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. Key takeaways.

1. RoadNet's proposed method.

Road detection is generally subdivided into the following tasks:

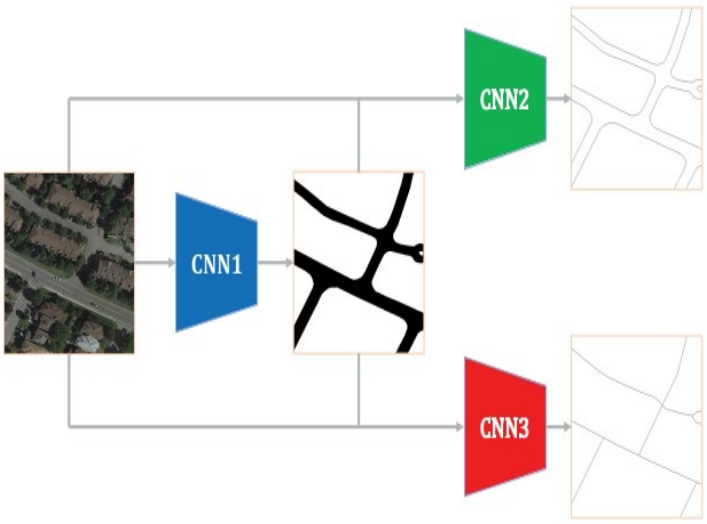
- Road surface segmentation (Semantic segmentation).
- Road edges detection (Edge detection).
- Road centerline detection (Object centerline detection).

Therefore it could be a challengeing task because of the following issues:

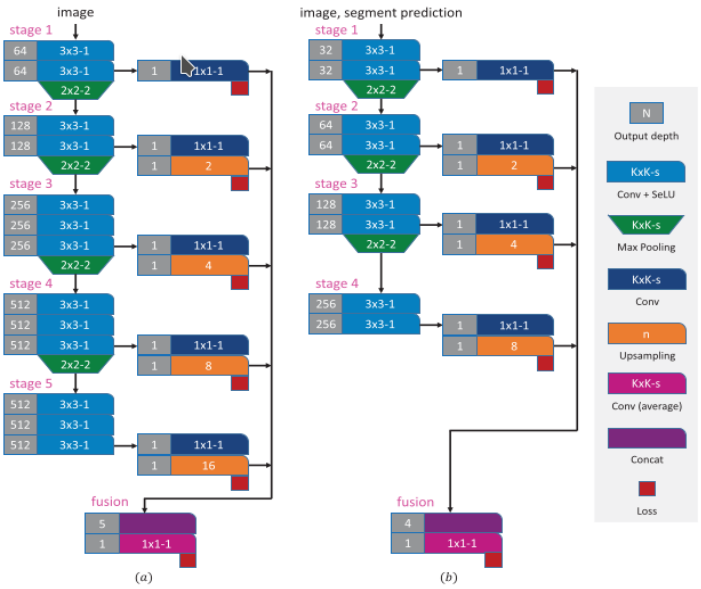
- Cars and buildings leading to confusing shadows.
- Cars and buildings leading to occlusions.

RoadNet proposed that the predicted segmentation map should aid the prediction of road centerlines and edges during training to benefit from both high level and low level features.

2. RoadNet architecture.



The general architecture of RoadNet



Architecture of RoadNet in details, with CNN1 (Figure (a)) and CNN2 – CNN3 (Figure (b))

The key difference between RoadNet and the previous method is that it is not a straight feedforward convolutional network. It consists of 3 sub-networks to to form a multi-tasking network with the segmentation network (CNN1) boosting the other two networks (just like a boosting technique). For each individual network, the output of each convolutional block also boost for the output of the subsequent blocks.

```
24
25      def get_model(self):
26          ### Stage 1 ###
27          inputs = Input(shape=self.input_shape)
28          conv_1 = Conv2D(64, kernel_size=(3,3), activation='selu', padding='same')(inputs)
29          conv_1 = Conv2D(64, kernel_size=(3,3), activation='selu', padding='same')(conv_1)
30          pool_1 = MaxPooling2D(pool_size=(2,2))(conv_1)
31
32          ### First output ###
33          ### Size = W * H * 1 ###
34          side_1 = Conv2D(1, kernel_size=(1,1), activation='relu', padding='same', name='surface_output_1', kernel_regularizer=
35
36          ### Stage 2 ###
37          conv_2 = Conv2D(128, kernel_size=(3,3), activation='selu', padding='same')(pool_1)
38          conv_2 = Conv2D(128, kernel_size=(3,3), activation='selu', padding='same')(conv_2)
39          pool_2 = MaxPooling2D(pool_size=(2,2))(conv_2)
```

Output of the first block boost the second block.

3. Training and loss formulation on the fused output layers.

a. *Balanced cross entropy*

When using the traditional cross-entropy to train RoadNet, the domination of background pixels will lead to mis-evaluation of the model. The accuracy may reflect the performance wrongly due to a high percentage of correct pixels are background pixels. Therefore, RoadNet proposed using balanced cross entropy.

b. *Construction loss.*

Although the balanced cross entropy well define the discrimination between two classes (foreground and background). There is still overfitting to cope with.

$$\begin{aligned}\mathcal{L}_{bf} &= \ell(\boldsymbol{\mathcal{W}}, \mathbf{w}) \\ &= -\beta \sum_{j \in \mathcal{Y}_+} \log \Pr(P_j = 1 | \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{W}}, \mathbf{w}) \\ &\quad - (1-\beta) \sum_{j \in \mathcal{Y}_-} \log \Pr(P_j = 0 | \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{W}}, \mathbf{w})\end{aligned}$$

```
25
26      def weighted_binary_crossentropy(self):
27          w1 = self.beta
28          w2 = 1 - self.beta
29
30          def loss(y_true, y_pred):
31              # transform predicted map to a probability map
32              y_pred = K.clip(y_pred, K.epsilon(), 1 - K.epsilon()) # need epsilon to avoid absolute zeros
33              ones = tf.ones_like(y_true) # create a mask
34              msk = tf.equal(y_true, ones)
35
36              ### Calculate weighted binary crossentropy loss with beta=0.1 to signify the imporance of loss where y==1 ###
37              res, _ = tf.map_fn(lambda x: (tf.multiply(-tf.math.log(x[0]), w1) if x[1] is True else tf.multiply(-tf.math.log(1 - x[0]), w2), x[1]),
38                                 (y_pred, msk), dtype=(tf.float32, tf.bool))
39
40              ### L2 normalization ###
41              ### l2_norm = 1/(2|X|) * ||Y- P||2
42              l2_norm = tf.nn.l2_normalize(y_pred - tf.cast(msk, dtype=tf.float32)) * (1/(y_pred.shape[1] * y_pred.shape[2]))
43
44
45              return res + l2_norm
46
47          return loss
```

$$\mathcal{L}_c = \frac{1}{2|\boldsymbol{\mathcal{X}}|} \|\boldsymbol{\mathcal{P}} - \boldsymbol{\mathcal{Y}}\|_2^2$$

c. *Weight decay loss.*

Weight decay loss is generally a l2 kernel regularizer. Therefore, it can be added as an attribute of each side output layer instead of being explicitly included in the loss function.

4. Side outputs supervision.

In RoadNet, we do not only supervise the final output of the surface segmentation, road edge and centerline layers but we also monitor the loss of each side output with respect to the ground truth. This will make it a total of 16 layers to monitor (3 final output maps, 5 side outputs for surface sub-network and 4 side outputs for each of edge sub-network and centerline sub-network).

For each side output, the loss is formulated as :

$$\mathcal{L}_{bs} = \sum_{m=1}^{M} \omega_m \ell_{\text{side}}(\mathcal{W}, \mathbf{w}^{(m)}),$$

Where ω (m) is the weightage of the loss at m-th side output and l(W, w(m)) is the balanced crossentropy of the side output with respect to the ground truth map. With the main loss formulation and the side loss supervision, the overall loss is formulated as:

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{bf} + \gamma \mathcal{L}_c + \eta \mathcal{L}_w$$

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{\text{total}} + \delta \mathcal{L}_{bs} \\ &= \alpha \mathcal{L}_{bf} + \delta \mathcal{L}_{bs} + \gamma \mathcal{L}_c + \eta \mathcal{L}_w. \end{aligned}$$

Compilation in tensorflow keras can be defined as (for simplicity, all the loss hyper-parameters will be set to 1 and the hyper-parameters of the side output layers will be set to 0.1 – 0.5 as the output proceeds deeper into the network):

```
losses = {
    'surface_final_output' : balanced_loss,# net.weighted_binary_crossentropy,
    'edge_final_output' : balanced_loss,# net.weighted_binary_crossentropy,
    'line_final_output' : balanced_loss,# net.weighted_binary_crossentropy
    'surface_side_output_1' : balanced_loss,
    'surface_side_output_2' : balanced_loss,
    'surface_side_output_3' : balanced_loss,
    'surface_side_output_4' : balanced_loss,
    'surface_side_output_5' : balanced_loss,

    'edge_side_output_1' : balanced_loss,
    'edge_side_output_2' : balanced_loss,
    'edge_side_output_3' : balanced_loss,
    'edge_side_output_4' : balanced_loss,

    'line_side_output_1' : balanced_loss,
    'line_side_output_2' : balanced_loss,
    'line_side_output_3' : balanced_loss,
    'line_side_output_4' : balanced_loss
}

loss_weights = {
    'surface_final_output' : 1,
    'edge_final_output' : 1,
    'line_final_output' : 1,

    'surface_side_output_1' : 0.1,
    'surface_side_output_2' : 0.2,
    'surface_side_output_3' : 0.3,
    'surface_side_output_4' : 0.4,
    'surface_side_output_5' : 0.5,

    'edge_side_output_1' : 0.1,
    'edge_side_output_2' : 0.2,
    'edge_side_output_3' : 0.3,
    'edge_side_output_4' : 0.4,

    'line_side_output_1' : 0.1,
    'line_side_output_2' : 0.2,
    'line_side_output_3' : 0.3,
    'line_side_output_4' : 0.4
}
```

```
y = {
    'surface_final_output' : labels_segments,
    'edge_final_output' : labels_edges,
    'line_final_output' : labels_centerlines,

    'surface_side_output_1' : labels_segments,
    'surface_side_output_2' : labels_segments,
    'surface_side_output_3' : labels_segments,
    'surface_side_output_4' : labels_segments,
    'surface_side_output_5' : labels_segments,

    'edge_side_output_1' : labels_edges,
    'edge_side_output_2' : labels_edges,
    'edge_side_output_3' : labels_edges,
    'edge_side_output_4' : labels_edges,

    'line_side_output_1' : labels_centerlines,
    'line_side_output_2' : labels_centerlines,
    'line_side_output_3' : labels_centerlines,
    'line_side_output_4' : labels_centerlines
}

adam = tf.keras.optimizers.Adam(lr=1e-3, beta_1=0.9,beta_2=0.999,amsgrad=True)
model.compile(optimizer=adam, loss=losses, loss_weights=loss_weights)
model.fit(train_images, y=y, epochs=EPOCHS, batch_size=BATCH_SIZE, callbacks=callbacks)
```

III. Remaining uncompleted tasks.

The following tasks are to be completed :

- Run sample training rounds on GPU to validate the effectiveness of the model.
- Implement blinear blending to match the cropped output maps into a complete map.