Skymap Global Pte Ltd

Weekly internship report

Intern's fullname : Nong Minh Hieu

Report period : 18th September - 25th 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 loss modification.

After a number of unsuccessful attempt to train the model with custom weighted cross entropy loss. The model’s loss and MeanIOU metric for segmentation of road has been successfully converged on validation dataset with the following new custom weighted crossentropy loss function definitions :

1. <u>Loss function 1:</u>

```
    def weighted_binary_crossentropy(self):
        def loss(y_true, y_pred):
            # y_pred = tf.math.sigmoid(y_pred)
            _epsilon = tf.convert_to_tensor(K.epsilon(), y_pred.dtype.base_dtype)
            y_pred = tf.clip_by_value(y_pred, _epsilon, 1 - _epsilon)
            # y_pred = -tf.math.log((1/y_pred) - 1)

            y_true = tf.cast(y_true, tf.float32)
            # 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)

            w1 = beta
            w2 = 1 - beta

            # ones = tf.ones_like(y_true)
            # msk = tf.equal(y_true, ones)
            ### Calculate weighted binary cross-entropy loss with beta=0.1 to signify the import
            # res, _ = tf.map_fn(lambda x: (tf.multiply(-tf.math.log(x[0]), w1) if x[1] is True
            #                    (y_pred, msk), dtype=(tf.float32, tf.bool))
            # res = - beta * tf.multiply(tf.math.log(y_pred), y_true) - (1-beta) * tf.multiply(t
            # res = tf.math.reduce_mean(res)
            b_ce = K.binary_crossentropy(y_true, y_pred)
            weight_vector = y_true * beta + (1. - y_true) * (1-beta)
            weighted_b_ce = weight_vector * b_ce
            res = K.mean(weighted_b_ce)

            # 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
```

Which is basically the mean of the sum of every loss pixel-wise multiplied by their weights (β for forground pixel and 1 – β for background pixel) and can be represented as the following formula :

$$\frac{1}{|Y|}\sum_{j \in Y}(\delta(P_j)(\beta\log(P_j)) + \delta(P_j)((1-\beta)\log(1-P_j)))$$

- $\delta(x) = 1 \; for \; x \geq 0.5 \; (positive \; class)$
- $\delta(x) = 0 \; for \; x < 0.5 \; (negative \; class)$

2. <u>Loss function 2:</u>

Since loss function 1 is not close to the suggested loss function in the original RoadNet paper, the loss function is supposedly modified as below:

```
    def weighted_binary_crossentropy(self):
        def loss(y_true, y_pred):
            # y_pred = tf.math.sigmoid(y_pred)
            _epsilon = tf.convert_to_tensor(K.epsilon(), y_pred.dtype.base_dtype)
            y_pred = tf.clip_by_value(y_pred, _epsilon, 1 - _epsilon)
            # y_pred = -tf.math.log((1/y_pred) - 1)

            y_true = tf.cast(y_true, tf.float32)
            # 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)

            w1 = beta
            w2 = 1 - beta

            # ones = tf.ones_like(y_true)
            # msk = tf.equal(y_true, ones)
            ### Calculate weighted binary cross-entropy loss with beta=0.1 to signify the imp
            # res, _ = tf.map_fn(lambda x: (tf.multiply(-tf.math.log(x[0]), w1) if x[1] is Tr
            #                    (y_pred, msk), dtype=(tf.float32, tf.bool))
            # res = - beta * tf.multiply(tf.math.log(y_pred), y_true) - (1-beta) * tf.multipl
            # res = tf.math.reduce_mean(res)
            b_ce = K.binary_crossentropy(y_true, y_pred)
            weight_vector = y_true * beta + (1. - y_true) * (1-beta)
            weighted_b_ce = weight_vector * b_ce

            loss_pos = weighted_b_ce * y_true
            loss_neg = weighted_b_ce * (1 - y_true)

            # res = K.mean(weighted_b_ce)
            res = K.mean(loss_pos) + K.mean(loss_neg)

            # 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

```

Which is the sum of mean loss for each class (foreground and background) and can be represented as the following formula :

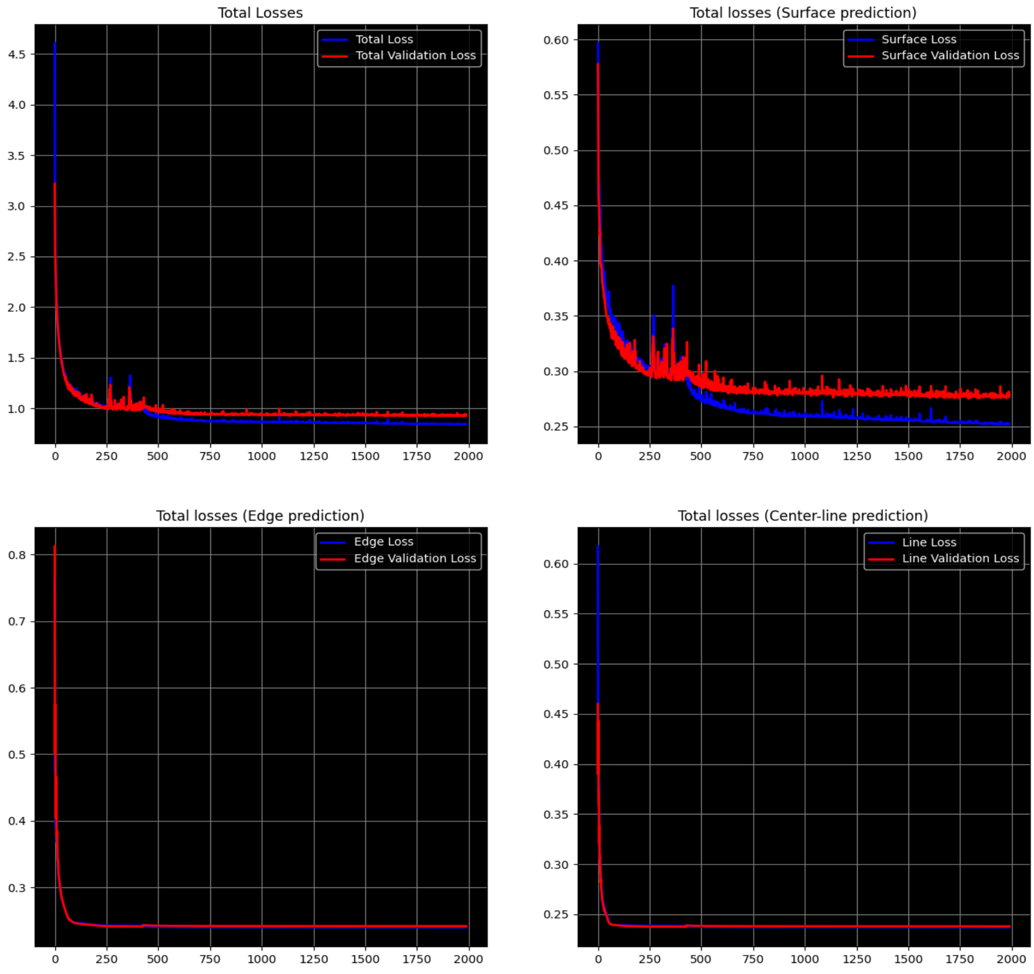
$$\frac{1}{|Y_+|}\sum_{j \in Y_+} \beta \log(P_j) + \frac{1}{|Y_-|}\sum_{j \in Y_-} (1-\beta)\log(Pj)$$

However, due to limited time and computation resource restriction, this loss function has not been tested. Therefore the remaining part of the report will be based on the training results using the first loss function.

III. Visualization of training loss and metrics.

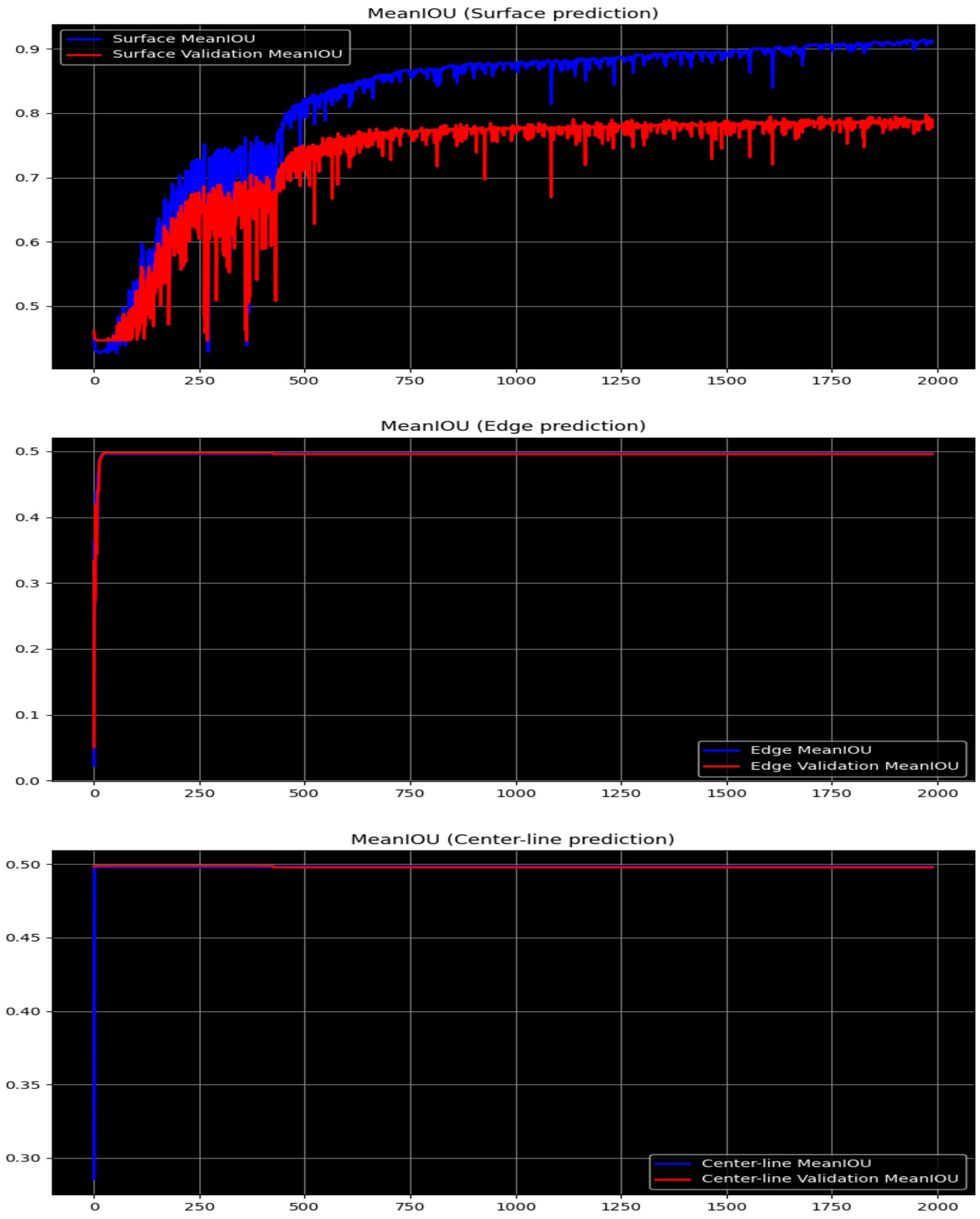
The loss and metrics has been improved for segmentation of road using the first loss function. However, prediction of center-line detection and edge detection saw no progress thus far.

1. <u>Training Loss.</u>



(Total training loss and losses for individual layer prediction)

2. Training metric (Mean IOU).

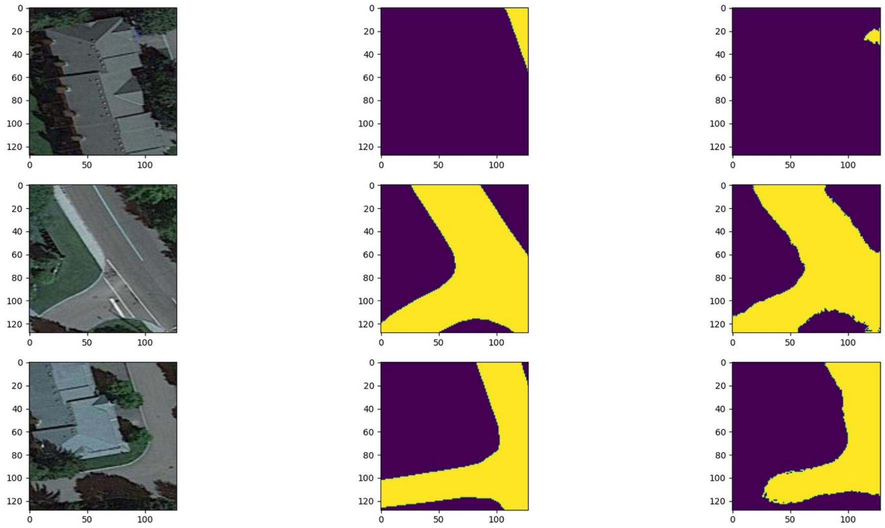


(Mean IOU of road surface segmentation, edge detection and centerline estimation respectively)

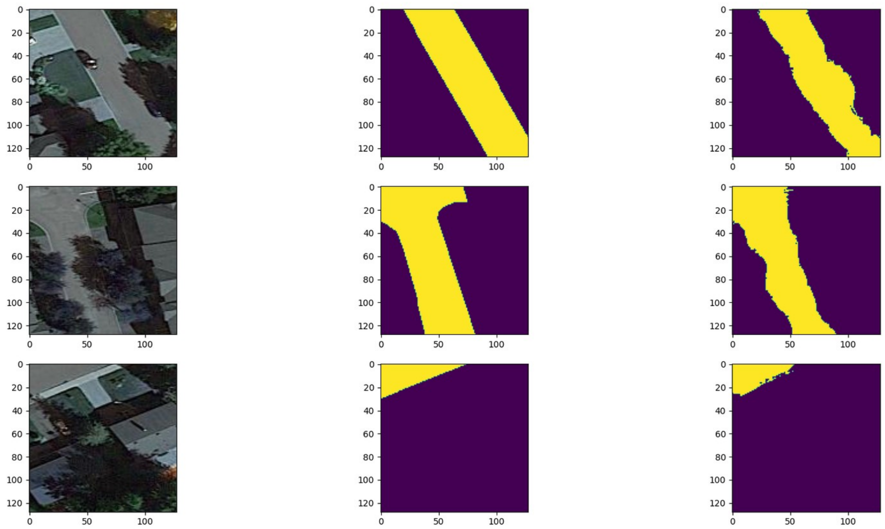
IV. Sample predictions.

1. Predictions by individual image patches.

The following figures represents the prediction of the current model by individual 128 x 128 x 3 image pathces. From left to right : raw image – ground truth segmentation – prediction.



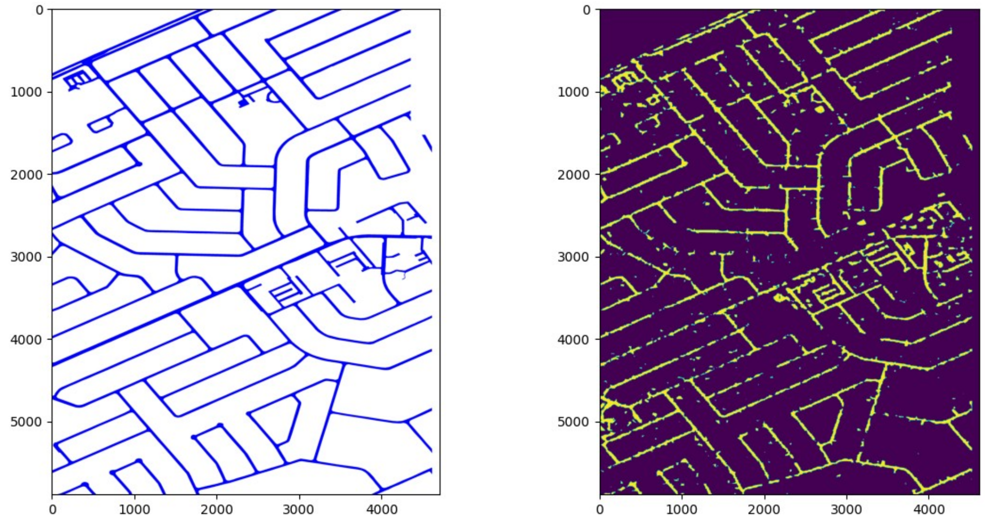
(Figure 1)



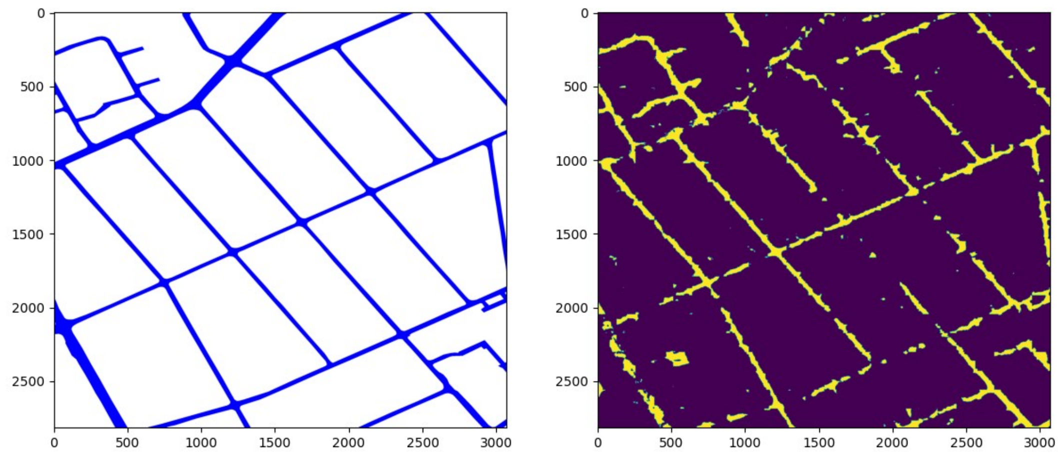
(Figure 2)

2. Prediction on full images of validation set.

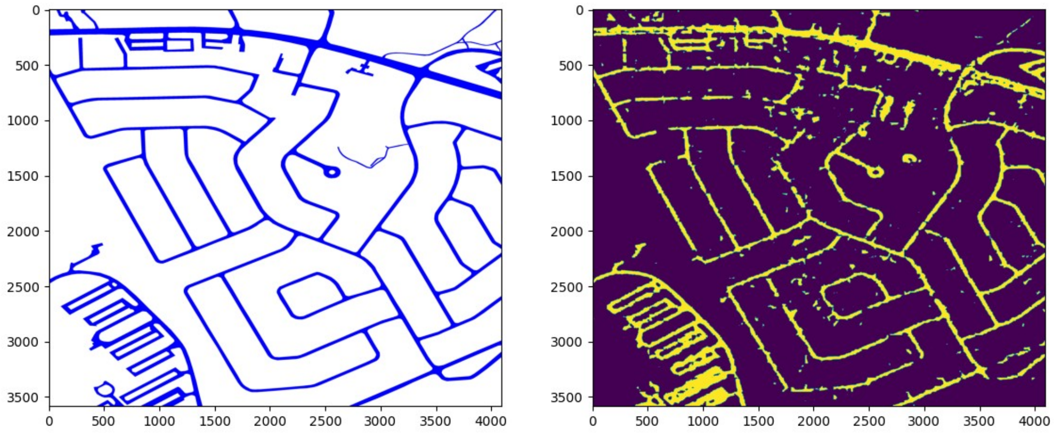
a. Full prediction on Ottawa-1.tif.



b. Full prediction on Ottawa-20.tif.



c. Full prediction on Ottawa-16.tif.



V. Remaining uncompleted tasks.

- Complete the model for centerline and edge detection.
- Run prediction to obtain full completed results including segmentation, centerline and edge to format into geojson files.