

# Bristol DAQ ML Studies

---

Bristol DUNE Meeting 12.3.21

Joel Greer + Raul Stein

- We have ~30k images in our SNNu dataset which we can make use of, so have set a run which trains over the entire set to try and avoid overfitting. We split this by 80%/10%/10% into train, validation and evaluation datasets
- We could also try finding a larger non-sparse dataset to use as a similar check
- We will also look at the effect of image augmentations, which can make a dataset effectively much larger, as long as it is already quite generalised. The following results are without any image augmentations.

The anchors boxes for this dataset are:

Coarse scale (1): [26,16], [39,51], [217,143]

Intermediate scale (2): [6,6], [9,12], [13,33]

Detailed scale (3): [3,2], [3,3], [4,3]

# YOLOv3 Loss

Looking into the loss function to understand why.

From each of the 3 scales of the network we get a contribution to the loss. They are currently all equally weighted.

At each scale, there are 4 contributions to the loss:

- $x,y$
- $w,h$
- objectness score
- class scores - expect this to be 0 if there is only 1 class

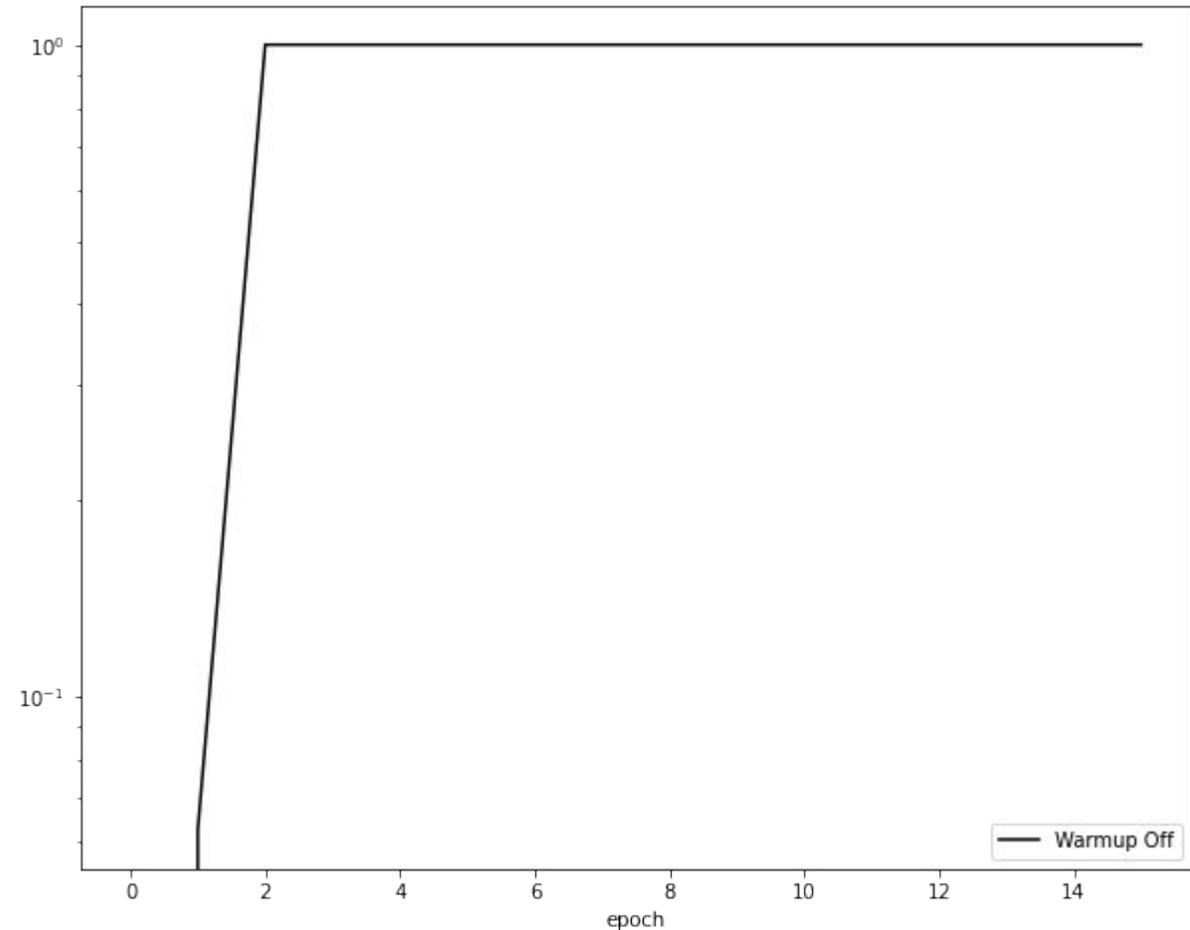
# Warmup Batches

Plot on the right shows whether the loss is calculated using a warmup batch.

(epoch 0 is first epoch)

- The value is averaged over all batches in each epoch. It is 0 if the batch uses warmup loss and 1 if the batch uses normal loss calculation

Warm-up batches are designed to let the network stabilize for the object classification problem and after that let it train on localization.

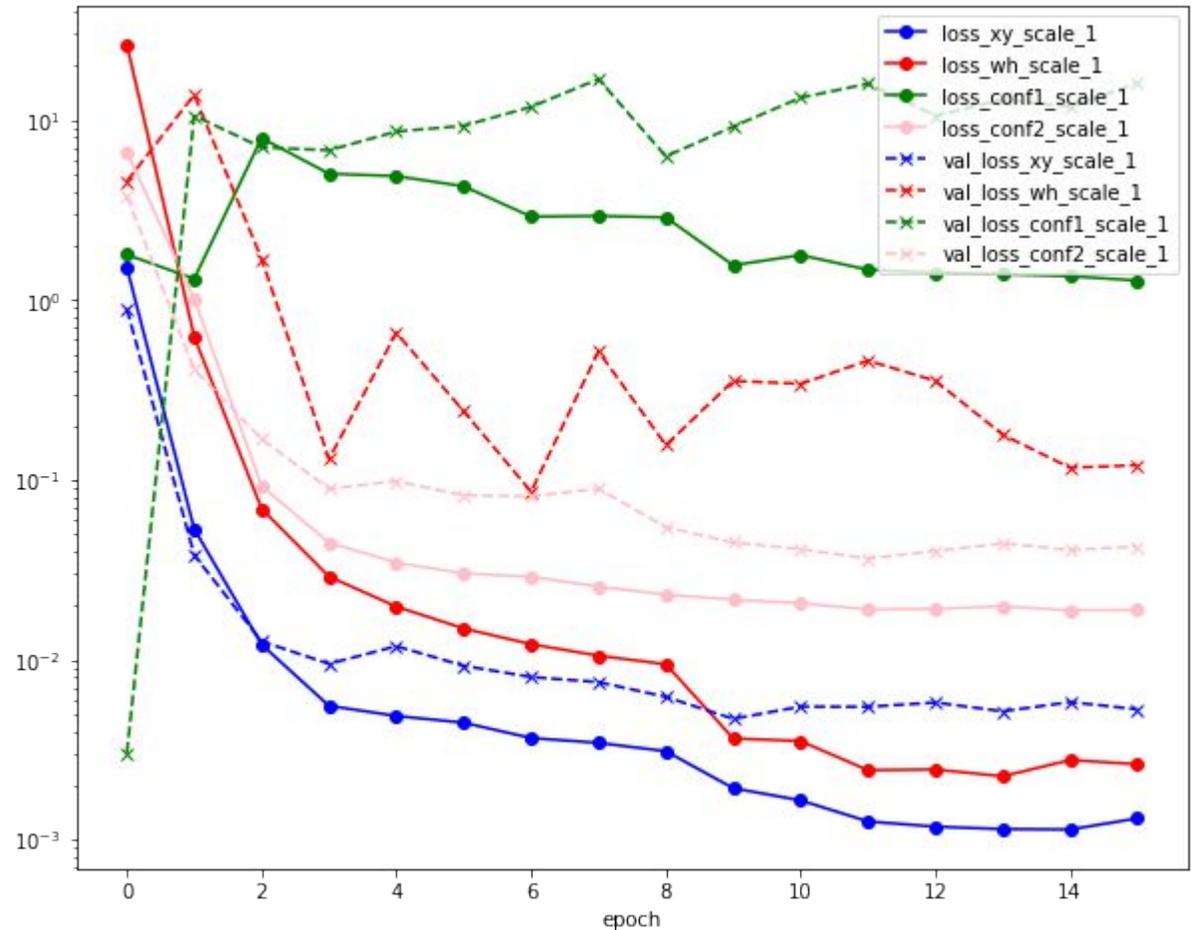


# Possible Overfitting - SNNu

Looking at the loss terms from the training set vs those from the validation set might help identify overfitting.

Looking at the contributions to the loss for the coarsest scale:

- Conf1 term doesn't optimise well for the training set
- There are also symptoms of overfitting for conf1
- W,h term also seems to be overfitting. The anchor boxes for this scale cover a huge range ([26,16], [39,51], [217,143]), so I would expect this term to be difficult to optimise for this scale
- Overfitting starts to occur for the conf2 and the x,y loss around the 4th epoch - a couple of epochs after warmup



# Investigate Loss - Part 1

## conf term

Also, separate out the contributions from predicted boxes with associated truth from those without before applying MSE cost function:

```
conf_delta1 = object_mask * (true_box_conf - pred_box_conf) * self.obj_scale
conf_delta2 = (1 - object_mask) * conf_delta * self.noobj_scale
```

So that:

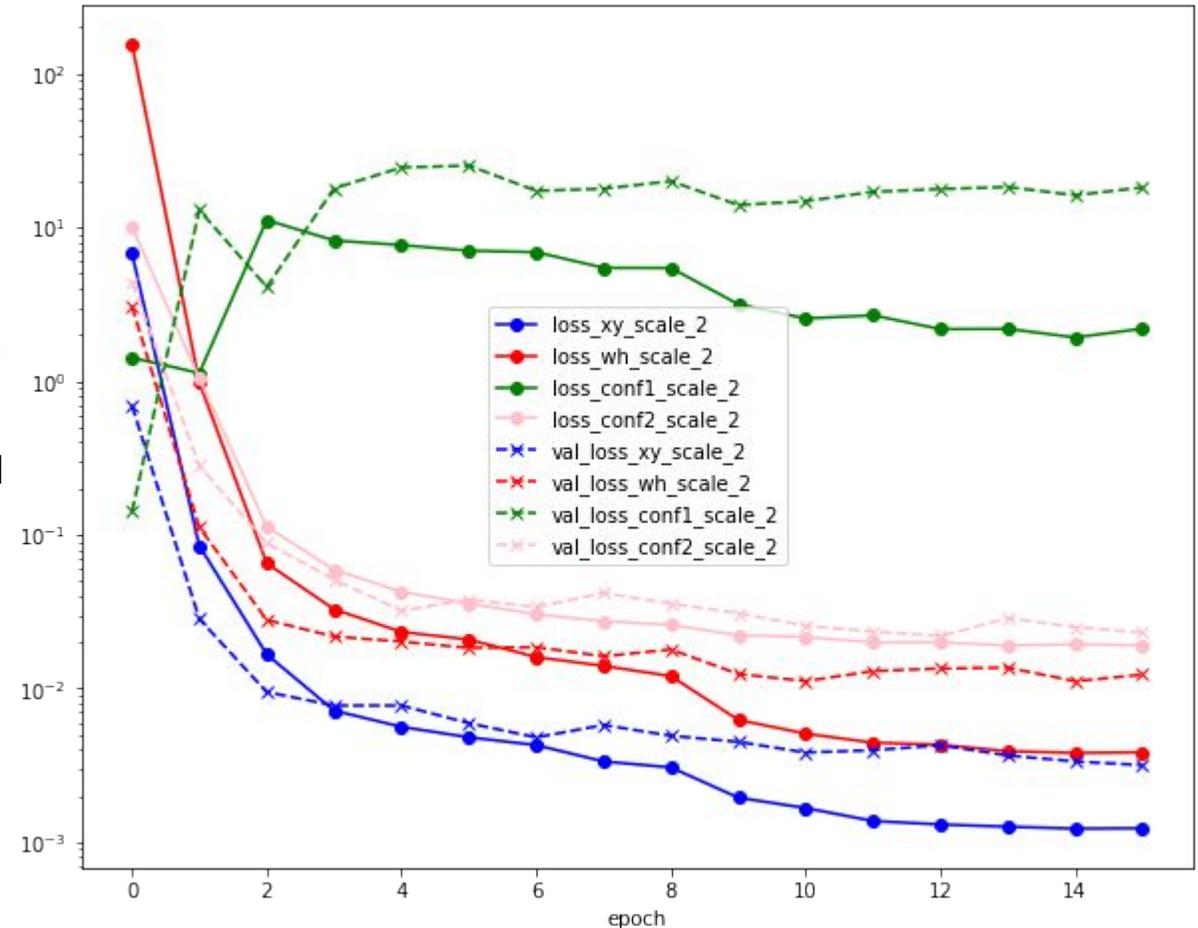
```
loss_conf1 = tf.reduce_sum(tf.square(conf_delta1), list(range(1, 5)))
loss_conf2 = tf.reduce_sum(tf.square(conf_delta2), list(range(1, 5)))
loss = loss_xy + loss_wh + loss_conf1 + loss_conf2 + loss_class
```

and compare these contributions

# Possible Overfitting - SNNu

Contributions to the loss for the intermediate scale:

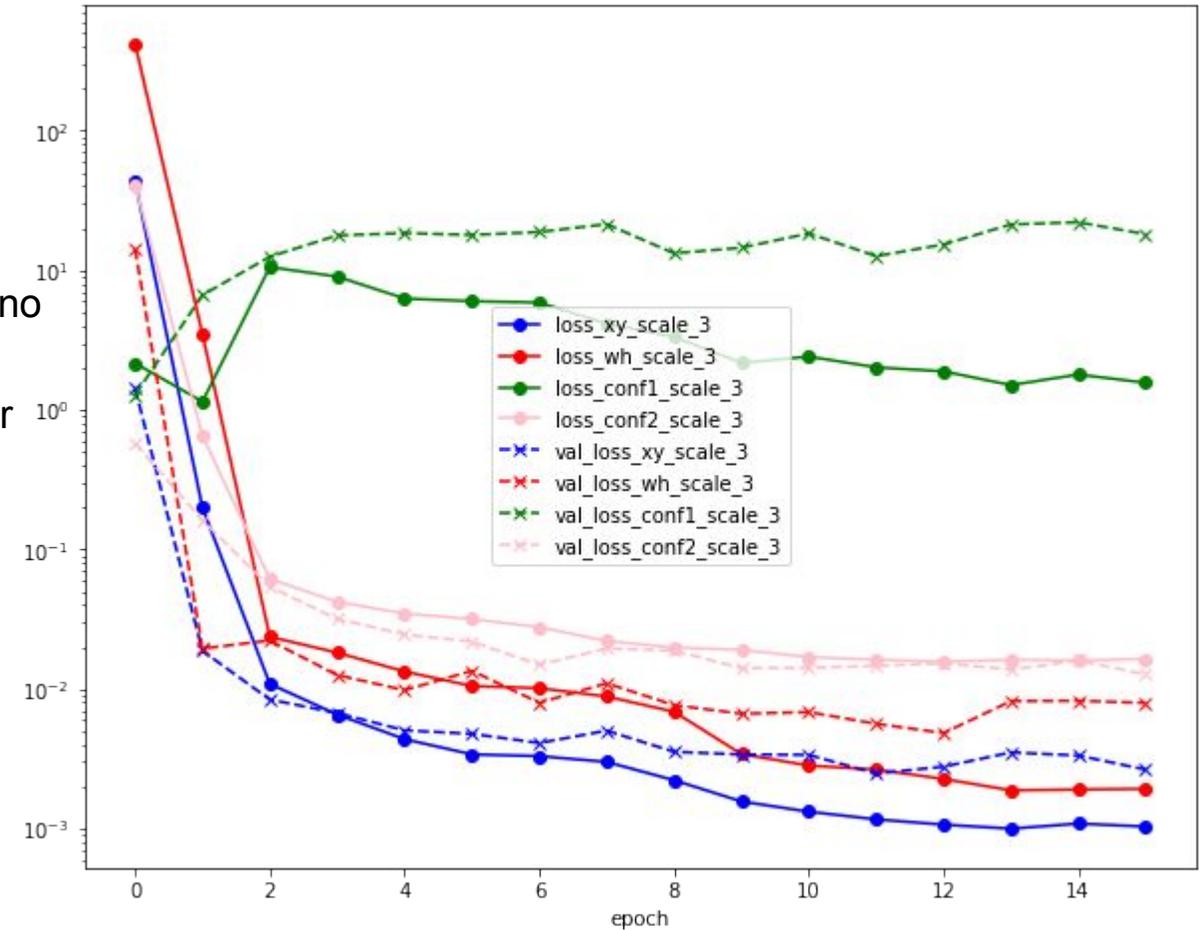
- Network again fails to optimise conf1 term well, although there is also possible overfitting for it
- All the other terms seem to optimise well, particularly conf2
- We see some overfitting after epoch 8 for the x,y and w,h terms



# Possible Overfitting - SNNu

Contributions to the loss for the detailed scale:

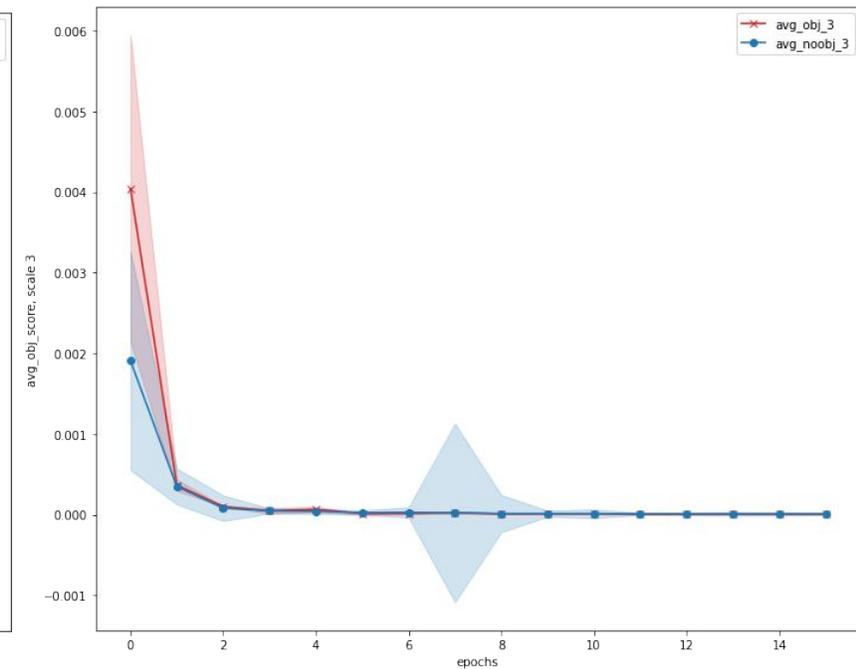
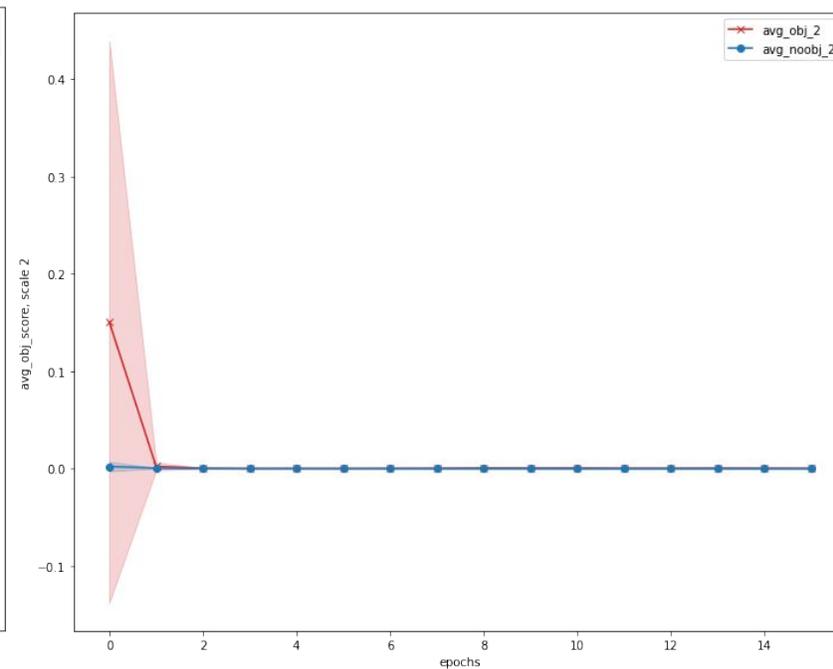
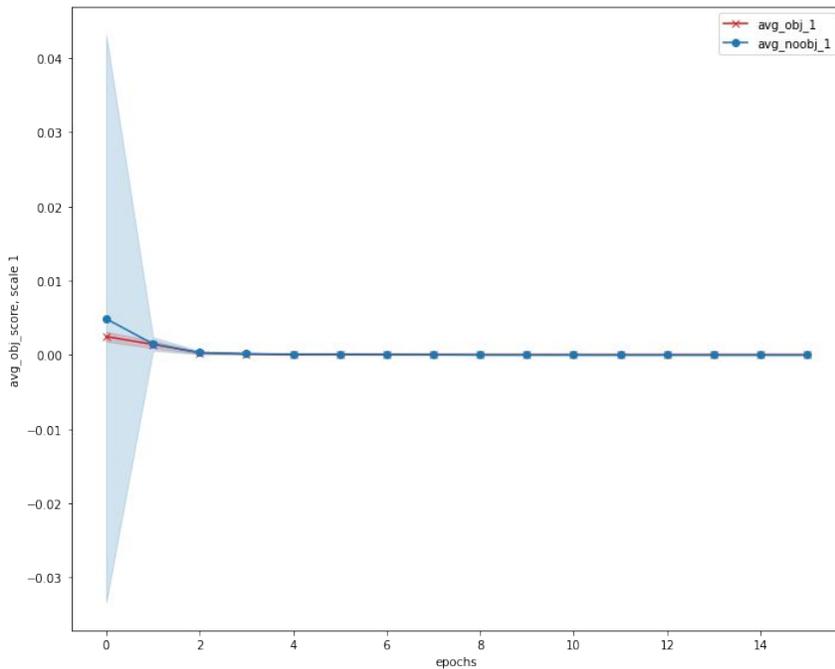
- A similar story to the intermediate scale
- Conf1 term fails to optimise well and possible overfitting
- Conf 2 term seem to optimise reasonably well with no noticeable overfitting
- Some possible overfitting for x,y and w,h terms after epoch 8



# Obj scores for Predicted Boxes

8 images used in this evaluation, so not same stats as loss curves. Need to add code for use of multiple batches in the metrics callback to this branch.

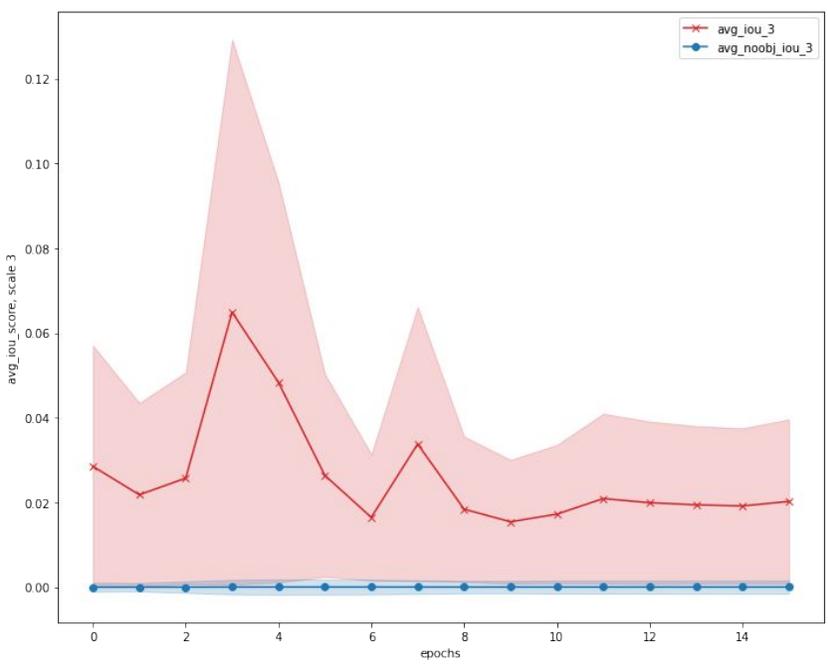
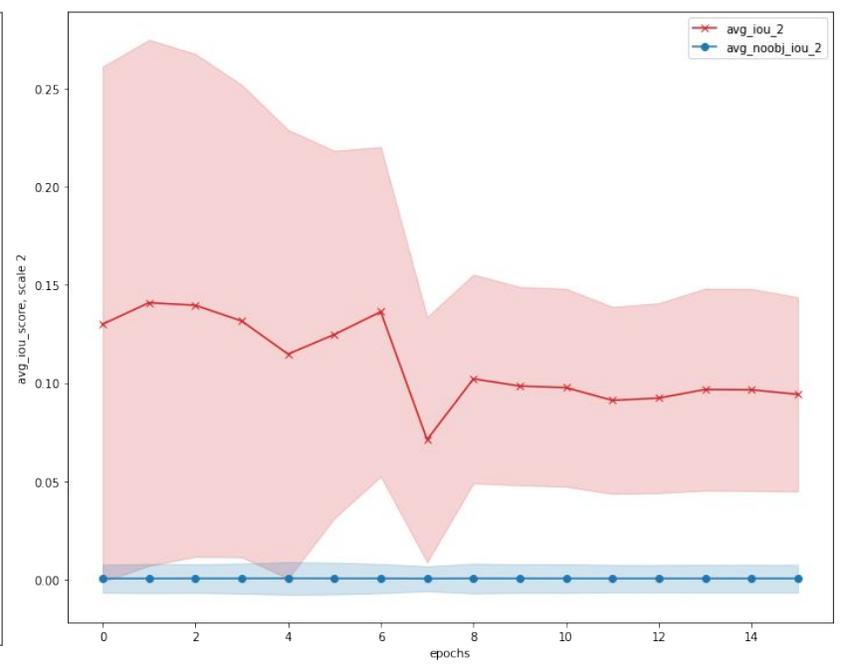
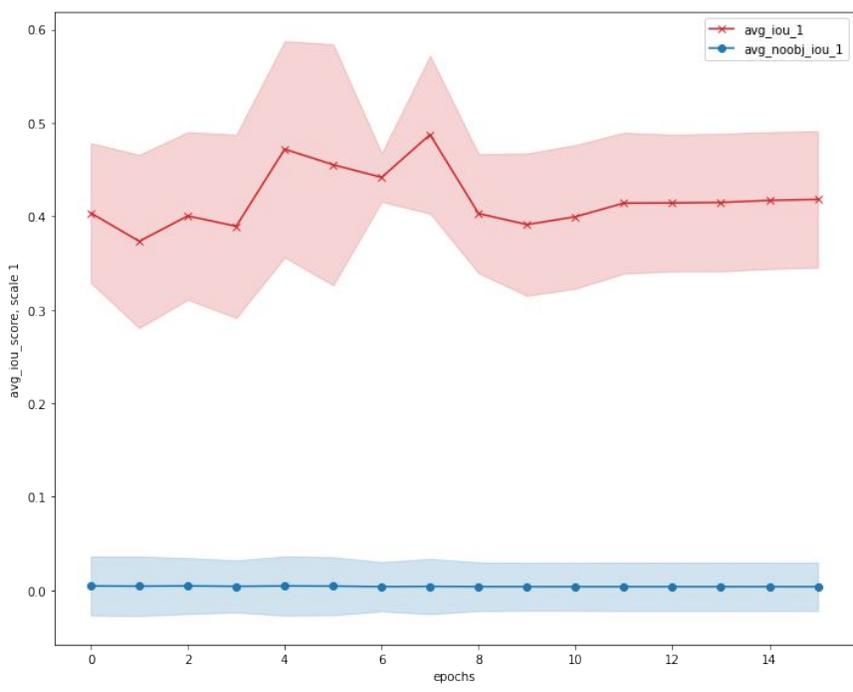
In these 8 images there are: 4 truth boxes for scale 1; 5 truth boxes for scale 2; 2 truth boxes for scale 3.



# IOU scores for Predicted Boxes

8 images used in this evaluation, so not same stats as loss curves. Need to add code for use of multiple batches in the metrics callback to this branch.

In these 8 images there are: 4 truth boxes for scale 1; 5 truth boxes for scale 2; 2 truth boxes for scale 3.



Low statistics data set is:

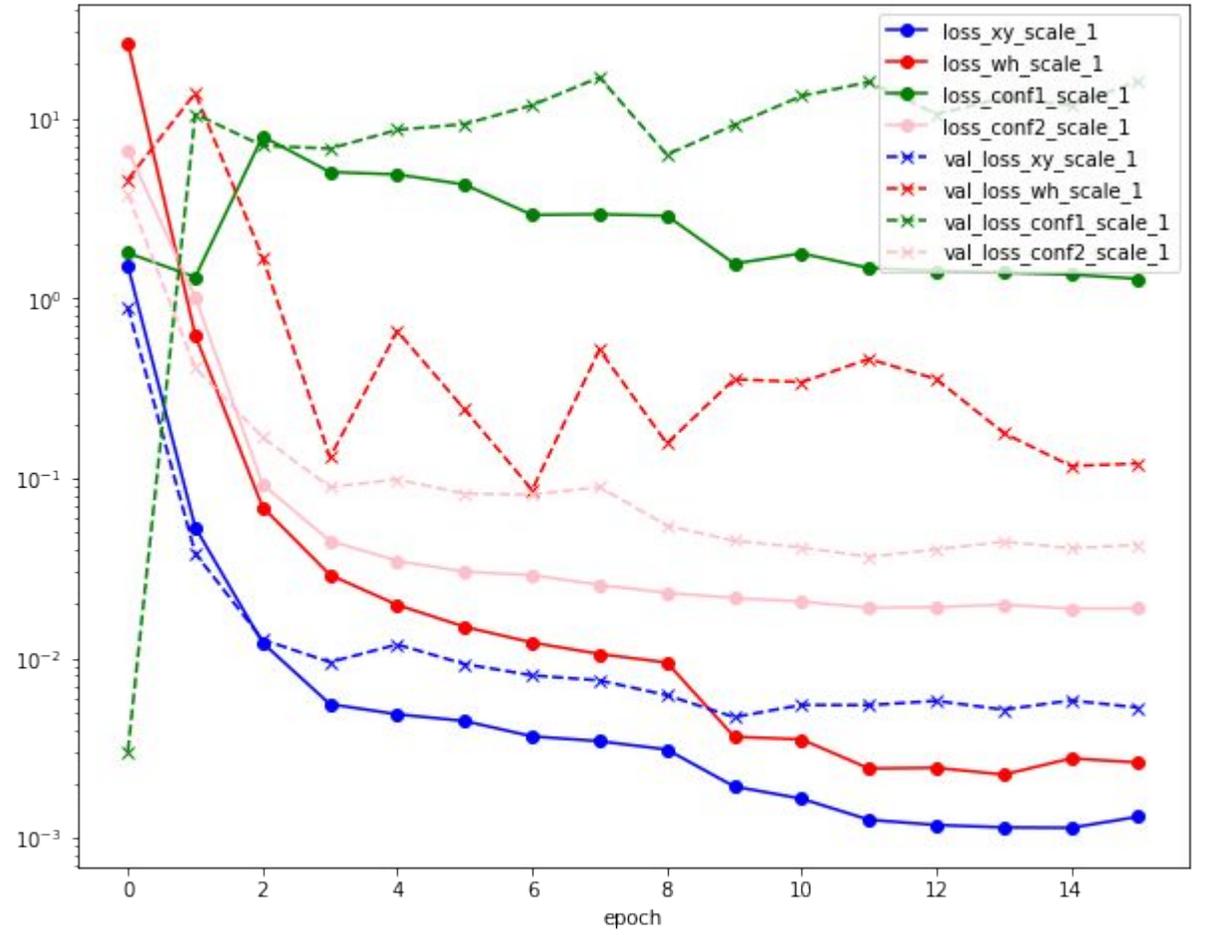
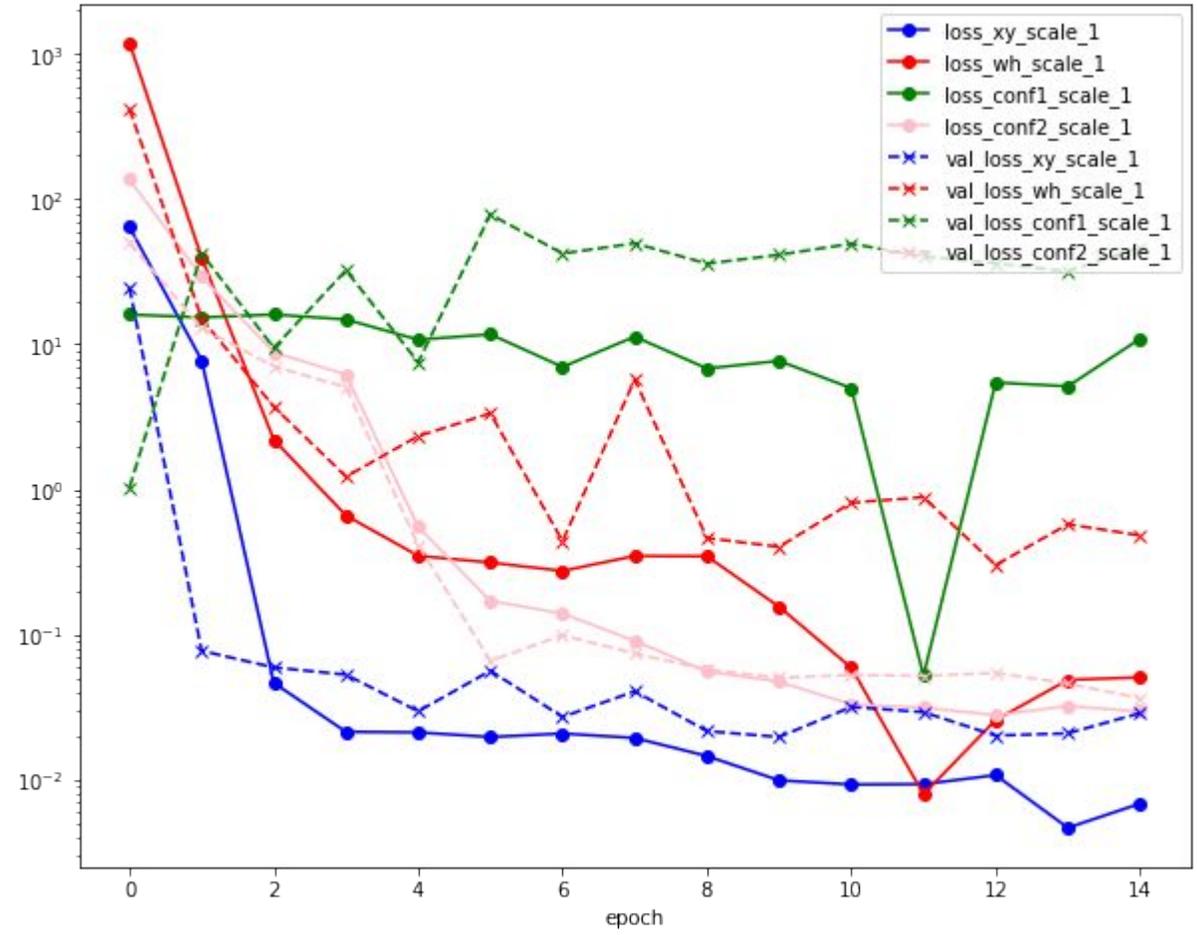
- 500 images
  - 500 train images
  - 500? validation images - checking

High stats dataset is:

- 30k images
  - 80% train
  - 10% validation

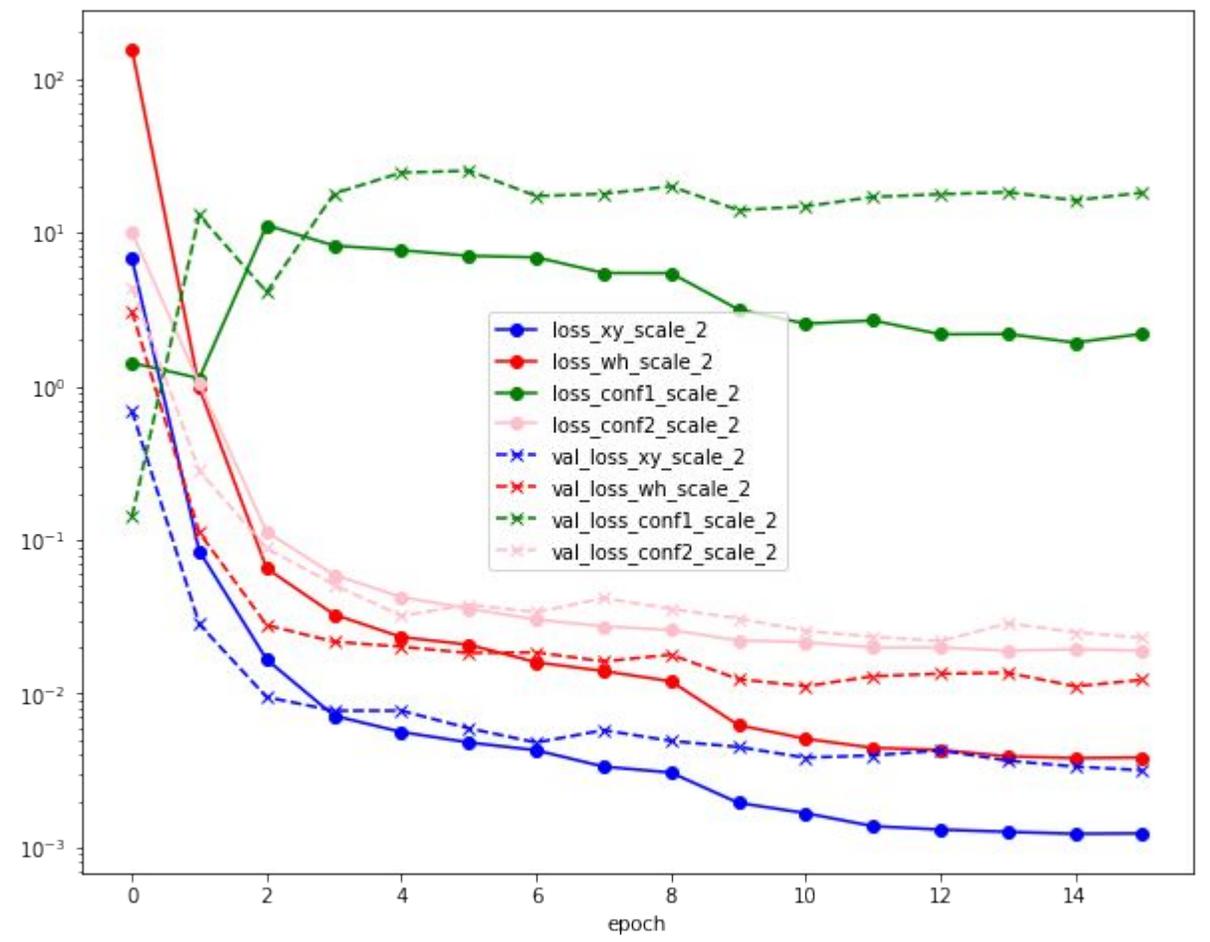
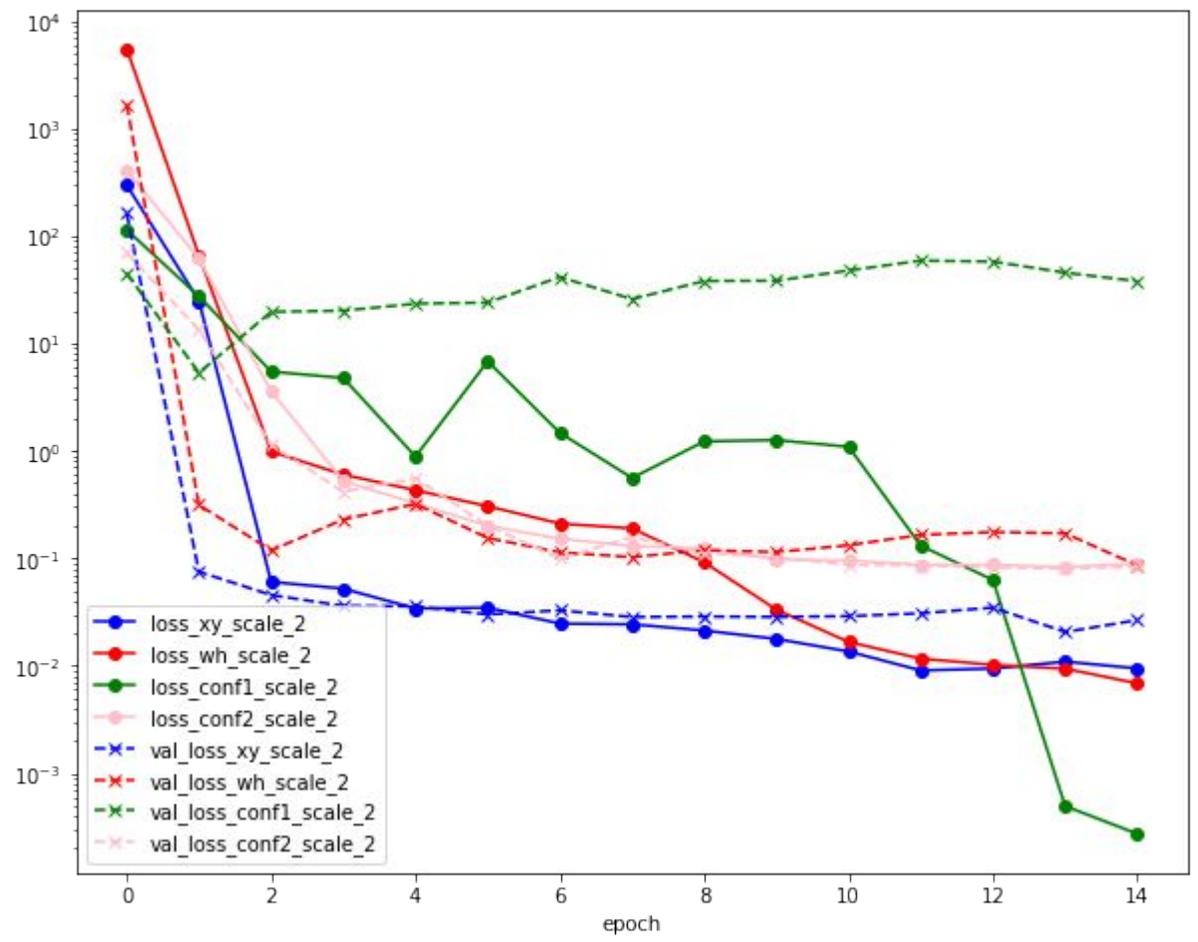
# Comparison Between Low and High stats

Looking at coarse scale, low stats left, high stats right.



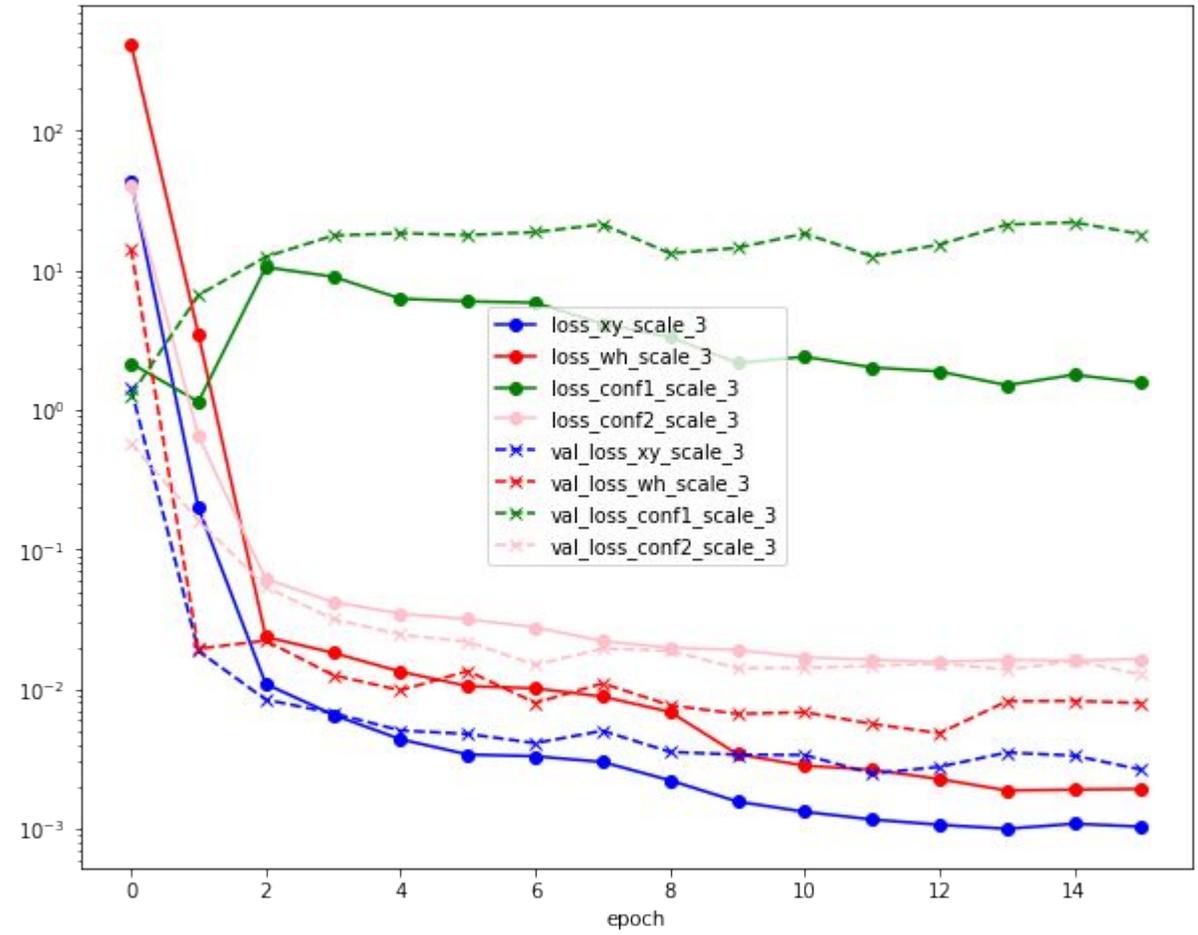
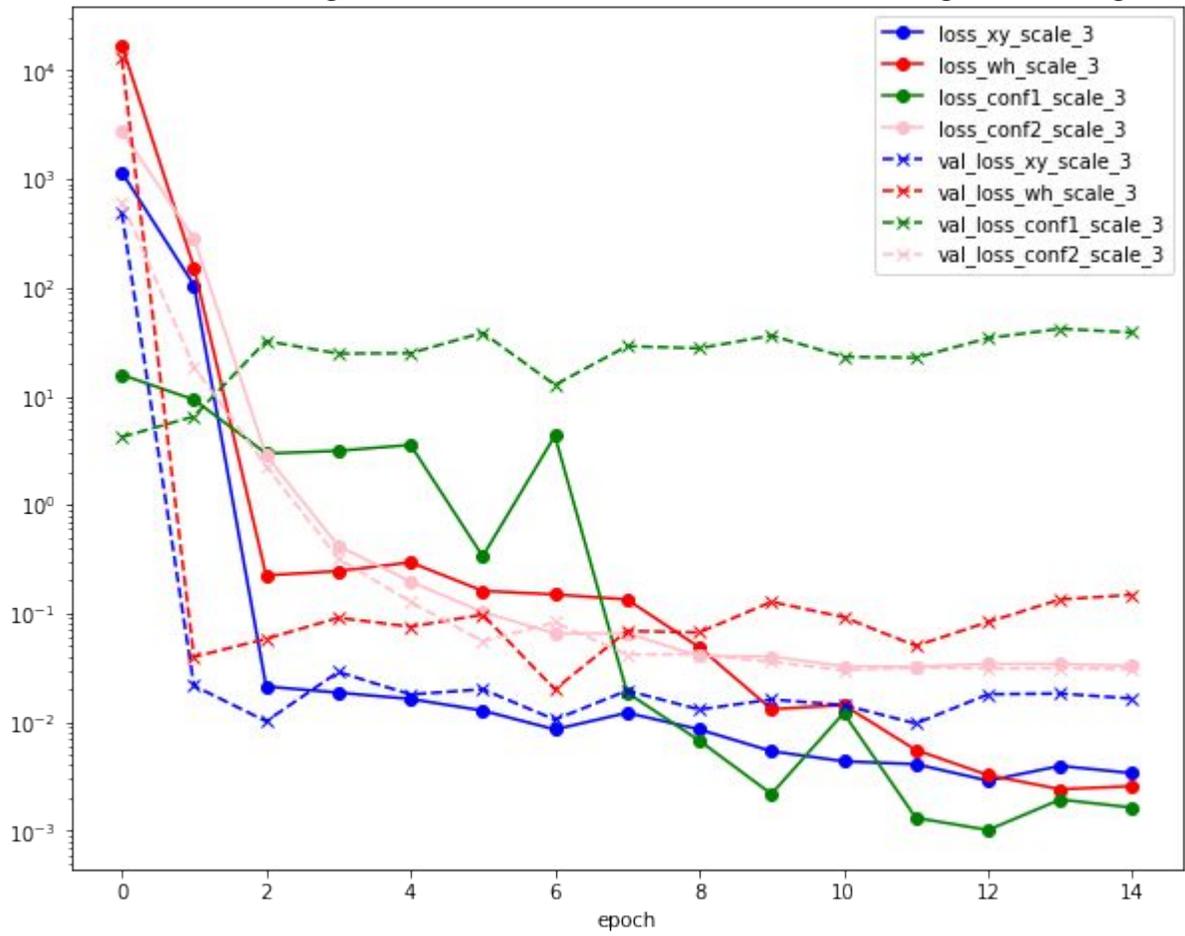
# Comparison Between Low and High stats

Looking at intermediate scale, low stats left, high stats right.



# Comparison Between Low and High stats

Looking at detailed scale, low stats left, high stats right.



# Summary

- 30k SNNu images without image augmentations is the dataset used
- The network always fails to optimise the objectness loss well (specifically, conf1 - predictions where we should have an associated truth box) and predicted objectness scores almost instantly decay to 0
- The anchors for the coarsest scale are diverse and it seems like this results in poor loss optimisation for the coarse scale, particularly in w,h term. The other terms in the loss also show more overfitting. Perhaps the overall effect is that this scale becomes more difficult to generalise for the network
- For the other scales, the loss seems to optimise well for all contributions to the loss except conf1. Some overfitting appears to occur in later epochs - this is the kind of thing I would expect image augmentations to have an impact on. Jitter would affect x,y overfitting, not sure if it would have much effect on w,h overfitting
- Possible image augmentation would help the objectness score, but seems unlikely. We'll try it.
- Potentially using a log likelihood loss for objectness term would help - will rerun using this
- We are also trying to understand more about any impact of batch size and learning rate and are running some basic exploratory tests

# Backup Slides

---

# Warmup Training

Some of the contributions to the loss depend on whether we are doing 'warmup training' or not. This forces predicted boxes to be the same size as the anchor box

```
batch_seen = tf.assign_add(batch_seen, 1.)
```

```
true_box_xy, true_box_wh, xywh_mask = tf.cond(tf.less(batch_seen, self.warmup_batches+1),  
                                              lambda: [true_box_xy + (0.5 + self.cell_grid[:, :grid_h, :grid_w, :, :]) * (1-object_mask),  
                                              true_box_wh + tf.zeros_like(true_box_wh) * (1-object_mask),  
                                              tf.ones_like(object_mask)],  
                                              lambda: [true_box_xy,  
                                              true_box_wh,  
                                              object_mask])
```

- If in warmup batch, effectively add a truth box for all predictions with no associated truth box which is in the centre of the grid cell of that prediction
- If in warmup batch add 0.0 to all entries in width and height tensor with no associated truth box - which makes the predicted boxes the same shape as the anchor boxes
- If in warmup batch include contributions from all predictions (via xywh\_mask), otherwise mask out any contributions to the box localization parts of the loss which have no associated truth box

# YOLOv3 Loss

## x,y term

```
xy_delta = xywh_mask * (pred_box_xy-true_box_xy) * wh_scale * self.xywh_scale
```

- xywh\_mask - all 1's if in warmup batch, otherwise only 1 for predictions with associated truth box
- pred\_box\_xy = `(self.cell_grid[:, :grid_h, :grid_w, :, :] + tf.sigmoid(y_pred[... , :2]))`  
which centres a box on the grid cell which it was predicted by and adds the sigmoided predicted output x,y values
- true\_box\_xy - if in a warmup batch, we care about all predictions and add `self.cell_grid[:, :grid_h, :grid_w, :, :] * (1-object_mask)` to the true\_box\_xy, so that we effectively create a truth box in the centre of the grid cell for all predicted boxes which actually have no associated truth box.
- wh\_scale - scale factor of  $1(480 \times 480) - 2(0 \times 0)$  applied to each element depending on the relative area of the truth box and the downsampled image
- self.xywh\_scale - scaling factor = 1.

# YOLOv3 Loss

## w,h term

```
wh_delta = xywh_mask * (pred_box_wh-true_box_wh) * wh_scale * self.xywh_scale
```

- `xywh_mask` - all 1's if in warmup batch, otherwise only 1 for predictions with associated truth box
- `pred_box_wh` - `y_pred[..., 2:4]` which is the natural logarithm of the scaling factor for the width and height of the predicted boxes. Which can be multiplied by `anchor_w,h/downsampled_image_w,h` to get output boxes
- `true_box_wh` - if in a warmup batch, we care about all predictions and add `tf.zeros_like(true_box_wh) * (1-object_mask)` to the `true_box_xy`, so that we effectively create a truth box in the centre of the grid cell for all predicted boxes which actually have no associated truth box which has a w,h of 0x0.
- `wh_scale` - scale factor of  $1(480 \times 480) - 2(0 \times 0)$  applied to each element depending on the relative area of the truth box and the downsampled image
- `self.xywh_scale` - scaling factor = 1.

## conf term

```
conf_delta = object_mask * (pred_box_conf-true_box_conf) * self.obj_scale +  
(1-object_mask) * conf_delta * self.noobj_scale
```

- `object_mask` - 1 for predictions with associated truth boxes, 0 otherwise
- `pred_box_conf` - `tf.expand_dims(tf.sigmoid(y_pred[... , 4]), 4)`, the objectness scores for all predictions
- `true_box_conf` - 1 for grid cell anchors with associated truth boxes, 0? otherwise - should check again
- `self.obj_scale` - scaling factor = 50
- `(1-object_mask)` - 0 for predictions with associated truth boxes, 1 otherwise
- `conf_delta` - predicted box objectness score for all predicted boxes which overlap with a truth box by 0.33 IOU or less
- `self.noobj_scale` - scaling factor = 1

Last week I found that the contribution to this term from predictions with associated truth boxes was negative! Due to `pred_box_conf-true_box_conf`. Will minimise cost more when the predictions are worse!

## class term

```
class delta = object mask * \  
tf.expand_dims(tf.nn.sparse_softmax_cross_entropy_with_logits(labels=true_box_class,  
logits=pred_box_class), 4) * \  
self.class_scale
```

- object\_mask - 1 for predictions with associated truth boxes, 0 otherwise
- true\_box\_class - one-hot class of truth box
- pred\_box\_class - vector of predicted classes. List should be 1 class long. - ought to explicitly check dimensions are okay in this calculation
- self.class\_scale - scaling factor = 1

# YOLOv3 Loss

```
loss_xy      = tf.reduce_sum(tf.square(xy_delta),      list(range(1,5)))  
loss_wh      = tf.reduce_sum(tf.square(wh_delta),      list(range(1,5)))  
loss_conf    = tf.reduce_sum(tf.square(conf_delta),    list(range(1,5)))  
loss_class   = tf.reduce_sum(class_delta,              list(range(1,5)))  
  
loss = loss_xy + loss_wh + loss_conf + loss_class
```

I find `loss_class` is always 0, as expected

# Investigate Loss - Part 3

Some parts of our loss function may be different from those used in the YoloV3 paper.

Firstly: *YOLOv3 predicts an objectness score for each boundingbox using logistic regression.*

We do apply a sigmoid function to the output object score, but use a MSE cost function for the loss from this term. It isn't clear which cost function is used in the paper. Online code seems to use MSE, which can be sub-optimal for classification problems.

If the object score isn't improved by changes to application of warmup batches, it may be useful to investigate the use of the cross-entropy (log-loss) for the objectness classification problem when using logistic regression. This should allow a smoother cost function for the objectness term.

# Investigate Loss - Part 4

---

Some parts of our loss function may be different from those used in the YoloV3 paper.

Secondly, YoloV3 paper doesn't specify a definition for the truth object score or the predicted objectness score. At the moment it is set to 1 for the anchor in a given cell grid which has the maximal overlap with a truth box. However, for some online YoloV2 implementations, this is multiplied by the IOU before the predicted object confidence is subtracted in the loss function.

Not clear in my mind if this or multiplying the sigmoid'ed output of the part of our network which is responsible for predicting 'objectness' by the IOU would be a better idea. This is how I think it is written in the YoloV2 paper though.

# Overall Loss

From a single scale, the loss seems to have the dimensions of 8 (ie:the batch size) and be left to aggregate over multiple batches. At the moment I understand this aggregation to be a mean over all batches. Not clear to me why the sum of the loss terms which I have logged doesn't equal the loss for this scale...

Maybe this is to do with the optimizer - but not really sure

```
loss_xy      = tf.reduce_sum(tf.square(xy_delta),      list(range(1,5)))  
loss_wh      = tf.reduce_sum(tf.square(wh_delta),      list(range(1,5)))  
loss_conf    = tf.reduce_sum(tf.square(conf_delta),    list(range(1,5)))  
loss_class   = tf.reduce_sum(class_delta,              list(range(1,5)))  
  
loss = loss_xy + loss_wh + loss_conf + loss_class
```

# Overall Loss

It is less clear how the contributions to the loss for a given scale actually sum together to give the loss value for that scale.

The network uses an Adam optimizer and calculates the final loss via:

```
Loss = tf.sqrt(tf.reduce_sum(y_pred))
```

Where:

```
Y_pred = [loss_yolo_1, loss_yolo_2,  
loss_yolo_3]
```

