison with the final implementation we used and a version of it that utilizes transfer learning from networks such as Inception. Although we determined that it could complicate our model, once the final architecture was created, these transfer learning networks would provide pre-trained weights for the convolutional layers that could improve overall accuracy and speed of the network. The specifics of the network design could also be tuned for the optimal number of hidden units and the number of convolutional and recurrent layers. Currently, real astronomical image sequence data is not available at large scale. Since the data used to train our model was simulated and only had 6 class types, future work could involve using real data with a more all-inclusive object classification range. This will be made possible once real observations are more readily available via new telescopes, such as LSST, coming online in the near future.

As an online classification tool, the question of whether an object has been properly classified at any given point is important. Future work could address the scientific questions of interest in determining how to decide if an object has been classified. Tradeoffs between misclassification and other scientific project goals can be considered when creating a metric as opposed to simply assessing the tension as we have described in this work.

As far as the visualization design, accessibility for color blindness should be addressed. Further, a feature to allow the analyst to remove objects form the dashboard would be needed.

Generally speaking, it would be interesting to use the final tool described in this project for exploration and discovery. We did preliminary testing on the Deep-HITS [1] survey to see how our trained model worked on real data. The results were promising as it was able to properly classify when shown the 6 object types it was trained on. Future work could look more carefully at Deep-HITS, with the assistance of astronomers, to see if objects that are not confidently classified by the model are of particular interest. Perhaps interesting objects or misclassifications in the Deep-HITS survey could be identified.

References

[1] G. Cabrera-Vives et al. Deep-hits: Rotation invariant convolutional neural network for transient detection. The Astrophysical Journal, 836:97 (7pp), 2017.

[2] R. Carrasco-Davis et al. Deep learning for image sequence classification of astronomical events. arXiv:1807.03869 [astro-ph.IM], 2018.

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. neural computation. vol. 9, no. 8, 1997, pp. 17351780., doi:10.1162/neco.1997.9.8.1735., 1997.

[4] A. Kimura et al. Single-epoch supernova classification with deep convolutional neural networks. 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), 2017.

[5] J. B. Lipton, Zachary C. and C. Elkan. A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019, 2015.

[6] A. Mahabal et al. Deep-learnt classification of light curves. ArXiv e-prints arXiv:1710.01422, 2017.

[7] T. Naylor. An optimal extraction algorithm for imaging photometry. Monthly Notices of the Royal Astronomical Society, vol. 296, no. 2, pp. 339346, 1998.

[8] I. Nun and P. Protopapas. The fats light-curve library. `https://github.com/isadoranun/FATS`, 2017.

[9] X. B. Shi, Baoguang and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. IEEE transactions on pattern analysis and machine intelligence39.11., 2017.

[10] C. Szegedy et al. Going deeper with convolutions. arXiv:1409.4842v1, 2017.