Skymap Global Pte Ltd

Weekly internship report

Intern's fullname : Nong Minh Hieu

Report period : 3rd September - 9th September

Supervisor : Le Hai Ha

Table of content

A. Disclaimer.

This report template is used to help supervisor track and monitor the learning progress of the intern at the designated company/organization. This report template shall not be kept, recorded or used for any formal reasons.

B. Report contents.

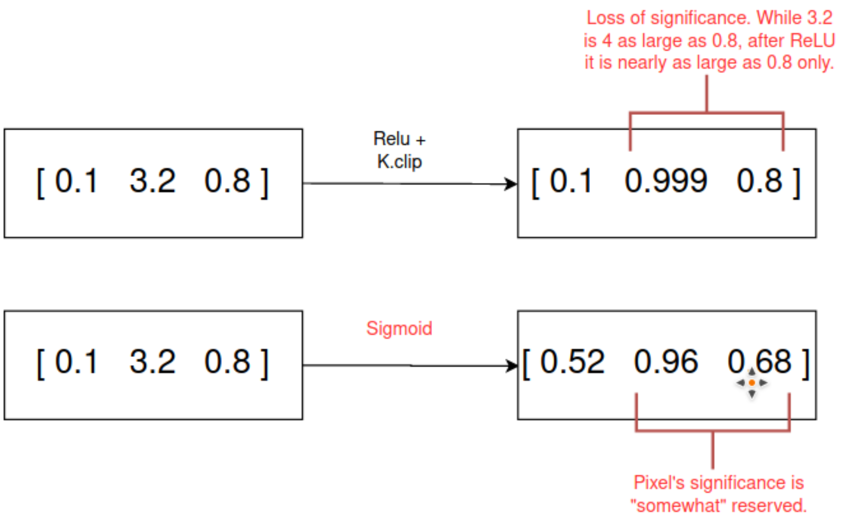
I. Introduction.

This report is a summarization on the progress of the aforementioned intern on "Researching on RoadNet's approach on road detection from high-resolution, remotely-sensed image" task. This report shall cover briefly on what the intern has learnt, what other tasks are remained and self reflection.

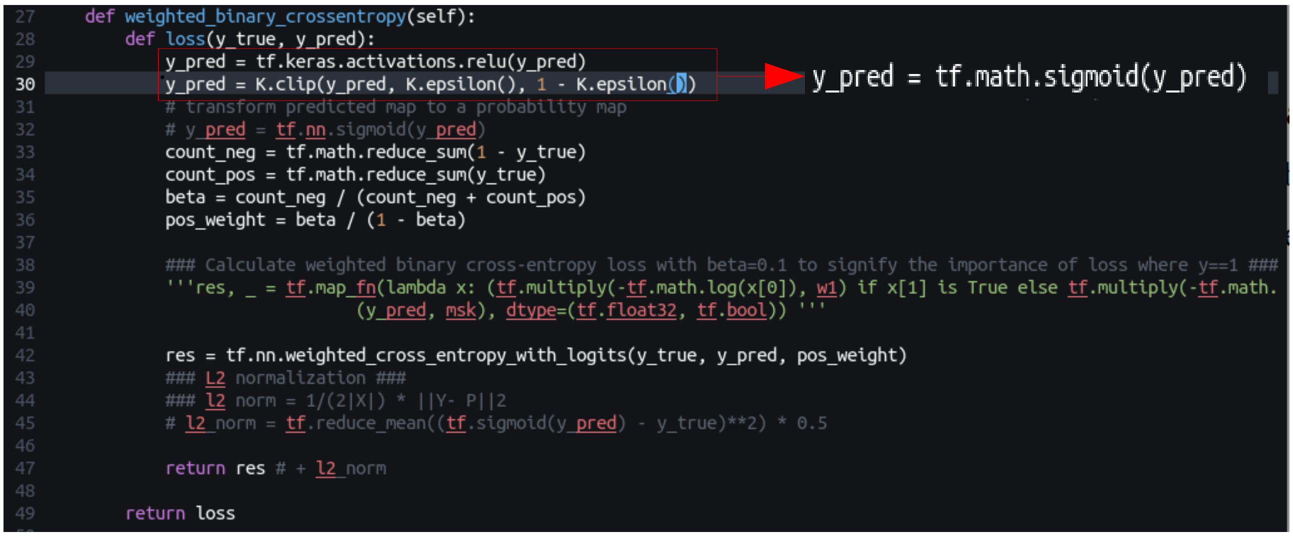
II. Training script modifications.

1. Side output activation functions.

The side outputs was previously activated using ReLU activation function then clipped within (0, 1) range to represent the pixel-wise probability. However, using this method will forgo the significance of the pixel in representing regions with roads.


Loss of significance. While 3.2 is 4 as large as 0.8, after ReLU it is nearly as large as 0.8 only.
[0.1 3.2 0.8]
Relu + K.clip
[0.1 0.999 0.8]
[0.1 3.2 0.8]
Sigmoid
[0.52 0.96 0.68]
Pixel's significance is "somewhat" reserved.

Therefore, the activation is changed to sigmoid to ensure all pixel-wise outputs are within (0, 1) range while maintaining significance of certain pixels.

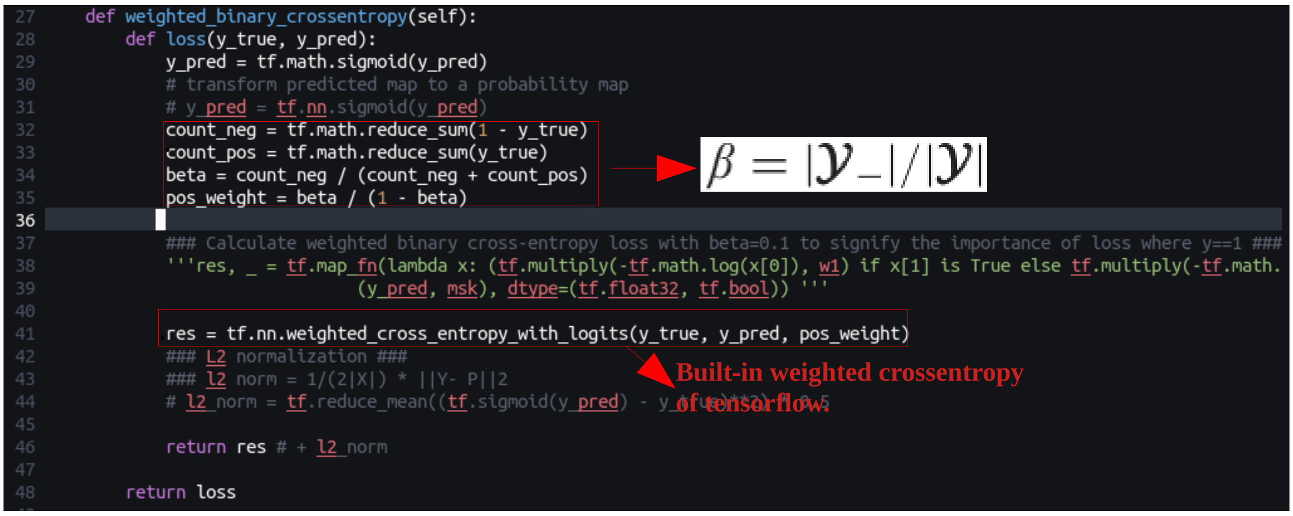


```python
    def weighted_binary_crossentropy(self):
        def loss(y_true, y_pred):
            y_pred = tf.keras.activations.relu(y_pred)
            y_pred = K.clip(y_pred, K.epsilon(), 1 - K.epsilon())
            # transform predicted map to a probability map
            # y_pred = tf.nn.sigmoid(y_pred)
            count_neg = tf.math.reduce_sum(1 - y_true)
            count_pos = tf.math.reduce_sum(y_true)
            beta = count_neg / (count_neg + count_pos)
            pos_weight = beta / (1 - beta)

            ### Calculate weighted binary cross-entropy loss with beta=0.1 to signify the importance of loss where y==1 ###
            '''res, _ = tf.map_fn(lambda x: (tf.multiply(-tf.math.log(x[0]), w1) if x[1] is True else tf.multiply(-tf.math.
                                 (y_pred, msk), dtype=(tf.float32, tf.bool)) '''

            res = tf.nn.weighted_cross_entropy_with_logits(y_true, y_pred, pos_weight)
            ### L2 normalization ###
            ### l2 norm = 1/(2|X|) * ||Y- P||2
            # l2_norm = tf.reduce_mean((tf.sigmoid(y_pred) - y_true)**2) * 0.5

            return res # + l2_norm

        return loss
```

```python
y_pred = tf.math.sigmoid(y_pred)
```

2. <u>Weighted crossentropy loss.</u>

Beta hyper-parameter used in weighted crossentropy was perviously assigned as a fixed hyper-parameter. However, the amount of positive and negative pixels in the ground truth maps varies by image patches. Therefore, beta should be assigned adaptively by taking the amount of background pixels divide by the total number of pixels. Then, we calculate the weight for loss at foreground (positive pixels) by z = β / (1 – β) (pos_weight).



```python
    def weighted_binary_crossentropy(self):
        def loss(y_true, y_pred):
            y_pred = tf.math.sigmoid(y_pred)
            # transform predicted map to a probability map
            # y_pred = tf.nn.sigmoid(y_pred)
            count_neg = tf.math.reduce_sum(1 - y_true)
            count_pos = tf.math.reduce_sum(y_true)
            beta = count_neg / (count_neg + count_pos)
            pos_weight = beta / (1 - beta)

            ### Calculate weighted binary cross-entropy loss with beta=0.1 to signify the importance of loss where y==1 ###
            '''res, _ = tf.map_fn(lambda x: (tf.multiply(-tf.math.log(x[0]), w1) if x[1] is True else tf.multiply(-tf.math.
                                 (y_pred, msk), dtype=(tf.float32, tf.bool)) '''

            res = tf.nn.weighted_cross_entropy_with_logits(y_true, y_pred, pos_weight)
            ### L2 normalization ###
            ### l2 norm = 1/(2|X|) * ||Y- P||2
            # l2_norm = tf.reduce_mean((tf.sigmoid(y_pred) - y_true)**2) * 0.5

            return res # + l2_norm

        return loss
```

$$\beta = |\mathcal{Y}_-| / |\mathcal{Y}|$$

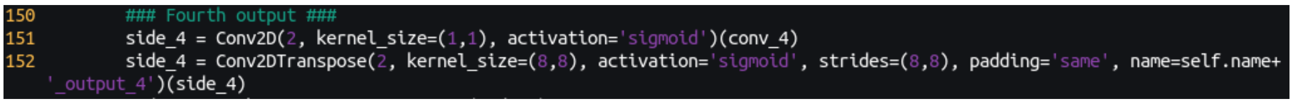
Built-in weighted crossentropy of tensorflow.

Then the final loss is defined as :

$$z \times -\log\left(\sum_{j \in Y_+}^{\square} \sigma(Pr(Pj = 1))\right) + (1 - z) \times -\log\left(\sum_{j \in Y_-}^{\square} \sigma(Pr(Pj = 0))\right)$$

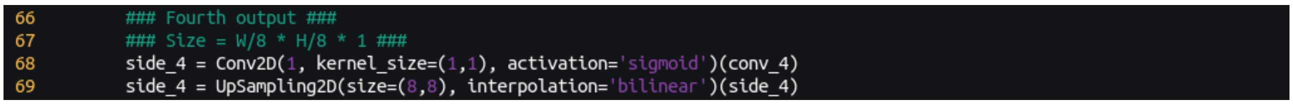
3. Upsampling method.

Previously the upsampling method used was transposed convolutional layer. Now it is changed to bilinear interpolation as in the research paper.



```
        ### Fourth output ###
        side_4 = Conv2D(2, kernel_size=(1,1), activation='sigmoid')(conv_4)
        side_4 = Conv2DTranspose(2, kernel_size=(8,8), activation='sigmoid', strides=(8,8), padding='same', name=self.name+
'_output_4')(side_4)
```

(Transposed convolutional block)



```
        ### Fourth output ###
        ### Size = W/8 * H/8 * 1 ###
        side_4 = Conv2D(1, kernel_size=(1,1), activation='sigmoid')(conv_4)
        side_4 = UpSampling2D(size=(8,8), interpolation='bilinear')(side_4)
```

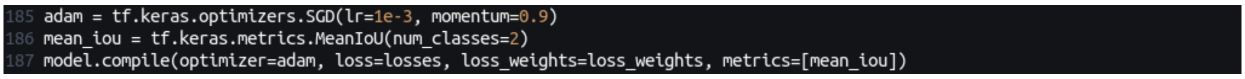
(bilinear interpolation)

4. Other modifications.

- Changed the image dimension from 128x128 to 512x512 and batch size from 32 to 1 to cater for larger image size per batch.
- Activate prediction using sigmoid before calculating l2 norm.

$$\|P - Y\|_2^2 = \sqrt{(\sigma(P) - Y)^2}$$

- Added MeanIOU as a training metric.



```
adam = tf.keras.optimizers.SGD(lr=1e-3, momentum=0.9)
mean_iou = tf.keras.metrics.MeanIoU(num_classes=2)
model.compile(optimizer=adam, loss=losses, loss_weights=loss_weights, metrics=[mean_iou])
```

III. Current issues with training process.

- The loss formula is implemented correctly but the loss of the model is not decreasing to a certain point as well as MeanIOU is not increasing.
- The model's prediction is completely biased towards predicting background pixels (very high false negative).

All of the aforementioned issues imply that the loss formulation does not successfully suppress the imbalance in the amount of background and foreground pixels in the training image patches. More researches will be needed to find out as to while the prediction of the model is still highly biased even when using weighted sigmoid crossentropy.

IV. Remaining uncompleted tasks.

- Remedy the biasedness of the model, re-run the training iterations and visualize losses and evaluation metrics (MeanIOU, precision, ...).
- Run prediction and bilinear blend the predicted image patch and format the results in geojson or shapely with rasterio.