Follow me project final write up

Duckhwan Kim

1. Network architecture and the role of each layer/block

   In this project, I designed three different block and concatenate them to build the fully-convolutional-network.

   1) Encoder block

      Stacked encoder layers take 3 channel input image and generates 20x20x1024, which contains 2D feature information. In other words, it reduces spatial domain but increases channel domain which represent more general information (less 2D dependent) but in high dimension (channel)

      It's purely (depthwise) separable 2D convolution network with batchnorm layer. Separable 2D convolution is used to decouple spatial information (X, Y) and channel information. Also it reduces total number of parameters required in this layer.

      I modified separable_conv2d_batchnorm() little bit to have kernel_size=3 as input parameter since I want to have different kernel size for each encoder layer. Therefore, the encoder block is designed as below:

```
def encoder_block(input_layer, filters, strides,
kernel_size=3):

    # TODO Create a separable convolution layer using the
separable_conv2d_batchnorm() function.
    output_layer = separable_conv2d_batchnorm(input_layer,
filters, strides=strides, kernel_size=3)
    return output_layer
```

   2) Decoder block

      Stacked decoder blocks take the output of 1x1 convolution and convert to the original input size using skip connections. From the extracted features by encoding blocks, it will generate prediction in the same size of input image.

      It has two layers as input: one is small, and another is large. It upsample small layer (x2) and concat with large layer in channel level to implement skip connections. Therefore the decoder block is designed as below:

```
def decoder_block(small_ip_layer, large_ip_layer, filters):

    # TODO Upsample the small input layer using the
bilinear_upsample() function.
    upsampled_layer = bilinear_upsample(small_ip_layer)
    # TODO Concatenate the upsampled and large input layers
using layers.concatenate
```

```
    concat_layer = layers.concatenate([upsampled_layer,
large_ip_layer])
    # TODO Add some number of separable convolution layers
    output_layer = separable_conv2d_batchnorm(concat_layer,
filters)
    return output_layer
```

3) 1x1 Conv layer

Since it's convolution in channel domain, we cannot use separable convolution. Therefore I used conv2d_batchnorm directly as below:

```
middle1 = conv2d_batchnorm(enc3_pool, filters=1024,
kernel_size=1, strides=1)
```

For entire network, I made and tried three different version, but I will explain the last version here.

```
def fcn_model3(inputs, num_classes):

    # TODO Add Encoder Blocks.
    # Remember that with each encoder layer, the depth of your
model (the number of filters) increases.
    ## inputs: 160 * 160 * 3

    enc1 = encoder_block(inputs, filters=64, strides=1,
kernel_size=3)
    #enc1 = 160 * 160 * 64

    enc1_pool =
keras.layers.pooling.MaxPooling2D(pool_size=(2,2),
strides=None, padding='same')(enc1)
    #enc1_pool = 80 * 80 * 64

    enc2 = encoder_block(enc1_pool, filters=128, strides=1,
kernel_size=5)
    #enc2 = 80 * 80 * 128

    enc2_pool =
keras.layers.pooling.MaxPooling2D(pool_size=(2,2),
strides=None, padding='same')(enc2)
    #enc2_pool = 40 * 40 * 128

    enc3 = encoder_block(enc2_pool, filters=256, strides=1,
kernel_size=7)
    #enc3 = 40 * 40 * 256

    enc3_pool =
keras.layers.pooling.MaxPooling2D(pool_size=(2,2),
strides=None, padding='same')(enc3)
    #enc3_pool = 20 * 20 * 256
```

```
    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    middle1 = conv2d_batchnorm(enc3_pool, filters=1024,
kernel_size=1, strides=1)
    #middle1 = 20 * 20 * 512

    # TODO: Add the same number of Decoder Blocks as the number
of Encoder Blocks
    dec1 = decoder_block(middle1, enc3, filters=256)
    #dec1 = 40 x 40 x 256

    dec2 = decoder_block(dec1, enc2, filters = 128)
    #dec2 = 80 x 80 x 128

    x = decoder_block(dec2, inputs, 64)
    #x = 160 x 160 x 64

    # The function returns the output layer of your model. "x"
is the final layer obtained from the last decoder_block()
    return layers.Conv2D(num_classes, 1, activation='softmax',
padding='same')(x)
```

1) No stride, but pooling layer

   To reduce spatial information, I can set stride as 2 when 2D convolution, then the output size is reduced in 50% in both X-Y dimension. However, it skips the computation during the convolution; in other worlds, it skip extract features for a single pixel and boundary from the input. I believe it can be downside for the last encoder layers, when the input's spatial information is already small. I want to keep those information as much as I can so I compute all input pixels (stride =1), but place max pooling layer to pick the maximum activation in 2by2 radius. It also lose the information, but not the maximum value in the neighborhood.

2) Different kernel size in encoder block

   The main advantage of stacking convolution layers is extracting different levels of features. In other words, I want to extract very small features from the first layer using small kernels (like line, curve). After that, I want to extract bigger features like square or circle. To do that, I placed different size of kernels for each encoder layer. Since drone's altitude is not fixed, the object's size is not always same. To distinguish between small human or hero and object in the simulation with similar color pattern, I think extract big feature size is critical. That's why I placed different kernel size in stacked encoder. However, it increases number of computing significantly, and one epochs (batch_size = 64) takes more than 1000 seconds ETA. Therefore, I need to switch p3.x2large powered by GV100. To use GV100, I need to set different CUDA, CUDNN environment in provided AMI in AWS. It took some time but it's worthy. Training becomes really quick.

3) Skip connections.
   I connect
   a) 1x1 output & encoder_3
   b) Decoder_1 & encoder_2

c) Decoder_2 & inputs.

What I think important is bring input image to the last layer of the network; bring the information from high-dim (160 by 160) to the last layer before classification. Since each encoder reduce spatial dim half, (both X-Y dim), I think matching schemes is reasonable in skip connections

2. Network Parameter
I need to select hyper parameters in the network. These are final hyperparameters.

```
learning_rate = 0.004
batch_size = 64
num_epochs = 20
steps_per_epoch = 200
validation_steps = 50
workers = 2
```

1) Learning_rate: I started from 0.001 when num_epochs is set as 20. I tried to reduce the training loss below 0.01 in default num_epochs. Since the network is small enough, I think 200 stpes * 20 epochs is good enough for this network. Moreover, the training data is very limited; therefore increasing steps or epochs can lead overfitting. When I set 0.006 as my learning rate, I can see the fluctuation in the training loss after epoch=8. Therefore, I set 0.004 my learning rate. I still see one fluctuation during the training, but it settles down soon.

2) Batchsize: My thought process: as big as possible. As there are some research (Samuel L. SmithPieter-Jan, 2018) about large batch size is good for training accuracy, I want to make it as big as possible until GPU memory can hold them. In both p2.xlarge and p3.x2large, batchsize 64 works.

3) Num_epochs: I set 20 but I guess ~13 is good enough. I can see one loss fluctuation around epochs= 8 but it settles down soon when epoch=10. Morethan 20, I guess it may lead overfitting

4) Steps_per_epoch: Total number of training IMG is 17826. Therefore, I first tried 250 (~17826/64) but can't see any difference. So I revert to the default value. I guess epochs=20 hide the impact of steps_per_epoch.

5) Validation_steps: I didn't try/touch since it's for validation not for training.

6) Workers: I think this value is set for AWS system. I didn't try to change

In conclusion, I changed batchsize and learning rate empirically.

3. Function of 1x1 convolution and fully connected layer

It does not have any meaningful operation in spatial domain. It just multiply the kernel values and accumulate in input channel domain (channel level reduction). Using 1x1 convolution, the channel is changed, which is exact same as fully connected layer which connects [1xN] layer to [1xM] layer with [NxM] weight matrix if we assume input channel is N, output channel is M. (the total number of parameters in 1x1 kernel is also 1*1*N*M = N*M same as NxM weight matrix in FC layer).

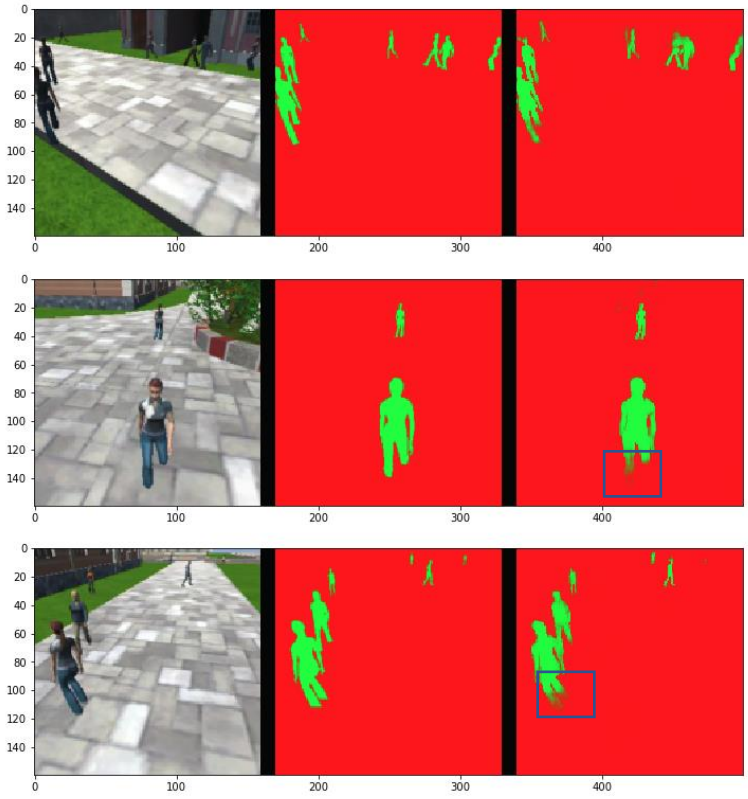
In this problem, we use 1x1 convolution at the end of stacked ender layers since we need to maintain spatial information. In the code, middle1 layer's input is enc3_pool: (20 x 20x 256). If we use FC layer, input is changed to a single vector, which length is 102,400 and output will become the vector which length is 409,600. Since input and output is 1D vector, it loses 2D information, which is critical in our project. Therefore, we used 1x1 convolution. Although number of information in input, output and weight is same, it can keep 2D information.

4. Final score = 0.59199 > 0.40

```
In [18]: # Scores for while the quad is following behind the target.
         true_pos1, false_pos1, false_neg1, iou1 = scoring_utils.score_run_iou(val_following, pred_following)

         number of validation samples intersection over the union evaulated on 542
         average intersection over union for background is 0.9969256515523762
         average intersection over union for other people is 0.46514603306997987
         average intersection over union for the hero is 0.9475759273610485
         number true positives: 539, number false positives: 1, number false negatives: 0

In [19]: # Scores for images while the quad is on patrol and the target is not visable
         true_pos2, false_pos2, false_neg2, iou2 = scoring_utils.score_run_iou(val_no_targ, pred_no_targ)

         number of validation samples intersection over the union evaulated on 270
         average intersection over union for background is 0.9828753401743201
         average intersection over union for other people is 0.6398523984169983
         average intersection over union for the hero is 0.0
         number true positives: 0, number false positives: 39, number false negatives: 0

In [20]: # This score measures how well the neural network can detect the target from far away
         true_pos3, false_pos3, false_neg3, iou3 = scoring_utils.score_run_iou(val_with_targ, pred_with_targ)

         number of validation samples intersection over the union evaulated on 322
         average intersection over union for background is 0.9958473450077014
         average intersection over union for other people is 0.45194692454706975
         average intersection over union for the hero is 0.4506669951604881
         number true positives: 207, number false positives: 1, number false negatives: 94

In [21]: # Sum all the true positives, etc from the three datasets to get a weight for the score
         true_pos = true_pos1 + true_pos2 + true_pos3
         false_pos = false_pos1 + false_pos2 + false_pos3
         false_neg = false_neg1 + false_neg2 + false_neg3

         weight = true_pos/(true_pos+false_neg+false_pos)
         print(weight)

         0.8467650397275823

In [22]: # The IoU for the dataset that never includes the hero is excluded from grading
         final_IoU = (iou1 + iou3)/2
         print(final_IoU)

         0.699121461261

In [23]: # And the final grade score is
         final_score = final_IoU * weight
         print(final_score)

         0.591991611919
```

5. Limitation: Can it detect dog, cat, car, etc.
   It can't since it never learnt! To detect cat, dog instead of human we need mask which indicates cat, dog and appropriate label. Since output of network dimension is same as number of classifications, the network size will increase due to the last layer.

6. Future improvements
    1) How to predict better (higher IOU?) Currently, I am not sure how use depth camera in this project. However, I can see we have information from the drone. It will be difficult to distinguish between human and hero using depth camera, but it will be helpful to distinguish between hero and road pavement, which is similar color.



References

Samuel L. Smith, P.-J. K. (2018). Don't Decay the Learning Rate, Increase the Batch Size. *ICLR*.