The Astrophysical Journal, 880:83 (14pp), 2019 August 1 https://doi.org/10.3847/1538-4357/ab275e

CASI: A Convolutional Neural Network Approach for Shell Identification

Colin M. Van Oort[1], Duo Xu[2], Stella S. R. Offner[2], and Robert A. Gutermuth[3]
[1] Complex Systems Center, University of Vermont, Burlington, VT 05405, USA; cvanoort@uvm.edu
[2] Department of Astronomy, The University of Texas at Austin, Austin, TX 78712, USA
[3] Department of Astronomy, University of Massachusetts—Amherst, Amherst, MA 01003, USA
Received 2019 March 20; revised 2019 May 16; accepted 2019 June 4; published 2019 July 29

Abstract

We utilize techniques from deep learning to identify signatures of stellar feedback in simulated molecular clouds. Specifically, we implement a deep neural network with an architecture similar to U-Net and apply it to the problem of identifying wind-driven shells and bubbles using data from magnetohydrodynamic simulations of turbulent molecular clouds with embedded stellar sources. The network is applied to two tasks, dense regression and segmentation, on two varieties of data, simulated density and synthetic ^{12}CO observations. Our Convolutional Approach for Shell Identification (CASI) is able to obtain a true-positive rate greater than 90%, while maintaining a false-positive rate of 1%, on two segmentation tasks and also performs well on related regression tasks. The source code for CASI is available on GitLab.

Key words: ISM: bubbles – ISM: clouds – methods: data analysis – stars: formation – techniques: image processing

1. Introduction

Forming stars influence their environments by injecting energy over a large dynamic range, with different sources contributing at different times and characteristic length scales. Stellar feedback has been invoked to explain a host of phenomena, including the relation between dense cores and the stellar initial mass function (IMF, Alves et al. 2007; Offner et al. 2014), the longevity of turbulence within molecular clouds (Cunningham et al. 2006; Wang et al. 2010; Offner & Liu 2018), the properties of multiple star systems (Offner et al. 2016), and the global efficiency of star formation (Krumholz et al. 2007; Federrath 2015; Lee et al. 2015). Nevertheless, the energetics and impact of feedback remain poorly constrained.

Identifying feedback signatures and quantitatively disentangling the interaction with the environment are notoriously difficult. For decades, astronomers have studied the distribution of gas in the interstellar medium by making 2D dust emission and absorption maps and 3D atomic and molecular spectral cubes. A variety of algorithms have been developed to identify peaks in the data, namely cores and filaments, including CLUMPFIND, DENDROGRAMS and GETFILAMENTS (Williams et al. 1994; Goodman et al. 2009; Men'shchikov 2013). However, simple structure identification algorithms like these fail to identify feedback signatures, which exhibit a variety of complex morphologies. Statistical approaches, such as principle component analysis and the spectral correlation function provide a means to quantify the underlying impact of feedback on the turbulent cloud structure; however, many statistics commonly applied to spectral line cubes are relatively insensitive (Boyden et al. 2018). Consequently, the imprint of feedback is usually identified "by eye" (Churchwell et al. 2006; Arce et al. 2010, 2011; Li et al. 2015).

The human brain is a superb tool for parsing complex images (Zhang & Zhang 2010), and a variety of papers have used visual identification to study feedback in surveys of individual regions (e.g., Könyves et al. 2007; Arce et al. 2010, 2011; Narayanan et al. 2012; Li et al. 2015). Features produced by stellar winds and outflows resemble shells, bubbles, or cones in intensity maps, which is one way they can be visually identified (Churchwell et al. 2006; Offner et al. 2011; Simpson et al. 2012). In spectral line data, such as CO observations, feedback often appears connected over a range of velocities (frequencies), so astronomers often identify feedback by searching for coherent three-dimensional structures (Arce et al. 2010, 2011; Li et al. 2015). Meanwhile, the explosion of data over the last decade and production of large surveys, such as those covering the entire Galactic plane, has outstripped the analysis capacity of professional astronomers. This has led to a variety of "citizen science" efforts, in which interested members of the public visually inspect and characterize the data. Galaxy Zoo, which has undergone a number of iterations, involved millions of people, and produced dozens of papers to date, has the highest profile of these initiatives (e.g., Lintott et al. 2008). Recently, the Milky Way Project applied the power of citizen science to the identification of stellar feedback in the *Spitzer* Galactic plane surveys GLIMPSE and MIPSGAL. This effort yielded a catalog containing the locations and sizes of thousands of new bubbles in the Milky Way (Simpson et al. 2012).

However, human classification, while formidable, has several disadvantages. Although numerous people devote significant time to data parsing, citizen hours are finite and only certain problems can be formulated into simple pattern searches for non-experts. Moreover, classifications are subjective and differ between people. This can produce different catalogs and conclusions for the same data even between experts (compare Narayanan et al. 2012 with Li et al. 2015, for example).

The very nature of feedback ensures that human identification will be ambiguous. Since stellar feedback acts on the interstellar medium, which by nature has a strongly inhomogeneous density and velocity distribution, signatures are usually asymmetric and often blend into the turbulent background (Arce et al. 2010, 2011). Voids, low-density regions that are produced by supersonic turbulence, may also masquerade as feedback-driven bubbles, causing false positives. Although stellar feedback can accelerate cloud gas to velocities above the mean cloud turbulent velocity, the peak velocity of the feedback is sensitive to the source orientation with respect to the line of sight and its location relative to the cloud boundary, where gas changes phase from molecular to atomic (Arce et al. 2010; Offner et al. 2011; Li et al. 2015). These complications mean that even experts have trouble unambiguously and accurately identifying feedback.

Algorithmic approaches to identifying bubbles have been utilized in order to remove some of the subjectivity of bubble identification, while also removing some of the burden from human identifiers (Giri et al. 2017). However, more traditional algorithmic approaches tend to lack the flexibility required for widespread application.

One alternative approach is machine learning, a sub-field of computer science in which algorithms adapt to patterns and correlations in data. Machine learning is now a mature field, and is commonly applied to pattern recognition problems, including topics ranging from genome sequencing to face recognition to drug discovery (LeCun et al. 2015). Machine learning can automate the process of feature identification, scale efficiently to large data sets, and produce repeatable catalogs. However, to date it has been applied relatively sparsely to problems in astrophysics.

In this work, we present our convolutional approach for shell identification (CASI; Van Oort 2019).[4] CASI is a convolutional neural network (CNN), a variety of artificial neural network (ANN) where the primary unit of computation is the convolution operation rather than simple matrix multiplication. For an overview of convolution arithmetic in the context of machine learning, see the Appendix for a brief overview and Dumoulin & Visin (2016) for a more comprehensive guide.[5] ANNs are a computational model that is loosely inspired by biological neural networks, where the fundamental unit of computation is a single neuron that receives one or more stimuli and provides one or more output signals. See Section 3 in Lieu et al. (2019) for a brief overview of ANNs that targets the astronomy audience.

CASI is designed to identify feedback signatures in molecular clouds, with a focus on wind-driven bubbles created by intermediate-mass stars. This is motivated by the observation that such shells identified in nearby star-forming regions, like the Perseus molecular cloud, have a huge impact on the cloud energetics and evolution (Arce et al. 2011). Magnetohydrodynamic (MHD) simulations with embedded sources are used to train our method and investigate its efficacy.

In the remainder of Section 1, we summarize relevant machine-learning applications in the literature. We describe our method in Section 2 and present results in Section 3. Finally, conclusions and a discussion are provided in Section 4.

1.1. Machine Learning for Image Tasks

Hubel & Wiesel (1968) identified specialized neurons in the visual cortices of cats and monkeys that process small, partially overlapping regions of their visual field. This pattern of local, overlapping connectivity inspired the design of the Neocognitron (Fukushima & Miyake 1982), a neural network-based approach to character and digit recognition. However, difficulties encountered when training networks with more layers and a lack of sufficient training data led to a decline in the popularity of ANNs, with alternative methods such as support vector machines (SVMs) receiving more attention.

More than a decade later LeCun et al. (1998) introduced LeNet-5, a CNN that broke the record for character recognition performance and became a baseline architecture for many applications of CNNs that followed, contributing to a resurgence in the popularity of ANNs. This resurgence ushered in a wave of research and targeted hardware improvements that allowed ANNs to overtake many competing machine-learning algorithms and attain state-of-the-art results on a variety of tasks, rivaling human performance in some cases (Nasser et al. 2017).

In addition to character recognition, CNNs have been successful at image classification, object detection, semantic segmentation, and image denoising/artifact removal to name a few. For a broad overview of the techniques involved in CNNs and their applications see Gu et al. (2018).

1.2. Previous Applications to Astronomical Data Analysis

Machine-learning techniques have been applied to structure detection in astronomical data several times with varying degrees of success.

Beaumont et al. (2011) used SVMs to segment ^{12}CO data containing a supernova remnant partially obscured by a molecular cloud, reaching >90% accuracy when classifying hand-labeled pixels as belonging to the supernova remnant or molecular cloud.

SVMs are a supervised learning method that classifies data by finding a decision boundary that simultaneously minimizes classification error and maximizes the distance between the boundary and the closest samples of any class. SVMs may also be applied to regression problems. Such applications are often referred to as support vector regression. Since SVMs attempt to maximize the margins about the decision boundary they tend to generalize well and feature robustness to minor perturbations of input data. Interested readers should refer to Bennett & Campbell (2000) for an overview of SVMs.

Beaumont et al. (2014) developed BRUT, a method that utilizes Random Forest classifiers, to identify bubbles and similar structures in color-composite images from the *Spitzer Space Telescope*. However, BRUT is sensitive to the position of the bubble in the image, making wide-field searches computationally expensive (Xu & Offner 2017).

Deep learning is a relatively new and rapidly evolving sub-field of machine learning that features ANNs with sophisticated architectures and greater numbers of layers. Relatively few astrophysical applications utilize deep-learning techniques, which may be partly due to the age and the rapid research pace of the field. Daigle et al. (2003) utilized a Multi-Layer Perceptron (MLP), a simple neural network architecture that features consecutive layers of densely connected artificial neurons, to identify expanding shells in the Canadian Galactic Plane Survey, obtaining a 0.6% false-positive rate. Later Daigle et al. (2007) compared the performance of the MLP against two alternative network architectures, the competitive network and the growing neural gas network, on similar data. There was no clear winner in this comparison, as all three networks were able to correctly identify 10 out of the 11 bubbles considered when evaluated using a leave-one-out cross-validation method.

Dieleman et al. (2015) applied a CNN to the morphological classification of annotated images from the Galaxy Zoo project, attaining an accuracy >99% for images where human annotators strongly agreed upon the classification label. The authors suggest that a machine-learning system could be used to classify the "easy" images, leaving the more difficult cases for human annotators. Filtering the images in this way could

[4] The source code for CASI is available on GitLab: https://gitlab.com/casi-project/casi-2d.

[5] An associated GitHub page provides helpful animations: https://github.com/vdumoulin/conv_arithmetic.

lead to a reduced workload for human annotators when processing large surveys.

Lanusse et al. (2018) trained a CNN to identify the existence of gravitational lensing in simulated data that was constructed to resemble Large Synoptic Survey Telescope observations. This approach reached a true-positive rate $\geqslant 80\%$ while maintaining a false-positive rate of 1% on samples with varying signal-to-noise ratios.

The network employed in Lanusse et al. (2018) utilizes residual connections, a network architecture feature introduced by He et al. (2016), where identity connections combine the input and output data of a block of operations. Residual connections effectively change the underlying model of a network, or network component, from $y = f(x)$ to $y = f(x) + x$, and encourage the network to learn iterative transformations of the input rather than a direct mapping (Jastrzebski et al. 2017). Networks and network components that incorporate residual connections can easily learn the identity function, which allows them to mitigate the effects of harmful or underperforming components during learning. These properties allow architectures with residual connections to effectively utilize a greater number of layers and a larger number of model weights than architectures that do not include residual connections. See Appendix A.6 for more details on residual architectures.

Primack et al. (2018) utilized a simple CNN to classify images from the CANDELS survey into one of three phases of galaxy evolution. The network is trained using simulated CANDELS-like observations and then applied to real data, reaching a so-called "Blue Nugget" phase galaxy identification accuracy of around 80%. This application involves a relatively small data set, thus the authors implemented several measures to keep the network from overfitting, including data augmentation and dropout.

Lieu et al. (2019) trained a CNN to classify solar system objects from other astronomical sources in simulated data. The network is initialized with weights that were trained on the ImageNet data set and then fine-tuned on 7512 simulated Euclid images. Similar to Primack et al. (2018), this work utilizes various techniques to mitigate overfitting, including batch normalization (see the Appendix), dropout, and data augmentation. After testing several modern CNN architectures, Lieu et al. (2019) are able to reach an accuracy of 95.6% when distinguishing between four classes of stellar objects.

Most recently, Diaz et al. (2019) investigated the classification of simulated galaxies into three classes. CNNs were applied to this task, using data generated from N-body simulations as training data, and they were able to obtain an accuracy exceeding 99%.

The extent of previous work in this area, as well as the lack of a comprehensive and automated solution, motivates further application of machine-learning techniques to structure identification in studies of star formation and the interstellar medium. In this study we apply the U-Net architecture, which is described in the following section, to several tasks derived from MHD simulation data.

2. Method Overview

2.1. Neural Network Architecture

In this work we employ a Residual U-Net, a variant of the U-Net architecture developed by Ronneberger et al. (2015) where the fundamental unit of construction is a residual block (He et al. 2016) rather than a single convolution. A residual block is simply a sequence of consecutively applied convolution operations that are spanned by a residual connection. See Appendix A.6 for more details.

The U-Net architecture and its derivatives have grown in popularity since their introduction, and residual U-Nets in particular have been applied to a wide variety of problems, including road segmentation (Zhang et al. 2018), detection of pulmonary nodules (Lan et al. 2018), segmentation of optic nerve tissue (Devalla et al. 2018), and several other medical segmentation tasks (Zhu et al. 2019).

Figure 1 displays our residual U-Net architecture and provides details on the structure of each sub-component. Beyond the addition of residual connections, we also make a few other small alterations to the original U-Net architecture.

In particular, we utilize padded convolutions[6] in place of unpadded convolutions, which results in feature maps with identical spatial dimensions at corresponding levels of the downsampling and upsampling paths. This removes the need to apply cropping to the cross-connections and allows the depth of the network to be modified more easily when a particular problem benefits from the use of higher-level features. We make use of batch normalization prior to each activation function, which was not used by Ronneberger et al. (2015), since it can stabilize training and act as a light regularizer (Ioffe & Szegedy 2015). Note that batch normalization is not strictly necessary and may have a negative effect on performance for some tasks, thus it may be useful to re-evaluate its use when applying this architecture to new problem domains.

2.2. Training

We utilize stochastic gradient descent (SGD) with momentum to train our networks, following results from Wilson et al. (2017) that suggest SGD may provide better generalization properties than adaptive step size methods, such as ADAGRAD (Duchi et al. 2011) and ADAM (Kingma & Ba 2014). Ruder (2016) provides an excellent overview of gradient descent algorithms, with a focus on variants used in deep-learning research and applications.

SGD is an optimization algorithm where the parameters of a function, such as the weights of a neural network, are adjusted using the gradient of a loss function with respect to those parameters. The loss function provides a performance criterion and the gradient of the loss with respect to the model parameters indicates how the parameters should be adjusted in order to reduce the loss. The backpropagation algorithm, an application of the chain rule from differential calculus, distributes the gradients backwards through the network starting from the final layer.

The behavior of SGD can be controlled via the use of several parameters including the learning rate, batch size, and momentum intensity. The learning rate scales the magnitude of weight updates applied to the network in each step of SGD. Utilizing learning rates that are too high can lead to divergence,

[6] Padded convolutions augment the convolution operation by extending the spatial dimensions, e.g., height and width, of the input with generated data. One common padding scheme is to apply a band of zeros that is half as wide as the spatial dimensions of the convolution filter in that direction. This scheme results in a convolution whose input volume and output volume have identical dimensions when the convolution filters have odd spatial dimensions (e.g., 3, 5, 7, ...).

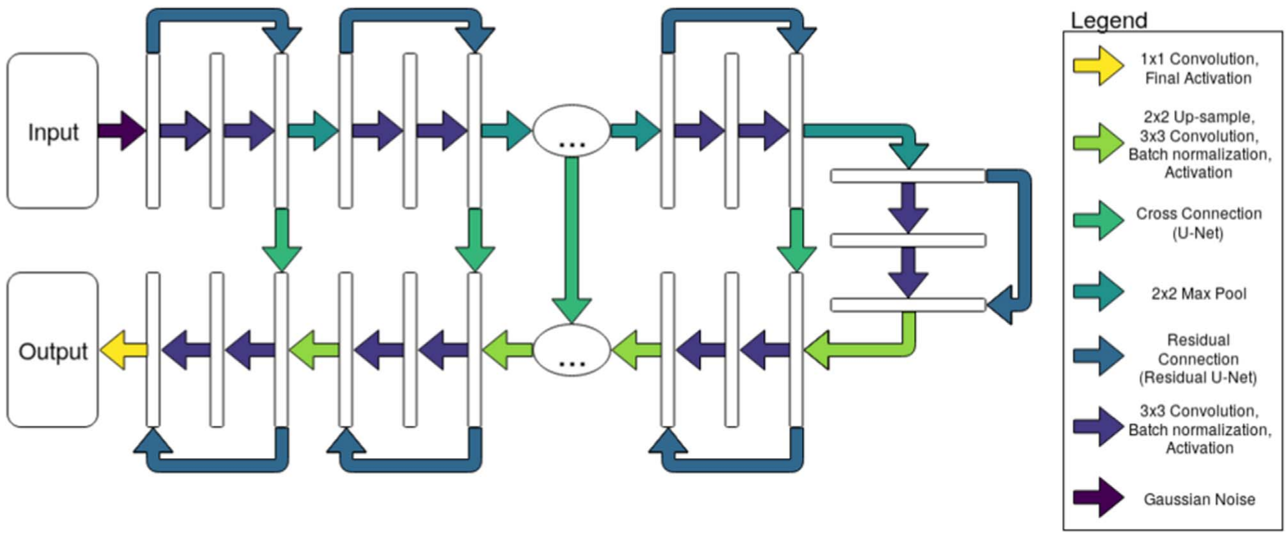
Input
Output
...
...
Legend
1x1 Convolution, Final Activation
2x2 Up-sample, 3x3 Convolution, Batch normalization, Activation
Cross Connection (U-Net)
2x2 Max Pool
Residual Connection (Residual U-Net)
3x3 Convolution, Batch normalization, Activation
Gaussian Noise

Figure 1. A flow-chart style depiction of our residual U-Net architecture, which maps a 4D stack of images with dimensions (number of images, height of images, width of images, number of image channels) to a 4D stack of images (number of images, height of images, width of images, number of output channels). In the context of astronomy data, the "images" in question may be slices of observational data volumes and the height/width are literally the height and width of the observational data (in pixels/voxels). In this work we only utilize a single input image channel (gas density in a voxel, CO intensity in a voxel, etc.); however, it is possible to supply the network with multiple varieties of data using multiple input image channels. The first half of the network uses downsampling operations to compress input features and construct higher-level representations, while the second half of the network uses upsampling operations to reconstitute the abstract representations. Cross-connections allow information from the downsampling path to be utilized in the upsampling path, leading to mappings that benefit from the combination of coarse and fine grained features. The ellipsis-within-oval graphics indicate that the depth of the architecture is variable and may be modified by the user.

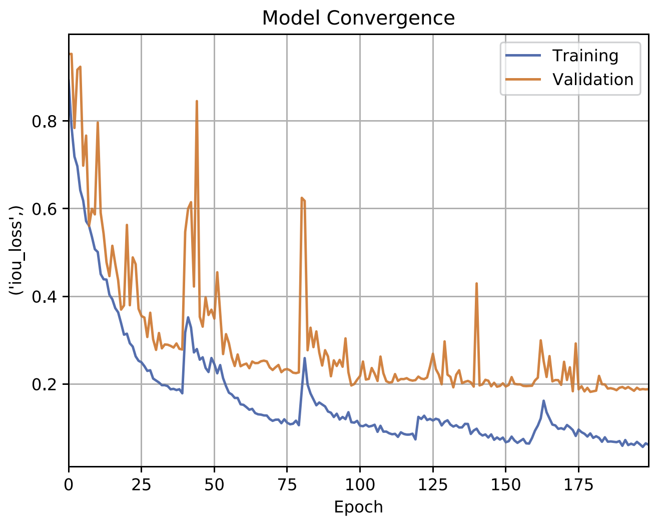

Figure 2. Example training and validation loss curves for a residual U-Net trained on the CO segmentation task using the Intersection over Union loss function. The spikes that occur every 40 epochs are a feature of the cyclic learning rate schedule. Note that by the 200th epoch, network performance appears to have converged, with little improvement in the validation loss for 50–75 epochs.

where the loss increases after each update and the network fails to learn, while learning rates that are too low may lead to premature convergence and extended training times.

Batch size determines how many training samples will be used to calculate the gradient at each step of SGD. Utilizing a batch size of one results in what is usually referred to as online SGD, while a batch size equal to the size of the training set results in batch SGD, and the use of batch sizes that fall between these two extremes results in mini-batch SGD. The batch size parameter features a trade-off between calculation speed and gradient accuracy when considering smaller versus larger batch sizes. Nearly all modern applications of ANNs use mini-batch SGD for training, since online SGD can introduce too much noise into the training process and batch SGD tends to take too long to converge, though there is not a strong consensus on the optimal batch size setting. Masters & Luschi (2018) indicated that smaller batch sizes, between 2 and 32, tend to work well in many cases. Hoffer et al. (2017), however, suggested that larger batch sizes may also be effective, as long as the training duration is extended accordingly.

Momentum is an extension to SGD where each weight update is a linear combination of the current gradient and the previous weight update, which can reduce oscillations in weight updates and speed up training convergence (Goh 2017). The momentum intensity parameter usually falls in the range [0, 1) and controls the fraction of each weight updated that comes from the previous update. For example, setting the momentum intensity to 0.9 will result in each update consisting of the previous update multiplied by 0.9 plus the current gradient multiplied by 0.1. Alternatively, the momentum intensity may simply act as a learning rate applied to the previous weight update, rather than also scaling the current update.

We train our networks for 200 epochs[7] of SGD with the momentum parameter set to 0.9 and a batch size of 8. Figure 2 shows that our models are able to converge under these conditions, thus 200 training epochs is sufficient for the conditions investigated here.

The networks are initialized with random weights using the Glorot initialization scheme (Glorot & Bengio 2010). We utilize the uniform distribution variant of this scheme, which draws samples from a uniform distribution over the interval $[-x, x]$, where

$$x = \sqrt{6/(\text{fan\_in} + \text{fan\_out})},$$

[7] A single training epoch involves several gradient updates, such that the network is exposed to each sample of the training set exactly once. In our case a single epoch consists of $\lceil N_T/B \rceil$ gradient updates, where N_T is the number of training samples and B is the batch size.

Table 1
Network Hyper-parameters

Name	Definition	Symbol	Value
Batch Size	Samples provided during each training iteration	B	8
Depth	Number of blocks used in network construction	D	4
Filter Count	Filters allotted for each convolution operation	F	16
Noise Strength	Std. dev. of the noise applied to network inputs	σ	0.003

fan_in is the number of input units for a weight tensor, and fan_out is the number of output units. Note that fewer training iterations may be required if the models are initialized with weights that have been previously trained on a similar data set and task, a process that is usually referred to as transfer learning (Pan & Yang 2010). During training, the learning rate is adjusted using the cyclic learning rate schedule described in Huang et al. (2017), with a maximum learning rate of 0.2 and 5 cycles of 40 epochs. Additionally, the training samples are shuffled at the end of each epoch, which effectively adds a small amount of noise to the gradient updates and can reduce the chance of getting stuck in a local optimum. Finally, the model state is saved, via a check-pointing utility, each time a new minimum error is observed on the validation set.

2.3. Hyper-parameters and Hyper-parameter Selection

This section provides a detailed description of relevant hyper-parameters and how they influence performance of the model discussed in Section 2.1. Table 1 provides a brief summary of these hyper-parameters and the values utilized in our experiments.

The batch size of the network, which controls how many images are provided to the model simultaneously during training and inference, is determined by B. As mentioned in Section 2.2, there are trade-offs to be considered when selecting the batch size parameter. Larger batch sizes allow samples to be processed in parallel and may reduce training and inference times at the cost of additional memory overhead. Batch sizes greater than one allow for the aggregation of gradients over several data samples, providing more accurate gradient estimates and potentially reducing the number of training iterations required for the loss function to converge. Small batch sizes have been shown to have a beneficial regularizing effect on deep neural networks that may improve generalization (Keskar et al. 2016; Hoffer et al. 2017; Masters & Luschi 2018). Though progress has been made toward improving the effectiveness of networks trained with large batch sizes (Hoffer et al. 2017; Smith et al. 2017), we tended to use small batch sizes due to the memory limitations of graphics processing units (GPUs) used to accelerate training.

The depth parameter, D, determines the number of fundamental blocks that are used in the construction of a particular network. For the residual U-Net this is the number of convolution blocks, e.g., pairs of convolutions and associated operations such as batch normalization and residual connections, present in both the compressive and decompressive paths.

Each block in the residual U-Net contains a spatial resampling operation, max pooling in the compressive path, and nearest-neighbor upsampling in the decompressive path. See Appendix A.3 for a brief overview of the mechanics and benefits of max pooling.

Thus, D governs the amount of dimension manipulation present in these architectures as well as the ability of the network to interact with the data at different spatial resolutions.

The depth parameter also contributes to the expressiveness of the network, as each fundamental block includes one or more convolution operations. The expressiveness of a particular network refers to its ability to accurately approximate various functions. When considering two competing networks, X and Y, network X is more expressive than network Y if the set of functions that X is able to accurately approximate is a superset of the set of functions that Y is able to accurately approximate. With this in mind, increasing D, and thus the number of model parameters, tends to improve the ability of the model to approximate functions and therefore increases its expressiveness.

The number of filters, F, indicates how many filters are allotted for each convolution operation. See Appendix A.2 for more information about how the filters are used in the model. Each downsampling operation increases the number of filters allotted to downstream convolution operations by a factor proportional to the dimension reduction, and similarly, each upsampling operation decreases the number of filters provided for downstream convolutions in proportion to the increase in spatial dimensions.

Additive Gaussian noise may be applied to the network inputs during training in order to avoid overfitting, and the standard deviation of this noise is controlled by the σ parameter. The application of random noise to training samples can improve the robustness of the resulting method to small data perturbations and reduce the chances of overfitting to the training data.

ANNs often require a computationally expensive hyper-parameter search process in order to reach desired performance levels. Some factors that contribute to the computational cost of this search are the number of hyper-parameters to be optimized, resources required to train the network, and complex nonlinear relationships between various hyper-parameters and final model performance. We did not utilize a comprehensive hyper-parameter optimization method in this work, as hand-tuning alone provided adequate performance to demonstrate the effectiveness and flexibility of the method. Instead, we refer interested readers to relevant literature.

The simplest, and arguably least efficient, hyper-parameter optimization algorithms are grid-search and random-search. Bergstra & Bengio (2012) investigated the relationship between these two methods and suggested that random-search may be a better choice.

Bayesian methods may offer a more intelligent method for exploring the space of network hyper-parameters, leading to a lower computational cost. Bayesian methods are generally more complicated than random-search or grid-search, thus there is a trade-off between compute time spent on the optimization and human time spent implementing more advance methods. Snoek et al. (2012) provides an overview

Table 2
Model Properties[a]

Model	$t_{\rm run}$(Myr)	$\dot{M}_{\rm tot}\,(10^{-6}\,M_\odot\,{\rm yr}^{-1})$	$B(\mu G)$
W1_T2_0	0.0	0	13.5
W1_T2_0.1	0.1	41.7	13.5
W1_T2_0.2	0.2	41.7	13.5
W2_T2_0.1	0.1	4.5	13.5
W2_T2_0.2	0.2	4.5	13.5
W2_T3_0.1	0.1	4.5	5.6

Note.
[a] Model name, output time, and the total stellar mass-loss rate. All models have $L = 5$ pc, $M = 3762\,M_\odot$, initial gas temperature $T_i = 10$ K, and $N_* = 5$. The calculations are first evolved without sources for two Mach crossing times to allow initial cloud turbulence to develop.

of Bayesian parameter optimization in the context of machine learning.

Evolutionary algorithms have also been successfully applied to hyper-parameter tuning. Examples include optimization of CNN hyper-parameters with a simple population-based evolutionary algorithm (Bardenet et al. 2013), optimization of SVM hyper-parameters with particle swarm optimization (Guo et al. 2008), and optimization of ANN hyper-parameters with covariance matrix adaptation evolution strategies (CMA-ES, Loshchilov & Hutter 2016).

3. Validation

3.1. Simulation Training Set

Our study uses outputs from the simulations presented in Offner & Arce (2015) as a training set. These calculations are performed with the ORION2 adaptive mesh refinement code and follow the evolution of a 5 pc turbulent piece of a molecular cloud with five randomly distributed embedded sources. The stellar sources are represented by sink particles coupled to a sub-grid model for isotropic main-sequence stellar winds. See Offner & Arce (2015) for additional details.

As a training set, the simulations have one essential advantage over observational data: they have complete information, including density, velocity, gas temperature, and magnetic field at every point in the 3D volume. Of particular importance, ORION2 has the capability to "tag" the gas launched in winds and follow its progress across the domain (e.g., Offner & Chaban 2017; Offner & Liu 2018). The wind tracer field is a passive scalar, advected with the gas density, which tracks the amount of wind material in each cell. This field allows us to distinguish wind material from pristine cloud material and provides an exact map of the shells and bubbles created by the feedback (see Section 3.2).

For our training sets, we adopt outputs at different times from simulations with two different stellar distributions and two different initial magnetic field strengths as listed in Table 2. For training, we use only the 256^3 basegrid, thereby neglecting the information at higher "adaptive" resolution. This corresponds to a spatial resolution of $\sim$0.02 pc.

3.2. Gas-density Training Set

We train our method with two different types of data. The first training set is constructed using the simulation gas density, ρ. We define the wind fraction as $f_t = \rho_t/\rho$, where ρ_t is the density of the wind material as tracked by the tracer field. Pixels with values of $f_t > 0.02$ are considered part of the feedback (e.g., Offner & Liu 2018). These pixels define the target regions to be identified during training, testing, and validation.

Due to the high expansion velocity of the wind shells, $v \gtrsim 1\,{\rm km\,s^{-1}}$, little mixing occurs outside the boundary of the swept-up material. Consequently, the feedback signatures are roughly spherical but are modulated by local density and magnetic field variations. Thus, in 2D image slices, the target training regions resemble irregular bubbles.

3.3. Synthetic CO Emission Training Set

The second training set is constructed from a suite of synthetic molecular line observations. We post-process each simulation output using the radiative transfer code RADMC3D[8] to obtain a spectral cube for the ^{12}CO(1–0) emission line. Following Offner & Arce (2015), we adopt the Large Velocity Gradient approximation, which calculates the level populations by solving the equations for local radiative statistical equilibrium. We use the gas densities and velocities on the 256^3 basegrid as inputs, where we convert from total mass density to molecular number density using $n_{\rm H_2} = \rho/(2.8 m_p)$ and $[^{12}{\rm CO/H_2}] = 10^{-4}$ (Frerking et al. 1982). Gas with temperatures exceeding 1000 K or with $n_{\rm H_2} < 50\,{\rm cm^{-3}}$, where all of the CO is likely dissociated, are assigned a CO abundance of 0. In addition, CO molecules freeze out onto dust grains in cold gas with densities $n_{\rm H_2} > 10^4\,{\rm cm^{-3}}$, and CO molecules are dissociated by strong shocks, e.g., where the gas velocity exceeds $10\,{\rm km\,s^{-1}}$, so we also set the CO abundance to 0 in these regions. We include turbulent line broadening below the grid resolution by adding a constant microturbulence of $0.25\,{\rm km\,s^{-1}}$, which is consistent with the line width–size relation on this scale (Larson 1981). The spectral cube resolution is $\Delta v = 0.156\,{\rm km\,s^{-1}}$.

The tracer field, which tracks the stellar winds, records the amount of wind material in a given voxel (3D pixel). In order to use these data to define the positive and negative detections, we combine it with the gas velocity information and construct a spectral cube (position–position–velocity) that complements the synthetic CO emission. The approach we adopt is to map the tracer field to a density regime where $50 < n_{\rm H_2} < 10^4\,{\rm cm^{-3}}$. We then carry out the radiative post-processing described above. The emitting regions in these cubes provide a map of the location of the wind-driven shells.

To account for observational resolution, we place each cube at a distance of 250 and 500 pc and convolve it with a 46″ beam, which is the resolution of the COMPLETE ^{12}CO (1–0) survey of Perseus (e.g., Ridge et al. 2006). We also add random noise assuming $\sigma_{\rm rms} = 0.15$ K, which is comparable to the noise in the COMPLETE data.

3.4. Performance Metrics

The prediction of gas density and CO emission can be phrased as at least two ways, regression and segmentation. In the regression phrasing, the network is expected to output a floating point value corresponding with some measure of bubble material present in each pixel (e.g., molecular line emission or a continuum map, depending on the training data

[8] http://www.ita.uni-heidelberg.de/~dullemond/software/radmc-3d/

set). In the segmentation phrasing, the network is expected to classify each pixel as containing a "low" or "high" amount of bubble material.

The regression phrasing may provide more detail about perceived structures, allowing for certain kinds of analysis, such as the measurement of total bubble mass independently from the non-bubble gas along the line of sight. However, regression methods will need to handle heavy-tailed distributions of input and output values, which could lead to poor performance. The segmentation phrasing removes the potential difficulty of learning a heavy-tailed output distribution, but in doing so loses some of the detail provided by regression methods.

3.4.1. Segmentation

Segmentation masks provide less detail when compared with regressed values, but they may be more useful for identifying interesting or important regions of the input data. For example, the outputs of segmentation models can be used to augment human efforts in processing large surveys by highlighting regions of interest or filtering regions without structures of interest.

The segmentation phrasing is achieved by selecting a threshold value, which may then be used to discretize the density and CO emission data. The threshold value may be selected arbitrarily by the user or using some sort of heuristic, such as selecting a certain portion of the range of the data to constitute the negative and positive classes (e.g., the lower 1% of the range is the negative class and the upper 99% of the range is the positive class). We utilize a 1% threshold because it closely aligns with features that may be visually identified.

The loss function used in the segmentation phrasing is based on the Intersection over Union (IoU) score, also known as the Jaccard Index, and is defined as

$$\mathrm{IoU}(y, y') = \frac{\mathrm{TP}(y, y')}{\mathrm{TP}(y, y') + \mathrm{FP}(y, y')},$$

where $\mathrm{TP}(y,\ y')$ counts the number of true positives in prediction y' using the training label y and $\mathrm{FP}(y,\ y')$ counts the number of false positives.

The IoU score traditionally operates on binary inputs and is non-differentiable. In order to facilitate the training of neural networks via gradient descent, the following differentiable approximation is used:

$$\mathrm{IoU}(y, y') = \frac{\sum_{i=1}^{N} y[i] \cdot y'[i]}{\sum_{i=1}^{N} y[i] + y'[i] - \sum_{i=1}^{N} y[i] \cdot y'[i]},$$

where N is the number of pixels in y and $y[i]$ is the ith element of y. The IoU loss is simply $1 - \mathrm{IoU}(y, y')$.

Trained models are evaluated using tools from binary classification, namely confusion matrices and derived statistics, such as accuracy, F1 score, and Matthew's correlation coefficient (Powers 2011).

Given a confusion matrix with a number of true positives, $\mathtt{TP}$, a number of true negatives, $\mathtt{TN}$, a number of false positives, $\mathtt{FP}$, and a number of false negatives, $\mathtt{FN}$, accuracy is calculated as

$$\mathtt{Accuracy} = \frac{\mathtt{TP} + \mathtt{TN}}{\mathtt{TP} + \mathtt{TN} + \mathtt{FP} + \mathtt{FN}},$$

the F1 score is calculated as

$$\mathtt{F1} = \frac{2 \times \mathtt{TP}}{2 \times \mathtt{TP} + \mathtt{FP} + \mathtt{FN}},$$

and Matthew's Correlation Coefficient is calculated as

$$\mathtt{MCC} = \frac{\mathtt{TP} \times \mathtt{TN} - \mathtt{FP} \times \mathtt{FN}}{\sqrt{(\mathtt{TP} + \mathtt{FP}) \times (\mathtt{TP} + \mathtt{FN}) \times (\mathtt{TN} + \mathtt{FP}) \times (\mathtt{TN} + \mathtt{FN})}}.$$

Receiver Operating Characteristic (ROC) curves are generated by plotting the true-positive rate against the false-positive rate of a model at different prediction threshold values and can provide more information about the predictive behavior of a classifier than single number statistics.

3.4.2. Regression

In the regression phrasing, models are trained using target values that have not been thresholded and the mean squared error (MSE) is used as the loss function.

Evaluation of regression models is traditionally dominated by the analysis of residuals, with the assumption that models featuring residuals that are tightly and symmetrically distributed about zero are better. We utilize the following scoring function in order quantitatively evaluate and compare models according to these assumptions

$$f(R) = -|\langle R \rangle| - \mathtt{std}(R) - |\mathtt{skew}(R)|,$$

where f denotes the fitness function and R denotes the computed residuals. Note that the first term directly penalizes residual distributions whose mean value strays from 0, the second penalizes residual distributions that feature a non-zero standard deviation, and the final term penalizes residual distributions with non-zero skew.

Additional qualitative evaluation of the residuals can be obtained using histograms, kernel density estimates, and scatter plots, each providing a slightly different perspective on the distribution of residuals.

3.5. Case Study 1: Gas Density

In both problem phrasings the network is provided 2D slices of a 3D molecular gas-density cube as input, though the expected output differs. As noted in Sections 3.4.1 and 3.4.2, the expected output for the regression task is the fraction of gas density associated with wind-swept bubbles and the expected output for the segmentation task is a binary mask that identifies regions with "high" levels of gas density associated with wind-swept bubbles.

We cut each 3D simulated density cube along its primary axes in order to form a stack of 2D slices, which are then divided into training, validation, and testing sets. We then normalize each set of 2D slices by subtracting the mean value and dividing by the standard deviation, after which it is ready to be used in training.

3.5.1. Density Segmentation

In Figure 3, we display examples of residual U-Net predictions on several samples randomly selected from an unseen test set. In the figure, the grayscale components depict rescaled density values and and the colored components depict network predictions and errors. Qualitatively, the model

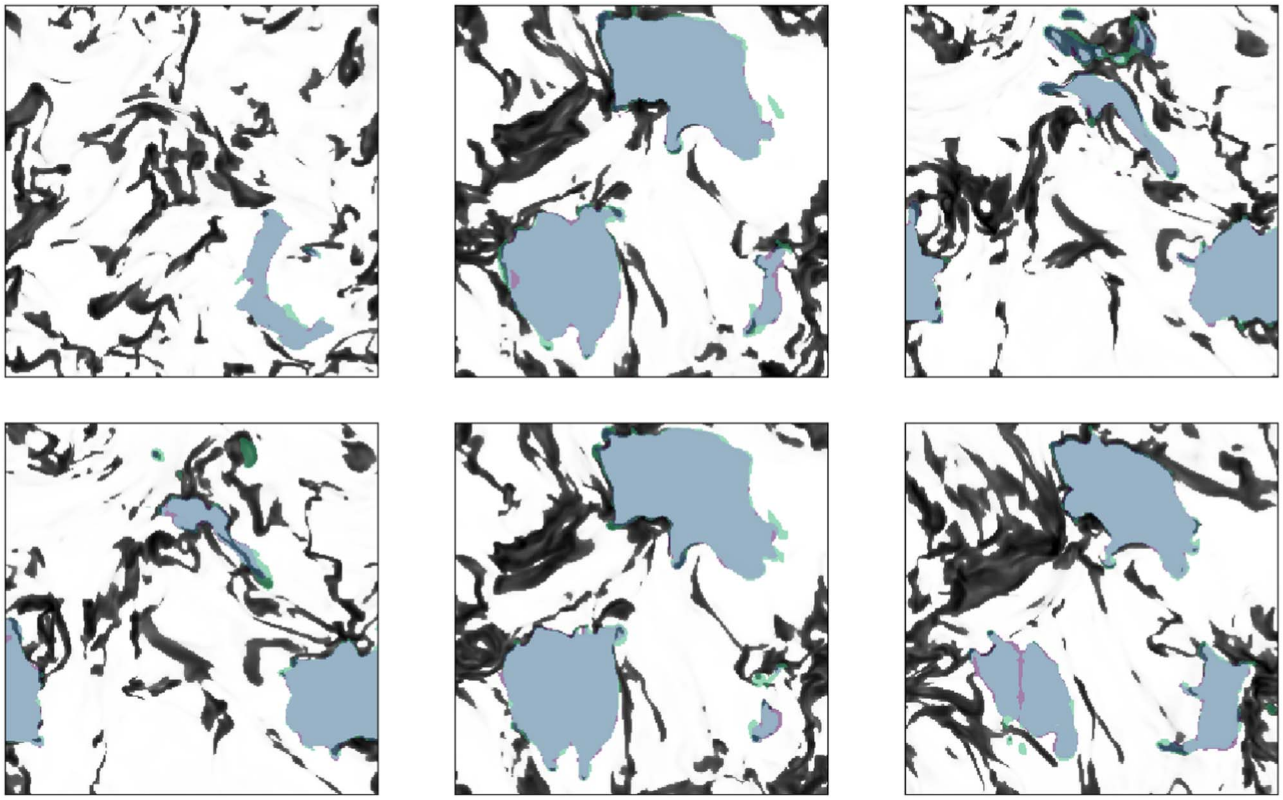

Figure 3. Density segmentation predictions from a residual U-Net on samples randomly selected from the test set. Each frame contains a single slice from a simulated density cube, which is shown in grayscale. Since each slice is taken from a position–position–position cube, the x and y axes of each frame represent spatial coordinates within the cube. The tiles presented here have a side length of 5 pc that is inherited from the simulation. In each frame, true positives are shown in blue, false positives are shown in purple, false negatives are shown in green, and true negatives are not displayed. As a pre-processing step the density data was normalized so that it is now unitless and falls approximately in the range [−0.4, 190], where lower-density regions correspond to lighter colors and higher density regions correspond to darker colors. The color scale for the density data is identical across all tiles, and a logarithmic transformation is utilized in order to improve contrast.

appears to correctly segment all major contiguous structures, though there may be some smaller structures that are missed. Additionally, note that the majority of errors are located on or near the edge of identified structures and would have little effect on whether or not a particular structure is identified. Finally, note that the upper left tile contains bubble structure that was correctly identified by CASI but may be difficult for a human to identify due to a lack of corresponding features in the density data.

In Figure 4 we present a ROC curve for the same model, which shows that the model attains a true-positive rate of 95.52% while maintaining a false-positive rate of only 1%. Supporting this, we summarize the distributions of several binary classification statistics in Table 3, where the classification statistics are computed across a test set of 154 samples. In particular, Table 3 clearly highlights the low error rate obtained by our model, where the maximum fraction of false positives is 1.52% and the maximum fraction of false negatives is 3.9%.

In order to better grasp the effect of random initialization on final model performance we trained 60 instances of the model using the same data and parameter settings, recorded the Receiver Operating Characteristic Area Under Curve (ROC AUC) statistic for each model, and then constructed confidence intervals for the mean of the ROC AUC distribution. The results of this experiment are presented in the first column of Table 4, which shows that CASI is able to consistently obtain ROC AUC scores close to the maximum value of 1.0.

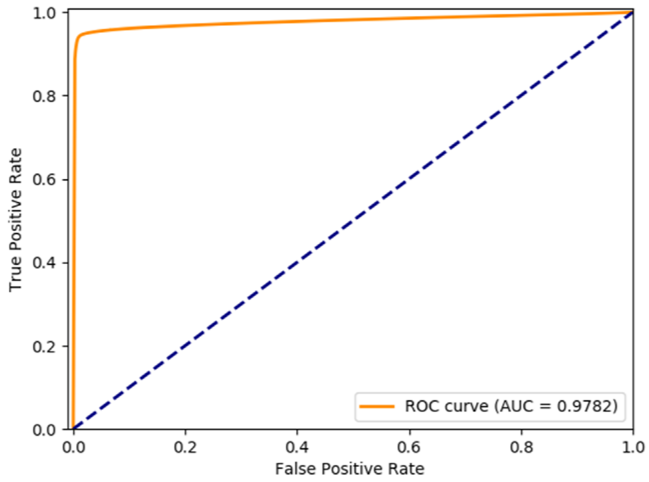

Figure 4. Example ROC curve for a residual U-Net trained on the density segmentation task. The dashed blue line represents $y = x$, which corresponds with the expected performance of a random binary classifier. A true-positive rate of 95.52% is obtainable with a false-positive rate of 1%, suggesting that this method may perform well as a content filter.

3.5.2. Density Regression

Applying a residual U-Net to the density regression task leads to a tight distribution of residuals that is not strongly correlated with the size of the input value, indicating that the

Table 3
Confusion Matrix Statistics for a Residual U-Net Trained on the Density Segmentation Task, Computed over a Test Set Containing 154 Samples

	True Positive	True Negative	False Positive	False Negative	Accuracy	F1 Score	Matthews Corr.
Mean	10.82	87.76	0.47	0.94	98.59	91.71	91.07
Std.	7.04	7.75	0.34	0.82	1.08	10.83	10.42
Min.	0.00	72.49	<0.01	0.00	94.58	0.00	0.00
25%	5.43	80.00	0.02	0.36	97.98	90.99	90.18
50%	9.77	89.17	0.37	0.72	98.85	94.82	93.77
75%	17.09	93.13	0.69	1.22	99.37	96.11	95.45
Max.	25.44	99.97	1.52	3.90	99.98	98.29	98.11

Note. True positives, true negatives, false positives, and false negatives are presented as a fraction of image pixels, thus assuming values between 0 and 100. The other three statistics, accuracy, F1 score, and Matthew's correlation coefficient, also assume values between 0 and 100, with higher values indicating better model performance. The minimum values observed in the F1 score and Matthew's correlation coefficient are caused by a few samples with no positively labeled pixels.

Table 4
Segmentation Task Performance Statistics Collected by Training and Evaluating 60 Randomly Initialized Networks on the Same Training, Validation, and Testing Splits

Performance Stat.	Distribution Stat.	Density Segmentation	^{12}CO Segmentation
ROC AUC	Mean	0.9768	0.909
	Std. Error	0.0016	0.0018
	85% Conf. Int.	(0.9745, 0.9792)	(0.9063, 0.9117)

Note. The first column indicates a statistic that was computed using the predictions of each trained network, while the second column indicates a statistic that was applied to the results of the column one statistic. The receiver operating characteristic area under curve (ROC AUC) statistic is calculated by computing the integral of the ROC curve, as seen in Figures 8 and 4.

model has captured much of the relationship between the input and output. Figure 5 displays a 2D histogram that shows the relationship between residuals and input values.

Figure 6 displays the example prediction residuals for several samples from the test set. Note that the larger residuals tend to be clustered together near the edges of structures, similar to what was observed in the density segmentation setting.

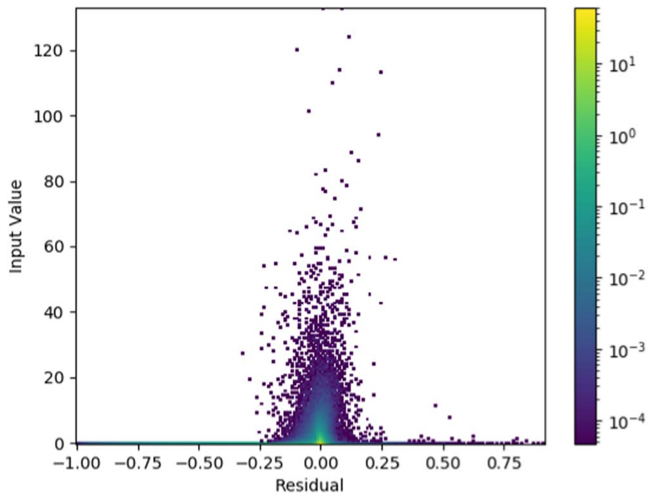

Figure 5. 2D histogram investigating the scaling of residuals with respect to the input value for a residual U-Net trained on the density regression task. The color scale is logarithmic in order to increase contrast and represents the density of points associated with each residual value-input value pair. Recall that the input values here are density values that have been scaled to have zero mean and unit standard deviation, thus the y-axis of this plot is unitless. Due to the heavy-tailed nature of the input values, this rescaling results in the data that fall approximately within the range [−0.4, 190].

3.6. Case Study 2: Synthetic Molecular Emission

The ^{12}CO data feature position–position–velocity coordinates, rather than the position–position–position coordinates used for the density data. In both the segmentation and regression tasks the input data are inspected along the velocity axis such that the network is provided with position–position slices, those slices are divided into training, validation, and testing splits, then each data split is normalized by subtracting the mean and dividing by the standard deviation.

3.6.1. CO Segmentation

The U-Net attains slightly lower performance in the ^{12}CO tasks, when compared with corresponding density tasks, even though the training set is more than a factor of two larger. This indicates that the relationship between the ^{12}CO observations and the constructed tracer data may be more complex than the relationship between the density data and corresponding tracer.

Figure 7 shows example predictions, which feature similar characteristics to the density segmentation predictions. The major structures are all correctly identified, with some smaller structures being missed and errors clustered along the edges of larger structures.

The ROC curve, provided in Figure 8, features a sharp curve that is pushed up toward the upper left corner of the plot, where the model reaches a true-positive rate of 91.45% while maintaining a false-positive rate of 1%. This accuracy is slightly lower than that achieved by the density segmentation task; however, the results still constitute excellent performance.

An investigation of final model performance variation due to random initialization is provided in column two of Table 4, which shows that CASI is robust to random initialization on the CO segmentation task.

3.6.2. CO Regression

For the ^{12}CO regression task an element-wise logarithm operation is applied prior to the normalization operations in order to further reduce the dynamic range of the data. Specifically we modify the data using

$$x' = \ln(1 + x - \min(x)),$$

where x is one of the training, validation, or testing sets, and $\min(x)$ is the element-wise minimum. Subtracting by the

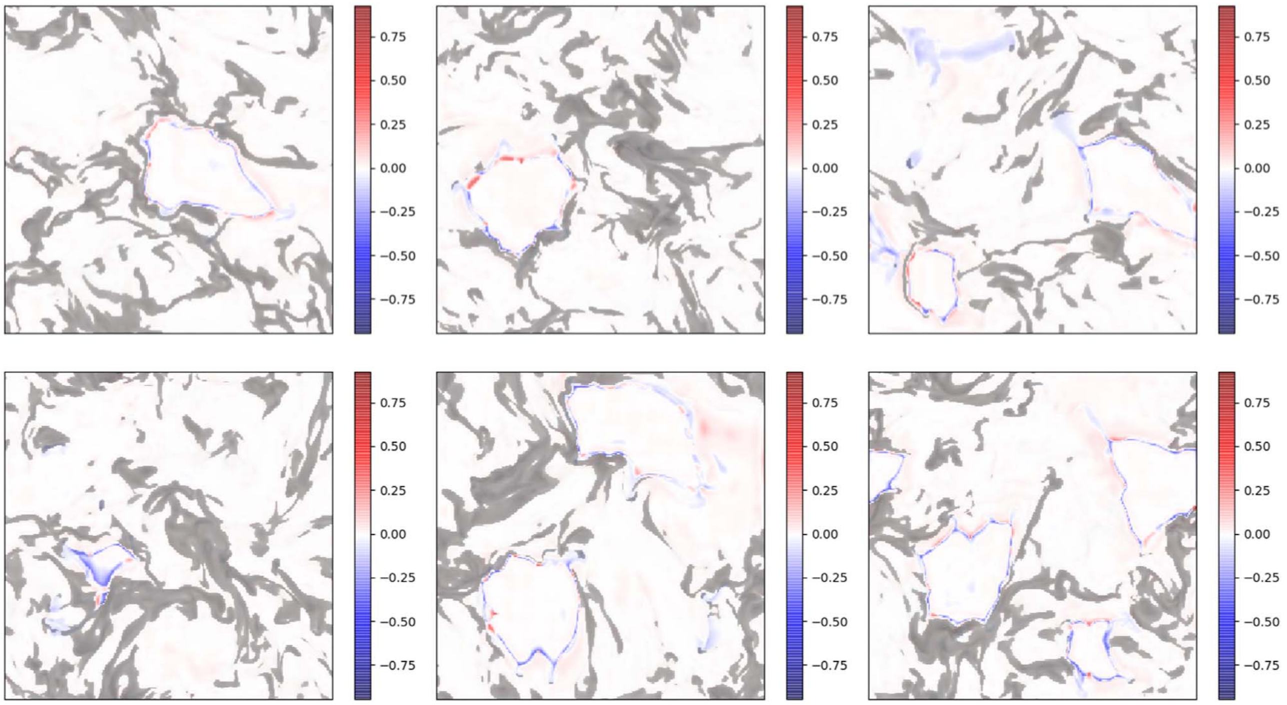

Figure 6. Example residuals from a residual U-Net trained on the density regression task using the mean squared error loss function. Positive residuals, shown in shades of red, correspond to overestimation, while negative residuals, shown in shades of blue, correspond to underestimation. As with Figure 3, the side length of each tile is 5 pc and the grayscale components represent rescaled density values.

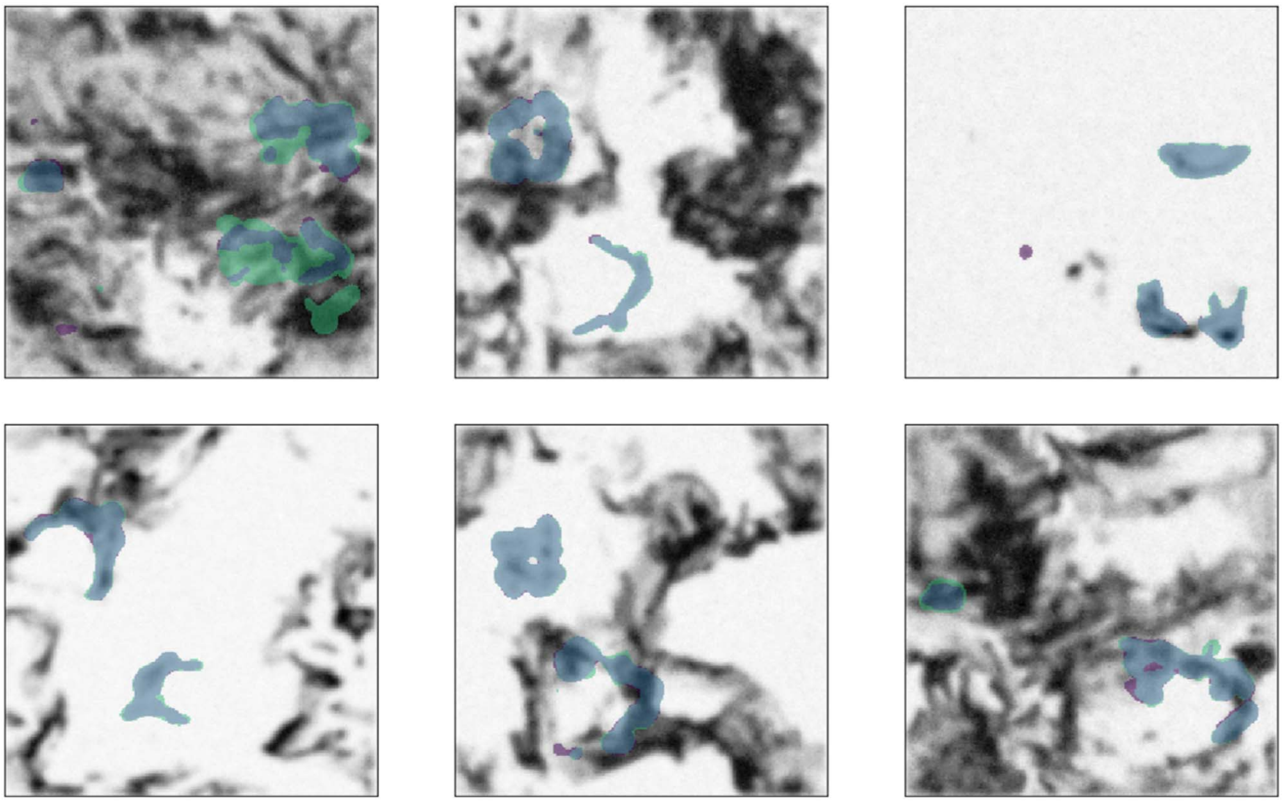

Figure 7. ^{12}CO segmentation predictions from a residual U-Net on samples randomly selected from the test set. As with Figure 3, the side length of each tile is 5 pc, the grayscale components represent rescaled density values, true positives are shown in blue, false positives are shown in purple, false negatives are shown in green, and true negatives are not displayed.

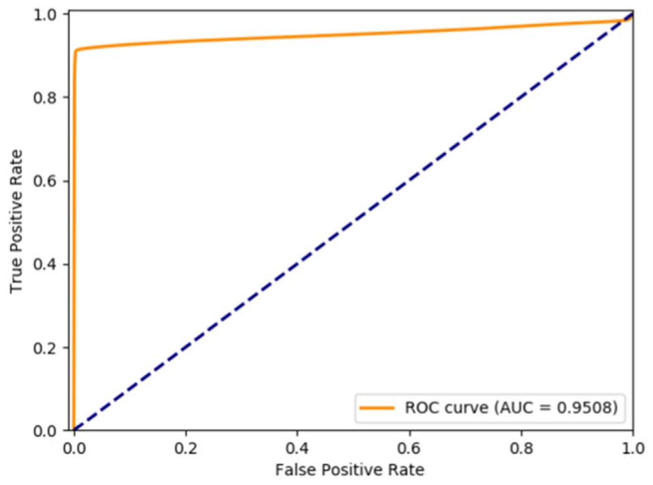

Figure 8. Example ROC curve for a residual U-Net trained on the ^{12}CO segmentation task. The dashed blue line represents $y = x$, which corresponds with the expected performance of a random binary classifier. A true-positive rate of 91.45% is obtainable with a false-positive rate of 1%, supporting the proposal that this method may perform well as a content filter.

minimum value and adding 1 ensures that there are no invalid output values and that all values fall within the compressive regime of the logarithm.

We investigated the effect of random initialization on this task using the same experiment structure seen for the other conditions. These results are reported in column two of Table 5.

4. Conclusions

Our results indicate that methods from deep learning, namely the U-Net and its variants, are a flexible and effective tool for learning relationships in simulated density and ^{12}CO data. Moreover, our algorithm is completely general and could be trained to identify other astronomical signatures, such as protostellar outflows, filaments and dense cores, given appropriate training sets.

Our models learn well under several different conditions and generalize to unseen data from the same distribution with a minimal performance impact. Additionally, CASI features a low false-positive rate and a clustering of errors that make it well-suited to assisting astronomers by filtering large-scale survey data that is being inspected by humans.

We also note that CASI is relatively quick to train, especially on smaller data sets, taking approximately 2 s per epoch for the density tasks (approximately 8 minutes for 200 epochs) and 7 s per epoch for the CO tasks (approximately 25 minutes for 200 epochs). After training, CASI can process more than 100 samples per second, allowing for rapid application to new data. With fast training times and even faster post-training predictions, CASI may be rapidly applied to new data sets with minimal overhead.[9]

Despite the generally positive results presented, there are several important research directions surrounding the application of deep-learning techniques to facilitate the analysis of astronomical image data that have not been addressed.

First, all results presented in this work focus on learning from simulated data, but in order to assist in the processing of survey data these models must operate on true observations that may greatly differ from the simulated data that they were trained upon. For example, we adopt a simple CO abundance model and do not take into account chemistry. Boyden et al. (2018) showed that self-consistently computing abundances and temperatures can produce statistically different emission maps. However, Xu & Offner (2017) demonstrated that synthetic dust emission maps of the simulations also utilized here can be used to successfully train a random forest algorithm to correctly identify observed bubbles. This makes us confident that our CO emission maps have, at minimum, similar underlying morphologies to observational data. We extend our study to observational data in D. Xu et al. (2019, in preparation) and demonstrate that training sets based on synthetic CO emission can indeed be applied to observed CO data. Beyond assessing and improving the simulations that are used to generate training data, a comprehensive investigation of regularization and data augmentation techniques may lead to models that are better able to bridge the gap between simulation and observations.

Second, our methods leverage the high-fidelity information and annotations provided by the simulations to learn relationships in a supervised setting. However, there exist considerable amounts of unlabeled survey and observational data that may be utilized in semi-supervised or unsupervised approaches. Semi-supervised and unsupervised approaches could reduce or

Table 5
Regression Task Performance Statistics Collected by Training and Evaluating 60 Randomly Initialized Networks on the Same Training, Validation, and Testing Splits

Performance Stat.	Distribution Stat.	Density Regression	^{12}CO Regression
Mean	Mean	−0.0527	−0.019
	Std. Error	0.0009	0.0008
	85% Conf. Int.	(−0.054, −0.0513)	(−0.0201, −0.0179)
Std. Dev.	Mean	0.2012	0.3483
	Std. Error	0.0031	0.0011
	85% Conf. Int.	(0.1968, 0.2058)	(0.3466, 0.3499)
Skew	Mean	−3.8254	−13.17
	Std. Error	0.0211	0.0854
	85% Conf. Int.	(−3.8562, −3.7946)	(−13.2945, −13.0454)
Score	Mean	−4.0793	−13.5372
	Std. Error	0.01778	0.0854
	85% Conf. Int.	(−4.1053, −4.0534)	(−13.6618, −13.4126)

Note. The first column indicates a statistic that was computed over the residuals of each trained network, while the second column indicates a statistic that was applied to the results of the column one statistic. The fourth element of the first column, score, refers to the regression score defined in Section 3.4.2. CASI is able to reliably obtain a residual distribution with a mean near zero and a small standard deviation, indicating a tight residual distribution that is clustered about the origin. For both tasks the mean and skew components feature negative values, indicating that CASI tends to underestimate values more often than it overestimates values. Additionally, the negative skew value indicates that the tail of the residual distribution is longer in the negative direction, thus the largest errors tend to be underpredictions. However, the fact that the residual distribution is tightly grouped about the origin indicates that the relatively large skew value is not concerning and due in part to the characteristics of the input data.

[9] The computer that was used to collect timing information was outfitted with an Intel i7-6700K CPU and a Nvidia GTX 1080Ti GPU.

remove the overhead involved with hand labeling and curating large data sets, while still drawing insights from said data.

Finally, only 2D models were investigated in this work, which ignore the 3D structure present in density and ^{12}CO cubes. We have found that 2D models seem to be sufficient for solving certain problems in this domain, certainly the benchmarks investigated here are well solved by 2D models, but some problems may require models with greater knowledge of 3D structure.

3D convolutional models have begun to find application in human action recognition (Ji et al. 2013), object detection in 3D point clouds (Maturana & Scherer 2015), medical imaging (Khosla et al. 2018), and other domains (Tran et al. 2015). These 3D models may also be well-suited to identifying structures in stellar feedback, and we begin to explore such models, as well as their application to observational data, in upcoming work (D. Xu et al. 2019, in preparation).

C.V.O., D.X., S.S.R.O., and R.A.G. were supported by NSF grant AST-1812747. S.S.R.O. also acknowledges support from NSF Career grant AST-1650486.

Software: Astropy, Keras, Matplotlib, Numpy, Pandas, Python, Scikit-learn, Tensorflow.

Appendix
Relevant Neural Network Operations

A.1. Batch Normalization

Batch normalization allows a network to renormalize data at arbitrary points during the forward pass using moving mean and standard deviation statistics calculated over training batches. Following the description of batch normalization provided by Ioffe & Szegedy (2015), if $\mathcal{B} = \{x_1, x_2, ..., x_n\}$ represents a batch of training samples then the mean and variance of the batch are calculated as

$$\mu_{\mathcal{B}} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \sigma_{\mathcal{B}}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2.$$

The data are then normalized using the batch mean and variance

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}},$$

where ϵ is an arbitrary constant used for numerical stability. Finally, the output of the batch normalization is calculated using

$$y_i = \gamma\hat{x}_i + \beta,$$

where γ and β are learned parameters that allow the network to reverse or modify the batch normalization procedure when beneficial.

Batch normalization is applied slightly differently during training and inference, though this is handled internally by most deep-learning frameworks. Interested readers should refer to Ioffe & Szegedy (2015).

A.2. Convolution

A 2D convolution in this context involves an image with dimensions (image height, image width, image channels), or (H_i, W_i, C_i), and a set of filters with dimensions (filter count, image channels, filter height, filter width), or (F, C_i, H_f, W_f). The convolution is computed by sliding each filter over the spatial dimensions of the image. At each location an element-wise product between the filter and the corresponding image pixels is computed, the results of which are summed and become a single pixel in the output of the convolution. The sliding behavior of the convolution is controlled by horizontal and vertical stride parameters, s_h and s_v, which indicate how far the filter should move in each direction after each calculation. The output of the convolution described above would have the dimensions

$$\left(\frac{H_i - H_f + 1}{s_v}, \frac{W_i - W_f + 1}{s_h}, F\right).$$

It is common to pad the image with zeros in order to force the dimensions of the output into desired values. Notably, if the spatial dimensions of the filter are odd and the image is padded by $\lfloor H_f/2 \rfloor$ on the top/bottom and $\lfloor W_f/2 \rfloor$ on the left/right then the the output of the convolution will have the dimensions (H_i, W_i, F). This is referred to as the "same" padding scheme, since the output has identical spatial dimensions to the input. In practice, this operation is usually applied to a batch of several images in parallel.

A.3. Max Pooling

The max pooling operation is designed to reduce the spatial dimensions of an image while keeping the most important data intact. It does this by inspecting small sub-regions of the image, commonly 2×2 windows, and filtering out the maximum value

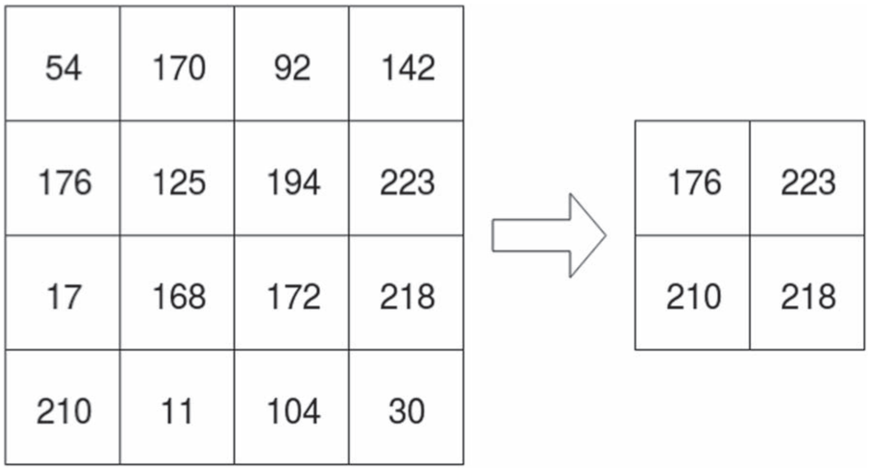

Figure 9. Max pooling with a 2×2 window, used to map a 4×4 input to a 2×2 output.

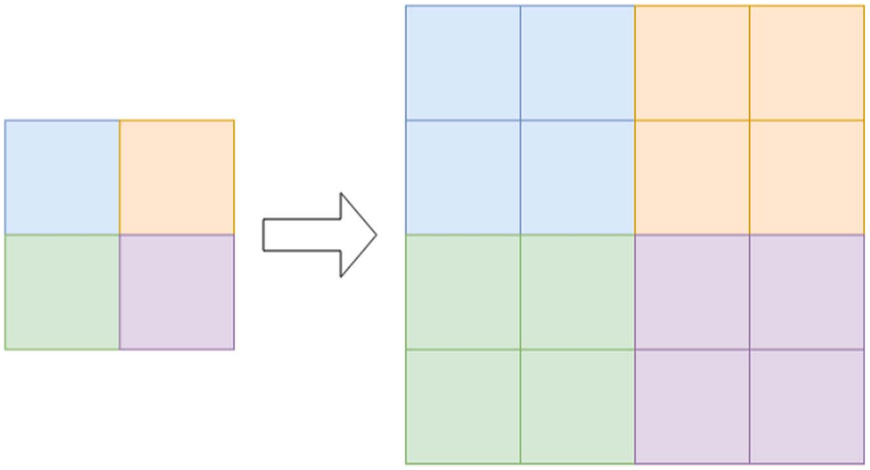

Figure 10. Nearest-neighbor interpolation with a 2 × 2 window, used to map a 2 × 2 input to a 4 × 4 output.

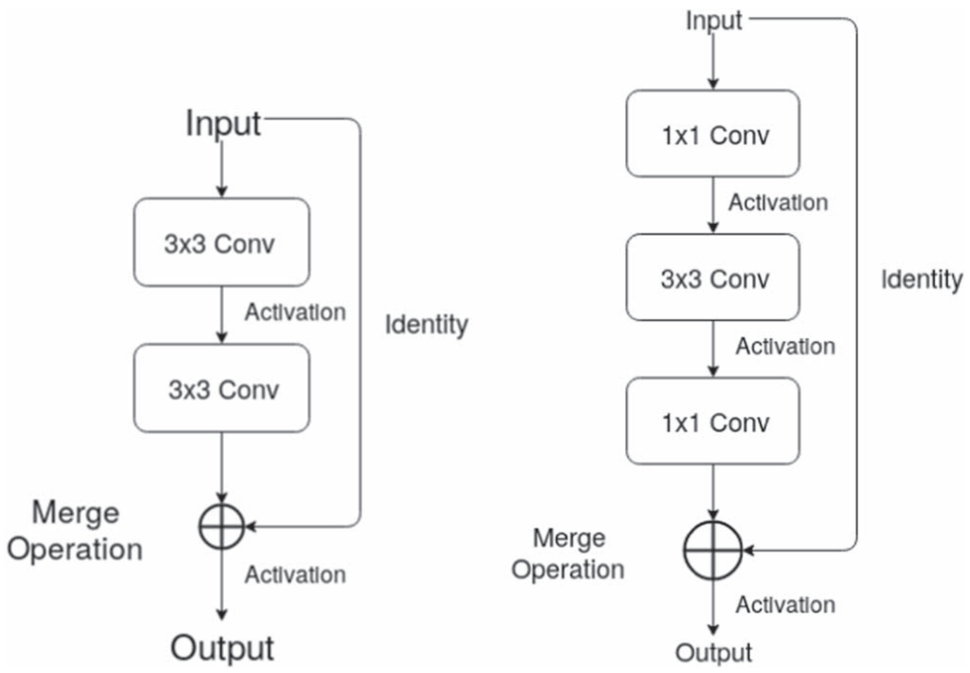

Figure 11. Left: a basic residual block using 3 × 3 filters; the number of filters used in each convolution is a free parameter that must be selected. Common activation functions include ReLU, sigmoid, and tanh. Common merge operations include concatenation, element-wise addition, and element-wise maximum (maxout, Goodfellow et al. 2013). If addition is used as the merging operation, then a projection skip-connection, commonly implemented using a 1 × 1 convolution, may be required in place of the identity skip-connection in order to obtain the correct dimensions for the merge operation. Right: a bottleneck residual block, which uses 1 × 1 convolutions in order to reduce the number of parameters required, relative to the basic residual block. If the input volume has n channels it is common to use $n/2$ or $n/4$ filters in the first two convolutions, followed by n channels in the final convolution, which compresses the data before the larger convolution is applied, resulting in a reduced number of parameters.

in that sub-region. The max pooling operation, like the convolution described above, has stride parameters that adjust the spatial relationship between the sub-regions. It is common to have strides that are equal to the size of the sub-regions, resulting in disjointed sub-regions that fully cover the input image. See Figure 9 for an example application of the max pooling operation.

Note that max pooling is applied to each channel independently, and thus the result of applying a max pooling operation with a 2 × 2 window and a stride of 2 to a (64, 64, 3) image would be a (32, 32, 3) image.

A.4. Nearest-neighbor Interpolation

Nearest-neighbor interpolation is an extremely simple upsampling operation that increases the spatial dimensions of an image by an integer factor, n, by expanding each pixel into an $n \times n$ block with identical values. This may be used to reverse the effects of a max pooling operation, though some detail is lost. See Figure 10 for an example application of nearest-neighbor interpolation.

A.5. Activation: Exponential Linear Units (ELUs)

Introduced by Clevert et al. (2015), the exponential linear activation function is defined as

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x \geqslant 0 \\ \alpha(e^x - 1) & \text{if } x < 0, \end{cases}$$

where α controls the negative saturation value of the function. Use of ELU has been shown empirically to allow faster and more robust training of deep neural networks when compared to rectified linear units (ReLU) and other common activation functions.

SELU, defined as

$$\mathrm{SELU}(x) = \lambda \begin{cases} x & \text{if } x \geqslant 0 \\ \alpha(e^x - 1) & \text{if } x < 0, \end{cases} \quad \lambda > 1,$$

exhibits similar properties to ELUs but with the added benefit of having a normalizing effect on network activations, similar to batch normalization. See Klambauer et al. (2017) for more details.

A.6. Residual Connections

Sometimes referred to as skip-connections, this architecture component can improve performance (He et al. 2016), reduce training instability in deeper networks (He et al. 2016), and encourage iterative inference (Jastrzebski et al. 2017). Figure 11 depicts two common variants of network blocks that contain residual connections.

ORCID iDs

Stella S. R. Offner https://orcid.org/0000-0003-1252-9916
Robert A. Gutermuth https://orcid.org/0000-0002-6447-899X

References

Alves, J., Lombardi, M., & Lada, C. J. 2007, A&A, 462, L17
Arce, H. G., Borkin, M. A., Goodman, A. A., Pineda, J. E., & Beaumont, C. N. 2011, ApJ, 742, 105
Arce, H. G., Borkin, M. A., Goodman, A. A., Pineda, J. E., & Halle, M. W. 2010, ApJ, 715, 1170
Bardenet, R., Brendel, M., Kégl, B., & Sebag, M. 2013, Collaborative Hyperparameter Tuning Proc. ICML 30 (New York: ACM), 199
Beaumont, C. N., Goodman, A. A., Kendrew, S., Williams, J. P., & Simpson, R. 2014, ApJS, 214, 3
Beaumont, C. N., Williams, J. P., & Goodman, A. A. 2011, ApJ, 741, 14
Bennett, K. P., & Campbell, C. 2000, Acm Sigkdd Explorations Newsletter, 2, 1
Bergstra, J., & Bengio, Y. 2012, Journal of Machine Learning Research, 13, 281
Boyden, R. D., Offner, S. S. R., Koch, E. W., & Rosolowsky, E. W. 2018, ApJ, 860, 157
Churchwell, E., Povich, M. S., Allen, D., et al. 2006, ApJ, 649, 759
Clevert, D.-A., Unterthiner, T., & Hochreiter, S. 2015, arXiv:1511.07289
Cunningham, A. J., Frank, A., Quillen, A. C., & Blackman, E. G. 2006, ApJ, 653, 416
Daigle, A., Joncas, G., & Parizeau, M. 2007, ApJ, 661, 285
Daigle, A., Joncas, G., Parizeau, M., & Miville-Deschênes, M.-A. 2003, PASP, 115, 662
Devalla, S. K., Renukanand, P. K., Sreedhar, B. K., et al. 2018, Biomedical Optics Express, 9, 3244
Diaz, J., Bekki, K., Forbes, D. A., et al. 2019, MNRAS, 486, 4845
Dieleman, S., Willett, K. W., & Dambre, J. 2015, MNRAS, 450, 1441
Duchi, J., Hazan, E., & Singer, Y. 2011, Journal of Machine Learning Research, 12, 2121
Dumoulin, V., & Visin, F. 2016, arXiv:1603.07285
Federrath, C. 2015, MNRAS, 450, 4035
Frerking, M. A., Langer, W. D., & Wilson, R. W. 1982, ApJ, 262, 590
Fukushima, K., & Miyake, S. 1982, Competition and Cooperation in Neural Nets (Berlin: Springer), 267
Giri, S. K., Mellema, G., Dixon, K. L., & Iliev, I. T. 2017, MNRAS, 473, 2949
Glorot, X., & Bengio, Y. 2010, PMLR, 9, 249
Goh, G. 2017, Distill, http://distill.pub/2017/momentum
Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. 2013, arXiv:1302.4389
Goodman, A. A., Rosolowsky, E. W., Borkin, M. A., et al. 2009, Natur, 457, 63
Gu, J., Wang, Z., Kuen, J., et al. 2018, Pattern Recognition, 77, 354
Guo, X., Yang, J., Wu, C., Wang, C., & Liang, Y. 2008, Neurocomputing, 71, 3211
He, K., Zhang, X., Ren, S., & Sun, J. 2016, in Proc. IEEE CVPR (Piscataway, NJ: IEEE), 770
Hoffer, E., Hubara, I., & Soudry, D. 2017, arXiv:1705.08741
Huang, G., Li, Y., Pleiss, G., et al. 2017, arXiv:1704.00109
Hubel, D. H., & Wiesel, T. N. 1968, The Journal of Physiology, 195, 215
Ioffe, S., & Szegedy, C. 2015, arXiv:1502.03167
Jastrzebski, S., Arpit, D., Ballas, N., et al. 2017, arXiv:1710.04773
Ji, S., Xu, W., Yang, M., & Yu, K. 2013, ITPAM, 35, 221
Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. 2016, arXiv:1609.04836
Khosla, M., Jamison, K., Kuceyeski, A., & Sabuncu, M. 2018, arXiv:1806.04209
Kingma, D., & Ba, J. 2014, arXiv:1412.6980
Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. 2017, arXiv:1706.02515
Könyves, V., Kiss, C., Moór, A., Kiss, Z. T., & Tóth, L. V. 2007, A&A, 463, 1227
Krumholz, M. R., Klein, R. I., & McKee, C. F. 2007, ApJ, 656, 959
Lan, T., Li, Y., Murugi, J. K., Ding, Y., & Qin, Z. 2018, arXiv:1805.11856
Lanusse, F., Ma, Q., Li, N., et al. 2018, MNRAS, 473, 3895
Larson, R. B. 1981, MNRAS, 194, 809
LeCun, Y., Bengio, Y., & Hinton, G. 2015, Natur, 521, 436
LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. 1998, IEEEP, 86, 2278
Lee, E. J., Chang, P., & Murray, N. 2015, ApJ, 800, 49
Li, H., Li, D., Qian, L., et al. 2015, ApJS, 219, 20
Lieu, M., Conversi, L., Altieri, B., & Carry, B. 2019, MNRAS, 485, 5831
Lintott, C. J., Schawinski, K., Slosar, A., et al. 2008, MNRAS, 389, 1179
Loshchilov, I., & Hutter, F. 2016, arXiv:1604.07269
Masters, D., & Luschi, C. 2018, arXiv:1804.07612
Maturana, D., & Scherer, S. 2015, in IEEE/RSJ Int. Conf. Intelligent Robots and Systems 922 (Piscataway, NJ: IEEE)
Men'shchikov, A. 2013, A&A, 560, A63
Narayanan, G., Snell, R., & Bemis, A. 2012, MNRAS, 425, 2641
Nasser, Y., Eckersley, P., Bayle, Y., et al. 2017, Measuring the Progress of AI Research, https://www.eff.org/ai/metrics
Offner, S. S. R., & Arce, H. G. 2015, ApJ, 811, 146
Offner, S. S. R., & Chaban, J. 2017, ApJ, 847, 104
Offner, S. S. R., Clark, P. C., Hennebelle, P., et al. 2014, in Protostars and Planets VI, ed. H. Beuther et al. (Tucson, AZ: Univ. Arizona Press), 53
Offner, S. S. R., Dunham, M. M., Lee, K. I., Arce, H. G., & Fielding, D. B. 2016, ApJL, 827, L11
Offner, S. S. R., Lee, E. J., Goodman, A. A., & Arce, H. 2011, ApJ, 743, 91
Offner, S. S. R., & Liu, Y. 2018, NatAs, 2, 896
Pan, S. J., & Yang, Q. 2010, IEEE Transactions on Knowledge and Data Engineering, 22, 1345
Powers, D. M. 2011, Journal of Machine Learning Technologies, 2, 37
Primack, J., Dekel, A., Koo, D., et al. 2018, ApJ, 858, 114
Ridge, N. A., Di Francesco, J., Kirk, H., et al. 2006, AJ, 131, 2921
Ronneberger, O., Fischer, P., & Brox, T. 2015, arXiv:1505.04597
Ruder, S. 2016, arXiv:1609.04747
Simpson, R. J., Povich, M. S., Kendrew, S., et al. 2012, MNRAS, 424, 2442
Smith, S. L., Kindermans, P.-J., & Le, Q. V. 2017, arXiv:1711.00489
Snoek, J., Larochelle, H., & Adams, R. P. 2012, arXiv:1206.2944
Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M. 2015, arXiv:1412.0767
Van Oort, C. M. 2019, CASI-2D, Zenodo, doi:10.5281/zenodo.2695533
Wang, P., Li, Z.-Y., Abel, T., & Nakamura, F. 2010, ApJ, 709, 27
Williams, J. P., de Geus, E. J., & Blitz, L. 1994, ApJ, 428, 693
Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., & Recht, B. 2017, arXiv:1705.08292
Xu, D., & Offner, S. S. R. 2017, ApJ, 851, 149
Zhang, C., & Zhang, Z. 2010, A Survey of Recent Advances in Face Detection, Tech. Rep. MSR-TR-2010-66, Microsoft
Zhang, Z., Liu, Q., & Wang, Y. 2018, IGRSL, 15, 749
Zhu, W., Huang, Y., Tang, H., et al. 2019, MedPh, 46, 576