Exo-Hunt: Discovering Planets with Deep Convolutional Neural Networks

Griffin Miller
Stanford University
gmill@stanford.edu

Moritz Stephan
Stanford University
moritzst@stanford.edu

Simon Camacho
Stanford University
scamacho@stanford.edu

Dr. Kate Follette*
Amherst College
kfollette@amherst.edu

Abstract

We propose a novel method for detecting exoplanets from H-Alpha and Continuum images using deep convolutional networks and achieve 36.1% overlap accuracy and 24.7% box-count accuracy, which is better than humans trying to visually find planets in said images. We build on existing methods for identifying planets in images using starlight removal, which are described in detail in the Related Works section. We were granted access to a proprietary dataset from an Amherst professor, which contains the largest collection of preprocessed telescope images of star systems potentially containing exoplanets worldwide. We explore different preprocessing and data augmentation techniques in order to downsample each image from 9 to 3 channels before passing them to CNN backbones. We also explore different network architectures, building upon Faster R-CNN. Our best model samples the 9-channel images down to 3 by taking the slice-wise mean and feeds them through a custom deep convolutional backbone before using Faster R-CNN for bounding box predictions. While still far from perfect, our work is suitable as a starting point for a more long-term research project to automate exo-planet detection.

1. Introduction

Discovering new planets is a crucial part of not only astronomy and astrophysics but also the advancement of human civilization. As the human population grows, the number of resources available for those on this planet shrinks. Identifying planets is a difficult task – starlight in images must be removed, and then these images must be studied carefully in order to discover what might be a new planet. More specifically, physicists create what are called "reductions" on the initial images, each of which is a different form of principle components analysis intended to reveal possible planets in the imagery. Furthermore, there are two different types of images of star systems: Continuum images and H-Alpha images. Continuum and H-Alpha images are identical, except that planets will appear on the H-Alpha images and will not appear on Continuum images because of the hydrogen wavelengths captured in the images. Continuum images are important because they allow for injecting fake exoplanets into the data as a statistical baseline, which we discuss later as a preprocessing scheme to create additional training data.

We are fortunate to have obtained proprietary and confidential data collected using the MagAO-X telescope. The images are taken from various star systems with either no known companions, candidate companions, or confirmed exoplanets. Images taken by the telescope are rotated so that a planet will always be found at the same location on an image on a given day, necessitating the introduction of data invariances to achieve robustness [8].

Our initial strategy for our project was to create a simple convolutional neural network, train it on images that either did or did not contain a real planet and verify the model's efficacy at this binary image classification task by predicting whether a planet was present. For the initial results tackling image classification, we used a three layer convolutional network as a baseline for our project (the architecture of this model is described in the Methods section). We decided to use a simple model in order to easily understand how the data would be interpreted by a neural network, and then experimented with incrementally more complicated models as we moved on to object detection. Since the dataset we received from Prof. Follette does not contain images without exoplanets, we we had to split all images up into quadrants, which we could easily label as either containing or not containing a planet. All images are symmetric with the circular star exactly in the center. Due to the RAM limitations of

*Dr. Follette is an Assistant Professor of Astronomy at Amherst College (not enrolled in CS 231n). She kindly gave us access to the dataset and met with us to explain how exoplanet detection traditionally works.

Google Colab and the fact that the images are rather feature sparse, we randomly selected 800 train and 200 test images from the stars that have confirmed companions. We then trained our baseline model using the Adam optimizer, and tested on our test set, achieving 98.53% accuracy.

We then extended this functionality to the object detection task, enabling us to use images of star systems and locate planets in them. This is an extremely difficult task since the telescope images are extremely sparse and low variance, making the use of pre-trained models difficult due to data mismatch. Further, the images contain lots of noise and planets often just occupy a handful of pixels. In addition, we had to undertake lots of data exploration to better understand how easily planets can be spotted and found it to be practically impossible for the human eye. Another challenge was the sheer volume of data with over 48k images of 17MB each, creating a total of 816G, so we had to build robust scripts to extract and move only the data we needed.

The input to our model is a 451x451x9 H-Alpha or Continuum image. We then use the Faster R-CNN object detection model in order to classify and predict different bounding boxes on the input image, yielding planet location predictions. For the Faster R-CNN structure, we experimented with many different convolutional backbones, including initially the ResNet-50 architecture (described in more detail in the Methods section). This performed extremely poorly and we were unable to overfit it even on a tiny subsection of the dataset. Subsequently, we explored other backbones and ultimately built a custom one that provided the best results. Through model selection and hyperparameter optimization, we were able to achieve 36.1% test accuracy.

In this paper, we also discuss our experimentation with different channel downsampling schemes (downsampling from 9 to 3 channels) as well as with different data transformations designed to introduce more robustness into our models.

2. Related Works

In the process of creating our initial methods for approaching the problem of planetary detection, we chose to focus on the characteristics of the planetary images themselves instead of on the light inside of them, as other projects have done. In Malik et al. (2021) [9], they chose to perform feature extraction, using characteristics of how light changed between images of star systems in order to design a binary classifier which could decide whether an object in an image was a planet. Although this method has resulted in productive results, the drawback is that the shifting magnitude of light can only be recorded when the planet's orbit is in exact alignment with the observer collecting the light sample. Additionally, this method also produced a significant amount of false positives, introducing the necessity for manual verification of the machine learning methods. Instead, we chose to engage in supervised learning using a convolutional neural network in order to learn associations between the pixels in the input images to reduce the amount of false positives [6].

In Adams (2021) [1], Adams uses the PyKlip package and its starlight subtraction algorithm in order to remove starlight to help reveal planets that are covered by stellar glare. However, an issue with this algorithm is that it requires precise parameter inputs, and incorrect parameters can completely rid an image of all emitted signals. Adams further explores this parameter tuning. For our project, we use several methods from the PyKlip package for artificial injection with starlight subtraction.

Previous exoplanets have so far been discovered by a handful of alternative methods. The majority of exoplanets have been discovered by transit methods [4] which observes when a planet passes through the observer and a star, looking at the temporary dimmed light. An example of this is Venus passing between the earth and the sun. Another method for discovering exoplanets is through the radial velocity method [3] which relies on the shift in stars by small amounts in response to gravitational pulls, changing the color of the observed light. A less fruitful yet still successful method of discovering exoplanets is through gravitational microlensing [10] that focuses on bending light focused by gravity as a planet moves between the observer and a star. Though all of these methods have successfully discovered exoplanets, we predict that direct imaging through supervised deep learning methods will be the dominant method to discovering a large number of exoplanets in the near future.

We are currently working with a dataset containing approximately 6700 images from two star systems with confirmed exoplanets, and we've supplemented this dataset by using images which have artificially injected planets to improve generalization for further experimentation, which has shown to be successful by Jung (2020) [6]. Further, approaches such as by Malik et al (2021) [9] have shown that artificially injecting fake planets and removing known planets can dramatically boost the performance of classifiers for limited datasets in the astrophysics domain.

Additionally, our data contains images with nine channels, so we looked for different methods of reducing the dimensionality. In Wach et al. [16], they explore using mean, maximum, minimum, and an added convolutional layer. They found that by using different crayon methods combined with convolutional layers, their model performed the best. We also looked into other works that contained sparse features and less complex inputs. In Ranzato et al. [11], they found that sequential and stacked models outputted the best results when working with a similarly sparse dataset.

3. Methods

3.1. Classification Baseline

For our classification baseline, we chose to implement a simple network in order to match the sparse features of our input images. More specifically, we employ a network of the shape: 5x5 convolution with 9 input channels and 32 output channels, ReLU, 3x3 convolution with 32 input channels and 16 output channels, ReLU. The output of this ReLU is then flattened and passed to a linear layer with 810,000 input features which maps to two classes: planet vs. no planet. We chose to use this model in order to benchmark our progress in the classification task.

3.2. Object Detection

For the object detection task, we employed a number of different model architectures as backbones for the classification portion of each object detection model. Importantly we use Faster R-CNN for object detection, with a Faster R-CNN box predictor and a variety of different backbones[1]. The backbones we experimented with are ResNet-50, ConvNext-Tiny, MobileNetV2, VGG16, InceptionV3, and our custom implementation. For each of these, we experimented both with pretrained and newly initialized weights. The implementations of Faster R-CNN and the non-custom backbones are designed by PyTorch.

3.2.1 Object Detection Architecture

For object detection, we employed the Faster R-CNN architecture, which relies on three main steps. First, the Faster R-CNN runs a backbone CNN on an input image, generating output features. These output features are then passed to a region proposal network (RPN), which selects a few thousand regions of interest (RoIs) from the features. Before the next step of the model, an object classification and bounding box regression loss are computed specifically for the RPN, which respectively map to whether the model has correctly predicted if the image does or does not have an object in the region and the bounding box coordinates of the area proposed by the RPN. This loss equation can be understood as the following:

$$\mathcal{L}_{\text{RPN}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}}$$

$$\mathcal{L}_{\text{RPN}}(\{p_i\}, \{t_i\}) =$$

$$\frac{1}{N_{\text{cls}}} \sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{box}}} \sum_i p_i^* \cdot \mathcal{L}_{\text{box}}(t_i - t_i^*)$$

[1]We used Pytorch boilerplate code from here: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html

where i is the index of a particular RoI, λ is L2 regularization, p_i is the predicted probability of a particular location being an object, p_i^* is the actual label of the location (planet vs. no-planet), t_i is the predicted box location coordinates for the object, and t_i^* is the actual box location coordinates for the particular object. $\mathcal{L}_{cls}$ is the RPN classification loss and $\mathcal{L}_{box}$ is the RPN box loss [12].

Next, the RoIs generated by the RPN are then mapped onto the CNNs features using RoI pooling. RoI pooling entails placing the proposed regions onto subregions of the output CNN features. A max pooling is then applied to each of these subregions, at which point the model predicts the object's class (as opposed to just proposing whether an object exists or not) and the offset of the final bounding box. These final losses are thus similar in nature to those mentioned above except that they are computed over the pooled RoIs as opposed to over the RoIs themselves. The final loss equation for the Faster R-CNN architecture is thus:

$$\mathcal{L} = \mathcal{L}_{\text{RPN}} + \mathcal{L}_{\text{Final}}$$

As part of our experimentation methodology, we explored a number of different backbone CNN architectures for the overall Faster R-CNN architecture. These are described in greater detail in the following subsections.

3.2.2 Custom Backbone

We explored using a custom backbone with many fewer parameters and layers than any of the larger, state-of-the-art architectures we mention above (and in more detail below). Our reasoning for doing so is that our input images are both incredibly sparse and have 9 channels, each of which is a form of principal component analysis performed on the original exoplanetary images. As a result, these channels do not correspond to RGB channels and the resulting images are effectively greyscale images with very few identifying characteristics. Therefore, we hypothesized that the larger, existing architectures would be too complex and deep for the simple images which compose our dataset. Thus, we designed a 3-layer convolutional network to function as our custom backbone. At first, we used just alternations of Conv2d and ReLU blocks but then experiment with regularization layer, which improved performance. More specifically, our final architecture is [Convolution-BatchNorm-ReLU-MaxPool] x2 - [Convolution-BatchNorm-ReLU]. The first set of layers assumes an input channel size of 3, expecting that the 9 channel input images have already been downsampled using one of the downsampling schemes we describe in the data preprocessing section. The first convolutional layer uses 7x7 filters while the second two convolutional layers use 3x3 filters (all three use a stride of 1 and padding to maintain

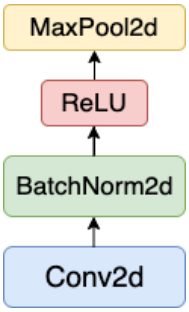

Figure 1. Structure of each block of the custom backbone.

image height and width). Both max pooling layers use 3x3 filters and a stride of 2.

3.2.3 ResNet-50

The first backbone we use is the ResNet-50 architecture. ResNet-50 is a 50-layer version of the ResNet architecture designed by He et al. [5]. The ResNet architecture involves many convolutional layers interspersed with max and average pooling layers, with the addition of residual connections. Residual connections involve combining the input of any individual layer with the output of the layer before passing the output to the next layer. This enables the model to maintain better efficacy for more layers as it is more capable of holding onto information from many prior layers. ResNet-50 is the smallest ResNet architecture (the other two versions have 101 and 152 layers); per our explanation in the custom backbone section, we believed that smaller, simpler models would be more effective for our less complex training and testing datasets. We use a PyTorch implementation of ResNet-50 and tested it both with pretrained and fresh weights. Both using pretrained and fresh weights, we were not even able to overfit to a small set of training data using this model. We expect that is because the architecture was just too complex for our task and its depth made it hard to train.

3.2.4 MobileNetV2

We also employ the MobileNetV2 architecture described in Sandler et al. (2019) [13]. MobileNet is a small CNN model which uses a combination of depthwise convolutions and inverted residual blocks in order to emphasize speed while still maintaining accuracy. Depthwise convolutions are effectively just the separate application of convolutional filters to each channel of input images. While original residual connections connect model layers with more channels and information, inverted residual blocks save information from smaller, "bottleneck" layers in order to help prevent information being lost to downsampling [13].

3.2.5 ConvNext–Tiny

The ConvNext model is a recent state-of-the-art convolution-based competitor to vision transformers. Based on the ResNet architecture, ConvNext also employs depthwise convolutions and inverted residuals (similar to MobileNetV2). Furthermore, the ConvNext architecture uses larger kernel sizes for convolution in order to simulate the self-attention mechanisms present in vision transformers; larger kernel sizes enable the model to learn longer-distance connections between different pixels. Finally, the ConvNext model also experiments with downsampling mechanisms, mainly changing how they are implemented in ResNet. In ConvNext, downsampling layers are split over the course of blocks, as opposed to being present at the start of each block. This more drawn out downsampling approach allows the model to progressively encode information as opposed to forcing the model to quickly consolidate features [7]. We use the "Tiny" variation of the ConvNext model per previously explained reasons – our images are sparse, so we hypothesize that smaller, less complex models should perform more accurately.

3.2.6 InceptionV3

The InceptionV3 model employs inception modules stacked on one another. Inception modules are characterized by their focus on maintaining similar depth and width. Each layer of the module can be described as a "wide" convolutional layer, meaning that there are multiple concurrent convolutional layers with different filter sizes which are applied to the duplicated layer input. Multiple of these "wide" convolutional layers occur in succession, after which the outputs of the final "wide" layer (meaning from each of the different convolutional filters) are then concatenated before being passed to another inception block. The InceptionV3 model improves on prior inception models by reducing feature dimensionalities progressively over the course of the model (similar to ConvNext-Tiny's improvement over ResNet). InceptionV3 also employs large filter sizes to learn longer-distance connections between pixels [15].

3.2.7 VGG16

VGG16 is a 16 layer convolution based model. Building on prior models, VGG16 employs more convolutions with smaller filter sizes, making the network deeper while maintaining the same number of overall receptive fields. Making the network deeper allows for more inter-layer activations which Simonyan and Zisserman (2015) [14] have shown to improve performance.

4. Dataset and Features

4.1. Acquiring Data

We interviewed Assistant Professor of Astronomy at Amherst College Dr. Kate Follette, an expert in the field of planetary detection. Prof. Follette runs the Follette Lab and graciously allowed us to train our model with her proprietary data. Each of the images in the input data is a 451x451x9 pixel image (meaning 9 channels), where the channels indicate different forms of principle component analysis in order to yield different reductions of the original images. These are performed in order to remove starlight and other image defects, which enable easier object detection. Furthermore, Prof. Follette provided us with the following useful information: whether the image contains a planet, whether the image is real or has an injected planet, the location of the planet in polar coordinates along with the distance from the center of the image of the planet (in pixels), and the date the image was taken. The whole dataset is composed of over 48k images with 17MB each, so 816GB total. We had access to the data over a shared Google Drive and built significant tooling to sync portions of the dataset automatically with our VMs in a time efficient manner.

4.2. Dataset Sizes and Types

4.2.1 Classification

For classifying each planet as having or not having a planet, we use 6,700 images containing real, confirmed exoplanets. After data preprocessing (described below), we have 26,800 images for training and test splits. Of these, we choose 800 train images and 200 test images at random.

4.2.2 Object Detection

As mentioned previously, our dataset has a combination of both real images and 'fake' images, which are images with injected planets. KLIP is an astronomical software program that was used to inject these 'fake' exoplanets. We were told that they are indistinguishable from real planets but later learned that this is most likely not actually the case. Since the dataset size is so large and images of the same star of the same day are fairly similar to each other except for random noise, we only used up to 100 images per star/date combination for the images containing fake planets and up to 150 for those containing real planets. There were images from 12 star systems containing fake images, taken on 27 different days. In addition, there were images from 2 star systems containing real planets taken on 9 different days. Due to our dataset size reduction, we use 4033 real and fake images, of which 1333 are real and 2700 are fake. After experimenting, we decided to only use 15 images per star-date combination since those are relatively similar. For our train/validation/test splits, we mix the real and fake images to yield a training dataset of 360 images, a validation dataset of 48 images, and a test dataset of 72 images.

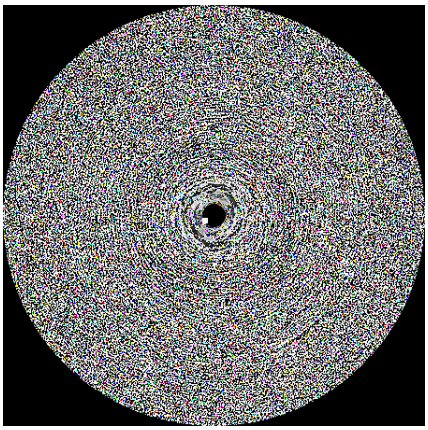

Figure 2. RGB visualization of image containing real exoplanet downsampled to 3 channels using mean. Box at star location.

4.3. Preprocessing Data

4.3.1 Baseline Image Classification

In order to preprocess the data, we had to contend with the issue of practically all of the images containing planets in them, meaning that our model would overfit to the training data and almost always predict that an image had a planet in it. As a result, we used a novel process in order to not only expand our training data but also fix our issue with regards to greedy image classification. Effectively, almost all of the images contain only one planet, in one "quadrant" of the image. Therefore, we divided each image into four separate images, with each image representing one quadrant of the original image. Using the polar coordinates given to us by Prof. Follette, we designated whichever quadrant of the reduction which contained a planet as the "true" image, thus creating three additional images per original image which did not have a planet. This has helped to resolve our issues with both data scarcity and data viability. While initially we had 6,700 images, after data augmentation we have 26,800 images to use for training/testing splits.

4.3.2 Object Detection

As mentioned previously, the images in our train, validation, and test datasets are a combination of real and fake images of dimension 451x451x9. Importantly, both our custom backbones and other state-of-the-art architectures we use expect a 3-channel input (i.e. RGB channels) so we explored many different downsampling schemes in order to reduce the images from 9 to 3 channels.

Namely, we attempted the following schemes: mean,

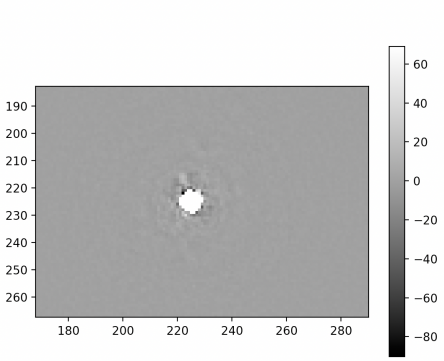

Figure 3. Single channel representation of a star (center) with planet just above it (the blurred shape).

max, min, sum, and convolution. For mean, max, min, and sum, we first grouped the 9 channels into 3 groups of 3. We grouped channels with their neighbors as these signify similar principal component analyses. We then performed the mean, max, min or sum operations over each group to yield one channel from each group. For convolution, we used a convolution with kernel size 1, stride 1, and padding 0, input channels 9, and output channels 3, which was prepended to the backbone network and trained. We experimented with each of these schemes when overfitting on a small train set and executed ablation studies to find out that mean and conv are the most performant downsampling models. More details are in the experiment section.

Part of Faster R-CNN is a builtin image normalization step to normalize pixels in each of the 3 channels between 0 and 1. However, the default values for the sample mean and standard deviation are taken from ImageNet, which is a highly unrepresentative dataset for our task. In order to achieve proper normalization, we computed the per-channel mean and standard deviations of our train dataset for each of the different downsampling methods separately and passed those into Faster R-CNN to ensure proper normalization.

Another issue with the data is the fact that the images have been aligned so that in each image of the same star at the same day, the exoplanets are in the exact same locations. That makes computing the bounding boxes easier, but it also means that we run at risk of overfitting to specific image locations. In order to prevent that, we randomly rotate every image between 0 and 360 degrees and adjust the target bounding box locations accordingly.

5. Experiments/Results/Discussion

5.1. Success Metrics

Apart from achieving low loss, we used two different accuracy metrics to measure quantitative model performance. The main job of our model practice is to find regions of images that might contain planets, which astrophysicists could then further inspect using statistical tools to strengthen or refute the model's hypothesis. Thus, we measured accuracy using the quality of the predicted bounding boxes. For overfitting and to get a general sense of which models and hyper parameters worked well, we used a relaxed accuracy metric that counted the number of images for which our model at least predicted one bounding box correctly containing a planet. We refer to this as the "overlap" accuracy. For further experiments, we then applied a stricter metric that divided the number of planets over all train, validation, and test images for which we predicted a bounding box correctly containing the planet divided by the total number of planets. We refer to this as "box-count" accuracy. In the Future Works section, we further describe other metrics we would like to include in future work such as precision, recall, F-1, and why they would be useful for this specific application.

5.2. Baselines

Our first baseline as part of the mid-project report was a simple 3-layer ConvNet that we used on the image classification formulation of our problem. Its structure is described in the methods section, and we trained it using the Adam optimizer for 3 epochs. We were able to achieve 98.53% test accuracy using this model.

Our baseline for the object detection task was the standard formulation of the Faster R-CNN pipeline using the mean to downsample images to three channels, and pretrained ResNet50 backbone and 2 epochs of training with Adam. We used a learning rate of 1e-4, 0.1 weight decay, and no learning rate scheduler. Those parameters are the result of hyperparameter tuning on the validation set, and our baseline achieved 0% test accuracy. We believe this poor performance was the result of ResNet50 being too complex of a model for our data and pre-training using a dataset that is too dissimilar from ours.[2]

5.3. Experimental Details and Hyperparameter Search

For specific overfitting experimentation and studies, we describe the appropriate hyperparameters in each of those sections. Generally, we use a mini-batch size of 1 image since Faster R-CNN can only deal with a batch size of 1. This is not an issue, however, since the presence of multiple bounding boxes per image effectively results in a "batched" training. Most images contain multiple planets, each of which presents a separate training opportunity for object classification and box regression.

We then selected a train set of 10 train images and tried to overfit overfit for each model backbone in order to create a fair comparison between them. There were many degrees

[2]We used the CS231N Assignment 2 training loop code

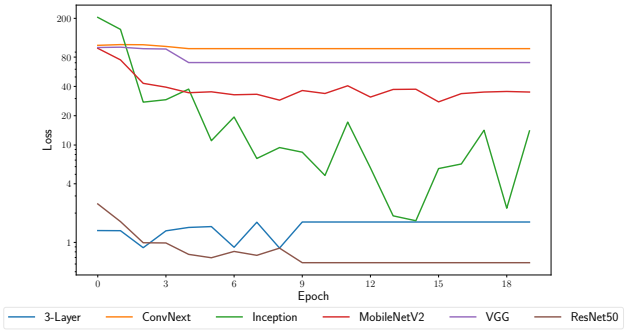

Figure 4. Loss graphs for overfitting on different model backbones. Epoch size varies due to model throughput speed differences and early stopping.

of freedom for our experimentation. We tried with and without different weight decays including ranging from 1e-5 to 1 in order to add regularization. For non-custom models, we trained both with and without loading pre-trained weights. The only-non custom model that achieved non-zero validation accuracies was InceptionV3. For this model, the version using newly initialized weights reached 20% test accuracy when overfitting compared to 0% with pre-trained weights, confirming our hypothesis that pre-training was not suitable for our task. We also adjusted learning rates, channel downsampling methods, learning rate scheduling, and number of training epochs, and tried Adam and SGD with Nesterov momentum as optimizers. We found that for overfitting, using our custom 3-layer backbone with convolutional channel downsampling, a learning rate of 1e-4, weight decay of 1e-5, a learning rate scheduler with gamma 0.1 applied every epoch and Adam optimizer (with betas 0.9 and 0.999) trained for 3 epochs worked best, achieving an 80%-100% validation overfitting accuracy depending on the seed. This was our starting point for tuning a model on the full dataset. The overfitting loss curves for different model backbones using their respective best parameters is depicted below.

5.4. Model Search

We tried to overfit on a subset of our data using the different backbone models. Out of them, only our custom 3-layer backbone and InceptionV3 without pre-training achieved some amount of overfitting with 80%-100% validation and 20% train accuracy respectively. We thus decided to use these models when fine-tuning on the full dataset, but still ran some experiments on other models as well in case the overfitting tests were unrepresentative.

5.5. Channel Downsampling

As mentioned in the data section, the raw training images have 9 channels, so we needed to experiment with methods to turn them into 3-channel tensors in order to use non-custom backbones. In order to choose the most effective downsampling scheme for larger trials, we experimented with each of the downsampling schemes described in the data preprocessing section on small portions of the train set (5) and test set (10) images. We use a learning rate of 1e-4, L2 weight decay of 1e-5, Adam optimizer with betas 0.9 and 0.999, step learning rate scheduler with step size of 1 and gamma of 0.1, and a training length of 10 epochs. We use the custom 3-layer network we describe in the Methods section as our model since it seemed to most capable at overfitting. For the mean, max, min, and sum schemes, we split the 9 channels into groups of 3 and then performed the respective downsampling operation along the channel dimension in order to end up with 3 groups of channel size 1, which we stacked to create a 3-channel input image. For the conv operation, we prepended a learnable Conv2d layer to the backbones which performed a 1x1 convolution with 9 input and 3 output channels without padding and stride 1. The results of these experiments are listed in the following table:

Scheme	Val. Accuracy	Test Accuracy
Mean	20%	0%
Max	0%	0%
Min	0%	0%
Sum	0%	0%
Conv	**80%**	**20%**

Table 1. Validation and test accuracies from different downsampling schemes using the "less" strict accuracy metric.

Notably, we found that the conv and mean schemes performed the best out of the various downsampling schemes. We attribute the fact that the max and min schemes perform poorly to the fact that information from 6 of the 9 channels is effectively disregarded in both of these methods. Furthermore, we propose that the mean outperforms the sum scheme as it provides some data normalization. Finally, the conv scheme likely performed the best because it enables the model to learn the relationships between the different principal components, enabling the model to weight each principal component analysis channel differently as opposed to equally, per the mean scheme.

5.6. Results

We then fine-tuned our best model configurations on the full dataset. Our best performing model was our custom 3-layer backbone using the mean channel downsampling scheme, a learning rate of 1e-1, no learning rate scheduling,

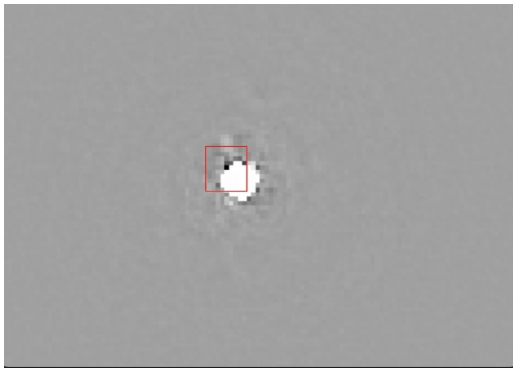

Figure 5. Correct example output of proposed bounding box from our model (single image channel).

and Adam to train for 3 epochs. We achieved a 36.1% accuracy on the test set using our overlap accuracy metric and 24.7% using the stricter box-count accuracy metric. Qualitatively, the model predictions frame the planet locations well. However, there are lots of false positives and boxes that are either extremely close to or overlapping each other. The same parameters but using conv as a channel downsampling method yielded a 12.5% test accuracy on the relaxed metric and 5% on the strict one, which was surprising since this method performed better during overfitting. We believe this is the case because the initial downsampling convolution adds more complexity to the network and reduces the number of channels in the first layer while normally, performant models have a larger number of output than input channels in the earlier layers. All other backbones did not correctly detect any planets in the test dataset, which we attribute to the aforementioned reasons relating to model complexity.

5.7. Visualization

We found it helpful to visualize our model by examining both saliency maps and the bounding boxes[3]. As seen in figure 4, the proposed box overlaps with the discovered exoplanet and stretches into the region of the star. We predicted that the box would be less accurate and were pleased to see that most of the planet was covered by the region.

We were interested in looking at the saliency maps to observe where the most important features for the image classification were as seen in figure 5. We predicted that the regions surrounding the planet and the star would be the most important, with the edges of the image as less important. By observing the saliency map, we found that the regions surrounding the planet containing only the sky were less important for classification. It is interesting to note that

[3]We used the saliency map code template from assignment 2 as a starting point.

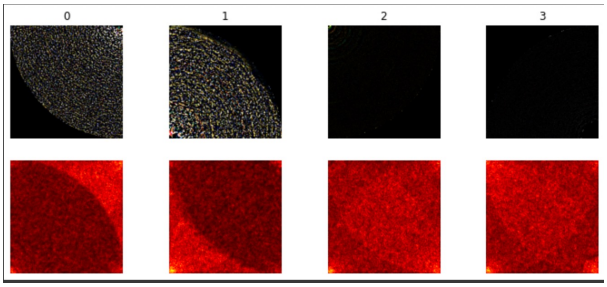

Figure 6. Four images with their corresponding saliency maps.

the saliency maps look inverted from the image. This might be caused by a diminishing necessity for information in the images as you move further away from the planet.

6. Conclusion/Future Work

6.1. Conclusion

In this project, we implemented both classification and object detection algorithms for exo-planet detection. Our image classification algorithm performs remarkably, achieving 98.53% test accuracy. Our object detection algorithm achieves 36.1% overlap accuracy and 24.7% box-count accuracy, which are impressive given the incredibly difficult nature of the task.

Overall, we experimented with data augmentation, hyperparameters, data invariances, and different CNN designs. We find that simpler, custom models (like the 3-layer model of our own design) work the best for the sparse features of exoplanetary images. Specifically, we found that using simpler backbone CNNs can on average improve overlap accuracy by around 60% based on overfitting tests.

6.2. Future Work

Moving forward, we would be interested in exploring the other methods of exoplanet discovery, such as as radial velocity, transit, and gravitational microlensing. It would be interesting to explore if deep convolutional networks could reduce the need for manual identification and verification as it currently stands. Additionally, there is one more method that has led to the discovery of a single exoplanet: astrometry [2]. This method observes micromovements of stars nearby caused by the orbit of a planet, so it would be interesting to learn more about this methodology and apply deep learning methods to astrometry as well. Further, we think iterating on the model architecture and hyperparameters would be important to reduce the number of false positives. In future work, we would want to measure precision, recall, and F-1 in order to monitor those metrics as well.

7. Contributions and Acknowledgements

Simon worked on model design, model research, hyperparameter experimentation, and data visualization. Moritz worked on the model design, hyper parameter tuning and dataset preparation. Griffin worked on research, data visualization, initial models, and channel dimensionality reduction.

Many thanks to Alexander DelFranco and Prof. Follette for providing the dataset and additional background information related to their research. We also thank Manasi Sharma for additional knowledge in the field and general support.

References

[1] Jeá Adams, Jason Wang, and Kate Follette. Optimization techniques for exoplanet direct imaging. In *https://baas.aas.org/pub/2021n1i344p05/release/ 1?readingCollection=27a4f9bb*, 2021. 2

[2] G. F. Benedict, B. E. McArthur, T. Forveille, X. Delfosse, E. Nelan, R. P. Butler, W. Spiesman, G. Marcy, B. Goldman, C. Perrier, W. H. Jefferys, and M. Mayor. A mass for the extrasolar planet gliese 876b determined from hubble space telescope fine guidance sensor 3 astrometry and high-precision radial velocities. In *https://iopscience.iop.org/article/10.1086/346073/pdf*, 2002. 8

[3] Bruce Campbell, G. A. H. Walker, and S. Yang. A search for substellar companions to solar-type stars. In *https://articles.adsabs.harvard.edu/full/1988ApJ...331..902C*, 1988. 2

[4] David Charbonneau, Timothy M. Brown, David W. Latham, and Michel Mayor. Detection of planetary transits across a sun-like star. In *https://iopscience.iop.org/article/10.1086/312457/pdf*, 1999. 2

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *https://arxiv.org/pdf/1512.03385.pdf*, 2015. 4

[6] Dawoon Jung. Recent machine learning applications in space. In *IEEE Potentials (https://ieeexplore.ieee.org/document/9132719)*, 2020. 2

[7] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *https://arxiv.org/pdf/2201.03545.pdf*, 2022. 4

[8] Jared R. Males, Laird M. Close, Kelsey Miller, Lauren Schatz, David Doelman, Jennifer Lumbres, Frans Snik, Alex Rodack, Justin Knight, Kyle Van Gorkom, Joseph D. Long, Alex Hedglen, Maggie Kautz, Nemanja Jovanovic, Katie Morzinski, Olivier Guyon, Ewan Douglas, Katherine B. Follette, Julien Lozi, Chris Bohlman, Olivier Durney, Victor Gasho, Phil Hinz, Michael Ireland, Madison Jean, Christoph Keller, Matt Kenworthy, Ben Mazin, Jamison Noenickx, Dan Alfred, Kevin Perez, Anna Sanchez, Corwynn Sauve, Alycia Weinberger, and Al Conrad. Magao-x: project status and first laboratory results. In *https://arxiv.org/pdf/1807.04315.pdf*, 2018. 1

[9] Abhishek Malik, Benjamin P. Moster, and Christian Obermeier. Exoplanet detection using machine learning. In *https://arxiv.org/pdf/2011.14135.pdf*, 2021. 2

[10] Shude Mao and Bohdan Paczyński. Gravitational microlensing by double stars and planetary systems. In *https://articles.adsabs.harvard.edu/full/1991ApJ...374L..37M*, 1991. 2

[11] Marc'aurelio Ranzato, Y lan Boureau, and Yann Cun. Sparse feature learning for deep belief networks. In *https://papers.nips.cc/paper/2007/hash/c60d060b946d6dd6145dcbad5c4ccf6f-Abstract.html*, 2007. 2

[12] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *https://papers.nips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf*, 2015. 3

[13] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *https://arxiv.org/pdf/1801.04381.pdf*, 2019. 4

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *https://arxiv.org/pdf/1409.1556.pdf*, 2015. 4

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *https://arxiv.org/pdf/1512.00567.pdf*, 2015. 4

[16] Hans B. Wach, Edward R. Dowski, and W. Thomas Cathey. Channel reduction and applications to image processing. In *https://opg.optica.org/ao/fulltext.cfm?uri=ao-39-11-1794id=44452*, 2000. 2