**Ed Roberts**

**Project Notebook - Use Machine Learning on JWST images to find early galaxies**

**Preface**

­This is my day-to-day rough notebook of everything I’ve done in the project. There are no doubt many spelling mistakes and typos as I haven’t read it all back. Throughout the project, I also grew to take more detailed notes the first few pages a much briefer than the later pages.

**Questions**

-28/10/22. Q (@Sandro) - Still have an issue with generate\_images.py will not load in email so assume we need another way of sending. Once this is done can we run through the generation process.

A – Set up Slack channel which shared resources

-28/10/22 Q (@Sandro) – Could we set up how I can access the JWSP images and quick demo of how the current recognition works process works?

A – Yes showed me website and DS9 (<https://ceers.github.io/dr05.html#nircam-imaging>)

-28/10/22 Q (@Pietro) – How can I connect to GPU for training? Not really important now but for next term when I’m doing the coding

A -

-29/10/22 Q(@sandro) – What the perfect system would look like (just a reminder to me to discuss on Tuesday)

-13/11/22 Q(@sandro) – Can we go into more detail regarding the segmentation maps.

-13/11/22 Q(Self note) – We can generate training data with definitions of locations of points but what about segmentation maps?

-22/11/22 Q(Sandro) – SNR histograms matching

-27/11/22 Q(Sandro) – What referencing form should I use? Particularly in the physical context section how often should I be referencing the literature?

- 27/11/22 Q(Sandro) – When was the cosmic reionisation period? I’ve seen a few different z values? Around z = 30? The galaxies we’re looking for, do they belong in this reionisation period? Is this the main physical context I should Be looking at?

Check Referencing is correct>>particularly the et al and format

!!!NOTE WE FOUND IT WAS MUCH EASIER TO ASK QUESTIONS ON SLACK SO FROM HERE ON QUESTIONS WERE ASKED ON THE SLACK CHANNEL AND NOT RECORDED HERE!!!

**Project Log**

**28/10/22**

**Reading about CNNs**

- Computer vision with PyTorch book: chapter 1 Building Blocks of Computer Vision

-Convolution and kernels: A kernel is essentially a feature extractor. The kernel moves around the image in a stride, s, and performs a matrix dot product on the pixels it covers. Padding is when extra 0 pixels are added to edges of image so that information is not lost at edges of image from convolution, it allows us to keep dimensions constant after convolution has taken place.

-Receptive fields (RF) <https://theaisummer.com/receptive-field/> and <https://arxiv.org/pdf/1701.04128.pdf>: RF is essentially the region of the image one unit of the neural network can ‘see’. Need to optimise receptive field size to specific problem so that it covers the entire relevant image region. Can increase RF linearly by adding more layers (every layer increases RF by kernel size). Sub-sampling increases RF multiplicity. Local RF is the layer-wise information strength carried forward by the convolutions of the kernel. Global RF is the cumulative information being carried to the last layer. WE want the last layer to have seen the whole image. 3x3 is the minimum RF that can capture up/down left/right

- Pooling: reduces the dimensionality of the problem. Like convolution layer but doesn’t perform convolution instead could pick max pixel value in given regions (max pooling). Other types of pooling include average pooling or global average pooling

-General rule of thumb in images > global information resolves what, while local information resolves where

**29/10/22**

**Reading about CNNs**

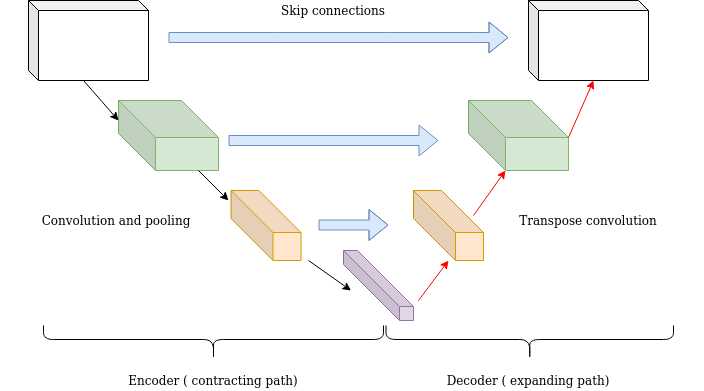
-skip connections (<https://theaisummer.com/skip-connections/>)

- These, as the name suggests, links non-adjacent layers together which can improve CNN performance i.e. they skip some layers in the neural network and feeds the output of one layer as the input of the next. Provide an alternative path for gradient (in backpropagation)

- ResNet: skip connections via addition. Some information captured in the first layers could be useful for later layers to also learn from so skip connections provide a way for them to do this; without skip connections this “information would have turned too abstract”. Need to ensure dimensionality of addition layer is same

- DenseNet: skip connections via concatenation. Allows maximum information flow between layers by connecting all layers directly to each other via concatenation.

- UNets: long skip connections. Long skip connections often exist in architectures where spatial dimensionality is reduced in the encoded part and gradually increased in the decoder part. Often used in image segmentation > so could useful in my project. Transpose convolution layers can be used to increase dimensionality. (see image below from <https://theaisummer.com/skip-connections/>)



-Types of Architecture

-AlexNet - first big jump (2012), Used the ReLu activation function and tries to use multiple GPUs to paralise training

-VGG (<https://arxiv.org/pdf/1409.1556.pdf>) classification network. Has 16 convolutional layers and 3 fully connected layers. Was the improvement on AlexNet

- ResNet – introduced addition skip connections to solve the problem of zero gradient in backpropagation

- Inception Architetures: Factorise convolutions. Essentially making network wider instead of deeper by using multiple kernels on the same level. Factorising comes in as you can add a 5x5 kernel as 2 layered 3x3s but the extend to 1xn and nx1. Also introduces auxiliary classifiers to deal with vanishing gradient problem.

**Reading Astro machine learning papers**

30/10/2022 - DEEPSOURCE: point source detection using deep learning

-Used a simple 5 layer CNN to increase signal to noise then use simple blob detection to mark the galaxies

- Concluded DEEPSOURCE was better than standard python blob detection algorithm

**8/11/2022 – Meeting with Sandro**

-Discussed how the image generation process works > how to run the file and change different configs

-Learnt about .fits format and how to open the files in DS9. Fits file (Flexible Image Transport system). Header contains lots of important information. Each pixel has 2^32 levels of numbers which is a lot bigger than jpeg which are 2^8> so saving to jpeg looks awful. Fits files reader is good at going to WCS (world coordinate system) ie to not get distortion.

-Opened a F444W image from JWST (<https://ceers.github.io/dr05.html#nircam-imaging>)

-Task for the week is to look into noise of JWST > essentially try and make the training images as close to the real images as possible. I need to plot image stats about from JWST images etc and try match.

-Astropy library > useful for manipulating .fits files

-Images will also need to be normalised > JWST has max pixel value of 1 but much higher in training data

**9/11/2022 – Downloading DS9 and image playing**

* Having discussed how to access the images and how to generate images with Sandro I spent time downloading the software to my computer and having a play around myself
* Looked at the generating image code and got to grips with it
* Learnt a crucial next step will be to generate training data that is as close to the real images as possible > ie need to find noise spectrums of real images and try match
* Open multiple image layers with ‘Open as > Multiple extension frames’

**11/11/2022 – NIRCam reading**

* Read about the NIRCam (Near Infrared Camera) <https://jwst-docs.stsci.edu/jwst-near-infrared-camera>
* ten 2K × 2K HgCdTe detectors: eight for 0.6–2.3 µm observations (0.031"/pixel) and two for 2.4–5.0 µm (0.063"/pixel)
* Each detector contains 2048 × 2048 pixels; the interior 2040 × 2040 pixels are light-sensitive photodiodes, and the outer 4-pixel wide border consists of reference pixels used to measure temperature and bias voltage drifts during exposures
* When imaging the sky, the short wavelength pixels have twice the spatial resolution (0.031″) as the long wavelength pixels (0.063")
* 29 filters of NIRCam have names “Fnnnx” > “nnn” refers to central wavelength of filter in µm > “x’ refers to filter width, “W2” for extra-wide (R~1), “W” for wide (R~4), “M” for medium (R~10) and “N” for narrowband (R~100) [Note R = λ/Δλ]
* <https://experts.arizona.edu/en/publications/overview-of-james-webb-space-telescope-and-nircams-role>
* Diagram

  Description automatically generatedSensitivity > <https://jwst-docs.stsci.edu/jwst-near-infrared-camera/nircam-performance/nircam-sensitivity>
* PSF <https://jwst-docs.stsci.edu/jwst-near-infrared-camera/nircam-performance/nircam-point-spread-functions> - each filter have predicted FWHM between 0.03" and 0.16" assuming nominal mirror wavefront errors. The NIRCam detectors achieve Nyquist sampling or better (FWHM > 2 pixels) above 2 µm in the short wavelength (SW) channel (0.6–2.3 µm) and above 4 µm in the long wavelength (LW) channel (2.4–5.0 µm). Below these wavelengths, the PSF is under sampled.

**12/11/2022 – Read paper “Morpheus: A Deep Learning Framework for the Pixel-level Analysis of Astronomical Image Data”**

* Uses a U-Net CNN architecture to assign pixel level classification (Spheroid, disk, irregular, point source or background)
* Directly inputs .FITS file which will be useful
* Trained using human classification labelled Hubble images
* Code all available
* Performed segmentation using the watershed algorithm (p10)

**13/11/2022 – Playing with AstroPy**

-Opened test image .fits and extracted location of the source points

-Need to rescale the pixel values so that they are in the same range as JWST image

-**IMPORTANT NOTE IN DS9 PIXELS COUNT FROM (1,1) BUT OBVISOULY WHEN READ INTO PYTHON COUNT FROM (0,0)**

**-ALSO I DISCOVERED WHEN READING IN USING FITS.OPEN() THE PIXEL COORDINATES READ AS (Y,X) NOT (X,Y)**

**15/11/2022 – Meeting with Sandro**

-Use the SCI-BKSUB images the first one as the image photo its just the picture with the background removed.

-The ERR images is just the error in each pixel ie readout error etc all put together (this is the 3rd image)

-The rest of the layers you don’t need to worry about at the moment.

-You can see which image layer its is in the […] end part of the file name in DS9

-ERR is the total error in each flux in the pixel

-The black marks in the centre of the image are just artifacts in the image > should come up in the error but can probably be ignored for now.

-Need to start writing report: first part on Why we have built JWST, what the need for identifying galaxies and in particular high z galaxies is, why im doing what im doing.

**19/11/2022 – Noise Histograms**

- Began plotting the noise histograms for both generated images and F444W JWST images. End goal is to match distributions so that generated training data is as similar as possible to the real NIRCam images

- Opened each image using Astropy then manually selected a region of the images with no signals by viewing in DS9

- Played with different bin numbers and different region sizes ­to get a feel for the distributions

- Need to discuss a normalisation technique to get the pixel value ranges in the same range for simulated images and real images >>>Could do linear scaling but I’m worried this will squeeze a lot of data into a small range losing information as the training data generates a much larger pixel value >>> better to generate data in the right range to start with

-Jupyter notebooks NIRCam\_hist.ipynb and Opening\_fits.ipynb

-**REMEMBER PYTHON READS PIXELS FROM (0,0) BUT DS9 FROM (1,1) AND PYTHON WORKS AS (Y,X)!**

-

**25/11/22 – Reading physical context papers**

-Cosmic evolution > reionisation in particular is period of time JWST will provide new data to.

**26-28/11/22 – Writing project plan.**

**-------------------------LENT---------------------------**

**20/1/23 – Learnt about diffusion models.**

* Possibility for use in removing noise from images as they essentially work by introducing noise to clean images bit by bit and then learn how to take noise out of the noisy image bit by bit until back to clean image
* Read paper “Gated-SCNN: Gated Shape CNNs for Semantic Segmentation” which Pietro had sent. Interesting way of using CNNs for segmentation by splitting the architecture into two streams: one for shape information and one for other information like a classical CNN.

**23/1/23 – Environment and package set up**

* Before starting the heavy coding this term, I needed to set up my computer. I checked and found out I had the wrong version of anaconda on my computer (as I am using an apple computer with M1 arm64 chip not an Intel chip the PyTorch version I had was not optimised for my computer)
* I uninstalled my whole python (conda) and reinstalled the correct version, created an environment (called torch) in which I then installed all the required packages (PyTorch, astropy etc). This is common practice in computing projects

**24/1/23 – PyTorch learning**

* I decided I needed to learn more about how to buuld models in PyTorch. I had used TensorFlow before but never PyTorch so followed some online tutorials to build simple classification models etc. Also followed PyTorch lessons on their Tensors, Datasets and DataLoaders.
* It became clear to me a larger amount of time than I’d anticipated would need to be spent preparing my Dataset. Because the file format (.fits) is not a standard outside of astrophysics there is no easy way to get the datasets into PyTorch. I will look at different ways to do this, possibly using AstroPy to form a dataset. I don’t want to just compress the fits files to JPG or PNG as that would lose quality in the images and ultimately would mean my final model wouldn’t fit nicely into astro pipelines. (I just remember the ‘morpheous’ paper I read last term said it slotted straight into astropipeline so they may offer a solution).
* <https://pytorch.org/tutorials/beginner/data_loading_tutorial.html>

**25/1/23 – PyTorch learning**

* Tutorial on model building again (layers and autograd)

**26/1/23 – Sandro meeting**

* Discussed desired output: ideally a probability map of centre of each galaxy i.e. probability centre of each galaxy is in each point. Also talked about training on SNR map ie dividing the SCI\_BKSUB map by the ERR map in real data as sometimes there are noise spikes (remember SCI\_BKSUB has the homogenous background of particles in the universe subtracted not the actual error subtracted.)
* Also talked about possibility of building additional diffusion model to look at noise reduction

**-PyTorch learning**

- really good answer on why flatten in standard nn for images (not for cnn: essentially to couple vertical and horizontal information <https://www.quora.com/Why-do-we-flatten-the-data-in-neural-networks-while-processing-images>

**27/1/23 – Pietro Meeting**

* Discussed possible noise handling technique and how important matching the training data noise is to the real image data would be

**30/1/23 – Loss function**

* Spent time viewing possible loss functions for my model. nn.MSELoss() found to not be appropriate due to the variable number of point objects required to find.
* At first I thought the order of the predicted points and the actual points would cause an issue, i.e. what if first point in the predicted data (y\_0) wasn’t the closest to the first point in the target data (x\_0). I don’t think this would be a problem because as the loss function is minimised it would pull the y\_0 point to be the closest point to x\_0. So there is fundamentally not a problem however this could mean minimising the loss takes longer. NEED TO CHECK IF THIS WORKS WITH MULTIPLE IMAGES IN BATCHES AS MODEL IS FED IN LOTS OF IMAGES IN BATCHES THEN LOSS IS Calculates Loss etc>>>I think I’ve thought about this wrong, we’re not trying to minimise completely for each image. Model is given a batch of images, guesses location of sources, calculates loss (error of guess) then updates the model and moves on to new images and repeats. Therefore, it shouldn’t be thought of as a loss function to specifically work on one image until that one image is optimised. The order of the model guesses may be important because sources move between guesses (new images) so loss function will need to ensure it calculates loss by using the predicted point and the true point CORRESPONDING to that prediction. An issue may be finding such a point may be computationally expensive. Could look at putting gaussian ‘holes’ in the loss function >>> point only has contribution to loss function if within a certain region close to each point
* Three types of gradient descent:

Batch Gradient Descent. Batch Size = Size of Training Set

Stochastic Gradient Descent. Batch Size = 1

Mini-Batch Gradient Descent. 1 < Batch Size < Size of Training Set (common batch size 32,64,128 etc)

SGD usually quickest but contains noise, will eventually oscillate about minimum.

* Model weights are only updated after each batch. Epochs is the number of times the learning algorithm will work through the entire training dataset
* Learning rate indicated the step size. That gradient descent takes towards local optima. Very important learning rate is close to optima otherwise algorithm may diverge if LR is too large, or if LR is too low then will take too long to converge. Can have adaptive learning rates.

**-Examining image simulation**

- First looked at the previous script used in Mich term. Upon looking at ways to match generated images as much as possible I found MIRAGE. From initial reading it appears Mirage is specifically created for simulating JWST and NIRCam images so this may produce much better-quality training data than the Python Galsim script. Need to examine further!!!! One issue is it seems it uses source catalogues which may be a problem since the whole point of the problem is to find new galaxies so if simulating which previous knowledge the model may learn to neglect others.

<https://github.com/spacetelescope/mirage/blob/master/examples/Catalog_Generation_Tools.ipynb>

Turns out images can be made from generated catalogues.

There is a bug during installing mirage for mac OS 12.5 so I have spent significant time trying to overcome the bug. (One of the packages in mirage doesn’t build properly so installation fails)

>>>Built environment to handle mirage in (named mirage), then installed the bug libraries separately, then installed full mirage package.

**31/1/23 – Image generation**

* Spoke to Sandro about Mirage. He said the final images would require reduction, i.e., MIRAGE images would have to be reduced in the NIRCam reduction pipeline so that you get a mosaic. This would make the image production more painful and ultimately reduce the number of images we could feed to the model which would be bad. Furthermore GalSim can produce better Sersic profiles. >>>Decided to move forward with GalSim instead of MIRAGE.
* Sandro also mentioned guitarra, another image generator. Guitarra is even more complicated but does have a full mosaic with a truth map we would be an interesting test of how good the Galsim trained model worked.

**-Loss Function**

- Came across focal loss, a very powerful loss function for object detection. <https://amaarora.github.io/2020/06/29/FocalLoss.html> <https://ieeexplore.ieee.org/document/8237586> (((((T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection,” in The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2980–2988.)))))

- Used in paper Point proposal network (<https://arxiv.org/pdf/2008.02093.pdf>) with excellent description of loss function (page 3). A two part loss function is used: confidence ground truth is 1 for origins which are within a radius r\_near from point source and zero if there is no point source close enough.

There is then also a regression part of the loss function which is the normalised offset from the origin to the nearest point source and is only calculated if confidence ground truth is 1.

HOWEVER there is still the issue that this loss function may become too computationally expensive when there are many sources in the images. Also there is the issue that some galaxies are much bigger in the image so the choice of r\_min could be varied and sometimes overlap.

Paper looks at computational speed times which could be an interesting experiment > ie how does it scale with pixel w x h and also density of galaxies.

**1/2/23 - Scripting python code to run in batches**

* Writing .sh script to run generation code many times to creat lots of training data.
* Issue encountered: for some reason the when ./ the script the output dir stings are not added correctly in the python script (the parts of the filename are added by over writing the beginning of the line instead of concatenating on the end). Strangely this problem doesn’t occur when the python call is done directly in the terminal. I believe its to do with hidden \r being in the script call however have spent a lot of time trying to debug but cannot find reason…..
* Bug solved!!! Firstly I had used ¡ not ! in my bash line (very annoying) and also there was a hidden ^M in by #!/bin/sh line.
* Ended the day with significant progress, had a working generative python program that I could call from a shell script to make 100s of images with noise, galaxy density etc easily controlled with a .yml file. All the generated images are neatly outputted into a fit directory with the location of each images centre outputted in appropriately named csv files.

**2/2/23 – Data processing and creating Dataset class**

* Wrote pre-processing code which reads all fits files, select appropriate layer of fits files then compiles all images into one 3d numpy array which was saved separately as an easily read .npy file.
* Had a meeting with Sandro to discuss various image simulator. Decided to stick with GalSim.
* Thought about the best way to handle the source centre data; because there are a varied number of sources in each image there is no easy numpy or csv data structure which would work (i.e. csv needs same number of columns for each row). It is possible to put NAN values into the arrays however I opted for the easier option of simply keeping all the data in separate appropriately named csv file. These can easily be read when required.
* Coded the dataset class required for reading in and combining the image data and the centres data into one dataset.
* Explored appropriate transforms for normalisation. Opted to normalise pixel values to a mean of 0 and std of 1. The other option was to normalise by dividing by the largest pixel value however I read online that usually the first mentioned method works better. This can be changed in the future if required.

**3/2/23 – Normalising/standardising pixel values**

* Encountered an issue regarding scaling of pixel values for training. Two options are either scale so that values are between 0-1 or scale such that mean is 0 and std is 1. I first coded the latter and plotted histograms before and after of the pixel values. The distribution looked rather strange with bands around certain values. I then opened a real JWST image and looked at histograms of its pixel values. The max value in JWST (around 10-100) is much less than max value in the generated images (1000-10000). This is due to the way in which the images are created, ie starting with noise of one across image and then scaling by a PowerLaw SNR ratio to get the pixel value at each point. Ideally, I want the real images to have a similar pixel value to the generated images so that a trained network is more likely to work well on real images. I need to look at adjusting the power law min and max to explore getting a good SNR ratio.
* Need to explore appropriate ways to scales, either to the 0-1 normalisation or to the gaussian with mean 0 and std 1. My instinct is the gaussian will be better because come sources are very faint and would become squeezed very close to all the noise in the simple scaling form. NEED TO WORK OUT IF SCALING TO 0 MEAN AND 1 STD IS OK IF IMAGE ISN’T GAUSSIAN, I.E. THE IMAGES HAVE A LONG THIN TAIL SO IS IT OK TO JUST MOVE TO GAUSSIAN???
* Played with DataLoader. Discovered DataLoader requires all elements of a batch to have the same shape>> this is an issue for the centres since each image has a varied number.
* Solved DataLoader varied number of centres using customised collate\_fn.
* Also realised a varied number of with a loss function is also going to form issues so need to look int this more…focal loss maybe?

**4/2/23 – Summary of where I’m at**

* Have python code which can simulate images along with .sh script to automate production of n number of images.
* Have a pre-processing pipeline which reads all fits images and selects appropriate layer (image layer) then compiles all images in a dataset into a single .npy file
* Have a custom dataset class and data loader suitable for loading data into PyTorch models. This is a very key step!

**Things I still need to do**

* Choose appropriate properties of training images to feed network: ie noise spectrum, density galaxies etc.
* Decide appropriate standardisation of pixel values>most probably mean 0 std 1 (I have coded this but haven’t decided whether I’m sure this is the best method to move forward with)
* Design model architecture (transformer or CNN?)

(IMPORTANT NOTE ON MODELS: CANNOT USE PRETRAINED MODEL SINCE THOSE MODELS DO NOT TRAIN WITH DATA THAT IS LIKE THE GALAXY DATA I AM LOOKING AT >> MY MODEL WILL NEED TO BE TRAINED FROM SCRATCH)

* Design loss function
* Train model and optimise hyperparameters
* Evaluate model (completeness, purity)
* Possibly compare to other models
* Possibly use model on real images

**Reading on object detection models**

DETR:

* Object detection algorithm. Has a CNN backbone which produces feature maps. These features are flattened and fed into transformer encoder. Transformer decoder separately has N input vectors (for N bounding boxes) and the flattened encoded images data is fed into the decoder as a side input. The N output of transformer decoder are then mapped to bounding boxes by FFN (Feed Forward Network).
* Uses Bipartite match loss to match the predicted boxes to the true objects. Padding on no object values to ensure both sets same dimension so there’s a one-to-one relation. Bipartite match loss could be useful for my project as images have varied number of objects.
* <https://www.youtube.com/watch?v=T35ba_VXkMY>

Vision Transformers (ViT):

* <https://viso.ai/deep-learning/vision-transformer-vit/>
* Becoming competitive to CNNs, however unless dataset is very large (order million images) ResNet or EfficientNet (maybe Fast R-CNN) are a better option. Article noted CNNs are easier to optimise than ViT.
* Key is multi-head self-attention which essentially allows ViT to connect part of images at any scale. Self-attention mechanism lets inputs interact with each other (“self’) and find out who they should pay more attention to (“attention”) <https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a>
* <https://viso.ai/deep-learning/object-detection/>

Graphical user interface, application

Description automatically generated

* <https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=9123&context=libphilprac> “One-stage detectors have high inference speeds and two-stage detectors have high localization and recognition accuracy.” >> Hence two stage detectors probably most relevant for this project. Remember though these models are for conventional bounding box detection not just centre detection. Also, since classification of boxes isn’t required, the process is inherently only the first step of the two-step process.

**7/2/23 – Researching architectures**

* **Point Proposal Network: Accelerating Point Source Detection Through Deep Learning** [**https://arxiv.org/pdf/2008.02093.pdf**](https://arxiv.org/pdf/2008.02093.pdf)
* Used a dropout rate of 20% (remember dropout is a method for preventing overfitting by randomly leaving out 20% of units in the neural network during training).
* ResNet-31 architecture to extract features, then two separate networks to extract confidence and regression values
* Output shape of feature maps determines number of sources that can be found in each region. Could train using images of 512x512 images input, then need to find shape of base layer output feature maps that accounts for enough origins. Eg PPN uses 7x7xk so there are 49 origins. Need to create images with similar galaxy density.
* Uses focal loss function for one part > focal loss ensures learning of hard to classify examples
* Used Adams optimiser (extension of stochastic gradient descent)

**- Reading on Faster R-CNN and RPN**

**- Reading on bound box anchors (**[**https://towardsdatascience.com/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9**](https://towardsdatascience.com/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9)**)**

- Essentially works by creating loads of “anchor boxes” for each predictor. These all have different location, shape, size etc. Then calculates IOU (Intersection over union) if IOU >50% then says should detect object. Otherwise, if greater than 40%, n.n. says its ambiguous and don’t use for learning. If IOU is less than 40% then anchor box should predict no object.

- Anchor box configurations need to be specialised, i.e. not to specific otherwise the n.n. may never see some objects.

- In general should ask yourself: 1) what is the smallest box I want to be able to detect? 2) What is the largest size box I want to be able to detect? 3) What are the shapes the box can take?

- Another option is to learn anchor box configuration <https://arxiv.org/abs/1807.00980>

- Can test by encoding ground truth boxes and then decoding them as though they were predictions. Should recover ground truth boxes.

**8/2/23 – ResNet Base layer**

* ResNet is a commonly used base of a model used for feature extraction. Would be sensible to use this in my model.
* <https://blog.paperspace.com/writing-resnet-from-scratch-in-pytorch/> how to build ResNet in PyTorch.
* Looked through PPN GitHub code on how to they built their base layers. They used tensorflow not pytorch
* Read how to build ResNets in Pytorch

**9/2/23 – PPN paper**

* Continued gaining a deeper understanding of the PPN model and looking at their source code on GitHub.
* To summarise: Has loads of origin anchors which are put in and then a regression values is output which gives an offset from a given origin to a predicted detected point centre. Along with a regression value for each origin anchor, a confidence output is also outputted which gives a confidence that a predicted point is actually a point.
* Loss function is combination of regression and confidence values, with focal loss used for confidence values which ensure difficult to detect galaxies aren’t ignored.
* Base layer ResNet-31 which extracts features from an image. Feature maps fed into two separate networks, a regression network, and a confidence network. Output of regression is a m x n x 2 matrix which contains m\*n origins with the x2 dimension being the dx, dy values of the offset from the origin anchor to the predicted point. The confidence network output a m x n x 1 matrix, with each values being a confidence score for that origin.
* One problem is that there are many duplicate proposals. Multiple origins may propose the same point and duplicates from patching of larger images. Duplicate removal was done using Non-maximum suppression (NMS) which takes the most confident point an removes all other points within a radius r\_NMS. Confidence scores below c\_NMS are also removed.
* Can only detect max of n\*m galaxies so in my images need to ensure origin density is large enough that all galaxies can be seen.
* Despite using focal loss to try and ensure model didn’t just focus on easy sources, they found the model was optimised when gamma = 0 ie loss became equivalent to a normal weighted binary cross-entropy. I could use this as a starting point in my project.
* Also found r\_near = r\_far as when r\_far > r\_near they found the number of false positives increased.

**10/2/23 – Begin building model!**

* Quick refresher on anchors first <https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html> and how they are much quicker than sliding window techniques. Note for my project I’m choosing to use similar anchors to the PPN model, i.e. not bounding boxes but a lattice of points each with an offset values and confidence values to a detected point.
* Began building model. Started with anchor creators, adapted code from PPN GitHub

**13/2/23 – Building ResNet base layers**

* Built a standard base layer residual block in PyTorch then the full ResNet31 architecture. This can be a starting point for my model > will need to play around with base layer structure etc and look at replacing final layers with transformer like DETR but this can at least be a starting point.

**14/2/23 – Building loss function**

* Began exploring focal loss and how to build in python
* Completed the anchor functions required to get labels for conf and reg truth maps

**15/2/23 – Building model**

* Decided to just start with binary corrs entropy loss for the confidence part as paper set their focal loss to be equal to this in hyperparameter section.
* Remember nn.CrossEntropyLoss() takes logits not normalised values so don’t sigmoid before applying this loss.
* Also BCEWithLogitsLoss takes logits so don’t need to sigmoid either.

**Table

Description automatically generated**

**16/2/23 – Lecture and supervision work**

**17/2/23 – Testing loss function**

* Had a few issues with loss function CrossEntopryLoss, moved to using binary cross entropy because the classification task is binary (0 for no galaxy and 1 for galaxy within radius).
* Coded the total loss function combining the regression and confidence parts

**18-20/2/23 – Lecture and supervision work**

**21/2/23 – Gluing model together**

* Began assembling all the part of my first model I’d built together. This took longer than anticipated as had to make sure all names etc matched and retesting each function to ensure I hadn’t messed up combining implementation.
* Also, I had used a different convention for the center corrdiants (x,y) and (y,x) in different sections of my model so had to fix this. In hindsight I could have built this using one convention to begin with however I had pulled different resources from various places which each used different conventions.
* Had an issue with the Dataset I had built> loading was too slow. I was just loading the data in a batch with image pixel values and then a tensor with pixel (x,y) positions of the centres. Once loaded in the centre values I then used the get\_anchors and get\_anchor\_label functions to assign the confidence and regression values of each origin anchor. This proved to be too slow in the model so I decided to rethink this

**22/2/23 – Redesigning dataset**

* At first, I tried to solve this issue by changing the centre location files to ‘.npy’ files instead of ‘.csv’ files. ‘.npy’ files are read quicker by python. I wanted to keep the generation script making csv files, however, because they are quickly readable by humans, which makes testing sections of my code easier. Once I’d implemented this in the pre-processing I ran a speed test and it turn out this method was slower!!! Time for another rethink!!!
* I next decided to move the anchor labelling into pre-processing. This way all the computation time spent on labelling wouldn’t have to be repeated each epoch of training. This should produce big improvements. I changed my gal\_ds class so that the \_\_getitem\_\_method reads and returns npy files of the image pixel values, the confidence anchor ground truths and the regression ground truth values. This no longer needed a collate function, which I had earlier coded to account for the varied size of arrays with different number of galaxies in each image, because each image has the same number of anchors. All the necessary labelled anchor data is now done in my pre-processing step, producing npy files the model then reads.
* When testing my new method it showed extreme improvements, as expected, with the read time of a batch of order < 1 second.

**23/2/23 – More error fixes**

* The next error I encountered when putting my model together was a dimension error of my model. The input data I was providing was missing a dimension eg it was like (a,b,c) but convolution layer required (a,1,b,c). Fixed by reshaping image array in pre-processing.
* Also had to fix type of data eg to float32 not double
* Next there was a new issue with the loss function. The mask I was providing for the regression part (i.e., to only do regression on points that have a source) wasn’t working. The .expand(batch, dim, dim, 2) seemed not to work at this higher dimension. Played around with .unsqueeze(-1) and possible other fixes however didn’t fix completely yet.
* On examining the JWST images, 224x224 maybe too small as will require 20 patches which may be computationally expensive. Perhaps look at 448x448 patches. Also, the galaxy density in per 224x224 images is much lower than the training data density I had produced thus far. Will need to decrease this. I had been looking at the images from a larger scale so obviously saw more galaxies. This is easily fixed when generating training data. Also works as a plus as effectively means less anchors are needed so feature layer can be smaller dimension (remember 7x7 mean max 49 galaxies can be detected in patch. I think I’ll work with 14x14 or 28x28 depending on input size as still need room in anchor separation to do regression)

**24/2/23 – Finding cause of bugs and errors**

* Fixed the mask issue with .unsqueeze(-1).expand(-1,-1,-1,2) however then ran into a new issue. The shapes of my ground truth data and output data of the model are the wrong shapes. I.e. model outputs shape (batch,2,7,7) but I had made the labelled data ground truth of the shape (batch, 7, 7, 2). It took a large amount of time to work out this was the problem because the error message from the console was ambiguous and ill-defined >>> hence meaning I spent practically all day trying to work out what the error was.
* Seems like a very simple thing when I realised but was such a small thing, I’d overlooked it meaning I spent a long-time debugging

**27/2/23 – Fixing error**

* Having found the cause of my error I played with various solutions but settled with changing the ground truth labelled anchor data to the shape (batch, 2, 7, 7) for regression part and (batch,1,7,7) for confidence part. Used the np.transpose function for this. Could have alternatively just transposed the output from the proposal layers but thought it was cleaner keeping model output as it is.
* Having changed this shape, I then had to go back to the masking issue I’d solved a few days ago. As the shape of the regression map had now changed I had to fix the masking so that it now fitted this shape.
* Finally, after fixing many bugs, the model ran training loops! This is good progress.
* I was scared at first as loss function didn’t monotonically decrease through training epoch’s however after some research, I found this is fine and should just be looking at my required statistics (precession ad purity functions) and maybe average loss of an epoch.
* Despite being able to train the model now, I haven’t yet coded precession and purity functions for the model so cannot see its progress throughout training. I began building these. Will also need to look at using a non-maximum suppression algorithm like PPN paper to get rid of repeat proposals (points predicted multiple times)
* Also noted when transforming back from feature map positions to real predicted positions that 7x7 probably won’t be enough as each anchor can only hole data for one galaxy, and it picks this by choosing the closest galaxy. But this galaxy may not be the brightest galaxy in the region. This could lead to many issues in learning; since we want model to use image data it should not be ignoring bright parts of the image in the loss function. Could solve by increasing the feature map dimensions. Possibly look at pyramid models as well. For now, I think priority is to get precession and purity working then see how model performs then begin experimenting to optimise…

**28/2/23 – Building metrics**

* First, I coded a function to convert the feature map regression to real image coordinates regression values (multiply by the step size and add origin anchor coordinates)
* Read about non-maximum suppression (<https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c>) as I need a way to eliminate duplicate parts
* Trained model for a few epochs (10) and looked at results. BIG ISSUE > model was predicted regression values bigger than 1 i.e. beyond step size so when multiplied by step size was giving unphysical pixel coordinates e.g. like -22 for x when we obviously can’t have a negative pixel values. Could be not trained long enough (was short) but could be bigger issue which will be a problem!!!!!
* Ran with a few different batch numbers and hyperparameters to see if training was better. Didn’t seem to affect much. Then tried running for a longer time, epochs = 20 (still not a lot of epochs but this takes around order of hours on my machine (try find a GPU to use!), still got strange results with no real improvement > something is fundamentally wrong in my model… time to systematically go back through testing of each section. (Checked realistic epoch numbers needed <https://arxiv.org/abs/1811.12019> , <https://datascience.stackexchange.com/questions/46523/is-a-large-number-of-epochs-good-or-bad-idea-in-cnn> which said models with 1,000,000 images and 1000 classes can be learnt in around 35 epochs.
* Began systematically retesting my model at the loss function. I figured this is essentially the most likely place an error is occurring since it is what provides the direction model should learn
* Tested the regression part of loss function. Checked masking was working correctly and then fed it a few test tensors with known values. Checked the expected Mean-Squared-Error value came out. Regression part seemed ok
* Upon testing the model I noticed when that the output changed when I passed the same image into the model multiple times. This is very strange. After some searching found this is because I’d forgotten to set model to evaluate mode so the mode was still using dropout (where 20% of nodes ‘dropped’ to avoid overfitting in training)
* A picture containing text

  Description automatically generatedInvestigated the binary cross entropy loss function and was happy it worked as intended as well. The mask also worked. (Noted when testing that full model uses the BCEWithLogitsLoss which includes sigmoid so if you pass truth map as input it’ll sigmoid it first then give a non-zero loss even though they’re the same but this is expected as ground truth is like its already been through sigmoid function).
* Went back to the trained network and explored what confidence values it was outputting. Each anchor was essentially giving 50% (well around 52% I expected likely higher than chance because model learnt more likely than not there is a galaxy close by). These values are clearly WRONG!
* Went back to generating different data. I thought if I gave the model ‘easier’ data then its likely it should be able to learn this easier so I could see if the data is too hard to learn or if my model was messing up somehow.
* ASIDE: also found a peculiar bug in the generation code; once in about a 1000 the code fails in a non-repeatable way. This leaves gaps in data set when pre-processing. This seems like a strange error so for now I will not fix it but simply run the generation manually for the few files that are missing (this is much quicker than spending a day fixing the bug).

**1/3/23 – Trying to train model successfully**

* Decided to run for a larger number of epochs to see what happens. Ran with 2000 images for 50 epochs. Found still not guessing correct values (i.e., still 50/50 for every anchor confidence values).
* I thought perhaps my training images have too many galaxies in. Reduced number of galaxies to 15 with std of 15 in image generation. Also cut off the lower end of SNR to 50 so that the images have less noise in, theoretically making training easier.
* Next decided I needed to bite the bullet an generate a much larger dataset. Generated 20000(=79\*128) images with a relatively low galaxy density (mostly with mean 15 but a few with mean 30 std 10). This took a lot of computation time meaning I couldn’t explore much else while doing this as my computer was busy churning the images out.
* !!!Access to a GPU would be very helpful because my training is very time consuming at the moment and takes my laptop time meaning I can’t work on testing my implementation is correct!!! Will speak to Sandro about this at my meeting tomorrow.
* Once the 20000 images are generated, I will run 50 epoch training run on them and see the results of this test. I will explore results of this run; if it is still giving poor results I think I can rule out the data/epoch number not being big enough and dive deeper into my implementation.
* Had to do a bit of data renaming; since I originally coded for the max number of files to be 9999 but now want a dataset of 20000 I need to pad an extra 0 to start of first 10000 files to ensure ordering in pre-processing is correct. Did this simply using os.rename() function. Once this was complete I changed my generating python script to handle 99,999 image names.

**2/3/23 – Meeting with Sandro**

* Discussed where I’m at with my model and how the first attempt at training hadn’t worked so why I was now looking into doing with a much larger dataset with many more training epochs. Explained how my model was designed i.e. ResNet feature extractor and then proposal layers. Talked about my current training images; we agreed that is was sensible to essentially start with images that are easy to learn then once was proven model worked we can experiment with different image types.
* Spoke to Pietro and got a google colab pro account off him to try experiment with GPU.
* Spent the rest of the day examining using GPU. Found my data was too big to upload to google for colab. Looked into other ways of doing this like zipping it all up.
* Found I can apply for time on the High Performance Cluster (HPC) which has a power GPU I can use for training.

**3/3/23 – Finding GPU and experimenting with data structure**

* Asked Sandro if he was happy for me to send an application to use the HPC GPU. He was happy so I sent the application. Was approved quickly and I began looking at how to use it.
* Using the GPU may take a little while to get my head around: have to SSH to the remote computer etc then will have to set up my environment with PyTorch and stuff so this could take a while to get running
* **Important find:** This day I also looked at the most efficient way to load data. Can’t store all data on GPU, i.e. not enough room in memory. Researched best practice and found its efficient to essentially get the CPU to prepare the data and get it ready in chunks then send the chunks to the GPU one at a time. This method ensures the GPU can handle all the data while also ensuring the GPU is using all its time on tensor calculations rather than collecting data. Moving data to the GPU can be slow so want to do this efficiently otherwise the benefits of tensor calculation speeds up massively outweighed by the time spent opening and moving data about. This will mean opening all the anchors in the \_\_init\_\_\_ eg self.acnhors then loading them one by one in \_\_getitem\_\_. This is quicker because doesn’t have to open files. SHORTSTORY is don’t want to be opening items in the \_\_get\_\_ item method. AND don’t want to load unnecessary data >> just load data in the batch currently being done.
* Also found the .hdf5 file format which could work perfectly for handling my data. Will look at setting this method up instead of my massive .npy image file <https://www.pythonforthelab.com/blog/how-to-use-hdf5-files-in-python/> “One of the most exciting features of the HDF5 format is that data is read from the hard drive only when it is needed. Imagine you have a large array that doesn't fit in the available RAM” … **perfecto!** <https://towardsdatascience.com/hdf5-datasets-for-pytorch-631ff1d750f5>

**6/3/23 – Setting up cluster**

* Now that I have access to GPU on cluster I need to work out how to use it
* This is a relatively steep learning curve as I haven’t used anything like this before.
* Began by SSH into it, setting up my account and my multifactor authentication
* Next need to work out three things: how to set up my environment with PyTorch etc, how the file system works, how to transfer code from my local machine to computer (github)
* !!! “Data to be read by jobs (including software installations) and output files from jobs should be placed in RDS areas, NOT the home directory. This is because /home is NFS-based and lacks the scalability of the lustre-based RDS. I/O to /home by jobs can create global system issues. “ !!!
* “Ampere GPU nodes” = “Wilkes3”
* “nano” command sued to edit text files
* Only allowed to run one job for 12 hours and per job per user limit is 32 GPUs (a lot!)
* <https://docs.hpc.cam.ac.uk/hpc/user-guide/a100.html>
* Next I need to set up my environment, and work out how to transfer code and data from my local machine to the system. Finally need to work out how to use SLURM to run
* “The batch job script is then submitted to SLURM with the sbatch command” <https://docs.hpc.cam.ac.uk/hpc/user-guide/batch.html>

**7/3/23 – Setting up cluster**

* Read docs on setting up environments on the cluster and docs on how to submit jobs using SLURM
* Read about the filesystem on CSD3 <https://docs.hpc.cam.ac.uk/hpc/userguide/io_management.html>
* Need to put data in the “/rds” directory not the /home because /rds is a scalable file system with much bigger storages. /home is suited for code, post processed results etc.
* Set up an environment “pytorch-env” in which I pip installed torch and torch vision (used virtualenv instead of conda)
* Looked at the SLURM script “slurm\_submit.wilkes3” to start to get a feel for how to call a job on the cluster. (remember wilkes3 = ampere which is the GPU cluster)
* I also cleaned up my pre-processing code so that it was neater and all functioned properly
* Did some testing on labels to ensure they’re correct. Opened train\_00000.fits and examined its labels as well. Found labelling was mostly good however one of the points was midway between 2 anchors so had picked up two labels > NMS would get rid of this when network is trained but during training this could cause issues because model should only predict one point here (i.e. shouldn’t be predicted multiples at same point) so the this point would cause issues in loss function. Need to examine whether this will actually cause an issue in the network.
* Reproduced an image dataset: lowered the number of galaxies to be mean 14 with std 7. This makes the dataset “easier” but mean proof of concept in network model could be done. Done this because then there’s a lower chance of galaxies being close together so more likelihood model would pick up all galaxies (basically just simplifying the problem before training). Simulated 30,000 of these images.
* Need to look at redesigning anchor labelling function as when points are too close it is missing some points and double counting other (had originally taken this from the PPN github)
* Tidied up my pre-processing code to be all in functions. Also made it write to the .hdf5 file format

**8/3/23 – A very productive day**

* First, I changed the anchor labelling so that it only looped through each anchor (not each point as well) then made it possible to label the 3 closest points to each anchor. This should ensures that all galaxies are labelled; before had an issue where if there were two galaxies in a given anchors range only one was labelled (the closest) even if the more distant galaxy was the dominate bright one, this could have produced issues in training.
* I also restructured all my code, moved from Jupyter Notebooks into simply .py in VS code. Then modalized it by adding files with \_\_init\_\_ so that I could simply import the functions I required in each. This made my code easier to read
* Set up a Git repo so I could get a version control
* Substituted the ResNet I’d coded from scratch for one from the Timms library. This will mean changing my feature extractor is much easier. When testing. Ensured feature\_extracter = True so that I only get the layers I want. Using Timms its much easier to just sub in a new feature\_extractor; can simply change the name in the call function. Eg from ‘ResNet3’4 to ‘ResNet18’
* Converted my PyTorch code into PyTorch lightning (PL). PL is “a lightweight and high-performance framework that organizes PyTorch code to decouple the research from the engineering, making deep learning experiments easier to read and reproduce. It is designed to create scalable deep learning models that can easily run on distributed hardware while keeping the models hardware agnostic.”
* Using PL will make experimenting with different models and hyperparameters a lot easier
* To run the model type “python -m ppn\_stars.train “

**Notes on PyTorch Lightning**

* Lightweight wrapper useful for experimenting with models
* Don’t have to worry about doing model.train() or model.eval()
* GPU done so don’t have to worry about .to(device)
* Don’t have to worry about doing the optimizer.zero\_grad(), loss.backward() and optimizer.step() calls in training either
* Don’t have to worry about the for loops in training step
* In the trainer look at using auto\_lr\_find=True

**9/3/23 – Getting ready to start training on Wilkes HPC**

* Set up a private GitHub repo so that I can easily push and pull code to the Wilkes HPC
* Decided it was important to go back through and comment all the changes I’d made yesterday while they’re still fresh in my mind. Just so down the line I remember what I’m doing.
* Looked through the submit job template and began filling in.
* Need to set up the environment properly
* Was all ok however had an issue when trying to get PyTorch-lightning set up in my env.

**10/3/23 – More house keeping**

* Set up my SSH key so I can access GitHub, it hadn’t actually worked toher day
* My GitHub and local git had become out of sync so I had to sort this out: started a new branch, “ellen”, then pulled my GitHub to be local, then copied my missing files into new branch then pushed back to github. My working dir with GitHub set up is not StaryNight
* Tried setting up git clone on HPC but didn’t work, permissions problems, tried sorting this all out but ran into loads of ssh issues (my knowledge of ssh isn’t perfect as have never formally studied it and seems to be an issue with where I put my private key)
* For now just using scp to copy files to HPC (but github would be preferable)

**13/3/23 – Another very productive day**

* Looked up possibly using FPN network as my feature extractor: <https://pytorch.org/vision/stable/generated/torchvision.ops.FeaturePyramidNetwork.html#torchvision.ops.FeaturePyramidNetwork> Could help with my varied size of galaxies.
* Before now I hadn’t coded how to translate specific outputs from model to useful human information (e.g. so that position of predicted points were in pixel values not just regression from each anchor points), so I coded this in the prediction step function.
* Also set up wandb in my code so that loging was even easier. This helped me see visualisation a lot.
* Set up ssh git key on wilkes so I can easily pull code to HPC. Tried this once but found a small issue (was missing one .py file). Think this issue could be due to branch issues. Will see tomorrow…

**14/3/23 – Sending off first job!**

* Sorted out git and GitHub; now have SSH’s set up to push and pull from my local machine and from wilkes allowing for easy code transfer. ‘ellen’ is now the name of my working branch.

To do a transfer: 1) locally git add –all 2) git commit -m “” 3) git push origin ellen 4) Wilkes make sure in right branch git checkout -b “ellen” 5) git pull origin ellen

* Next I wrote my SLURM bacth submission script. This took a bit of trial and error to get right. I wrote my python run command in a run.sh script then. On my first runs I had a version problem, the cuda installed on GPU wasn’t the one matching my env. Took a while to find solution but fixed after finding out that when installing pytorch it automatically install a new version of nvidia cuda so simply had to “pip uninstall nvidia\_cublas\_cu11” (<https://stackoverflow.com/questions/74394695/how-does-one-fix-when-torch-cant-find-cuda-error-version-libcublaslt-so-11-no>)
* Finally got my job script working and sent an initial test job to GPUs.
* Set up wandb in my pytorch lightning code to track the loss, hyperparameters and save model etc. Now works very well and can see all the graphs in live time when job is running
* Once this was all working I sent a long, 100 epoch, run to GPU and looked at outcome on wandb. Got a decent drop in loss.
* Need an easy way to test model by feeding in image and seeing where it guesses points are!! Code tomorrow but so far productive day!
* Wrote presentation slides

**15/3/23 – Experimenting with hyperparameters**

* Sent of loads of various jobs with different hyper parameters. Started with SGD optimiser. Observed that loss reduced quite a bit at the start but then flattened off and didn’t reduce much further for all my runs.
* For main parameters performed a sweep of their values, i.e. for learning rate swept lr = 1e-2, 1e-3, 1e-4, 1e-5. Found 1e-2 was too large as loss became NAN
* Loaded some checkpoints and performed some inference. Found model wasn’t working > not finding galaxies even when loss was flattening off
* Presented my project presentation to Sandro and the group

**16/3/23 – Continuing experimenting**

* Tried using the ADAM optimiser. Started looking good, as it moved below all previous loss functions however it then became unstable and oscillated heavily. Loading the model checkpoint showed it wasn’t much better than previous models
* Implemented focal loss function and tried a test run with it
* Try experimenting with batch size
* Decided I need to simplify the problem further to see where my issues lie. I generated 10000 images with only one source in. I think, if the network can’t find one source then we have an issue in my code and I need to perform EVEN MORE TESTING.
* While my some new generation code and tests ran I decided I should retest some of my functions. Started with the data labelling !!!**FOUND AND ERROR IN THE LABELLING!!!!** This is great news as explains why model wasn’t learning. I had messed up some of the indices in the labelling so fixed this and tested new code to make sure its right. Checked a few images and found all points were now correctly labelled, some points are labelled twice which could cause an issue but worth just seeing if new labelling function works better in training now.
* Re-ran the pre-processing to get a new dataset then ran a training run to see if model now worked…

**17/3/23 – Testing fixed labelling**

* First ran into an issuewith SSH keys, seemed to have been corrupted so had to sort this out first
* Ran a training run on the single galaxy images. **Found that the confidence score were working!! Was successfully identifying the correct anchor.** This is a big step forward showing essentially the first time the model is producing anything useful
* Despite the confidence scores now working for one galaxy images, they still aren’t working to regression values. Model is just spitting out essentially zero values for regression. This could be because the regression part of loss function is scaled too small. So I increased the regression scaling by 2 and re-ran training. But got an error when trying to train. Wandb wasn’t giving a weird OS ERROR – not enough space.
* First tried fixing in the wandb error by deleting the file it quoted, this didn’t work. After some thinking I remembered the file High Performance Cluster has two parts to the file syster: “rds” and the initial user files ~. The wandb file in question was on the initial ~ dir part. I remembered reading this part of file system is capped at 50GB. Looked at the files which wandb was having trouble with and found they were small… so not the issue. But then looked at all files on this part of the system and found a massive 50GB wandb .cache file, so **issue found!** To fix this problem I read docs on the wandb artefacts cache and found “wandb artifact cache clean-up [OPTIONS] TARGET\_SIZE” can be used to clean it up. This fix worked
* During this process I had corrupted my torch virenv so had to create a new one “torch\_2” and pip install all relevant packages again.

**19/3/23 – Experimenting with hyperparameters**

* Sent a few jobs to queue to experiment with getting the regression working better. The queue was very long…
* While I waited for my jobs to go through the queue I decided to generate a new clean dataset, some files on in my dataset had become corrupt last time I tried to pre-process them, hence the Text

  Description automatically generatedneed for a new one. Used the config:
* After setting up the environment I tried running a training run. Found an error with one of my pytorch-lightning (PL) hook names > said it should be on\_validation\_epoch\_end not validation\_epoch\_end. Realised this is because I pip installed the latest PL into my new env. Decided it was probably best to just adjust my code to suit the updated PL instead of remaking env because won’t harm if my code is up-to date with new PL. (Remember a hook is just a name to given to a function which is called at a certain point in a libraries code > it just allows you to enter the library code at specific points)
* Had another issue with the new version of PL > said now optimizer idx wasn’t needed in my optimizer\_zero\_grad(). Infact I read a few docs and don’t think this function is needed at allot now so I removed it. But its worth noting it may need to be put back in again later if I discover it is needed for some reasons
* Played around with the scaling between confidence and regression scores. Found regression became slightly better when around 10 times bigger than confidence. But too large and the network learns nothing (see run honest-paper-51). I will play with this until I find a good spot for one galaxy images
* For some reason running with SGD seems to cause NAN loss values. I DON”T KNOW WHY!?!? This is weird because works fine with ADAM optimizer. Upon further experimentation this error went away on some runs but occurred ono others?!?!?! (See runs lyric-plasma-53 which worked and smart-field-55 which didn’t)
* Ran many tests varying N\_reg, batch size, learning rate, feature extractor, pretrained etc. Found in general around N\_conf:N\_reg = 0.5:1 or 0.5 showed very good confidence values but the regression part wasn’t working.
* I tried to get the egression working by increasing its ratio in the total loss but I found this didn’t work as above about 0.5:3 the whole system became unstable in training and would shoot up to very high values. Also values above this would ruin the confidence score meaning that part was completely pointless.

**20/3/23 – Experimenting to get regression working**

* Now that I found the model could correctly find anchor of single galaxy images I need to get the regression working. I’d found increasing N\_reg made no effect on the regression values. I brainstormed a few ways to get the regression working: changing the feature extractor, adding more layers to the regression proposal parts, perhaps this is because there isn’t enough data in my simple dataset (cold try full training run)
* Found hrnet was a popular newer feature extractor, so ran a training run with this. Results showed hrnet performed similarly to resnet50/101, regression values still rubbish.
* In general I found the deeper resnets worked better, 18 and 34 seemed to shallow and resnet50/resnet101 worked on the whole better.
* In my experiments yesterday I found that overall after 50 epochs batch\_size, learning rate etc had little effect on final valuation loss value. It was simplt N\_reg which had the biggest effect but this is obvious and someone unimportant because scaling N\_reg scales the value the loss function.
* Decided to try implement more training layers to the regression part. Also perhaps I need a completely sperate network to perform the regression calculation?
* Pre-processed my new large dataset (that I’d generated a few days ago) and ran a few training runs with it. Some **GREAT NEWS** but also some not so great news. The confidence scores were mostly very good, with the model identifying some very faint galaxies!!!! But the regression score of these galaxies were still poor. Slightly improved from previous runs with single galaxies but still a lot of work to do with regression. But the confidence scores were very good, for example take a look at training run ‘decent-deluge-87’, its output for the train\_00009.fits file successfully identified all 3 galaxies in the image even with the one at (220,81) being very faint. It predicted it to be at (210,80) but this is still decent.
* I’d also like to note that overall my model seems to perform better using the Timm pretrained library despite my images being vastly different from the images used to train these models. (A pretrained network is just when you start your training with weights initialised to the values found when the network was previously trained on other data, essentially transfer learning, allowing you to skip training from scratch and just fine tuning training). This technique is sometimes referred to as transfer learning. <https://www.sciencedirect.com/science/article/pii/S2666651021000231>
* Added some extra layers to the regression layer to see if this improved regression values. Found 3 layers had no effect.
* Coded Non-maximum suppression function and included it in the predict\_step function.

**21/3/23 – Further investigation into regression bug**

* Performed rigorous unit steps step by step in the pipeline of my regression values, right from the initial labelling all the way up to the output scaling. Tested each step and function. I did find a somewhat obscure result when analysing the regression loss function. I had previously set the reduction method to “mean” thinking this would be better than “sum” since then the regression value would be similar between batches (if a batch has more sources it would jump in loss function) but when looking at the numerical value of this loss function I found it was very small, too small. I decided it might be worth setting the reduction to “sum” and seeing if training performs better. The idea being that if the loss function for regression is so small it won’t contribute to the learning. I had previously been trying to fix this by scaling the loss function relative to the confidence part. In theory this should have a similar effect. The job queue was huge today so took a long time for my experiment to run…in fact I send the job around 3pm and wasn’t done by 1am when I went to bed so I worked on other parts of the model
* I coded the completeness and purity metrics. When the model is working, I will have to play around with the values of r\_tp to see what works.
* Also started planning my diss. Sketched out a structure with ideas of what to include in each section.

**22/3/23 – More hyperparameter experimenting**

* I got the results of my “sum” regression fix. The results were marginally better but still not good enough. I had ran a lot of experiments with this new loss function so took a while to analyse results. I decided to brain storm new ideas to fix my regression.
* I did a training run with only the confidence part of the loss function (I set N\_reg=0), I found the conf scores were good as expected. The values I saw in the regression matrix were actually quite large. The model hadn’t been allowed to learn ay regression. In previous experiments, with N\_reg = N\_conf, I’d found the regression values were very small so the fact they are large here suggests that the model, with N\_reg = N\_conf, was in fact learning to give very small regression values. A different shaped regression loss function like L1 not MSE might help this then!
* Another idea I had was to simply increase the lattice size of the anchors. This could easily be implemented because I’d made my code easily adaptable. I decided to try this before trying a new separate regression model completely because it was easy to implement. I changed the pre-processing code to account for extra lattice points and sent off a load of jobs to see what happens

**23/3/23 – More experimenting**

* The queue had been long yesterday again so only got my test results back in the afternoon. Found the increased lattice size worked better. Got a few more double predictions than before but this is to be expected since the points are now closer together. There is still work to do though as the accuracy is about 10 pixels at the moment which isn’t really good enough. Possibly increase the lattice size further. I sent a few jobs off to trial 28x28 (separation of 8 pixels between points).
* Also research some more loss functions. Found the Huber loss function which can be used for regression. It is a merge of L1 and L2 (MSE) with a cut off at transition. I implemented options for Huber along with standard L1 in my code. Sent off a lot of jobs experiments to see what happens.
* Found that when the reduction was set to ‘mean’ for both confidence and regression the model didn’t work great but when both were set to “sum” it worked pretty well.
* Also fixed the bug with my focal loss code. Ran experiments with focal loss. Strangely focal loss proved worse at finding galaxies, even loss intensity ones, than standard cross entropy did.

**24/3/23 – Breakthrough!**

* Analysed results of the 40 experiments I’d sent off last night. Found a lot weren’t very good but… an expreiement (“rose-tree-170”) with the Huber loss and cross entropy with 28x28 lattice of anchors seemed to work rather well!!!! There was a slight issue with sometimes predicting 2 anchors but rough estimates showed a weighted average of the neighbouring predictions worked well at finding the true point. The accuracy seemed to be within 4 pixels and looked good at finding all galaxies. I sent off a few further tweaks to see if this model could be improved but time to start analysing its and checking it wasn’t overfitted to my dataset.
* It’s time to try deploy model to perform tests. I researched the best way to do this. In the past I had had issues with using torch.save and torch.load because my model was trained on GPU with Python Cuda complied, my local machine is a Mac which doesn’t have Cuda support so I use a different Python version, this meant I couldn’t run the inferencing locally. I did some research and found the most common pipeline for general deployment was to export the PyTorch model to ONNX format then load it in Caffe2. I began implementing this.

**26/3/23 – Fine tuning**

* Now I have a pretty good model I did some extra sweeps of learning rate, batch size etc to try and fine tune the model.
* I looked at caffe2 again but ran into the same troubles I had a couple days before. Caffe2 doesn’t seem to support my mac M1 chip. Need to look at another way to deploy model.
* Upon looking at more images I noticed the regression still wasn’t amazing but noted that when a galaxy was in the middle of 2 anchors it activated both. I also noted a sort of weighted average between the two predicted points worked well at getting the proper point. I coded this weighted average

**27/3/23 – Metrics**

* Thus far I hadn’t been logging metrics as I ran because the results were pretty rubbish before this point so there was no point logging purity and completeness when the model didn’t work. I began coding these metrics
* Had difficultly in measuring the metrics because the each image has a different number of galaxies the calculations can’t be vectorised because there isn’t a set number. I thought perhaps just logging accuracy of the labels might be good enough to roughly compare model during training. Once I have an easy way to get models into production, I’ll properly log purity and completeness.

**28/3/23 – Deploying to production**

* I found there is in fact no need for caffe2 at all! I can simply use onnxruntime to run my onnx compiled model. This is great! I did a long training run of the optimised hyperparameters I’d found and saved the model as onnx. At first had an issue because onnx couldn’t read the conv2d operation with padding = ‘same’ but fixed this by just replacing same with the appropriate numerical value. I then downloaded the model and found it ran locally on my machine!!!! WOOHOO!
* I began building the inference functions around the model

**29/3/23 – Inference**

* I built the appropriate inference functions to test my model in production.
* I then coded the metrics (purity and completeness) so that I could test the model on my machine

**30/3/23 – Showing Sandro and implementing feedback**

* I gave a production model to Sandro to see what he thought. He suggested better ways to output the data but seemed pleased with the results
* I spent the rest of the day implementing his suggestions (outputting regions maps that could be shown in DS9 on top of the actual fits file)
* Cleaned up a bit of my code so everything looked nicer. All that’s left to do now is to get my statistics out and write my report!

Graphical user interface

Description automatically generated

**31/3/23 – Finished implementing Feedback**

* I changed the run command so that the name of the fits file could be included in the command line
* Also, the python code now outputs the confidence and position in a csv file and more importantly creates a regions file which can be opened directly on top of the fits file

**7/4/23 – Producing graphs for my report**

* I made a graph of purity and completeness using a varied size for r\_tp (the radius in which a galaxy centre is confirmed)
* Generated images with a set SNR ratio, need to calculate the specific SNR for images to get exact values for my plot.

**8/4/23 to 26/4/23 – Mostly doing revision for my exams**

**28/4/23 – Meeting with Sandro**

* Discussed how to calculate more accurate values for SNR for my images > need to calculate the flux in R\_half (total sum of all pixels) and divide by the noise. The noise should be calculated by adding the noise of each pixel in quadrature. This can be analytically calculated from the input q=b/a axis ratio and the single pixel SNR.
* Also discussed a bit more on why the tool is useful; at the moment the art of labelling and deblending is a very subjective so would be useful to have a more solid understanding of what to do. My project is essentially step one on seeing if ML could be used to do this more accurately and less subjectively.
* Also talked about why we want to find more galaxies at high redshifts. Essentially there are lots of models which are well-fitted at low redshifts, because there’s lots of data in this region already, but all these models predict different outcomes at high redshifts so we need to test these models to see which ones work. Properties like Cosmic star-formation rate density ρSFR, UV Luminosity Functions, stellar mass. Compare to high-resolution simulations
* Talk about Lynman alpha and how redshift is calculated

**29/4/23 – 15/5/23 Report writing.**