[Machine Learning in Science:] Performance of advanced object detection algorithms on astronomical images

Dominic Askew and Jack Rowbotham

Department of Physics and Astronomy, University of Nottingham

Submitted: 31 May 2019

Abstract

We investigated the use of machine learning, more specifically convolutional neural networks for automatic detection and classification of objects in Hubble space telescope images. Manually processing the vast amounts of astronomical data collected daily is hugely time consuming and inefficient. Embracing new machine learning techniques could be the solution to this. Moreover, we investigated the use of transfer learning in conjunction with CNNs, by retraining networks first trained on pictures of everyday objects. We started by tailoring a pre-existing architecture called Mask R-CNN to our image types. We then created a standard model, changed specific parameters and measured the effect this had. We found that for the standard model the results were $mAP = 0.1927 \pm 0.0042$ and *classless* $mAP = 0.3294 \pm 0.0072$ with 3 'good' overlaps. Changing the pre-processing stretch function s to $s = 0.1$ had a positive effect on the models' performance. A smaller number of dividing classes also seemed to improve the models' performance. The peak performance for number of epochs occurred at around 120 epochs, slightly higher than the standard model. We found that using the backbone ResNet50 reduces training time with only a small negative effect on the performance, having big connotations for optimisation. Finally we compared our model to existing models and discussed the future direction of convolutional neural networks.

1 Introduction

The rate at which astronomical data is being collected vastly outweighs the rate at which it is being processed. With advancements made in the computer and data sciences, the rate at which data is collected has grown exponentially. For example, the Hubble space telescope transmits roughly 140 gigabits of data every week.

A proposed solution can be found in the newly developed field of machine learning, specifically using a convolutional neural network to detect and then classify astronomical objects at a much faster rate than any technique used currently. The next paradigm shift in astronomy must embrace data science techniques and technologies, one of these techniques being machine learning. Machine learning algorithms can be used for automatic object detection and classification in astronomical images and at the rapid processing rates of a computer and, if implemented correctly, could be a solution to astronomy's data overload.

This research explores the use of machine learning techniques within astronomy and more specifically applies a convolutional neural network to astronomical images in order to train a model to accurately detect and classify galaxies into their respective morphology sets. We started by finding a pre-existing architecture as a point of departure and aimed to tailor an algorithm to the far-field data we gathered from the Hubble space telescope in order to reach an accuracy comparable to models within the current literature.

1.1 Convolutional Neural Networks

The primary machine learning technique used in image recognition today is called a convolutional neural network (CNN). First inspired by biological processes, CNNs are designed to emulate the structure and connectivity of the animal visual cortex [1].

Deep CNNs have no hand-engineered feature detectors involved. The pertinent features for the specific classification task are purely determined by the network during training, whereby free parameters of the CNN are tuned to a given task with an optimisation algorithm. This is known as feature learning and is the basis of machine learning.

Coined by Hubel et al. [2–4] the idea of the receptive field is what sets the CNN algorithm apart from other neural networks, making it faster and more efficient. Early models of neural networks all possess fully-connected layers, within which every neuron in one layer is connected to every neuron in another layer. Traditional multi-layer perceptrons (MLP), early neural networks, used fully connected layers, although these became their downfall. MLPs suffer from the curse of dimensionality due to their fully connected nodes[5]. Therefore, although MLP models were successfully used in image recognition, modern high-resolution images have too many dimensions, including colour channels, to be processed quickly and efficiently. This is where receptive fields become relevant. In modern CNNs, neurons are connected to

only some, rather than all, of the neurons from the previous layer, which is called the receptive area.

1.1.1 Layers

A convolutional designed neurological network contains an input and output layer, as well as multiple hidden layers commonly consisting of convolutional, rectified linear unit (ReLU), pooling, fully connected and normalisation layers.

Pooling layers compute the mean (or max) value of a feature matrix over a region of the image, producing a summary statistic matrix of a lower dimension[6]. This reduces over-fitting which can improve results compared to using the complete extracted feature matrix, and is an example of a receptive field within a convolutional neural network.

1.1.2 Weights

Every neuron in a CNN computes an output from an applied filter, composed of vector weights and a bias which represents a specific feature of the input, such as a shape. A distinguishing feature of CNNs are their shared weights, meaning that all neurons in a convolutional layer react to the same feature in the same way. Therefore, features can be detected regardless of their position in the visual field, making CNNs' translation invariant as well as reducing the memory footprint. The adjustments in weights recorded from learning in the neural network are determined by minimising the loss function and recording the change. This process is called backpropagation. An error is computed at the output and distributed back through the networks layer. The weights are corrected using the error obtained.

1.2 Current literature

As of 2019, CNNs have boomed, with various open-source repositories of neural networks available. In their 2017 paper, Focal Loss for Dense Object Detection[7], Lin et al. compare their object detection method to various other top performing methods, such as Faster R-CNN, YOLOv2 and SSD5[8–11]. They found that their model, RetinaNet, outperforms other state-of-the-art models, regardless of whether they are one-stage or two-stage. To achieve this, they proposed the idea of focal loss which applies a modulating term to the cross-entropy loss in order to focus learning on hard negative examples. The most recent version of YOLO, YOLOv3 has since been released and achieves roughly the same performance but at 3.8 times the speed. However, these models are focussed on, and therefore tailored to, the detection and classification of common objects in everyday scenarios.

These techniques have been applied within the field of astronomy. In 2018, Gonzalez et al. published a paper called Galaxy detection and identification using deep learning and data augmentation[12]. It highlights a method for auto detection and classification of galaxies using DARKNET and YOLO13[9] as its point of departure. The paper outlines an issue with astronomical type FITS data, which has to be converted to a 3-channel colour image data type to be compatible with the CNN. This conversion is not unique and depends on a number of factors, most importantly the conversion method used to scale photon count to colour scale. The effect of changing these factors is investigated in this research.

A paper on the use of transfer learning to detect galaxy mergers [13] investigates how deep CNNs can be retrained, similarly to that described by Gonzalez et al. It looks at testing the hypothesis that transfer learning improves performance for small training sets. This is useful since galaxy mergers are rare so the data is limited. The issue addressed in this paper is that, when networks are trained with a limited number of data samples, the algorithm does not generalise well to new data and overfits. With large neural networks and increasingly smaller training set sizes with rare astronomical objects, overfitting becomes especially problematic. The solution emphasised is the use of regularisers to combat the overfitting. The IMAGENET [14] dataset is used to pretrain a classifier CNN, which acts as a regulariser. The classifier CNN is initialised with pre-trained weights and its performance is compared with a random weighting initialisation. Results obtained show that there was an improvement in the generalisation performance when used to train the merger classifier when it is initialised with the IMAGENET dataset.

Most current CNN architectures require images to be in the 3-colour channel RGB format. Astronomical data, however, is found in FITS format, and so data conversion methods are required before images can be input. Conversion methods from FITS in igr bands to RGB can be computed using methods highlighted by Lupton et al [15].

From [15], the usual algorithm is

$$R = f(r);\ G = f(g);\ B = f(b) \tag{1}$$

where,

$$f(x) = \begin{cases} 0, & x < \mathrm{m}; \\ \frac{F(x-m)}{F(M-m)} & \mathrm{m} \leq x \leq M; \\ 1 & \mathrm{M} < x. \end{cases} \tag{2}$$

giving a mapping between 0 and 1. From Lupton et al.[16], *F(x)* is given by

$$F(x) \equiv a \sinh(\frac{x}{\beta}) \tag{3}$$

where β is the softening parameter. β, determines the point at which we change from a linear to a logarithmic

transformation as

$$a\sinh(x) \sim x \; for \; x \ll 1; \qquad a\sinh(x) \sim ln(2x) \; for \; x \gg 1 \quad (4)$$

The linear part is used to show faint features in the data, while the logarithmic part enables us to simultaneously illustrate structures such as spiral arms or the cores of star clusters that are significantly brighter[15]. These parameters have a great effect on the images and, therefore, on the performance of any CNN model.

2 Method

Our background research and literature review revealed which performance and object detection algorithms could form the best foundation for our astronomical object detector. During our investigation of the YOLO and Detectron architectures, amongst others, we discovered that the size of these networks is often so large that running a training configuration from arrays of pre-trained weights could take an unfeasible amount of time. Due to the nature of our comparatively simplistic data, we chose to use a simpler network that utilises fewer layers: the architecture Mask-RCNN[17]. This is one of the algorithms implemented in Detectron, a model built by the Facebook AI research team that has won various awards including the Marr Prize at ICCV in 2017. Essentially, Mask-RCNN is the product of two pre-existing state-of-the-art models and is composed of two parts: a region proposal network (RPN) and a binary mask classifier. Conceptually, this is a simpler framework for object instance segmentation. It is flexible and, therefore, more compatible with our relatively simple HST dataset, the images of which, computationally speaking, appear as high-intensity peaks on a black background. We aim to use this framework to increase the performance of instance segmentation and bounding-box object detection.

2.1 Mask-RCNN Demo

Following the Github repository of Mask-RCNN, we found an implementation of Mask-RCNN on Python 3, Keras, and TensorFlow. The source code of Mask-RCNN is built on a Feature Pyramid Network (FPN) and a ResNet101 backbone architecture. Research indicates that training a residual network (ResNet101) is easier than training simple deep CNNs and partly mitigates the problem of degrading accuracy. This residual network backbone is a 101 layers deep and has been trained using more than a million images from the ImageNet database. We familiarised ourselves with the network by using the demo notebook in the Github repository. This notebook loads a network that has been pre-trained by the MS COCO dataset[18] and performs as an object detector for an input image we provide. It uses pre-trained weights and a defined classification index set based on the images contained in the MS COCO dataset. After creating the model and loading the weights, we ran the object detector to process our HST test image. The scores and class ID were printed with the corresponding mask and bounding box. For our test image, the object detector classified our image as a TV with a 0.708 certainty.

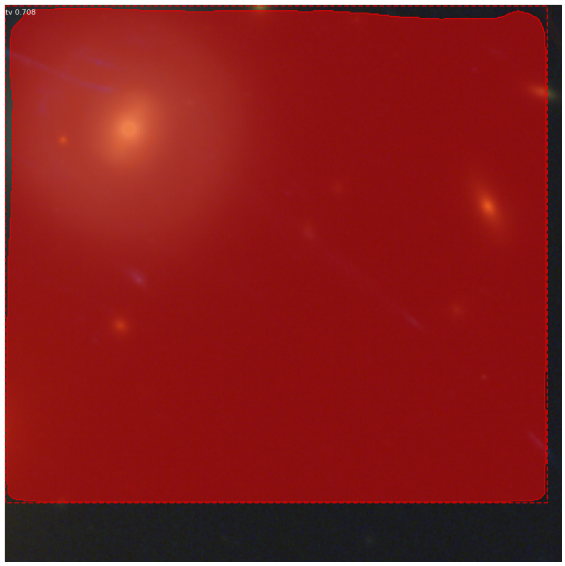

Figure 1: The result of running our astronomical dataset through a CNN machine learning model pre-trained on a dataset of common objects in context (COCO [18]). The model is trained on 81 classes, including common objects like 'person', 'bicycle' and 'tv'. The model predicts a single object with the classification of a television with a confidence of 0.708. The model has massively overfit, grouping all objects in the image into 1. This is expected as the model has been trained using much more complex images with much steeper and more frequently changing pixel intensity gradients.

2.2 Pre-processing Methods

Our HST dataset of galaxies was given to us in the FITS format, a standard digital file format for astronomical images. For our model, we required an RGB colour file format. We set up a notebook for processing this data prior to implementing it into our model. Lupton et al. describe a method that converts FITS formats to RGB given three wavebands are provided.

The function based on Lupton et al.'s methods[15] takes in three parameters: a) the minimum, m as seen in Eq.2 and defined as the intensity mapped to black, b) the stretch function, s, and c) the arcsinh softening parameter, β from Eq.3, which is taken into the function as Q. Figure (2) shows a linear stretch translation as seen in Eq.2.

With reference to Figure (2), the input values have a black intensity of 55. The output value, however, has a black intensity value of 0. In summary, changing the minimum, m, will change the black level, while the stretch, s and Q, will change how the values between

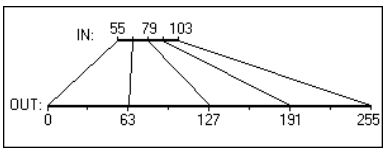

Figure 2: A visual example of the linear stretch function. Input values range from 55 to 103 and the output values range from 0 to 255.

black and white are scaled.

By pasting the raw code of this function into our notebook and by applying some indexing to read-in the three wavebands of 814, 606 and 435 respectively, we were able to successfully convert our data into a the format required for our model. However, the dataset we were given is a large satellite sky scan of approximate dimension 10000 x 10000 pixels. The model we built on reads in images of dimensions 256 x 256, so we needed to crop our image into smaller windows. This made up the training and validation datasets, so cropping into smaller images increased their size. This was achieved using a simple nested loop iterating over steps of 256, which then saved into a new data directory.

2.3 Mask-RCNN Model

After starting with object detection with pre-trained weights, we expanded the notebook to train Mask-RCNN on our HST data. As a basis, we adapted the shapes demo model.

2.3.1 Dataset

The demo shapes model already had an accompanying dataset to train the model. Instead of using this, we trained our model using the HST data. This required us to change the set class IDs for various shapes to a simple standalone classification of object. Having a single class allowed for a pure test of performance of object detection. The HST dataset was broken down into multiple cluster images. To retrieve the data and input it into the model, some indexing was required. We filtered through the root directory for the output images defined as 'crops' in the pre-processing stage. From the total number of images retrieved, we categorised them into two sets, training (80%) and validation (20%).

2.3.2 Configurations

The configurations for training on the astronomical objects dataset are derived from the base configuration class, so we needed to override values specific to the dataset such as crop size, the number of steps per epoch, and validation steps. It is also worth noting that the configuration name we set here will appear on the log files outputted from training, so it was useful to set it as something recognisable for each run. The image dimensions were set to 256 x 256 by default, although larger images could be used at the expense of a reduced training and validation set size as well as longer training times. The number of epochs tells us how many times our dataset will be run through our network. Following one complete epoch, a checkpoint will be outputted in the log files containing the updated weights following that run through. A training step represents one gradient update per batch-size processed. The number of steps per epoch defines the batch-size, which is dependant on the size of the training dataset. Traditionally, one epoch is a complete run through of the entire training dataset. Therefore, the size of the dataset is equal to the number of steps per epoch multiplied by the batch size. In the demo shapes model we are expanding, the number of epochs and the number of steps per epoch were set to 100 and 300 respectively. These values were chosen due to the data being simple and the fact that a number of epochs exceeding 100 would lead to overfitting in training. Although our data was simple, we could not be sure what value would be optimal, or whether 100 epochs would lead to overfitting. To combat this, we set our values to be the same as those used for the demo shapes model with the intention to vary these when analysing the performance of our model.

2.3.3 Training

Following configuration of the model, we began training. Firstly, we needed to create the model. After defining the model and pasting the configurations we had pre-set, we set up the model in 'training' mode using the Mask-RCNN import 'modellib.MaskRCNN'. We then defined which weights to start with in the model. By default, this was initialised with weights pre-trained on the COCO dataset[18]. Layers that were different due to a different set/number of classes were skipped and the weights were randomly initialised. These could be altered to initialise with 'last' if the model crashed during the training process. This would extract the last checkpoint log file and load the weights from there.

The model's training was completed in two stages:
(1) We trained only the 'head' layers by freezing all the backbone layers and training only the randomly initialised layers instead. This allowed the model to configure all the weights for the layers which did not use the pre-trained weights. This influenced quicker learning and more precise selection of weights per epoch, since there were far fewer layers to alter following each step per epoch. To train only the head layers, we passed layers equals 'heads' to the train() function. In configuration we set the number of epochs to 100. We assigned 50 epochs to the first stage of the training process and 50 epochs to the second stage.
(2) We fine-tuned all the layers. This unfroze the backbone and allowed the algorithm to alter the weights of the full model during training. It is worth noting that, following the second stage of training, we expected the losses to follow a similar trend to that in the first stage of training, since the model had

to adjust its weights accordingly. A plot of loss is described in figure (3).

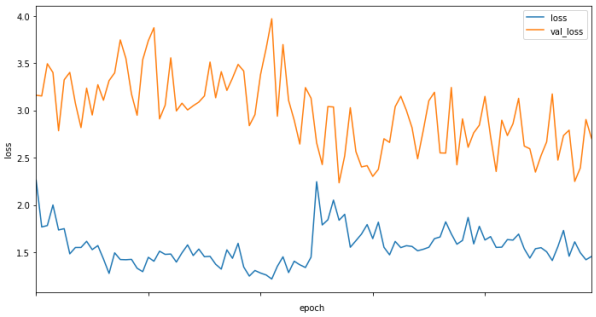

Figure 3: A plot of loss (blue) against number of epochs for our standard model. With each epoch iteration, back propagation occurs, adjusting the weights and lowering the loss for the next epoch. Although noisy, the general trend is a decrease of loss as the epochs increase, implying the model is learning. We can see that, as we pass the 50 epoch mark, the second stage of training begins. This is why we expect, and see, an initial increase in loss since the model now allows all layers to pass through the network. A plot of validation loss (orange) against number of epochs describes the measure of how much the model's predictions in that epoch differ from what they should be. This trend shows huge fluctuations, although the general trend indicates that as the number of epochs increases, the model's predictions are becoming more accurate.

2.3.4 Detection

In object detection, we kept the source code the same since it loaded the weights from the last saved log file outputted in training. We then selected a random image from our validation dataset and used this to compare how our model performed versus Source Extractor. For consistency, we decided to select an image from the validation set which had some distinct features so we could see how well the detector performed, outputting a corresponding mask and classification ID. Firstly, we loaded in the ground truth parameters of bounding boxes, masks, and class IDs from Source Extractor and ran this on whichever image we chose to test. Then, from our model, we loaded in the proposed masks, class IDs, and corresponding scores for comparison.

2.4 Evaluation Methods

To assess the performance of a model and compare it to other models, we used a number of evaluation metrics. The first, and perhaps most commonly used, metric is mAP or the mean average precision[19–21]. For the VOC2007 challenge, the interpolated average precision was used to evaluate both classification and detection. It is the mean of the precision of a number of factors, where precision in the case of pattern recognition is the number of true positives divided by the sum of true positive and false positives and can be visualised in figure (5).

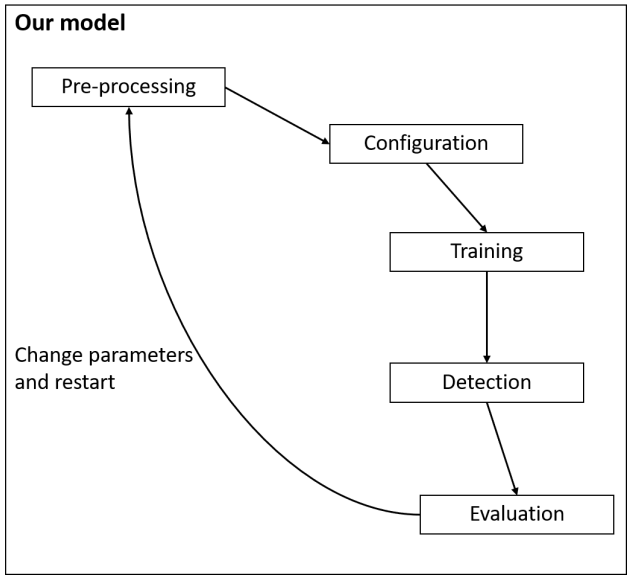

Figure 4: A flow diagram of our models function. Starting with pre-processing where our data is converted from FITS format to RGB. The crop size and RGB parameters are determined here. Next, with the prepared data, the configuration is where the model parameters are determined, such as the number of epochs and the models backbone. Training is where the model, initialised with COCO[18] weights, uses back propagation to adjust its weights each epoch and train itself on our astronomical dataset. Once training is complete, the model can be used to detect and classify. A crop from pre-processing is selected at random to assess the detection and classification abilities of the now trained model. The evaluation involved calculating our evaluation metrics. Once this is calculated and recorded, certain parameters can be changed and the model can be initialised from pre-processing once more.

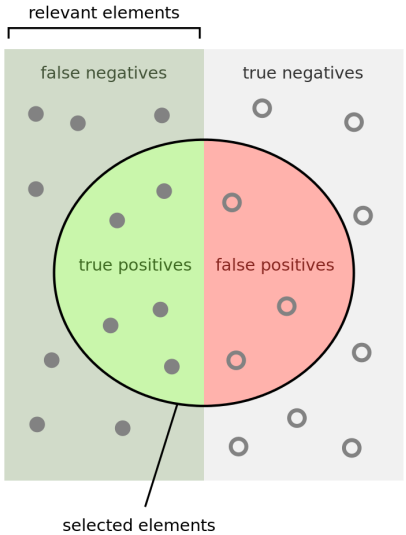

Figure 5: A visualisation of precision in the context of pattern recognition where precision is TP / (TP+FP) where TP is the number of True Positives for example in our case, the correct classification and FP is the number of false positives which following on for our example would be correct detection of an object but the wrong classification.

The mean average precision is given by

$$mAP = \frac{\sum_{q}^{Q} AvgP(q)}{Q} \tag{5}$$

where q is a relevant factor, Q is the number of total relevant factors and $AvgP$ is the average precision of each factor. More specifically, the factors that are assessed using this metric are bounding boxes, masks, and the classification of the objects. These give a good measure of the model's overall performance, including object detection and object classification measures. This is calculated by an in-built Mask R-CNN function that takes in the number of images to average over and the factors to take into account (More on this in 4.1.7).

The next metric used is classless mAP, the same mean average precision, but with the classification factor excluded. By nullifying the classification factor, this metric gives a good measure of the models object detection abilities.

N good overlaps is another evaluation method used and is important specifically in the context of astronomical images. An issue with object detection and classification in astronomical images is that the objects the models are trying to detect are often very similar and very close together. For example, a galaxy merger may have two identical objects in the same pixel region which can then be very easily assumed to be one, single object, whereas as in a dataset with more common objects like COCO[18], there are more features and higher contrast that makes objects more easily identifiable to the trained model. N good overlaps computes the positional overlap of the masks produced by the model with the ground truth masks, counting only overlaps of larger than 50%.

2.5 Expanding Model

The model we have used up to this point has only a single classification ID being object. This means that, currently, our model is purely an object detector minus classification. Expanding on our model, we looked to implement multiple classes to define the objects detected in our images. This required a rerun of Source Extractor.

2.5.1 Source Extractor

Source Extractor (SExtractor) [22] is used for the automated detection and photometry of sources in FITS image files. Running SExtractor over the data provides a ground truth from to which we can compare our model. The image processing algorithm calculates masks, bounding boxes, and classifications to a much larger degree of accuracy than a purely machine learning algorithm could currently produce. It is important to note then that the ground truth from which we compare may still carry over some computational error (more on this in 4.1.6).

2.5.2 Extra Classes

From the masterSEx FITS file we obtained from SExtractor, we could then implement classes with parameters consistent with the SExtractor parameters we had used. This allowed us to expand our classification from a single object class to individual classifications of magnitude, colour, and concentration. We then constructed a nested loop to extract each individual classification value for each parameter and combined all three groups to give a more detailed classification. Our model then output a class ID in the format colour_conc_mag. This then expanded our classification network from 1 to 13 class combinations. Making a small tweak to the class IDs outputted, we identified the general characteristics of galaxy morphologies and renamed some of the classifications. We found that red, high concentration generally implies an elliptical galaxy, whereas blue, low concentration generally implies a spiral galaxy[23].

N classes	Class Combination
2	Object
3	Concentration
3	Colour
4	Magnitude
5	Colour Concentration
7	Colour Magnitude
7	Concentration Magnitude

Table 1: The number of classes for each class combination. The background adds a class to each combination. Concentration divides objects into two sub-classes: high and low concentration. Colour also divides objects into two sub-classes: blue and red shifts. Magnitude divides objects into three sub-classes: faint, medium, and bright. Our standard model includes thirteen classes: a combination of background, colour, concentration, and magnitude. For example, a single object may be classified (within our standard model) as red, low concentration, and faint.

2.5.3 Optimised FITS - RGB

In fine tuning our pre-processing method, we defined a multicrops function which used indexing across all raw data files, inputting the R, G and B wavebands. We then defined a desired crop size and read in the size of the image to find out how many crops we could collect. From this, we iterated over n crops, collecting crops of dimensions defined in crop size over the whole image. The corresponding masks were also collected for their respective crops. We then defined an argument where, if a crop contains an object mask, it is saved in the image directory and indexed based on the row/column of the crop from the initial image. If the crop does not

contain an object mask, it is not saved. Therefore, our dataset is entirely built up of galaxy images with at least 1 region of interest.

2.6 Experimental Procedure

2.6.1 Standard Model

Prior to investigating the performance of our model, we defined a standard model which was configured with control variables. Our standard model will also be used as a comparison for parameter changes. The standard model is trained on a dataset pre-processed with parameters $m = 0$, $s = 0.5$, and $Q = 8$. These were defined as our control data parameters since the function used to convert FITS to RGB, as highlighted in Lupton et al., set these parameters as a recommended default for astronomical images. Into model parameters, our standard model was configured with a ResNet101 backbone, a classification index of 13 class combinations, and was set to iterate over 100 epochs total split equally in two training stages: 'heads' and 'all' layers. All other model parameters stayed consistent throughout our investigation. For visual representation, a test image was defined to display the masks, bounding boxes, and scores from both SExtractor and our model. This image is used consistently throughout our project to give us a visual indication of how the variation of parameters affects performance in detection and classification. This image is displayed in figure (6). The respective SExtractor detection and standard model detection is also highlighted in figure (6).

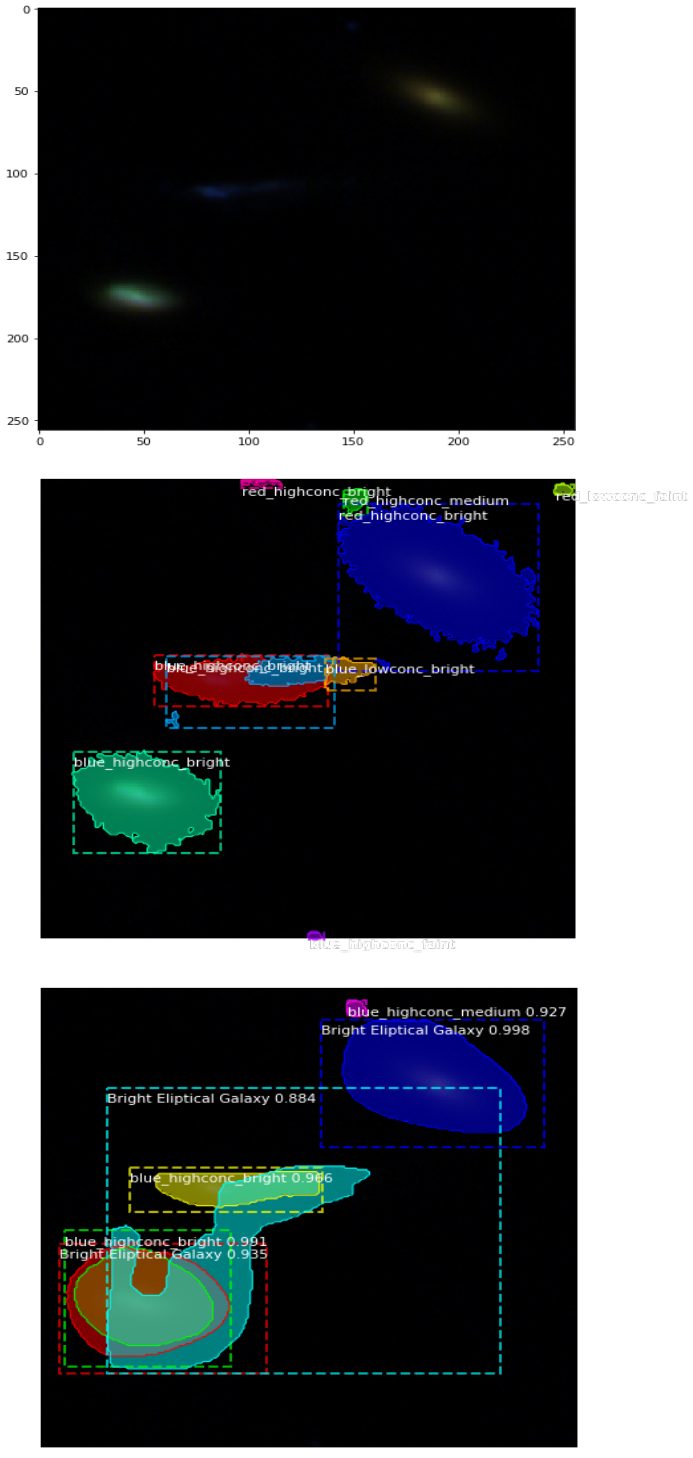

Figure 6: (Top) Image is the test image used consistently throughout the project. This has been processed with parameters $m = 0$, $s = 0.5$, and $Q = 8$. (Middle) Image is the respective SExtractor masks, bounding boxes, and classifications overlaying the test image. This is the image all models will be compared to for visual analysis of performance. (Bottom) Image shows the output masks, bounding boxes, classifications and corresponding scores from our standard model. Compared to the SExtractor image, the model performs relatively well visually, identifying, at the very least, the three main regions of interest in the image.

2.6.2 Errors

Errors on our results of mAP and classless mAP highlight how precise our measurements are. As explained in 2.4, the mAP parameter is calculated from an average over 150 random images from our validation set. To obtain an error on this result, we run the mAP calculation 10 times and take the mean value. This becomes our stated result for each varied parameter for both mAP and classless mAP. To obtain the error on this result we first calculate standard deviation, measuring the spread of the data about the mean value. From this, we divide the standard deviation by the square root of the number of repeats to get the standard error on the mean. This standard error is the error we quote on each result. For each parameter variation, we take 10 repeats of each mAP and classless mAP result and quote the mean value as our result and calculate its respective standard error.

2.6.3 Pre-processing Parameters

We initialised our model with data converted from FITS to RGB. The Lupton function has pre-set parameters which are recommended for galaxy images. We decided to vary the conversion parameters, minimum, stretch and Q to see how the clarity of image impacts the performance of detection and classification. We

varied the minimum of the data from a scale of 0 to 0.15 in increments of 0.05. Following this, we set the minimum back to its preset value of 0 and varied the stretch parameter about the preset value of 0.5, ranging from 0.1 to 0.9 in increments of 0.2. Finally, we varied the Q parameter from 0 to 32 in increments of 8 focused about the preset parameter of 8.

2.6.4 Model Parameters

The type of backbone we used was ResNet101. The other compatible backbone with this model is ResNet50. This is a smaller backbone with only 50 layers compared to ResNet101's 101 layers. We ran our standard model with the ResNet50 backbone to see again how this affected the performance of the model. Using the mAP to measure the performance of our model, this parameter takes into account the number of objects detected validated using the SExtractor model, as well as how accurate the classification was. From this, we decided to vary the number of classes used in our model from a single object class to a combination of all thirteen classes. Before we varied this, we understood that the optimal mAP value would come from having a single class since this will always give the correct class output. However we wanted to analyse if increasing the number of classes affected the performance of the mAP from our model. We took combinations of classes from each of the three parameter groups varying the number of classes by excluding the number of parameters classified each run.

The number of epochs in the model defines how many times the model cycles through the whole training dataset. This consists of a number of steps per epoch which defines the batch size of the model. We looked to vary the number of epochs at a consistent 300 steps per epoch to see how this affected the performance of training in our model.

Variable Change	mAP	Classless mAP	n overlaps
StdModel	0.1927 ±0.0042	0.3294 ±0.0072	3
m = 0.05	0.0137 ±0.0003	0.0303 ±0.0007	1
m = 0.10	0.0108 ±0.0002	0.0111 ±0.0002	1
m = 0.15	0.0063 ±0.0001	0.0061 ±0.0001	1
s = 0.1	0.2121 ±0.0046	0.4091 ±0.0089	3
s = 0.3	0.1857 ±0.0041	0.3192 ±0.0070	3
s = 0.7	0.1232 ±0.0027	0.2574 ±0.0056	3
s = 0.9	0.1501 ±0.0033	0.2549 ±0.0056	3
Q = 0	0.1676 ±0.0037	0.3181 ±0.0070	2
Q = 16	0.1796 ±0.0039	0.3239 ±0.0071	3
Q = 24	0.1606 ±0.0035	0.3252 ±0.0071	2
Q = 32	0.1705 ±0.0037	0.3757 ±0.0082	4

Table 2: Pre-processing parameters results. The first result is from the standard model, the basis from which we compare. The results highlighted in green are parameters that outperformed the standard model. The parameters for our standard model are $m = 0$, $s = 0.5$ and $Q = 8$. The change in the minimum, m showed no improvement to the standard model (where $m = 0$). The stretch s performs best when $s = 0.1$. The arcsinh softening parameter, Q, fluctuated but performed best in classless mAP and number of good overlaps when $Q = 32$ and the standard model ($Q = 8$) performed better than any other variations of Q in terms of mAP.

3 Results

The standard model from which we compare has the following set parameters; $m = 0$, $s = 0.5$ and $Q = 8$. It includes 13 different classes, a combination of background, magnitude, concentration and colour. It runs over 100 epochs and runs on the backbone ResNet101. The results for the standard model are $mAP = 0.1927 \pm 0.0042$ and $classless\ mAP = 0.3294 \pm 0.0072$ with 3 'good' overlaps.

3.1 Pre-Processing

Making small variations to our standard model, we varied the pre-processing parameters in our dataset. Initialising new models with adjusted parameters, we ran the model and recorded the mAP, classless mAP and the number of overlap masks compared to SExtractor on a consistent test image. The results of varying these parameters are displayed in table (2)

A visual depiction of the test image processed when $m = 0.1$, and our models output detection is displayed in figure (7). A plot of how the mAP and classless mAP varies upon changes of minimum intensity mapped to black is shown in figure (8). A visual depiction of the test image processed when $s = 0.1$, and our models output detection is displayed in figure (9). For comparison, figure (10) displays the test image processed when $s = 0.9$, and the respective model detection. A plot of how mAP and classless mAP varies upon changes in the stretch operator is displayed in figure (11). A visual depiction of the test image processed when $Q = 32$, and our models output detection is displayed in figure (12). A plot of how the mAP and classless mAP varies upon changes in the *arcsinh* function Q is shown in figure (13).

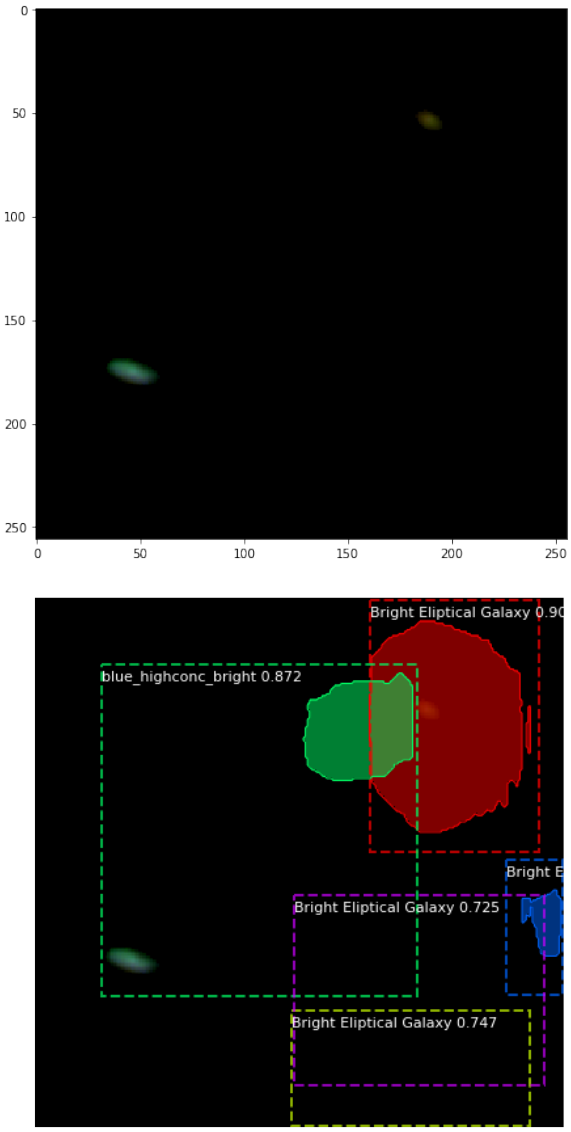

Figure 7: (Top) Image is the test image processed with a varied minimum parameter, $m = 0.1$. Analysing this image, its evident that we have lost some distinct features compared to our standard test image. (Bottom) Image is the respective model output overlaying the $m = 0.1$ image with defined masks, bounding boxes, classification and respective scores. Comparing to SExtractor, our model fails to identify 2/3 distinct regions of interest and hence results in significantly low mAPs indicating poor performance.

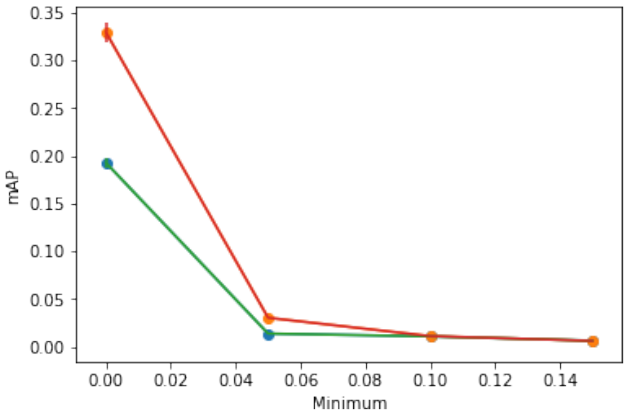

Figure 8: A plot of mAP(green) and classless mAP(red) against minimum intensity mapped to black with corresponding error bars. From both plots, it is shown that as the minimum parameter increases, the mAP and classless mAP decreases. This fits an approximate exponential decay trend.

3.2 Model Parameters

3.2.1 Number of Classes

How the variation of classes affects the performance of our model is highlighted in table (3).

Variable Change	mAP	Classless mAP	n overlaps
StdModel (13)	0.1927 ±0.0042	0.3294 ±0.0072	3
Obj (2)	0.4881 ±0.0106	0.5088 ±0.0112	3
Conc (3)	0.2689 ±0.0059	0.4202 ±0.0092	2
Colour (3)	0.3643 ±0.0080	0.4394 ±0.0096	1
Mag (4)	0.3296 ±0.0072	0.3603 ±0.0079	4
ColMag (7)	0.2867 ±0.0063	0.4330 ±0.0095	4
ConcMag (7)	0.2079 ±0.0046	0.3516 ±0.0077	3
ColConc (5)	0.1944 ±0.0043	0.3676 ±0.0081	2

Table 3: Results of varying the number of classes used. The number following the variable change in parenthesis give the number of classes used in that model, for example Obj(2) is comprised of 2 classes, background and object. Refer to table (1) for more details on the combinations. The standard model has 13 classes which includes a combination of all magnitude, concentration and colour classes plus background. There is a clear correlation indicating that the less classes used, the more precise model. One particularly interesting results is the poor performance of models containing the concentration class.

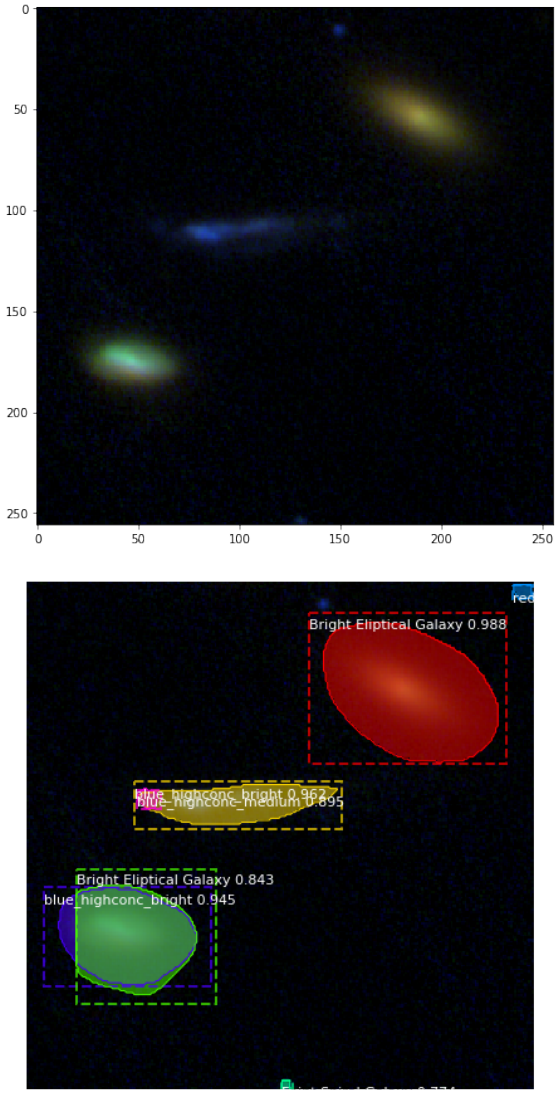

Figure 9: (Top) Image is the test image processed with a varied stretch parameter, $s = 0.1$. Analysing this image, We can see that the 3 distinct regions of interest are more defined compared to the standard test image. (Bottom) Image is the respective model output overlaying the $s = 0.1$ image with defined masks, bounding boxes, classification and respective scores. Comparing to SExtractor, our model performs relatively well, correctly masking all 3 main regions of interest with IoUs above 50%.

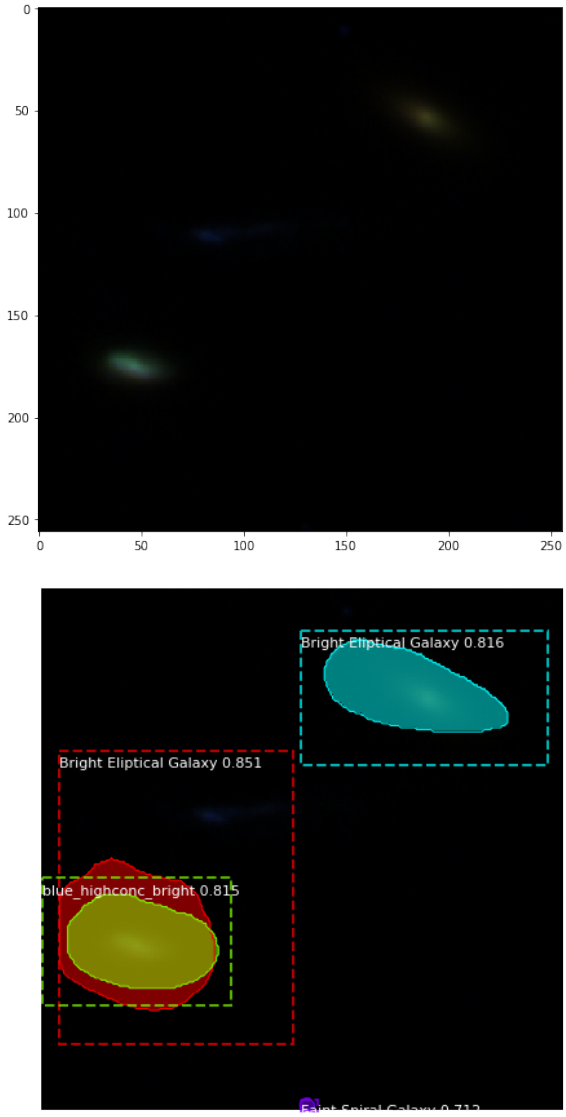

Figure 10: (Top) Image is the test image processed with a varied minimum parameter, s=0.9. Analysing this image, we can see that the image loses clarity compared to a lower stretch parameter, and we almost entirely lose the the regions of interest located in the centre of the image, becoming indistinguishable to background. (Bottom) Image is the respective model output overlaying the s=0.9 image with defined masks, bounding boxes, classification and respective scores. Comparing to SExtractor, our model manages to highlight 2/3 main regions of interest however, fails to identify the centre regions of interest.

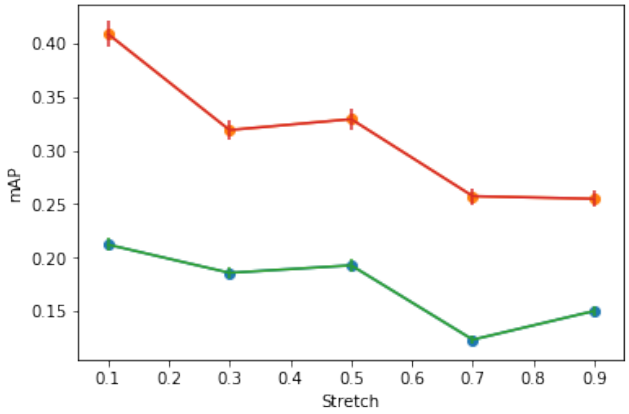

Figure 11: A plot of mAP(green) and classless mAP(red) as a function of the stretch operator with error bars. From both plots, its shown that as the linear stretch increases, both mAP values decrease. This fits an approximate negative linear gradient.

A visual depiction of how our model performed upon variation of number of classes is displayed in figure (14). Here we focus on the best performing model which has 2 classifications, and two models which performed significantly worse than other models with similar number of classes.

3.2.2 Number of Epochs

Results of how the number of epochs in training effects the performance of the model are displayed in table (4). As explained in (2.3.3), the training stage is divided into two sections, 'heads' and 'all' layers respectively. Therefore in each stage, the number of epochs is half the total number of epochs displayed in the results table.
A plot of how mAP and classless mAP varies upon changes in the number of epochs is highlighted in figure (15). A visual depiction of how our model performed upon variation of number of epochs in training is displayed in figure (16). Here we focus on instances where the number of epochs in training is 40 since this is the lowest performing model. We also focus on the instance where the number of epochs in training is 160 since we see a significant drop in mAP values from this model.

3.2.3 Backbone

Results of how the type of backbone effects the performance of the model is displayed in table (5). We also display the average time it takes for each epoch to complete in training.

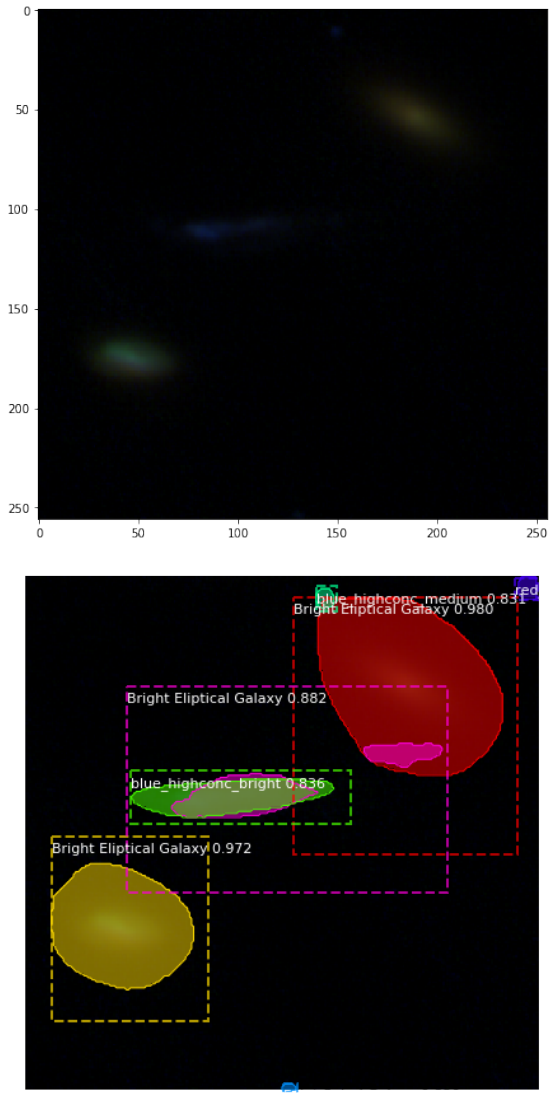

Figure 12: (Top) Image is the test image processed with a varied *arcsinh* parameter, $Q = 32$. Analysing this image, we see that the 3 regions of interest appear slightly fainter compared to the standard test image regions of interest. (Bottom) Image is the respective model output overlaying the $Q = 32$ image with defined masks, bounding boxes, classification and respective scores. Comparing to SExtractor, our model performs well, correctly detecting the 3 main regions of interest aswell as picking out some smaller regions of interest similarly to that found in SExtractor.

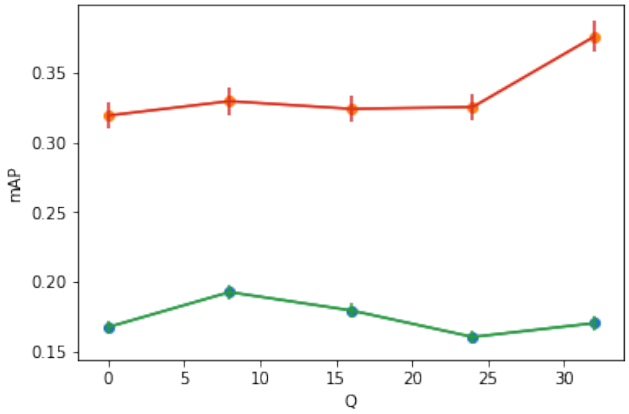

Figure 13: A plot of mAP(green) and classless mAP(red) against the *arcsinh* function Q with corresponding error bars. Plot(red) of classless mAP remains approximately constant as Q increases, aside the case where $Q = 32$. Plot(green) of mAP remains approximately constant, aside the case of the standard model ($Q = 8$). Overall, the value of Q shows to not alter the performance of the model.

Variable Change	mAP	Classless mAP	n overlaps
StdModel (Epochs=100)	0.1927 ±0.0042	0.3294 ±0.0072	3
Epochs=40	0.1328 ±0.0029	0.2649 ±0.0058	2
Epochs=60	0.1435 ±0.0031	0.2702 ±0.0059	2
Epochs=80	0.1730 ±0.0038	0.3112 ±0.0068	3
Epochs=120	0.1957 ±0.0043	0.3214 ±0.0070	3
Epochs=140	0.1925 ±0.0042	0.3297 ±0.0072	3
Epochs=160	0.1357 ±0.0030	0.2514 ±0.0055	2

Table 4: Results of varying the number of Epochs. The results highlighted in green are parameters that outperformed the standard model. The standard model contains 100 epochs. When the number of epochs = 120 the model outperforms the standard in terms of mAP. When the number of epochs = 140, the model outperforms the standard in terms of classless mAP.

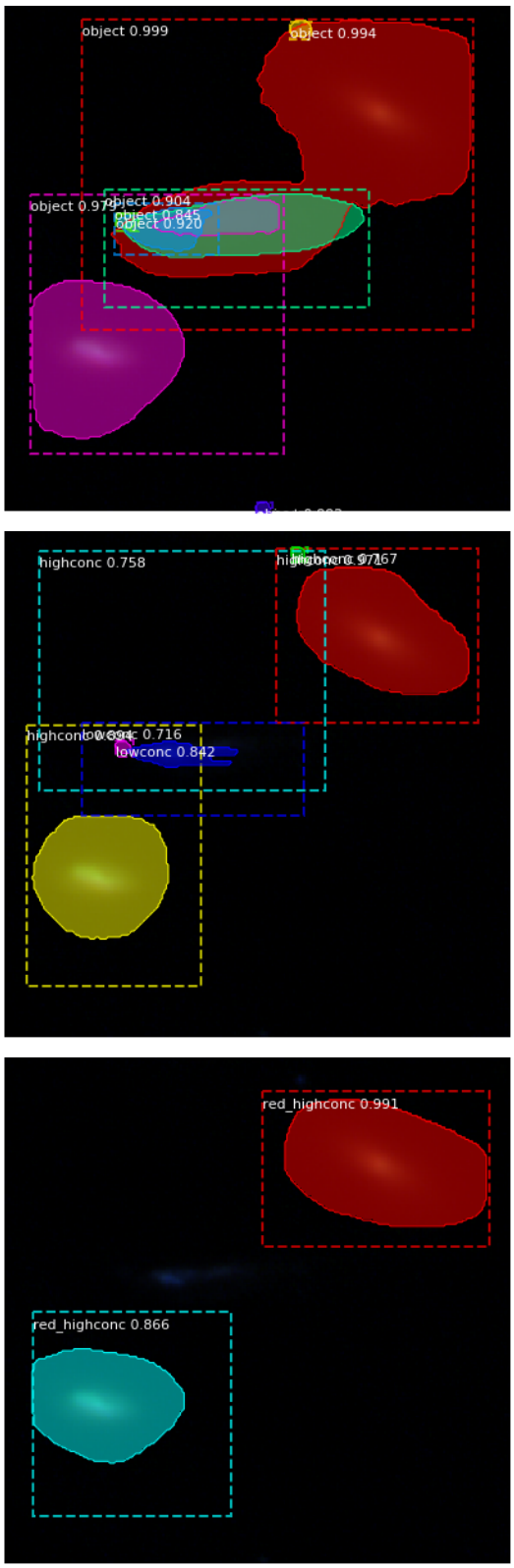

Figure 14: (Top) Image highlights the output of our model for 2 classification groups - Object and Background. This performs well, correctly detecting the main regions of interest, however it struggles to differentiate between galaxies, merging two regions of interest into one mask. (Middle) Image highlights the output of our model for 3 classifications - Concentration and Background. Visually, this performs relatively well, detecting the main regions of interest. (Bottom) Image highlights the output of our model for 5 classifications - Colour, Concentration and Background. This struggles to detect many objects, only picking out 2 out of 3 of the main regions of interest.

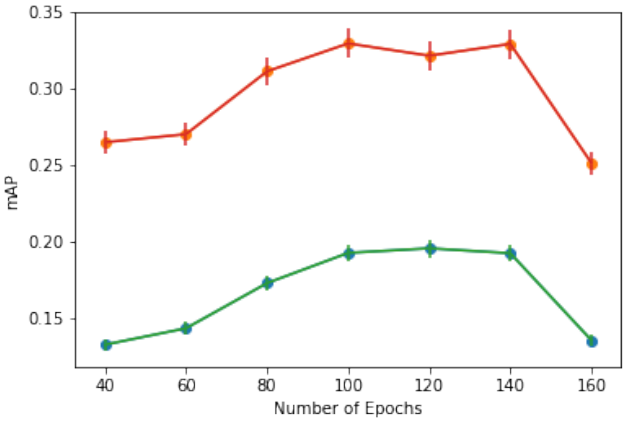

Figure 15: A plot of mAP(green) and classless mAP(red) as a function of number of epochs with corresponding error bars. From the plot, it is shown that as the number of epochs increases, the mAPs increase up to approximately 140 epochs. Increasing the number of epochs further than this causes the mAPs to decrease. From the theory on deep learning, we hypothesise that this decrease in performance is due to over fitting in the training stage of the model.

Variable Change	mAP	Classless mAP	Training time (s)
StdModel (ResNet101)	0.1927 ±0.0042	0.3294 ±0.0072	85(H) 134(A)
ResNet50	0.1913 ±0.0042	0.3158 ±0.0069	74(H) 99(A)

Table 5: Results of changing the backbone. Training time is given in seconds per epoch and is for the (H) - head layers and (A) - all layers of the model. Differences in training time are almost negligible for all the models using ResNet101, but in experimenting with ResNet50 it immediately became clear that the training time was a major difference. This decrease in training time (head layers 11 seconds per epoch faster, all layers 35 seconds faster per epoch) however, also had the effect of decreasing performance (mAP decreased 0.0014, classless mAP decreased 0.0134). Considering the errors, these results may not be relevant.

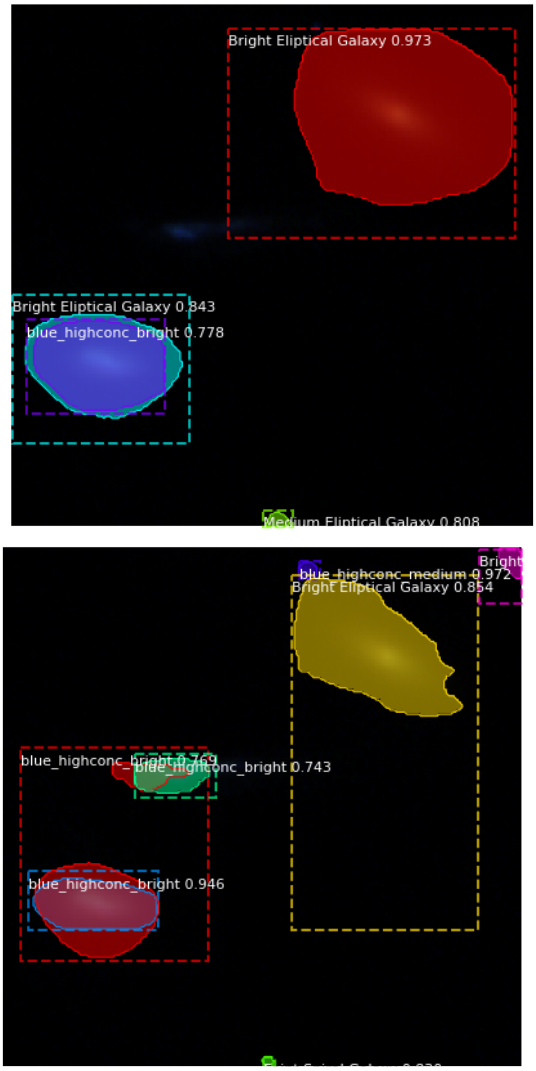

Figure 16: (Top) Image highlights the output of our model when trained in 40 epochs. We can see that the model performs relatively well, given that it has only seen the training data a reduced amount. It manages to detect 2/3 main ROIs as well as picking out some smaller ROIs. (Bottom) Image highlights the output of our model when trained on 160 epochs. Given that the model has seen the training data 160 times, we expect it to perform extremely well, however, aside detecting the 3 main ROIs, it shows signs of overfitting given the sizes of the bounding boxes defined around the masks.

4 Discussion

4.1 Assessing results

4.1.1 Pre-processing

From table (2) it can be clearly seen that two variable changes outperform our standard model (highlighted green) while the other changes performed less well than the standard. The parameters for our standard model are $m = 0$, $s = 0.5$ and $Q = 8$. When $s = 0.1$, the model outperformed the standard in terms of both mAP and classless mAP.

Analysing the variation of the minimum intensity mapped to black, m, it is shown that upon small increment increases the performance of the model falls tangentially to zero. Visually, the clarity of the images as the minimum increases becomes less defined and

we lose some of the key features such as the intensity profile. It is shown from these results that an increase in minimum value leads to an exponential decline in detection performance in both mAP and classless mAP.

Upon variation of the stretch operation, s, we notice that a smaller stretch leads to an improvement in performance of mAP and classless mAP. Visually, when the stretch is 0.1, the clarity of the image is improved and the galaxies defined in the image are more visible with larger intensity profiles. From these results, an approximate negative linear gradient could be fitted to the plot seen in 11, displaying that as the stretch increases, the mAP and classless mAP decrease.

Varying the value of the arcsinh function Q, it is shown that the values of mAP and classless mAP remain consistent and therefore we can justify that a variation of Q doesn't affect the performance of the model within the respective errors of the results. Visual inspection shows that an increase in Q decreases the brightness of the galaxies contained in the image up to a threshold of approximately 24, at which point the brightness appears to remain consistent upon increasing Q any further.

4.1.2 Number of classes

Analysing table (3) we see a clear trend that shows as the number of classes defined in the model decreases, the mAP and classless mAP increase. However, it is worth noting that the classification group 'concentration' performed significantly lower than other classifications with similar number of classes. We hypothesise that the reason the concentration performs significantly lower is due to SExtractor. The concentration parameter describes the size ratio between the main region of the galaxy and the apparent maximum dimension of the galaxy. In our far-field dataset, we have galaxies that are relatively close to each other, and therefore predict that SExtractor had trouble distinguishing between neighbouring galaxies, classifying them as a single galaxy instead. The mAP classless mAP values imply that the more classes there are, the worse the precision. However, for the n overlaps metric it is not so distinct, with the 7 class colour magnitude combination retrieving joint most 'good' overlaps with the 4 class magnitude only model. N overlaps gives a good indication of the models' object detection abilities. This then implies that the models with more classes are just as good in terms of object detection, but are less precise in classifying the objects, thus bringing down the mAP.

4.1.3 Number of epochs

Increasing the number of epochs increased the performance of our model up to a plateau mAP of 0.1957 and classless mAP of 0.3297 at around 140 epochs. Beyond this number of epochs, the model's performance significantly declines. We predict from visual inspection of the proposed masks and bounding box regions in figure (16) that the model is overfitting. This implies that our model has learned the training data too well that it has negatively impacted how it performs when proposing masks, bounding boxes, classification IDs, and scores on the standard test image.

4.1.4 Backbone

ResNet50 consists of 50 layers compared to ResNet101, used in the standard model, so it is expected that it will take less time to run. The use of ResNet50 reduced the training time in the head layers by 11 seconds per epoch on average and 35 seconds per epoch on average when training all layers. The trade-off is, however, that the mAP and therefore the performance drops; the mAP was down 0.0014 from the standard model (using ResNet101) and the classless mAP down 0.0134 on the standard model. Considering the error, these results may not be significant. With more time, the potential trade-off between speed and performance using the ResNet50 backbone would be investigated further, considering optimisation and errors.

4.1.5 Comparing Results to Literature

From Gonzalez et al.[12] a similar method is presented for automatic detection and classification of galaxies. Using similar performance evaluation methods, the results they present show that their model achieved mAP values of 0.6812, 0.6852, 0.6579 for datasets containing 55050, 22020 and 32290 images respectively. Our highest performing model achieved an mAP of 0.4881 for a dataset of 7867 images. Comparing our results, its clear that their model significantly outperformed our model. We believe this is because as part of their data processing method, they added a data augmentation stage. We predict that by utilising data augmentation, we could come close to achieving mAP values in the region Gonzalez et al. reach.

4.1.6 Errors

It is important to note then that the ground truth from which we train our model may still carry over some computational error. Although robust, SExtractor is not completely accurate. It is hard to quantify the error SExtractor carries across as a whole but individual elements can be assessed. For example, the error on the magnitude class set can be calculated using, as given by [22],

$$\Delta m = 1.0857 \frac{\sqrt{A\sigma^2 + \frac{F}{g}}}{F} \tag{6}$$

where A is the area (in pixels) over which the total flux F (in ADU) is summed, σ the standard deviation of noise (in ADU) estimated from the background, and g the detector gain (GAIN parameter, in e/ADU). This may be quantifiable, but other factors involved are less easy to compute and so including this error is difficult.

Human-labelled images, rather than SExtractor, would be better as a ground truth in the interest of discarding this error.

4.1.7 What we would do with more time

If the project were to continue in the long term, an area that we would look to be improved is optimisation. In this case, where large amounts of data are processed, reducing the models' run time, as well as simplifying the code, can significantly increase the rate at which you can collect results and therefore use said results to improve performance. One example that could be utilised in the future to improve run time is the backbone; our results show that ResNet50 reduced run time significantly, as seen in table (5) and discussed in 4.1.4.

In pre-processing, our notebook splits up our HST datasets into smaller crops, with an added function to only keep crops which have a minimum of 1 object mask in. We could add to this function by introducing overlaps in our crops. This would essentially introduce images with the same object masks in them, but in different positions in the image (data augmentation). This would increase the size of both the training and validation training sets and it is hypothesised to improve the performance of the model. With an increased data size, we would expect the run time of our model to increase. We would also vary the number of images in our data set by increasing the intersection of overlap in pre-processing to see how this affects the performance of our model. We predict that an increased data size will improve the performance of our model, but at the expense of a longer run time, so some sort of compromise would need to be made.

Also within pre-processing, Lupton et al.[15] describe a method that can bring out more detail in dust lanes of spiral galaxies. They find that a $\sqrt{asinh}$ stretch function can have this effect. With more time, a model tailored only to spiral galaxies using this adjusted stretch function may have an interesting affect on performance.

It would be interesting to see how adapting architectures such as YOLO and Detectron perform when trained on our data. Both YOLO and Detectron are huge networks with implementations of several architectures including Mask-RCNN. We would expect them to outperform our model, although it would be useful to observe the level of performance and how long it would take to train a model using these architectures in accordance with similar configurations as our model.

When calculating the mAP for each model, we used a Mask R-CNN function that takes in a given number of random crops from the dataset. Due to some unsolved computational issue, we were limited to 150 random crops per mAP calculation. If we increased the number of crops above this the kernel would frequently die, resetting all outputs in the model. We would ideally investigate this further and resolve the issue, since increasing the number of images over which the AP is averaged would give a more accurate value of mAP and reduce its associated error.

Gonzalez et al. [12] highlight a potential improvement for current CNN models. The leading models only take in images with three colour channels, an example being mask R-CNN used in this research. Models in the future should look to expand the colour band possibilities which, while increasing the memory required, may lead to higher precision as more bands allow for more specific feature recognition. The paper also highlights the potential that data augmentation has in improving performance.

5 Conclusions

We have shown how some parameters can affect the performance of a machine learning based object detection and classification algorithm. Changing pre-processing parameters generally decreased performance compared to the standard model, but a number of tweaks led to a higher level of performance. Setting the stretch to $s = 0.1$ and the arcsinh softening parameter to $Q = 32$ were particularly beneficial changes. We also found that decreasing the number of classes used in a model actually improved performance, although this may be to do with the metric used for evaluation, mAP, and its encapsulation of the precision of classification. We found that reducing the backbone size only had a minor effect on performance and should be pursued in the future. The best performing model we produced achieved an mAP of 0.4881 ± 0.0106.
We have explored the future of CNNs and discussed potential methods for improvements. For example, optimisation methods such as using smaller backbones such as ResNet50 and increasing the colour band used for the data might improve feature detection.

Acknowledgments

Firstly we would like to thank the University of Nottingham for providing the facilities to enable us to complete this project. We would like to give special thanks to our supervisor Steven Bamford for his dedicated support and commitment in sometimes mammoth meetings. We would also like to extend thanks to our families and friends for their continued support.

References

[1] Matsugu, Mori, Mitari, and Kaneda, Subject independent facial expression recognition with robust face detection using a convolutional neural network., *Neural Networks* **16**, 555 (2003).

[2] Hubel and Wiesel, Receptive fields and functional architecture of monkey striate cortex, *J Physiol* **195(1)**, 215 (1968).

[3] Hubel and Wiesel, Brain and visual perception the

story of a 25-year collaboration., *Oxford University Press* **729pp** (2004).

[4] Hubel and Wiesel, Receptive fields of single neurones in the cat's striate cortex., *J Physiol* **148(3)**, 574 (1959).

[5] K. Simon, *Neural Networks - A comprehensive foundation* (Prentice-Hall, 1998).

[6] D. Scherer, A. Muller, and S. Behnke, in *20th International Conference on Artificial Neural Networks (ICANN)* (Thessaloniki, Greece, 2010).

[7] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, Focal loss for dense object detection, arXiv:1708.02002 (2017).

[8] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, arXiv:1512.03385v1 (2015).

[9] J. Redmond and A. Farhadi, Yolo9000: Better, faster, stronger, arXiv:1612.08242v1 (2016).

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, Ssd: Single shot multibox detector, arXiv:1512.02325v5 (2016).

[11] J. Redmond and A. Farhadi, Yolov3: An incremental improvement, arXiv:1804.02767v1 (2018).

[12] R. Gonzalez, R. Munoz, and C. Hernandez, Galaxy detection and identification using deep learning and data augmentation, *Astronomy and Computing* **25**, 103 (2018).

[13] S. Ackermann, K. Schawinksi, C. Zhang, A. K. Weigel, and M. D. Turp, Using transfer learning to detect galaxy mergers, arXiv:1805.10289v1 (2018).

[14] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks, *0302-9743* (2016).

[15] R. Lupton, M. R. Blanton, G. Fekete, D. W. Hogg, W. OMullane, A. Szalay, and N. Wherry, Preparing red-green-blue (rgb) images from ccd data, arXiv:astro-ph/0312483v1 (2003).

[16] R. H. Lupton, J. E. Gunn, and A. Szalay, *AJ* **118** (1999).

[17] K. He, G. Gkioxari, P. Dollar, and R. Girshick, Mask r-cnn, arXiv:1703.06870v3 (2018).

[18] T. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollar, Microsoft coco: Common objects in context, arXiv:1405.0312v3 (2015).

[19] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval* (McGraw-Hill, New York, NY, USA, 1986).

[20] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, The pascal visual object classes (voc) challenge, *International Journal of Computer Vision* (2009).

[21] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun, Object detection networks on convolutional feature maps, arXiv:1504.06066v2 (2016).

[22] E. Bertin, *SExtractor v2.13 User's manual* (Institut dAstrophysique, Observatoire de Paris, Paris, France, -).

[23] K. Schawinski, C. Lintott, D. Thomas, M. Sarzi, D. Andreescu, S. Bamford, S. Kaviraj, S. Khochfar, K. Land, P. Murray, et al., Galaxy zoo: A sample of blue early-type galaxies at low redshift., *Monthly Notices of the Royal Astronomical Society.* **396**, 818 (2009).