Draft version December 18, 2017
Typeset using LaTeX **twocolumn** style in AASTeX61

HISTOGRAMS OF ORIENTED GRADIENTS FOR KECK/NIRC2 NOISE DETECTION

Justin Kang
University of Texas at Austin

ABSTRACT

Keck/NIRC2 uses adaptive optics to produce the highest-resolution ground-based images and spectroscopy in the 1-5 micrometer range. It is one of the best instruments for planet discovery and characterization, with 44 papers published just this year that have used it. However, there are several systematic errors that reduce its effectiveness and usability. One of the more well-known examples of this is the noisier lower-left quadrant, where there are erratic vertical streaks of intensity repeated every eight columns. The main method of dealing with this noise currently is using a three-point dither pattern to avoid the effects of the bad quadrant, but this is an inadequate solution as it discards the entire quadrant of the instrument. To remedy this, we employ techniques from computer vision and train a support vector machine (SVM) using the histogram-of-gradients features to detect such errors. Using relatively small training and testing sets, we draw some early conclusions. Due to the lack of well-annotated data, unfortunately no quantitative conclusions or evaluations of the method can be made at this moment. However, the qualitative results suggest that the method works well in terms of detecting the presence of this elevated noise. This method could then be used as a quick check to see which images in large sets have this characteristic noise, allowing observers to quickly know which images to avoid. Several improvements that can be done to the specific implementation and the overall experimental procedure are discussed, as well as possible extensions to this project.

Corresponding author: Justin Kang
justin.kang23@gmail.com

1. INTRODUCTION

1.1. *Direct Imaging*

Direct imaging for planet detection involves directly taking images of a planetary system or debris disk. From the planet's photometry, colors, and spectra, we can estimate the planet's orbit, size, temperature, atmosphere, and other properties. This method favors planets that can be resolved from their host star(s) - these are thus young and hot planets that are large, still radiating away energy from their creation, and widely separated from their host. Unlike most other detection methods, direct imaging works best with face-on orbits rather than edge-on, as it can then accurately measure the entirety of the planet's orbit around the star. This gives direct imaging a unique parameter space in relation to the other detection methods - it finds large planets with large orbital radii, occupying the space beyond what current radial velocity methods detect. Thus this method well complements the other planet detection methods, such as radial velocity and transits.

1.2. *Keck/NIRC2*

The second generation Near-Infrared Camera (NIRC2) at Keck Observatory uses the Keck Adaptive Optics system to produce the current highest-resolution ground-based images and spectroscopy in the 1-5 micrometer range. Its first major planet discovery was with Marois et al. (2008), where it was used to observe the first multiplanet system HR 8799. Since then it has been widely used by the direct imaging community, with 44 papers published in just 2017 that used the instrument[1].

However, there are several systematic errors that reduce NIRC2's effectiveness and usability. The best example of these is shown in the lower-left quadrant of the array; as noted by many that have used it (Bowler et al. (2015), Crossfield et al. (2017), Rodriguez et al. (2017)), this corner suffers from elevated noise levels. This noise presents itself in the form of vertical stripes - we can see groups of increased intensity every eight columns across the entire quadrant. The current popular method of dealing with this quadrant is adopting a 3-point dither pattern (as suggested in the NIRC2 observer's manual[2]), which omits the lower left and central positions of the detector. Another method is to place the coronagraph far from the region, keeping the objects of interest outside the region. Both of these "solutions" avoid the use of the error-prone region, losing an entire quadrant of the detector. Thus, these aren not really useful or efficient methods - we need a better way of dealing with this error.

[1] https://www2.keck.hawaii.edu/library/biblio/2017.html

[2] https://www2.keck.hawaii.edu/inst/nirc2/Manual/ObserversManual.html

1.3. *Linear SVMs and the Sliding Window Detection*

We now introduce a technique well known in the field of computer vision - the sliding window classification from Dalal & Triggs (2005) and Felzenszwalb et al. (2009), used commonly in pedestrian detection. This algorithm employs a sliding window to independently classify image windows as being an object or non-object. Given heterogeneous training and testing data, we transform images to a histogram-of-gradients (HoG) representation, train a linear classifier, and use this to classify up to millions of sliding windows. Here the training data is a combination of positive and negative examples from of the listed image sets - images with the noise are positive, and those without are negative. The classifier is then trained, and the negative examples are searched again for false positives. The classifier is then retrained using the initial set and the false positives to improve accuracy.

The benefits of using linear classifiers are that they are compact, fast to train, and fast to execute, allowing for quick verification of the results. The one we use here is a support vector machine - given a set of training examples, which are annotated as belonging to one of two categories (e.g. object or non-object), SVMs build a model that assigns new examples to one of the categories. The model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by the widest gap possible. New examples are then mapped to that same space and are assigned a category based on which side of the gap they are on.

2. METHODS

2.1. *Obtaining Data*

To run our analysis, we must first obtain training and testing data for the errors in question. In the field of computer vision this available data is now large, easily accessible, and well-annotated; unfortunately the field of astronomy is not so lucky. We thus obtain raw images from the Keck Observatory Archive[3] to create this data; the specific image sets used were those from Kraus et al. (2006) and Bryan et al. (2016).

[3] https://koa.ipac.caltech.edu/cgi-bin/KOA/nph-KOAlogin

Because the first image set is quite large (a total of 641 images), we easily find images where this elevated noise in the-lower left quadrant is obvious. To create the positive samples, we choose six ”generating” images from this set. For each of these generators, we save each 16x16 pixel window as a sample. One reason we choose this method of generating positive samples is that we are not necessarily interested in the accuracy of this technique, but rather its applicability. The second reason we choose this method is that the noise is presented as vertical groups of increased intensity repeating every eight pixels, so this sampling can better capture this property of repetition.

We use the second image to create the negative samples. The image set provides 28 relatively similar images without the elevated noise - thus we can keep the quality of the positive and negative training samples relatively consistent this way (that is, relatively simple and primitive).

2.2. *Training*

After generating the data in Section 2.1, we want to train a classifier on these samples. We have a template size of 16x16 pixels from the positive samples, so we choose a HoG cell size of 4x4 (the dimensions of the HoG cell must divide the template size). Using VLFeat, we first transform all of the positive samples into HoG representations. We then append these as features into a feature vector following the method of Felzenszwalb et al. (2009). For the negative images, we choose to sample 20000 random windows. To do this we loop through each image in the negative set, choose a random window, then turn that into a sample (with the same dimensions as the positive samples), and repeat until we reach 20000 samples. We then also transform these samples into HoG representations, and turn them into features and append them to the vector. We then generate a vector that labels each of these features as originating from a positive or negative sample, then again use VLFeat to train a linear support vector machine on these samples.

After training our support vector machine, we can do some early tests. We can run the classifier on our training samples, where we expect small errors, as a sanity check. We can also visualize how well-separated the examples are at training time to identify any biases in the training data as shown in Figure 1, and visualize the learned detector as shown in Figure 2.

2.2.1. *Hard Negative Mining*

We now want to tune our trained classifier. To do this we implement hard negative mining, following Dalal & Triggs (2005). Here we set a very low threshold to try to get as many false detections as possible. After we obtain all of the false positives (hard negatives) from this run, we turn them into HoG representations and features as in Section 2.2. We then retrain our support vector machine using these new features as more examples of negatives samples, so that we can reduce our detection rate of false positives.

2.3. *Testing*

We now move into our testing phase. We run our classifier on a set of completely new testing data - images that were used for neither training nor turning. Here we choose all of the remaining 635 images from Kraus et al. (2006) as our testing set.

For each of the testing images, we run a sliding window detector. For each window in the images, we turn the window into its HoG representation and then into a feature. We run our classifier on this feature, and determine whether it is considered as an error or not. We then run non-maximum suppression - because we used a sliding window, there is the possibility of overlap between each of the detections. We just want to obtain the highest confidence detections, so we suppress all of the overlapping ones through this method (in this case, overlap mentions centers - the center of one detection should not be in the bounding box of another). Because our testing method is not robust due to the lack of data in astronomy (as discussed in Section 2.1), we simply display bounding boxes over each of the images and manually go over them for each of the images to determine performance.

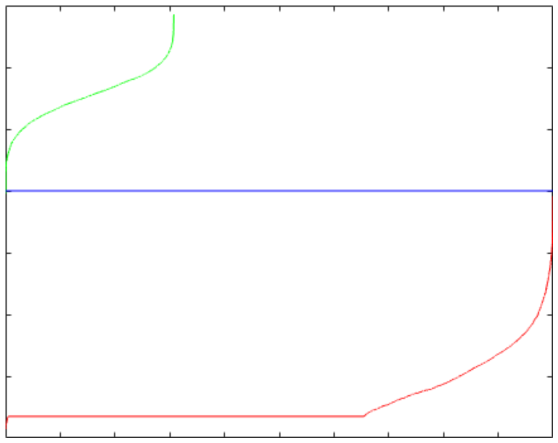

Figure 1. The visual representation of the separation of the positive and negative examples at training time. The odd separation indicate biases in the training data and/or limitations in separation from using a linear classifier.

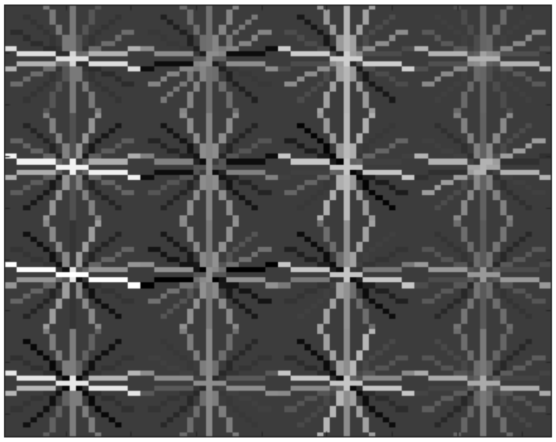

Figure 2. The visual representation of the learned sample in the trained detector. Here we can see that the HoG representation shows very strong intensities repeated in a pattern and that the gradient is strongest when going horizontally, as we expect from the behavior of NIRC2's noise.

3. RESULTS

For the training error, the classifier had an accuracy of 100%. In terms of detections, the true positive rate was 23.5%, the false positive rate 0%, the true negative rate 76.5%, and the false negative rate 0%. Thus as our sanity check expected, our classifier performed very well on the training data.

Qualitatively, our classifier performed very well on this test set. The classifier always had several detections on the obvious positive images - those with very strong intensity differences between the elevated noise and the background. However, the classifier did not perform as well on less obvious positives. This is indicative of the training data - because the classifier was trained on images with very strong gradients in the itensities for the noise, detecting the more subtle and less intense noise proved to be a greater challenge for the detector. For negative images, the classifier again performed well, showing very few false positives in these images. The greatest error rate for the classifier seemed to be on the images with the elevated noise. As shown in Figure 3, the classifier struggled to find any errors in the images where the noise was very subtle. The classifier also had a problem with showing many false positives in images with lots of error. We can see that there is a high concentration of detections in the lower-left quadrant, where we expect them to be, but also a few false detections in the other quadrants. Better choices of the hyperparameters could improve this in the future.

Due to the lack of sophisticated labeling of the testing data, quantitative results do not exist at this time.

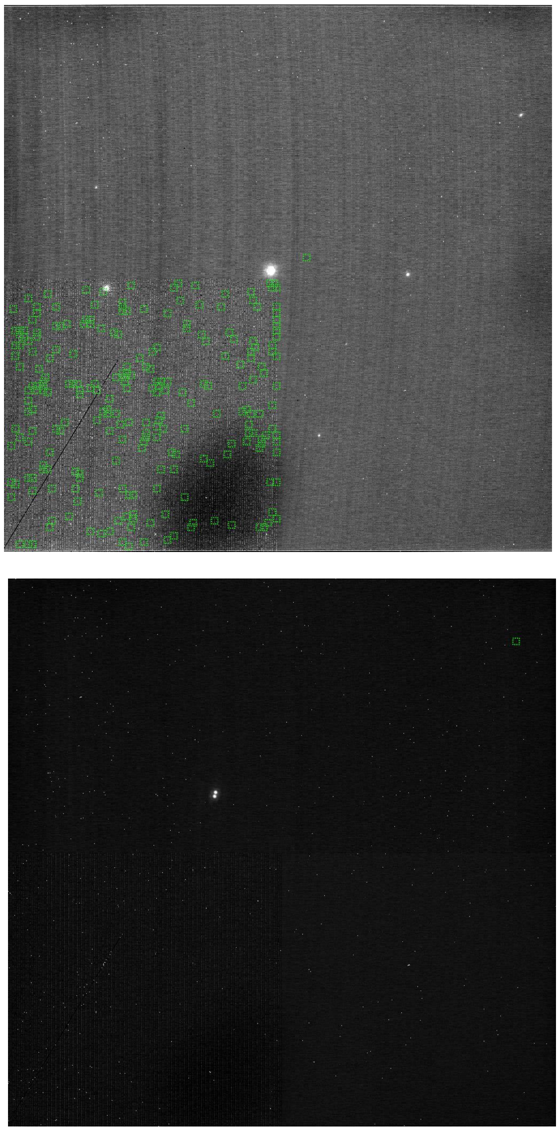

Figure 3. The detections from the classifier. In the first image we see many true positives and a false positive detected (green). In the second we see no true positives and a false positive detected, where we expect many true positives.

4. CONCLUSION

Because we do not have any quantitative results about the performance, making any concrete conclusions is difficult. However, there are some things we can infer from the results. The first is that the qualitative results suggest this method works well for detecting the presence of this elevated noise. Thus one could use something general like a count of the detections to tell whether there is a large amount of noise in their images or not. The second is that because this technique seems like it could be one that could apply well to this field, there are many possible improvements and extensions. These are further discussed in section 4.1.

4.1. *Recommendations*

There are many improvements and extensions that can be done on this analysis. We will first go over the possible improvements, in order of relative increasing difficulty. We then discuss two extensions of this project.

4.1.1. *Make the program more performant*

The program takes about 10 minutes to run on the given data set. The inputs are relatively small - there are 6144 positive samples and 20000 negative samples for training and 635 images for testing. One of the methods to make this program more performant and scalable is to introduce parallelization. This program involves a lot of loops, which are particularly slow on MATLAB (the language of choice for this project), thus parallelizing these loops can provide a good boost in terms of time performance when training and testing the classifier. Saving the resulting trained classifier could also save time from having to retrain every time.

4.1.2. *Find the performance in accuracy.*

As discussed in Sections 3 and 4, we can't make any quantitative conclusions from this analysis. By labeling the testing data, we can determine the exact rates of true positive, false positive, true negative, and false negative detections. By doing this we can make better statements of the performance of this analysis method and thus take more informed steps to improve it. This could range from being very simple, such as a boolean saying whether the image contains the elevated noise or not, to very complicated, such as including the specific number of segmented errors or even the bounding boxes of all of their locations in their image. By doing this, we can get better and better ideas of the exact quantitative performance of this analysis.

4.1.3. *Improve the training data.*

The training data generated for this analysis was very crude. It did not take into consideration any properties of the noise itself or specify the errors in any way - the generation code only exploited the fact that the errors were only present in the lower-left quadrant.

Thus, there are many ways to improve the training data. The first improvement is quantity. One of the reasons why computer vision and machine learning work so well today is because of the large availability of good data. By just increasing the quantity (as well as variety) of the training samples, we can greatly improve our performance. This is seen in the tests of subtle errors - with better training data, these likely could have easily been detected.

The second improvement is quality. By making the training data more specific to the noise in question, we can increase the accuracy of our classifier - particularly so for false positives, which our classifier struggles greatly with.

4.1.4. *Make the program portable.*

As mentioned in Section 4.1.1, this entire analysis is implemented in MATLAB[4]. This is not very portable, as MATLAB requires a license to use. One obvious switch is switching over to Python, which is completely free to use and a much more commonly-known language. Many of the language features between MATLAB and Python are the same, and libraries such as OpenCV allow for comparable computer vision techniques as MATLAB does. This would allow more people to easily access and develop the code.

4.1.5. *Tune the hyperparameters.*

In our "tuning" section, we simply used hard negative mining to improve accuracy. In the field of machine learning, what is more commonly done in the tuning step is tuning the hyperparameters. Hyperparameters are parameters not inherently from the data but rather from the analysis process. In our program, these are: the template sample size, the HoG cell size, the number of negative samples, the lambda for training the classifier, and the thresholds used for hard negative mining and detection. By adding a small structural change to our program, we can use machine learning to try many random combinations of hyperparameters and determine which one produces the best detection rates. This would result in a much more rigorous selection of hyperparameters than what was done for this project - just picking values that seemed to work well.

4.1.6. *Apply the kernel trick.*

As discussed in Section 1.3, there are many benefits to using a linear classifier. However, we can see from Figure 1 that the separation the support vector machine obtains is a bit wonky. By using a kernel for determining similarity rather than a simple linear classifier (which uses inner-product similarity), we introduce a black-box similarity function. By changing this function, we can then increase the number of dimensions in the parameter space. What this allows us to do is increase the number of dimensions until we find one where the samples are all separable. By making our classifier nonlinear, we can improve the separation between the examples, and thus theoretically improve the performance of our classifier.

[4] The program and data are entirely available on Github, at `https://github.com/justin-kang/AST-381-Final/`

4.1.7. *Use a neural network.*

On the topic of nonlinear classifiers, we can change this program from using a support vector machine to a neural network. The immediate benefit this gives us is that we can easily introduce a nonlinear classifier to the program leading to better separation. The difficulty in doing this is determining the best architecture for the program. The current method of determining this is looking at literature for similar problems and copying their architecture - this could also apply here. Neural networks can also lead to much more robust classifiers, as we can train them to learn for themselves what features are necessary than trying to specify it ourselves.

4.1.8. *Subtract the noise.*

We now move from improvements to extensions. Once we are able to determine the presence (and eventually location) of noise in our images, the most natural extension is removing it. This would require two things: a classifier that can accurately detect every instance of error in the image, as well as a means of segmentation. The classifier has to be very accurate so that we can find every noisy pixel for removal, as we want to be thorough in our removal process. We need a form of segmentation because after removing all of the noise, we want to determine the nature of the remaining pixels - did the photons originate from the same object, or were there multiple? Implementing an accurate segmentation algorithm is likely to be considerably more difficult than making the classifier more accurate, and is one of the bigger challenges of modern astronomy.

4.1.9. *Use this in new applications.*

One of the main benefits of this method is its generality. The core problem in detection is trying to find an object. If there is training and testing data available for this object, then this detection method should work. This is evidenced by the applicability of this project - taking something that was originally used for pedestrian and face detection and applying it to noise detection in astronomical images. The field of deep learning is in a huge boom right now in the computer science industry - it would be a great disservice to any to not use these powerful techniques, even across disciplines.

4.2. *Acknowledgements*

We would like to thank Dr. Adam Kraus for his guidance on the AST 381: Planetary Astrophysics course at UT, as well as his advice on this project.

REFERENCES

Bowler, B. P., Liu, M. C., Shkolnik, E. L., et al. 2015, ApJS, 216, 53

Bryan, M. L., Bowler, B. P., Knutson, H. A., et al. 2016, ApJ, 827, 16

Crossfield, I. J. M., Ciardi, D. R., Isaacson, H., et al. 2017, ApJ, 153, 255

Dalal, N., & Triggs, B. 2005, CVPR, 1063-6919, 05

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., et al. 2009, IEEE, 32, 1627

Kraus, A. L., White, R. J., & Hillenbrand, L. A. 2006, ApJ, 649, 306

Marois, C., Macintosh, B., Barman, T., et al. 2008, Science, 322, 1348

Rodriguez, J. E., Vanderburg, A., Eastman, J. D., et al. 2017, arXiv, 1709.01957

Vedaldi, A., & Fulkerson, B. 2008, VLFeat, 0.9.20