AstroStatLearn Project: Galaxy segmentation

Camilla Saverese 1838890, Giulio D'Erasmo 1859130, Arturo Ghinassi 1863151
Andrea Potì 2008416, Amedeo Ranaldi 1982926

Contents

1 Introduction

The topic of our project was **Image segmentation and detection** of satellite images of galaxies using different technique involving typical methods as Random Forest and KMeans and neural networks like U-Net. Image segmentation is a process of assigning a label to every pixel in an image such that pixels with the same label are part of the same source.

2 The Dataset

The Dataset is composed of three images (25000x25000 pixels) in FIT format, which allow to preserve the intensity of large images when imported and saved. The first image < img.fits > is the raw satellite image, then we have the < rms.fits > which is the error of the image, and < true.fits > which is the true segmentation.

In order to train the various model we split the images in patches of 256x256 pixels increasing the number of sample for our training.

2.1 Pre-processing

We use different technique to pre-process the images.

1. normalize the image using normalize of < astropy.visualization > library;

2. enhance the contrasts using FITS Liberator 3;

3. add the log stretched rms to the previous step image.

The software FITS was applied and tuned in order to polish the image from the noise and augment the contrast between the dark background and the white galaxy. While adding the log stretched rms let us to show in a better way the galaxy thanks to the way the curve is defined which drop to 0 lower value and maps to 1 galaxy-pixel-value.

Figure 1: In order from left: original image, software enhancement and rms adding.

We decide to pre-process the images in order to do data augmentation on the training dataset and achieve better performance on the scores.

3 Metrics

In order to measure the performance of our models we use different metrics: Iou, Dice coefficient, Recall, Accuracy.

The metrics IoU and Dice coefficient are specific to image segmentation and detection. The first is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth while the latter is 2 * the Area of Overlap divided by the total number of pixels in both images, they both range from 0 to 1. The importance of this metrics is because in all reality, it's extremely unlikely that the (x, y)-coordinates of our predicted bounding box are going to exactly match the (x, y)-coordinates of the ground-truth bounding box, because of this, we need to define an evaluation metric that rewards predicted bounding boxes for heavily overlapping with the ground-truth.

We also use accuracy as standard score but we are aware that we can't based an entire evaluation of the models using this score because of the class unbalance between the background and the true galaxy led to achieve always an high percentage of accuracy due to the well predict of true negative value.

4 Baseline Methods

This are the classical method we apply at the beginning in order to have a baseline scores with respect more advance models like neural networks.

4.1 Astropy - image cleaning

Following photutils cleaning in order to segment the image we need to apply several step to achieve the desired results. First we subtract the background and background noise from image, then we convolve the data with a 2D Gaussian kernel with a FWHM of 3 pixels because the image is usually filtered before thresholding to smooth the noise and maximize the detectability of objects with a shape similar to the filter kernel. In the end we need to define the detection threshold (the threshold level is usually defined as some multiple of the background noise above the background) and we are ready to detect the sources that have 10 connected pixels that are each greater than the corresponding pixel-wise threshold level defined above.

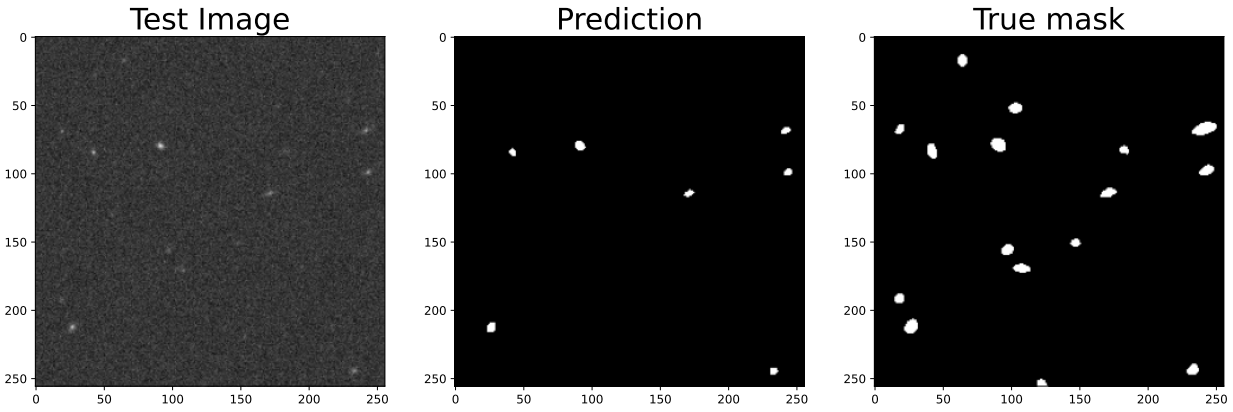

Figure 2: Prediction using the original image with MinMax normalization.

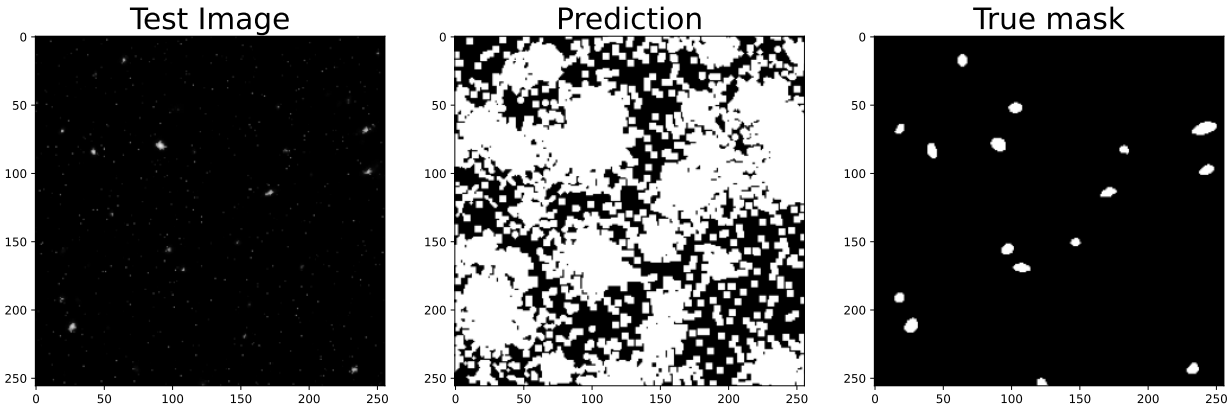

Figure 3: Prediction using the software pre processing image.

As shown above, this technique doesn't always works. This because it is not able to recognize the multiple white pixels as noise instead of galaxies due to the multiple applied preprocessings.

Here are the scores we were able to achieve with the different datasets used.

X	IoU	recall	dice_coeff	Accuracy
Image w/Normalize	0.2509	0.9942	0.3971	0.9612
Image software w/Normalize	0.1125	0.1235	0.1980	0.7463
Image software + rms w/Normalize	0.0795	0.0918	0.1415	0.4669
Image software+ rms w/Normalize and streatch	0.1063	0.1181	0.1783	0.5320

4.2 KMeans

Following this guide about KMeans for image segmentation we were able to segment the images in a faster way and with good results. The idea behind this implementation is to group similar color to the same cluster in order to separate the various part of the image, in our case background (black) and galaxy (white).

Figure 4: In order from left: original image, kmeans segmentation and true image.

Here are the scores we were able to achieve with the different datasets used.

X	IoU	recall	dice_coeff	Accuracy
Image w/Normalize	0.0315	0.2711	0.0594	0.2554
Image software w/Normalize	0.1526	0.5696	0.2624	0.9534
Image software + rms w/Normalize	0.1531	0.6202	0.2633	0.9542
Image software+ rms w/Normalize and streatch	0.2874	0.7008	0.4254	0.9452

4.3 Random Forest

The RandomForest classifier is trained with a training dataset of 500 images, divided thanks to the train-test split of the library sk-learn in training sample (70%) and validation sample (20%). For each image, there are 13 features computed. They are:

- Spectral features:
    1. Red
    2. Green
    3. Blue
- Texture feature:
    1. Local binary pattern
- Haralick (Co-occurance matrix) features:
    1. Angular second moment
    2. Contrast
    3. Correlation
    4. Sum of Square: variance
    5. Inverse difference moment
    6. Sum average
    7. Sum variance
    8. Sum entropy
    9. Entropy

Haralick features are statistical features of texture (Contrast, Correlation, Homogeneity and Energy) that are calculated from gray level Co-occurrence matrix (GLCM). In this project they are computed by the library OpenCV.

The Random Forest classifier was trained with the following parameters:

- Number of estimators = 250
- Max depth = 12

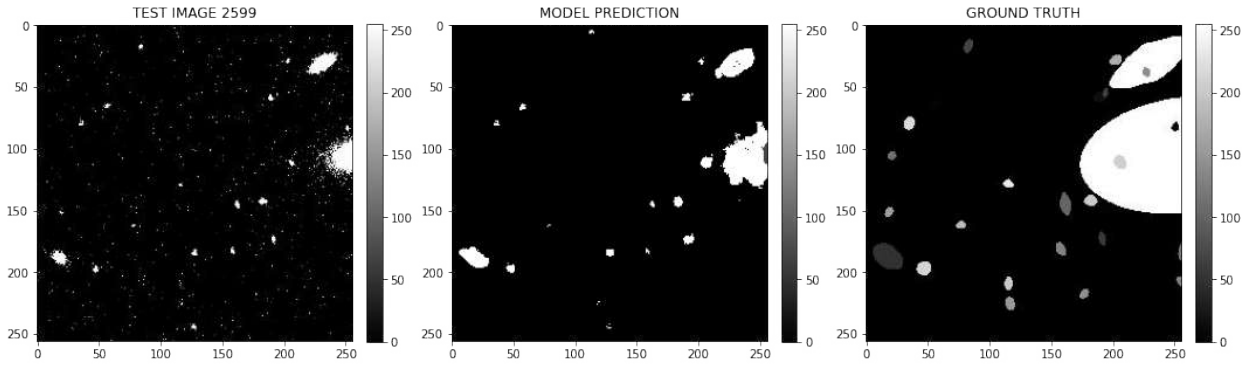

Figure 5: In order from left: original image, random forest segmentation and true image.

Here are the scores we obtain training the model using the dataset in which we pre process the image using the best data augmentation we were able to achieve as shown in the dedicated section.

	Precision	Recall	Accuracy	F1
score	0.1259	0.4434	0.76	0.1873

5 Neural Networks

5.1 U-Net

U-net is the first neural network architecture we tried. The name is given by the shape of the architecture that consists of two parts, the encoding part and the decoding part. In the encoding part there are 5 different layers, at each layer the input is passed to a 3x3 convolution and a ReLU activation two times and then the pixels are halved with a 2x2 max pooling moving to the next layer, and in the mean time the image gets copied and cropped to be concatenated with the resulting image of the same layer in the decoding part.

In the decoding part we have the same functioning but this time inverted, instead of the max pooling we have an upward convolution 2x2 that halves the number of features each time we move to the next layer. In the last layer the 64 remaining features are mapped to a single channel (galaxy or not) for each pixel (conv 1x1). In the end we get in output the segmentation map of same size of the inputs.

The main differences we applied to the original architecture were changing the input size because smaller patches were able to return better results. We added padding after each convolution, this means instead of copying and cropping we just had to copy and concatenate.

In order to train the model we divided the original images in patches of size (256x256) and we ended up with almost 2000 training samples. We had to binarize the masks in order to have 0 values for pixels not belonging to a galaxy and 1 otherwise.

The training set has been divided with an 80/20/20 split in train, validation and test set. We ran the model with batches of size 8 for 30 epoques but due to an early stopping monitoring the validation loss the model training finished earlier with good results.

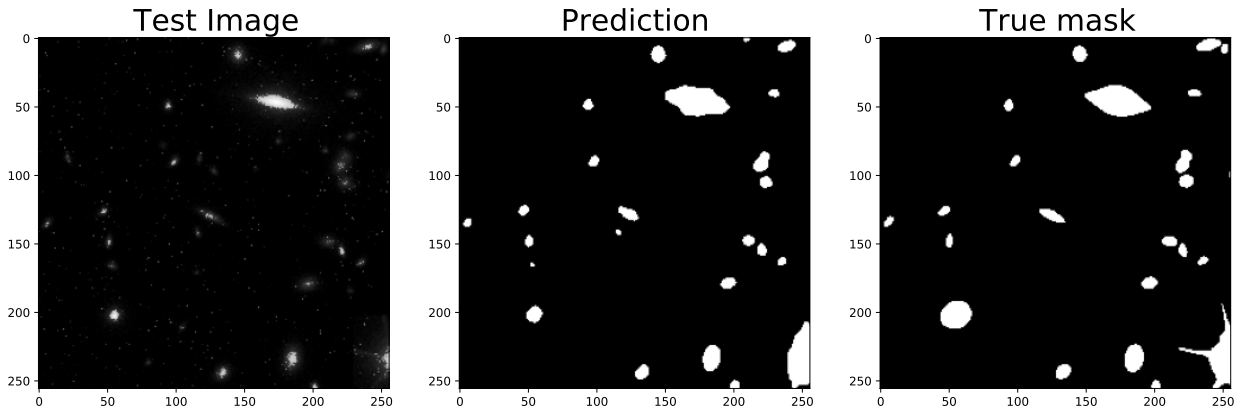

Figure 6: In order from left: original image, U-Net segmentation and true image.

Here are the scores we are able to achieve with the different datasets used.

X	loss	IoU..0.3	Iou..0.5	recall	dice_coef	accuracy
Image w/Normalize	0.0757	0.5871	0.5855	0.6587	0.5686	0.9763
Image software w/Normalize	0.0694	0.6077	0.6164	0.6931	0.6120	0.9781
Image software + rms w/Normalize	0.0656	0.6276	0.6623	0.7486	0.5984	0.9806
Image software+ rms w/Normalize and streatch	0.0393	0.7116	0.7076	0.7751	0.7493	0.9838

As we can see the model is able to obtain very good scores for the most relevant metrics when coming to image segmentation. The IoU is above 70% and the Dice score reaches almost 75%, this by using the full preprocessed dataset that surely is the main reason for the model to perform like this also comparing to the different type of preprocess applied.

5.2 Mask-RCNN

Mask R-CNN is a framework for instance segmentation which unify object detection based on Faster R-CNN, and semantic segmentation.

Mask R-CNN starts with two stage based on Faster R-CNN: the first which is called Region Proposal Network (RPN), proposes candidate object bounding boxes called Region of Interest or RoI. With the second stage, that improves the one who based off, called RoIAlign is used to classify the object label class in the RoI and refine the bounding box, and with respect the Faster R-CNN will output a object mask using the pixel-to-pixel alignment property of RoIAlign using a FCN.

The model is trained with a specific dataset format called COCO, which consist in the simpler format of dataset of images and the annotation dictionary containing a series of fields, including the category id and segmentation mask of the object.

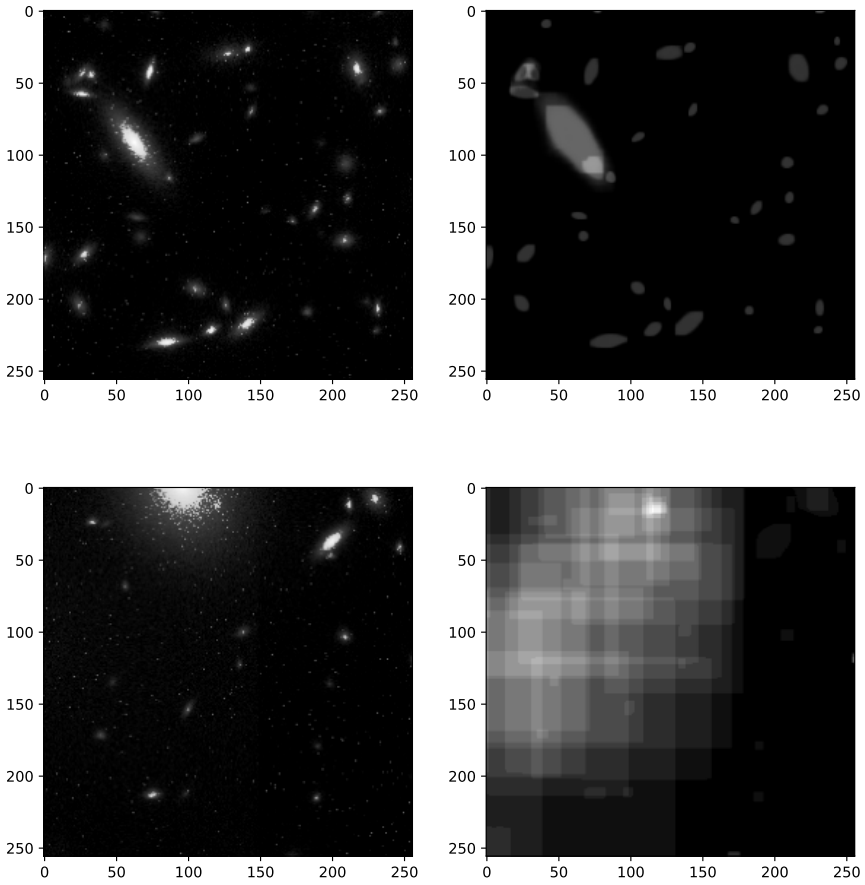

Figure 7: Image segmentation prediction of our trained RCNN model.

The model is trained using 20 epoques dividing the dataset in train/validation/test with a batch size of 8, Adam optimizer and a learning rate updater. Here we show the scores we obtain by our trained model. The metrics used here is called Average Precision at IoU tresholds, meaning that, for example referring to the object detection, we will predict True Positive only if the IoU is greater than the treshold and than compute the Precision metrics for all the class and images.

We can see that we obtain modest result using the normalize version of the image, with an higher score training the model using the cleanest and sharpest image we achieved.

	bbox	bbox.1	bbox.2	bbox.3
X	AP @[IoU=0.50:0.95]	AP @[IoU=0.50]	AP @[IoU=0.75]	AR @[IoU=0.50:0.95]
Image	0.005	0.016	0.002	0.036
Image w/Normalize	0.112	0.422	0.025	0.559
Image Software w/Normalize	0.107	0.356	0.041	0.51
Image Software + rms w/Normalize	0.245	0.718	0.086	0.581

	segm	segm.1	segm.2	segm.3
X	AP @[IoU=0.50:0.95]	AP @[IoU=0.50]	AP @[IoU=0.75]	AR @[IoU=0.50:0.95]
Image	0.007	0.023	0.003	0.023
Image w/Normalize	0.214	0.596	0.074	0.552
Image Software w/Normalize	0.193	0.532	0.096	0.533
Image Software + rms w/Normalize	0.389	0.78	0.338	0.629

5.3 DeTr

Unlike traditional computer vision techniques, DETR approaches object detection as a direct set prediction problem. It consists of a set-based global loss, which forces unique predictions via bipartite matching, and a Transformer encoder-decoder architecture. Given a fixed small set of learned object queries, DETR reasons about the relations of the objects and the global image context to directly output the final set of predictions in parallel. Due to this parallel nature, DETR is very fast and efficient.

The overall DETR architecture is surprisingly simple and depicted in Figure 2. It contains three main components, which we describe below: a CNN backbone to extract a compact feature representation, an encoder-decoder transformer, and a simple feed forward network (FFN) that makes the final detection prediction. Unlike many modern detectors, DETR can be implemented in any deep learning framework that provides a common CNN backbone and a transformer architecture implementation with just a few hundred lines. Inference code for DETR can be implemented in less than 50 lines in PyTorch.

5.3.1 Panoptic Segmentation

Panoptic segmentation unifies the typically distinct tasks of semantic segmentation (assign a class label to each pixel) and instance segmentation (detect and segment each object instance).

5.3.1.1 Panoptic Segmentation Format The format for panoptic segmentation is simple to define. Given a predetermined set of L semantic classes encoded by $\mathcal{L} := \{0, \ldots, L-1\}$, the task requires a panoptic segmentation algorithm to map each pixel i of an image to a pair $(l_i, z_i) \in L \times N$, where l_i represents the semantic class of pixel i and z_i represents its instance id. The z_i's group pixels of the same class into distinct segments. Ground truth annotations are encoded identically.

Each annotation struct is a per-image annotation rather than a per-object annotation. Each per-image annotation has two parts: (1) a PNG that stores the class-agnostic image segmentation and (2) a JSON struct that stores the semantic information for each image segment. In more detail:

1. To match an annotation with an image, use the `image_id` field (that is `annotation.image_id==image.id`).
2. For each annotation, per-pixel segment ids are stored as a single PNG at `annotation.file_name`. The PNGs are in a folder with the same name as the JSON, i.e., `annotations/name/` for `annotations/name.json`. Each segment (whether it's a stuff or thing segment) is assigned a unique id. Unlabeled pixels (void) are assigned a value of 0. Note that when you load the PNG as an RGB image, you will need to compute the ids via `ids=R+G*256+B*256^2`.
3. For each annotation, per-segment info is stored in `annotation.segments_info`. `segment_info.id` stores the unique `id` of the segment and is used to retrieve the corresponding mask from the PNG (`ids==segment_info.id`). `category_id` gives the semantic category and `iscrowd` indicates the segment encompasses a group of objects (relevant for thing categories only). 4. The bbox and area fields provide additional info about the segment.
4. Finally, each category struct has two additional fields: `isthing` that distinguishes stuff and thing categories and `color` that is useful for consistent visualization.

5.3.2 Application

We decide to select just a smaller portion of the starting image in order to avoid long processing times as we did with the previous models. We first generate the dataset of images from the initial image and then we generate all the annotations following the semantic of COCO panoptic. Using the Pytorch Lightning library allowed us to create a class dataset for the annotations, and extend the DETR model to work with panoptic segmentation.

Finally we use `detr-resnet-50-panoptic` as a pretrained model to work with and we train our model with 3 epochs and a batch size of 3.

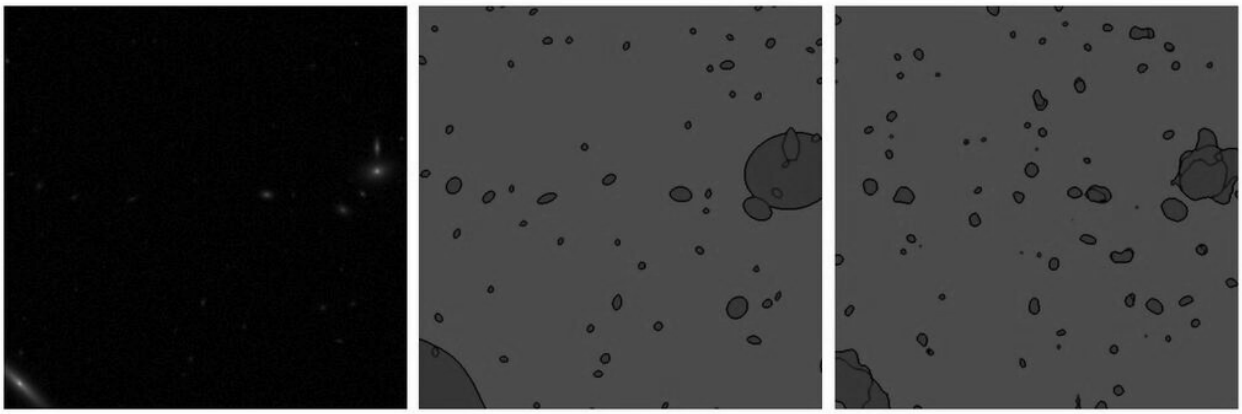

Figure 8: Image segmentation of RCNN model.

6 Conclusion and possible improvement

In the end what we discovered is that the baseline method didn't achieve to recognize the difference between noise and actual galaxy while neural networks where almost perfect for this aim also with few epoch. Using GPU with Google Colab free we weren't able to train the models with a modest amount of epoch and a vary big sample dataset which certain will result in better performance. Also we didn't augment so much the dataset with rotation, change in color and affine transformation that helps the NN to generalize better. Other better methods as MPVit are known to perform better in image segmentation task but it was to difficult to implement.

References

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-end object detection with transformers. *CoRR*, *abs/2005.12872*. Retrieved from https://arxiv.org/abs/2005.12872

He, K., Gkioxari, G., Dollár, P., & Girshick, R. B. (2017). Mask R-CNN. *CoRR*, *abs/1703.06870*. Retrieved from http://arxiv.org/abs/1703.06870

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, *abs/1505.04597*. Retrieved from http://arxiv.org/abs/1505.04597