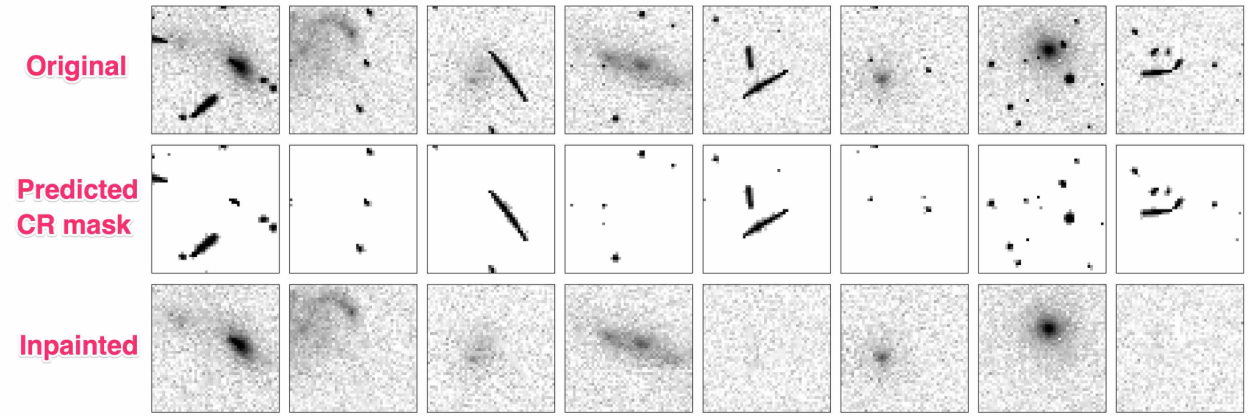
Project Description
deepCR: Cosmic Ray Rejection with Deep Learning



deepCR is an advanced solution designed to address the challenging issue of cosmic ray removal from astronomical images using Convolutional Neural Networks (CNNs). Cosmic rays, high-energy particles, often create unwanted artifacts in astronomical data, which can lead to inaccuracies in scientific analysis and hinder the extraction of reliable information.

Technology Stack:

Python: The core programming language for implementing deep learning models and image processing.
PyTorch: A popular deep learning framework used to build and train CNNs.
Astropy: A library for astronomical data analysis that provides essential tools for working with astronomical data.
NumPy and SciPy: Fundamental libraries for numerical and scientific computing, used for data manipulation and analysis.
Matplotlib: A data visualization library used for creating plots and figures.
Scikit-image: An image processing library that enhances image manipulation capabilities.
Jupyter: An interactive computing environment for creating and sharing documents that contain live code, equations, visualizations, and narrative text.
Astroscrappy: A Python package for cosmic-ray detection in single images.

Machine Learning Workflow

Training Phase: deepCR is trained on a substantial dataset of astronomical images containing cosmic rays. Each image is meticulously labeled, marking the locations of cosmic rays. During training, the CNN learns to recognize the unique signatures and features of cosmic rays.

Cosmic Ray Detection: After training, the CNN can be applied to new, unprocessed astronomical images. It scans the image pixel by pixel, identifying regions that match the patterns learned during training. These identified regions are potential cosmic ray locations.

Artifact Removal: Once the cosmic ray locations are detected, deepCR employs sophisticated interpolation techniques to remove the cosmic ray artifacts from the image. This interpolation process effectively replaces the affected pixels with plausible values, restoring the original appearance of the astronomical scene.

By automating the cosmic ray removal process, deepCR significantly reduces the manual effort required to clean astronomical data, especially in large-scale studies. Moreover, it ensures a more accurate and consistent treatment of cosmic rays, which is crucial for reliable scientific analysis.

Practical Applications

Astronomical Research: deepCR is invaluable in various astronomical studies, such as galaxy morphology analysis, exoplanet detection, and stellar population studies. It ensures the removal of cosmic ray artifacts that could obscure crucial celestial features.

Observational Surveys: Astronomical surveys often generate massive datasets. deepCR can efficiently process such datasets, saving valuable time and resources by automating cosmic ray removal.

Archival Data: For researchers working with archival astronomical data, deepCR can be a useful tool to preprocess and clean older images contaminated by cosmic rays.

Additional Information

deepCR supports both CPU and GPU, making it versatile and efficient for various computing environments.
The API of deepCR includes functionality for both applying pre-trained models and training new models on custom datasets.
deepCR has demonstrated robustness and speed compared to existing cosmic ray rejection methods, making it a powerful tool for astronomers.

About deepCR

deepCR: Unveiling the Universe with Clearer Data

DeepCR is a pioneering solution in the field of astrophysics, leveraging the power of deep learning to enhance the quality and reliability of astronomical data. With a mission to unveil the mysteries of the universe, deepCR focuses on cosmic ray removal, a crucial step in ensuring the accuracy of scientific analyses in astronomy.

Problem Statement

Astronomical images are susceptible to cosmic rays, high-energy particles that create unwanted artifacts in the data. These artifacts compromise the integrity of observations and hinder the

extraction of meaningful insights. Manual removal of cosmic rays is a labor-intensive and error-prone process, especially when dealing with extensive datasets.

Solution

deepCR harnesses the capabilities of Convolutional Neural Networks (CNNs) to automate the detection and removal of cosmic rays. It follows a systematic workflow:

Training Phase: deepCR is trained on a diverse dataset of astronomical images containing cosmic rays. Through meticulous labeling, the CNN learns to identify the distinctive characteristics of these cosmic intruders.

Cosmic Ray Detection: Once trained, deepCR applies its knowledge to new images, scanning them pixel by pixel. It identifies regions matching the learned cosmic ray patterns, marking them as potential cosmic ray locations.

Artifact Removal: deepCR employs advanced interpolation techniques to eliminate cosmic ray artifacts. It replaces affected pixels with plausible values, restoring the pristine appearance of the astronomical scene.

Impact

deepCR's automated cosmic ray removal brings numerous benefits:

Enhanced Data Quality: Astronomical data becomes cleaner and more reliable, enabling precise scientific analysis.

Time and Resource Savings: Researchers can process extensive datasets efficiently, reducing manual effort.

Archival Data Revival: Valuable older images can be reclaimed from cosmic ray contamination.

Future Prospects

deepCR is a growing project, and its community-driven "model zoo" ensures compatibility with various instrument configurations. Researchers and astronomers worldwide can benefit from this powerful tool to explore the universe with unprecedented clarity.

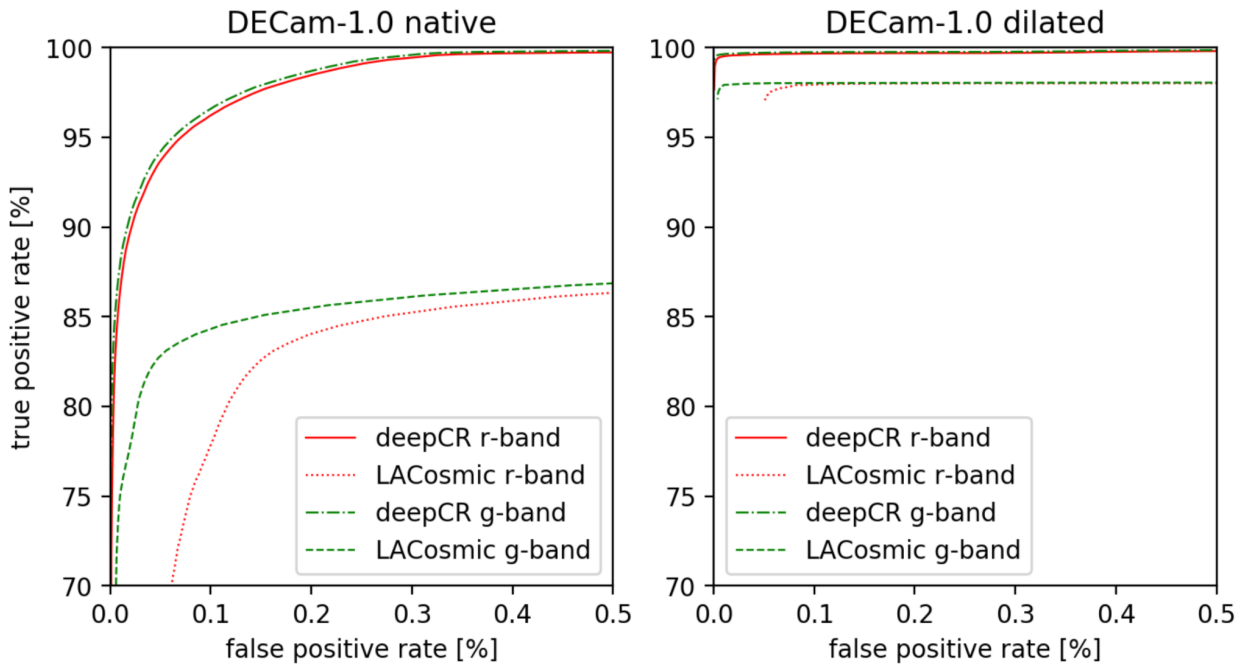
Join us on our cosmic journey with deepCR and uncover the universe's hidden wonders.

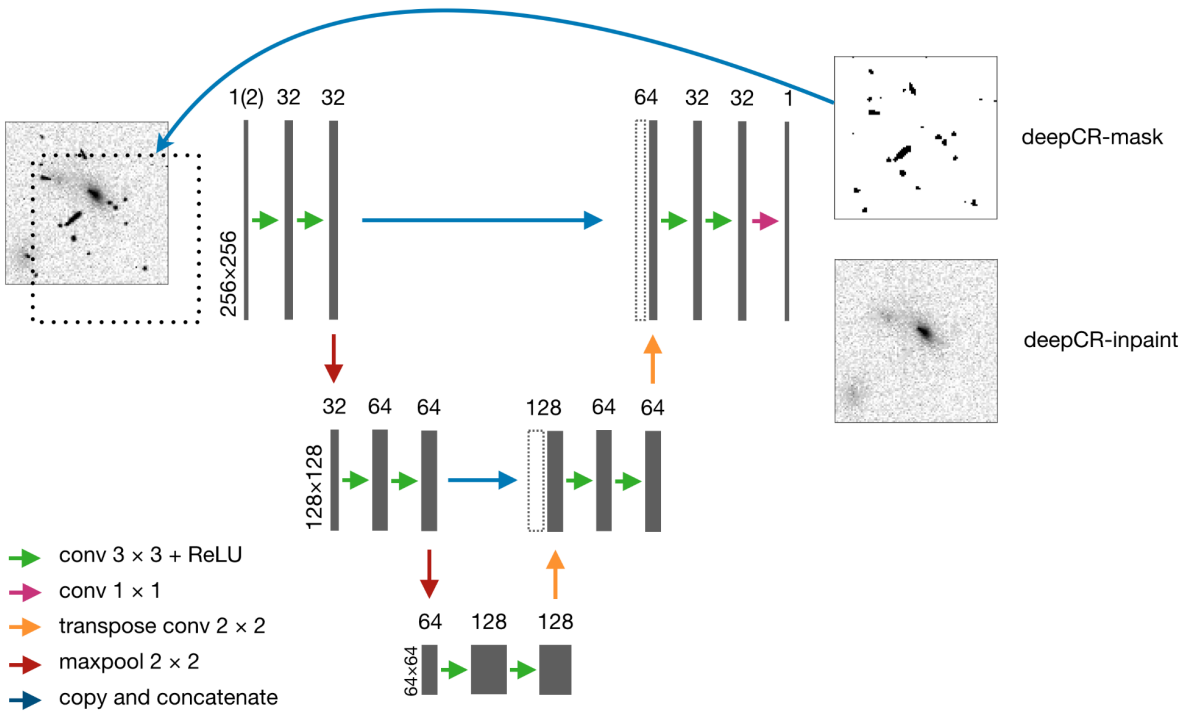
Dependencies

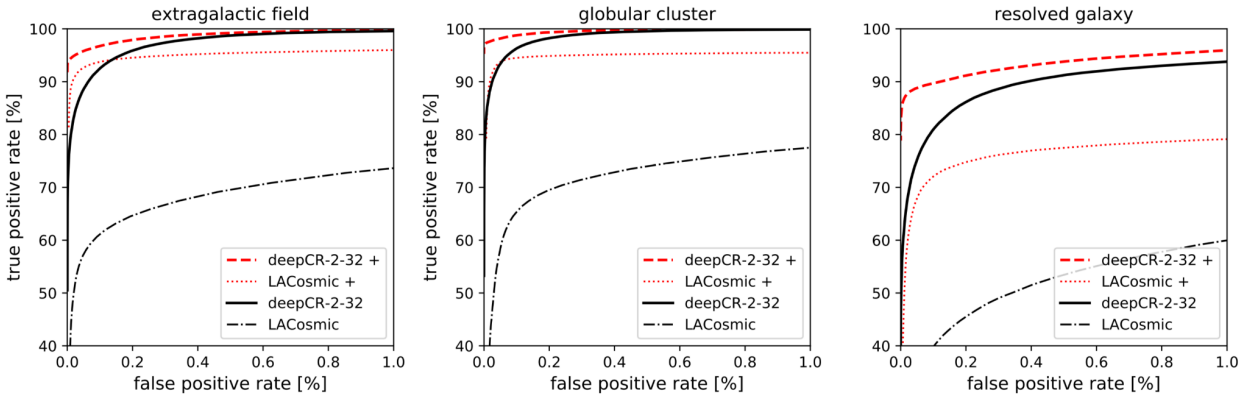
deepCR relies on a set of essential dependencies to function effectively. These components work seamlessly together to drive the cosmic ray removal process. Here's an overview of the key dependencies:

Python: The core programming language for implementing deep learning models and image processing.
PyTorch: A deep learning framework for building and training neural networks.
Astropy: A library for astronomical data analysis, providing essential tools for working with astronomical data.
NumPy and SciPy: Fundamental libraries for numerical and scientific computing, used for data manipulation and analysis.
Matplotlib: A data visualization library used for creating plots and figures.
Scikit-image: An image processing library that enhances image manipulation capabilities.
Jupyter: An interactive computing environment for creating and sharing documents with live code, equations, visualizations, and text.
Astroscrappy: A Python package specialized in cosmic-ray detection in single images.
These dependencies form the backbone of deepCR, ensuring its efficiency and effectiveness in cosmic ray rejection. Researchers and astronomers rely on this powerful stack to enhance the quality of their astronomical data.

Images:







References

deepCR GitHub Repository
Astropy
PyTorch
Matplotlib
SciPy
Scikit-image
Jupyter
Astroscrappy

Installation

pip install deepCR

Or you can install from source:

```
git clone https://github.com/profjsb/deepCR.git
cd deepCR/
pip install .
```

Quick Start

Quick download of a HST ACS/WFC image

wget -O jdba2sooq_flc.fits
https://mast.stsci.edu/api/v0.1/Download/file?uri=mast:HST/product/jdba2sooq_flc.fits

With Python >=3.5:

For smaller sized images (smaller than ~1Mpix)

```
from deepCR import deepCR
from astropy.io import fits
image = fits.getdata("jdba2sooq_flc.fits")[:512,:512]

# create an instance of deepCR with specified model configuration
mdl = deepCR(mask="ACS-WFC-F606W-2-32",
             inpaint="ACS-WFC-F606W-2-32",
          device="CPU")

# apply to input image
mask, cleaned_image = mdl.clean(image, threshold = 0.5)
# best threshold is highest value that generate mask covering full extent of CR
# choose threshold by visualizing outputs.

# if you only need CR mask you may skip image inpainting and save time
mask = mdl.clean(image, threshold = 0.5, inpaint=False)

# if you want probabilistic cosmic ray mask instead of binary mask
prob_mask = mdl.clean(image, binary=False)
```

There's also the option to segment your input image into smaller pieces (default: 256-by-256) and process the individual piece seperately before stitching them back together. This enables multi-process parallelism and saves memory.

Segment-and-stitching is enabled by n_jobs>1, which specified the number of processes to utilize. n_jobs=-1 is the number of available virtual cores on your machine and is optimized for time when your torch is not intel MKL optimized (see below for more details).

```
image = fits.getdata("jdba2sooq_flc.fits")
mask, cleaned_image = mdl.clean(image, threshold = 0.5, n_jobs=-1)
```

If your torch is intel MKL optimized, it's not necessary to open up many processes and one process should utilize half of the CPUs available. Monitor CPU usage -- if CPU usage for single process is > 100% it means intel MKL is in place. In this case, ** n_jobs<=4** is advised.

For single process segment-and-stitching, you need to manually enable segment = True because the default n_jobs=1 assumes segment = False.

```
image = fits.getdata("jdba2sooq_flc.fits")
mask, cleaned_image = mdl.clean(image, threshold = 0.5, segment = True)
```

Setup of the project:

```
import ast
from setuptools import setup
import sys

setup_requires = ['setuptools >= 30.3.0']
if {'pytest', 'test', 'ptr'}.intersection(sys.argv):
    setup_requires.append('pytest-runner')

# Get docstring and version without importing module
with open('deepCR/__init__.py') as f:
    mod = ast.parse(f.read())

# read the contents of your README file
from os import path
this_directory = path.abspath(path.dirname(__file__))
with open(path.join(this_directory, 'README.md'), encoding='utf-8') as f:
    long_description = f.read()

__doc__ = ast.get_docstring(mod)
__version__ = mod.body[-1].value.s
```

```
setup(description=__doc__.splitlines()[1],
    version=__version__,
    include_package_data=True,
    long_description=long_description,
    long_description_content_type="text/markdown",
    setup_requires=setup_requ
```

Quickstart

```
pip install deepCR
```

Or you can install from source:

```
git clone https://github.com/profjsb/deepCR.git
cd deepCR/
pip install .

from deepCR import deepCR
import numpy as np
mdl = deepCR(mask="ACS-WFC", device="CPU")
image = np.zeros((1024, 1024))
binary_mask = mdl.clean(image, segment=True, n_jobs=-1)
```

Model zoo: a collection of models with benchmarks

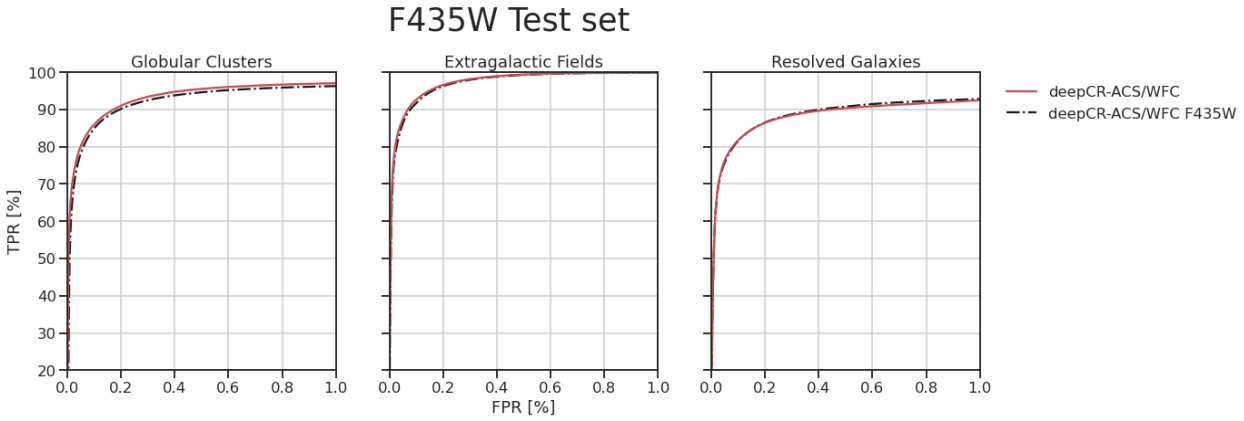
Hubble ACS/WFC

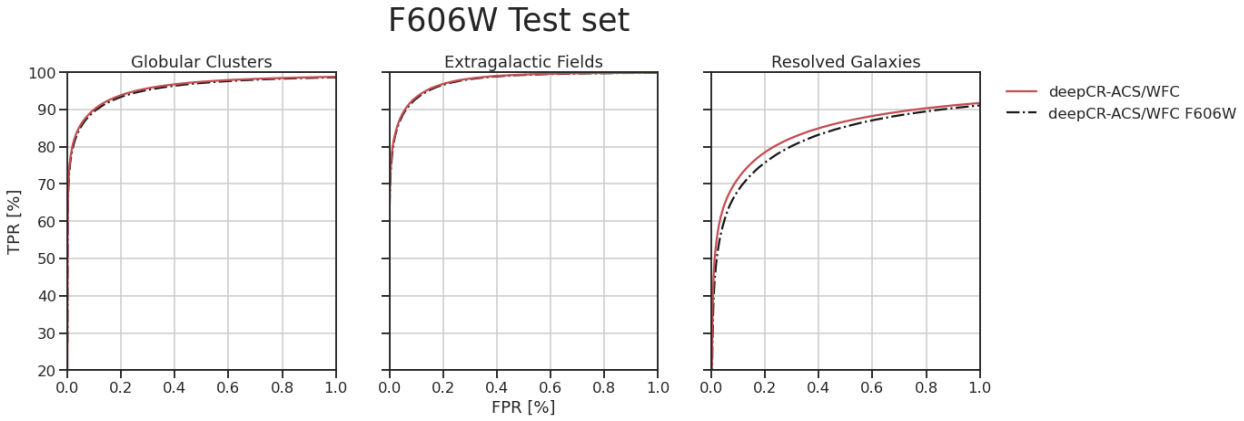
```
from deepCR import deepCR

model_cpu = deepCR(mask=MODEL_NAME, inpaint='ACS-WFC-F606W-2-32',
device='CPU')
```

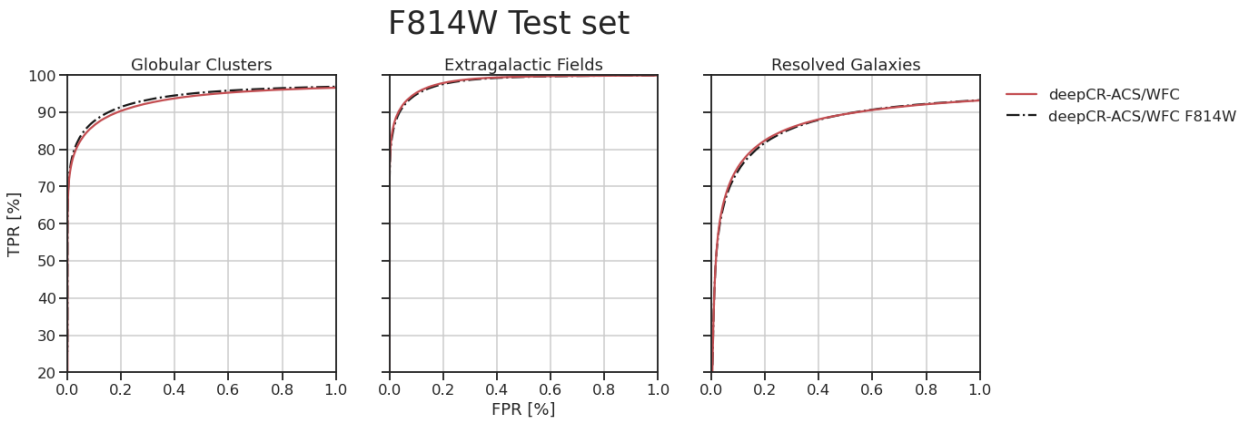
model_gpu = deepCR(mask=MODEL_NAME, inpaint='ACS-WFC-F606W-2-32', device='GPU')

ACS-WFC-2-32 model is trained with image sets from ACS/WFC F435W, F606W, F814W. Individual models are also available as ACS-WFC-F435W-2-32, ACS-WFC-F606W-2-32, ACS-WFC-F814W-2-32. The global model was tested with all three filter test datasets to demonstrate that there is no significant performance gap between the global model and the individual models. Refer to the ROC curves below. The Y-axis denotes the true positive rate (TPR), and X-axis denotes the false positive rate (FPR).

Since the training and test datasets comprise images from three different fields (extragalactic field, globular cluster, resolved galaxy), the models were tested for each field. As shown, there is no significant discrepancy between individual models' performance and the global model.



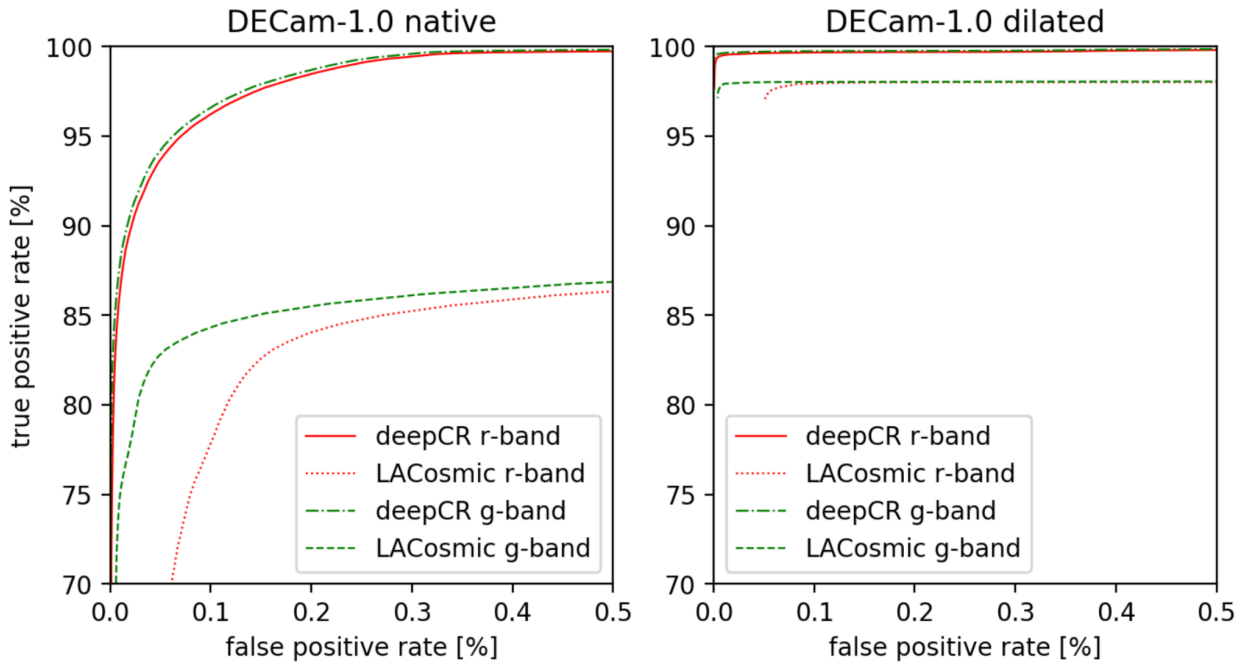




DECam

DECam is a high-performance, wide-field CCD imager mounted at the prime focus of the Blanco 4-m telescope at CTIO.

```
from deepCR import deepCR

model_cpu = deepCR(mask='decam', device='CPU')

model_gpu = deepCR(mask='decam', device='GPU')
```



The ROC curves above are produced from a test set that contains noise in cosmic ray labels. This causes TPR to be lower than actual because the deepCR predicted CR labels is essentially noise-free.

Note 0: The current DECam model as preliminary as it is trained on median-coadd science images which may cause false positives of stars in single frame images in some cases.

Note 1: Output will include some bad pixels and columns, and sometimes blooming patterns. In particular, the blooming artifacts in the CR mask might be 1-2 pixels larger than the actual blooming size and cannot be excluded by subtracting by a saturation mask.

Note 2:Input images should come from calibrated images in the original unit (adu).

Note 3: Model is trained on g-band images but is expected to work on other filters as well. We have benchmarked on g-band and r-band and are working i-band and z-band but only expect minor differences from the ROC curves above.

Note 4: For extremely short or long exposures (t_exp < 10s or t_exp > 1800s), please visually verify mask output.