Follow Me

Myoungki Jung

November 5, 2017

1 RUBRICS

1. Provide a write-up / README document including all rubric items addressed in a clear and concise manner. The document can be submitted either in either Markdown or a PDF format.
    - You are reading the documentation.
    - Section 3.5
2. The write-up conveys the an understanding of the network architecture.
    - FCN (2.1)
    - Encoder (2.1.1)
    - Decoder (2.1.2)
3. The write-up conveys the student's understanding of the parameters chosen for the the neural network.
    - General parameter selection (3)
    - run 0 to 2 (3.1)
    - run 3 (3.2)
    - run 4 (3.3)
4. The student has a clear understanding and is able to identify the use of various techniques and concepts in network layers indicated by the write-up.
    - 1X1 convolution layer (2.1.1)

- Fully connected layer (2.1.1)

5. The student has a clear understanding of image manipulation in the context of the project indicated by the write-up.
   - Image manipulation (3.1)
6. The student displays a solid understanding of the limitations to the neural network with the given data chosen for various follow-me scenarios which are conveyed in the write-up.
   - limitations (4.2)
7. The model is submitted in the correct format.
   - weights (weights directory)
8. The neural network must achieve a minimum level of accuracy for the network implemented.
   - 43% (3.1)

2 Neural Network Architecture

2.1 Fully Convolutional Neural networks

The use of fully convolutional network allowed to conduct scene segmentation for images. Previously, convolutional neural networks were used in classification of images such as MNIST, and Imagenet. However, these normal convolutional network only supports in classification. The type of learning for this project is supervised learning, using sized images and masks for the target object in the scene.
Yann LeCun stated that:

> In Convolutional Nets, there is no such thing as âĂIJfully-connected layersâĂİ. There are only convolution layers with 1x1 convolution kernels and a full connection table.

It is essential an autoencoder replaced the middle fully-connected layers with 1x1 convoluion layer.

2.1.1 encoder

```
def encoder_block(input_layer, filters, strides):

  output_layer = separable_conv2d_batchnorm(input_layer, filters, strides)

  return output_layer
```

Listing 1: Endcoder block code

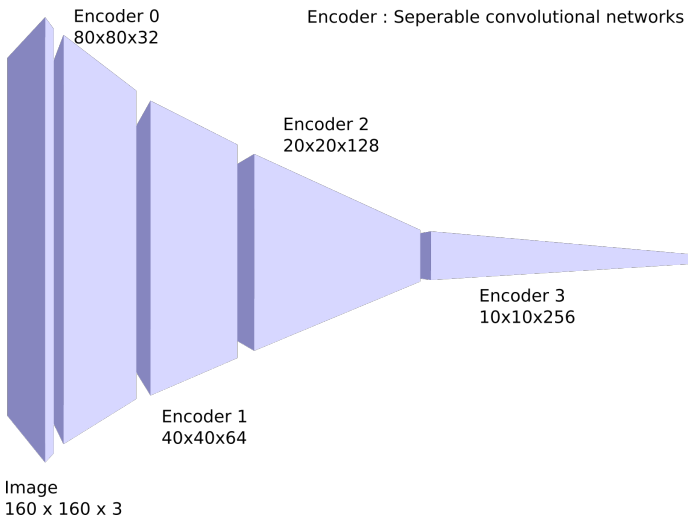


Figure 2.1: Fully connected convolutional network used in the project

Encoder consists of four layers of neural network modules and each module consists of a separable convolution layer. Later a conv2d_batchnorm with the same filter size was applied in the FCN model. Each layer of this network act as the shared spatial filter to recognise objects. The deeper layers contain a filter for more complex features, the shallower layer contains simpler features of objects.

ENCODING Once this parts are trained this encodes the image into the input to the 1x1 convolution layer, which contains the activated weights of the image. During the encoding process, some information may be lost due to the coarse strides of convolution or pooling layer and the type of pooling (average, max).

SEPERABLE CONVOLUTION The separable convolutional network is the key part which reduces the number of parameters needed to increase efficiency for the encoder network. The seperable convolutional network includes a convolutional network, and an activation function. The function used in the encoder module is class SeparableConv2DKeras and it performs depthwise spatial convolution and a pointwise convolution and controls output depth. In short, it is a way to factorize a convolution kernel into two smaller kernels.

WHY NOT FULLY CONNECTED LAYER? Fully connected layer is usually used to evaluate the neural network weights before sending it to activation layer, for classification classification. The output of this layer loses the spatial information contained in the convolutional networks and this cannot be used in our application as the network needs to contain the spatial network information to reproduce the scene segmentation as output. 1x1 convoluion layer is used as a pivot point to decode the weights of the encoder to generate scene segmentation images which perceived by convolutional neural networks in the encoder.

1X1 CONVOLUTION Google used this in their Inception paper to reduce the computational load but still make the network deeper.

> in our setting, 1x1 convolutions have dual purpose: most critically, they are used mainly as dimension reduction modules to remove computational bottlenecks, that would otherwise limit the size of our networks. This allows for not just increasing the depth, but also the width of our networks without significant performance penalty.

Aaditya Prakash also discussed 1x1 convolution in his github blog that this convolution is a strictly linear coordinate-dependent transformation in the filter space which leads to dimensionality reduction and increase in depth of network. It is also less likely experiencing over-fitting because of its small 1x1 kernel size.

2.1.2 DECODER

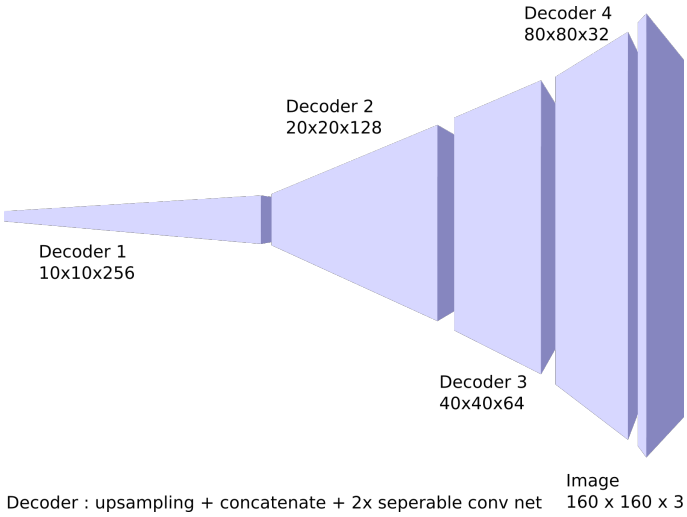


Figure 2.2: Fully connected convolutional network used in the project

Decoder is more complicated than the encoder. This block consists of an upsampling layer, concatenate filter, and two separable_conv2d_batchnorm layers.

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

  upsampled_output_layer = bilinear_upsample(small_ip_layer)

  output_layer = layers.concatenate([upsampled_output_layer,large_ip_layer])

  output_layer = separable_conv2d_batchnorm(output_layer, filters)
  output_layer = separable_conv2d_batchnorm(output_layer, filters)

  return output_layer
```

Listing 2: Decoder block code

UPSAMPLING The upsampling layer uses BilinearUpSampling2D, which repeats the rows and columns of the data, in this case 2x2. This upsampling increases the size of the matrix back to original output size by applying the same number of separable_conv2d_batchnorm in the encoder block.

CONCATENATE tensorflow's concatenate connects the output layer of upsampling function and the equivalent sized encoder block.

SEPARABLE CONVOLUTION is the same as ones in the encoder blocks but these layers will be directly linked to the encoder equvialents and allows skip connection operations.

DECODING Decoding of an image from the information inside 1x1 convolution layer is done by multiple upsampling.

2.1.3 FULLY CONNECTED CONVOLUTIONAL NETWORK

FCN consists of multiple blocks of encoder and decoder, and 1x1 convolution layer. Differently from the normal convolutional neural networks, a seperable network links a pair of enconders and decoders and creates sharper image output as shown in the lesson.

CHOSEN ARCHITECTURE The Architecture of the Neural network is Fully connected convolutional network (FCN). This network can decode the weights in the newural network back to an image, which shows the true and false in pixels. This decoding capability enables the scene segmentation, identification of target object in this project.

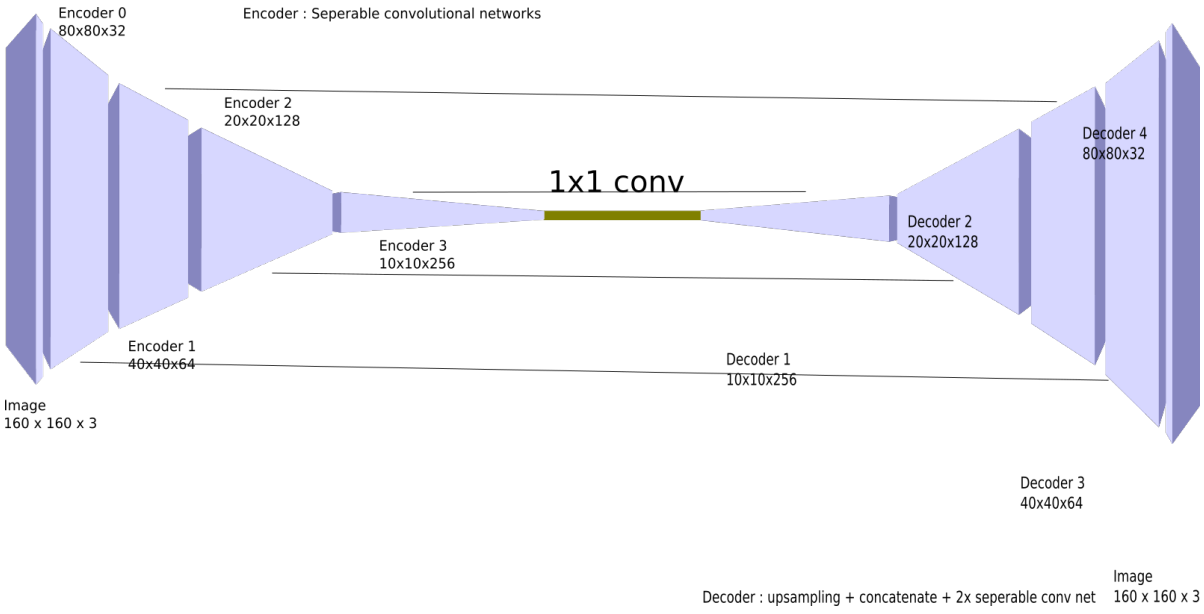


Figure 2.3: Fully connected convolutional network used in the project

The gold colour component in Figure 2.3, 1x1 convolution layer was used to make it as an adaptor filter between encoder and decoder module beacuse this filter stil conveys weights with the spatial data in the previous filters, unlike the classifer losing all the data and flattened, and only reduce dimension of the matrix, which removes computational bottlenecks. 1x1 convolution was used after 'relu' layer of 'SeparableConv2DKeras' to reduce the dimensions and to prepare for connections to following seperable convolution layers.

```
def fcn_model(inputs, num_classes):

    filter= 32

    encoded_0 = encoder_block(inputs, filter, 2)
    encoded_0 = conv2d_batchnorm(encoded_0, filter, kernel_size=1, strides=1)

    encoded_1 = encoder_block(encoded_0, filter*2, 2)
    encoded_1 = conv2d_batchnorm(encoded_1, filter*2, kernel_size=1, strides=1)
    encoded_1 = conv2d_batchnorm(encoded_1, filter*2, kernel_size=1, strides=1)

    encoded_2 = encoder_block(encoded_1, filter*2*2, 2)
    encoded_2 = conv2d_batchnorm(encoded_2, filter*2*2, kernel_size=1, strides=1)
    encoded_2 = conv2d_batchnorm(encoded_2, filter*2*2, kernel_size=1, strides=1)

    encoded_3 = encoder_block(encoded_2, filter*2*2*2, 2)
    encoded_3 = conv2d_batchnorm(encoded_3, filter*2*2*2, kernel_size=1, strides=1)
    encoded_3 = conv2d_batchnorm(encoded_3, filter*2*2*2, kernel_size=1, strides=1)

    oneToOne = conv2d_batchnorm(encoded_3, filter*2*2*2, kernel_size=1, strides=1)

    decoded_0 = decoder_block(oneToOne, encoded_2, filter*2*2*2)
    decoded_1 = decoder_block(decoded_0, encoded_1, filter*2*2)
    decoded_2 = decoder_block(decoded_1, encoded_0, filter*2)
    decoded = decoder_block(decoded_2, inputs, filter)

    return layers.Conv2D(num_classes, 1, activation='softmax', padding='same')(decoded)
```

Listing 3: Fully convolutional network code

The function conv2d_batchnorm was applied after each encoder block to facilitate convergence of model by making the weights more normalised.

3 TRAINING

Training is all about how to lead the model to the global minima of loss. The models were trained to the point where the slope of loss is less than 0.00001, assuming the model had arrived to a local minima of loss function. With trained model, the performance of model was gauges with the intersection over union rating (IoU) as noted by instruction of the project manual.

PREPARATION Installing tensorflow from source code was an essential part before using tensorflow to improve the efficiency and utilisation of GPU and CPU as the precompiled version did not optimised for each computer and ommitted many CPU and GPU instruction features like MMX, SSE. This instruction was well documented in Tensorflow.org. After installation of the wheel installation package, optimised for the local machine, follow the project instruction provided by Udacity to download and isntall dependency packages for python.

PARAMETERS The parameters used to tune the network and comments for them are shown below.

1. Filter size: 32. this filter size is relatively small to easily process on my GPU.
2. Num of Epoch: the bigger epoch number, the more iteration of training for the neural network - this value was set as high as possible, more than 20.
3. Learning Rate: learning rate controls how stably converge to the minima of loss function, by experimenting 0.0005 was reasonable for this network. If more epoch are available, this value can be decreased futher by magnitute. Although it would take longer time to train, the weights of neural networks will be finely tuned as the smaller learning rate can help optimiser to converge to the minima. Therefore, this value can be decreased when the neural network is about to converge after multiple training sessions.
4. Batch Size: batch size was constrained by the specification of the local machine. The size of VRAM of my GTX960 could not allocate massive memory blocks, which created by batch size higher than 32, especially for this 7 layer network. Smaller batch size will allocate/ disallocate memory blocks more frequently than the higher ones and, consequently, add more training time for that memory operations. A mini batch setting with 8 or 16 enabled my GPU process the neural network on a local machine. In addition, the lessons in udacity re
5. steps per epoch: due to the randomisation in training data, providing the entire data set is not neccessary. The randomisation fosters a balanced training compared with sweeping entire batch because such method provides higher possibility to develop the weights.
6. validation steps: the number of iteration to complete a validation. The more validation, better reliability of the validation, this needs to be more than 32 to get a proper statistical mean.

parameter	run0	run1	run2.1	run2.2	run2.3	run2.4	run 2.5	run2.6
learning rate	0.001	0.0005	0.0005	0.0005	0.0005	0.0005	0.00025	0.0005
batch size	8	8	8	8	8	8	8	8
num of epoch	50	25	10	10	10	100	20	20
accumulated trained epoch	50	75	85	95	105	205	125	125
steps per epoch	32	64	512	32	32	32	64	32
validation steps	16	32	256	16	16	16	32	16
workers	2	2	2	2	2	4	4	4
number of training sets	4131	4131	4131	4131	4131	5250	4131	4131
input weights	new	1	1	2_1	2_2	2_3	2_3	2_3
output weights	0	1	2_1	2_2	2_3	2_4	2_5	2_6
iou hero close	0.863	0.912	0.920	0.917	0.916	0.903	0.920	0.911
iou hero distant	0.084	0.157	0.141	0.180	0.239	0.181	0.165	0.217
final score	0.331	0.391	0.387	0.411	0.432	0.405	0.401	0.407

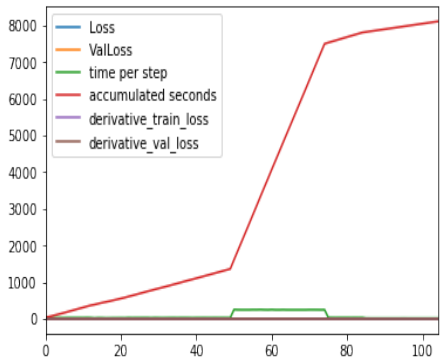
Table 3.1: The parameters and validation results of Run 0 to 2.6

3.1 RUN 0 TO 2.6

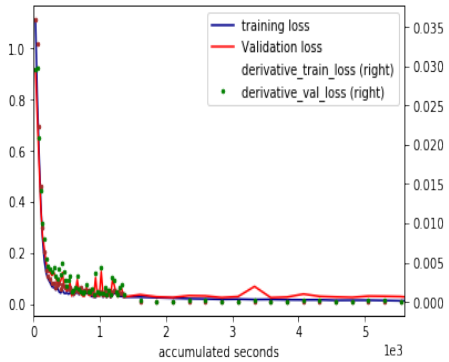
Table 3.1 shows the hyper parameters and results of each run. A notable one is that run 2.3 is the split point for experiment of using more collected data (run2.4: modeltrainingedit24.html) and (run2.5:modeltrainingedit25.html). The major parameter in this table is the learning rate which is the most sensitive factor to decide the final convergence of weight after iterations of trining. Other parameters related to the training sample batch size and training steps in an epoch merely decides the model.
run 0 to run 2.2 was further training to get a local minima for the training with the same training parameters. The training parameters and environments diverges from run 2.2. Run 2.3 was to train the output weights of 2.2 with more epoch with the same parameter as run 2.2. More collection of data was introduced to run 2.4 and run2.5 was trained the output of run 2.4 with the same parameter as Run 2.4.Run 2.5 is based on the weights of run 2.3 and the only difference is double the size of 'steps of epoch' and 'validation steps' and Run 2.6 is further trained weight of run 2.5.

RUN2.3 The highest score (43%) in Run 2 series were recorded with Run 2.3. The total traine

RUN2.4 run 2.4 was trained with additional 1 thousand images from the simulator, especially aimed to get the distant hero scenes on high default altitude, and crowds' eye height. However, Run 2.5 added more training with extra epoch and validation steps with a lower learning rate (0.00025) to relieve the oscilation of adam optimiser observed in training runs between 0 to 2.3. run2.4 with new collected data trained for 100 epochs showing seriously many spikes. validation data has different contents and the training data. A change in used training data set did not affect the performance of neural network positively. Additional 1200

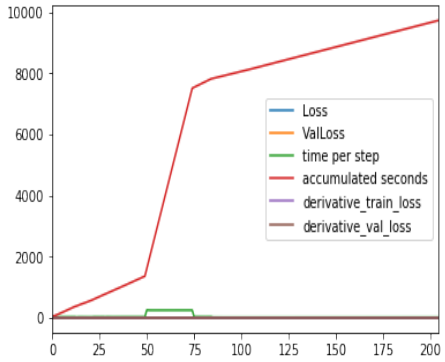


(a) Analysis 1 of run2.3

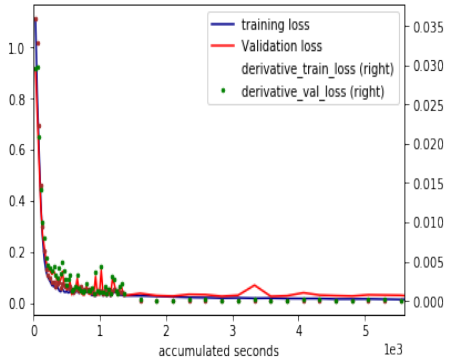


(b) Analysis 2 of run2.3

Figure 3.1: Analysis for run 2.3

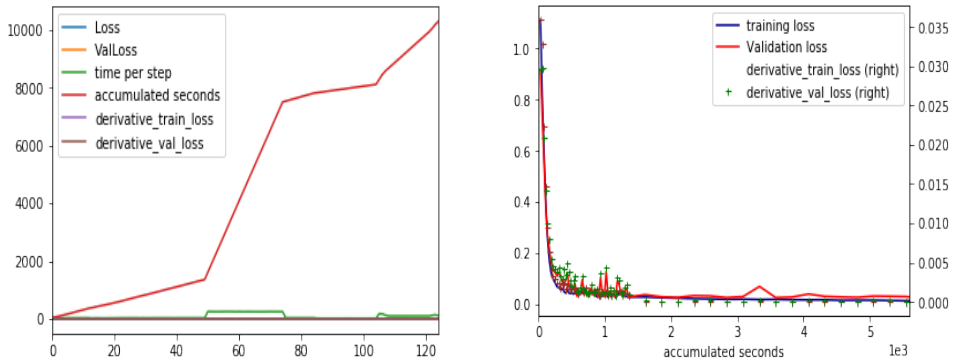


(a) Analysis 1 of run2.4



(b) Analysis 2 of run2.4

Figure 3.2: Analysis for run 2.4



(a) Analysis 1 of run2.5

(b) Analysis 2 of run2.5

Figure 3.3: Analysis for run 2.5

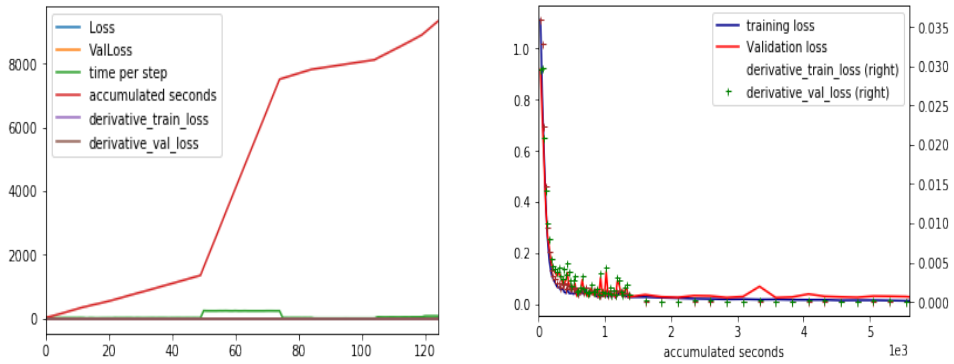
images were collected in simulator specificaly targetting the heroin in distance settings. Run 2.4 included additional 1230 images to train and validate. The final score went down by 27% due to the changes of neural network weights by the newly introduced dataset. Also, run 2.5 shows a weight corruption with even lower overall performance than run2.4 in both distant and proximity and it needs more training iteration to stablise it. When a data set is injected to the training set, the collection of data must be carefully selected. Run 2.5 with new collected data trained for 20 epochs on learning rate 0.0025 slight changes and some spikes for the same reason. both are not performing well in IoU tests.

IMAGE MANIPULATION A image manipulation script provided by the project repository was used to pre-process the images captured in the simulator to feed into neural network. In this supervised learning scenario, preparation of input for the neural network is essential. The raw simulation data was in jpeg format with rgb colour space, and the neural network needed to feed inputs as the resized images and a mask of the target object, in this case a girl in red outfit.

RUN2.5 AND RUN2.6 Extra training after arriving at a local minima of loss does not improves the overall performance. The oscillation of optimiser can corrupt a well performing model with extra training. A mechanism to stop the training at the best performing model is required. Figure 5.4 in 5 shows the slight deterioration in target reocgnition and slight false positiveness in model weights.

3.2 RUN 3

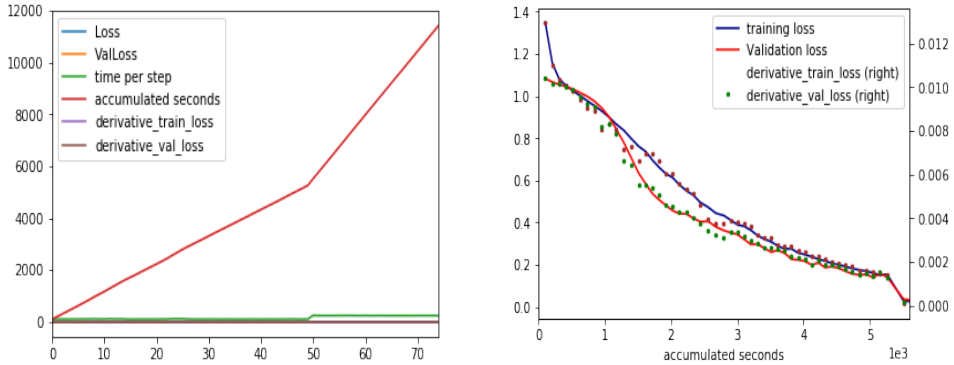
After observing the effect of additional training data sets at run 2.4 and further training of weights, it was a time for training new model with new learning rate. Using lower learning rate, it will be slower to converge to a minima, but it will be lower than the one with higher learning rate.



(a) Analysis 1 of run2.6

(b) Analysis 2 of run2.6

Figure 3.4: Analysis for run 2.6



(a) Analysis 1 of run3

(b) Analysis 2 of run3

Figure 3.5: Analysis for run3

PARAMETERS Table 3.2 shows the new parameter sets to induct a lower global minima, instead of stuck in a local minima with higher learning rate. The only difference with run0 is the learning rate of one tenth of that in run 0. With this learning rate, a slower learning rate was observed in Figure 3.5 and it achieved its own local minima to a half of the previous local minima. As can be seen in the file modeltrainingedit3.html and modeltrainingedit31.html, the output of FCN shows much traces random wavy stains (Figure 5.5), which indicates that the learning rate was too small to overcall the randomised weights after 50 times of epoch. After additional 25 epochs, these wavy stains in the output was removed (Figure 5.6), but still the overall performance was not as great as the result of run 2.3.

3.3 RUN 4

A new learning rate which is half of run0 to run2 and five times bigger than run 3 was experimented and the hyperparameters are shown Table 3.3. The majority of hyper parameters

parameter	run3	run3.1
learning rate	0.0001	0.0001
batch size	16	8
num of epoch	50	25
accumulated trained epoch	50	75
steps per epoch	32	64
validation steps	16	32
workers	2	4
number of training sets	4131	4131
input weights	new	3
output weights	3	3_1
iou hero close	0.802	0.751
iou hero distant	0.0402	0.018
final score	0.259	0.252

Table 3.2: The parameters and validation results of Run 3

except learning rate was changed compared to run 3.

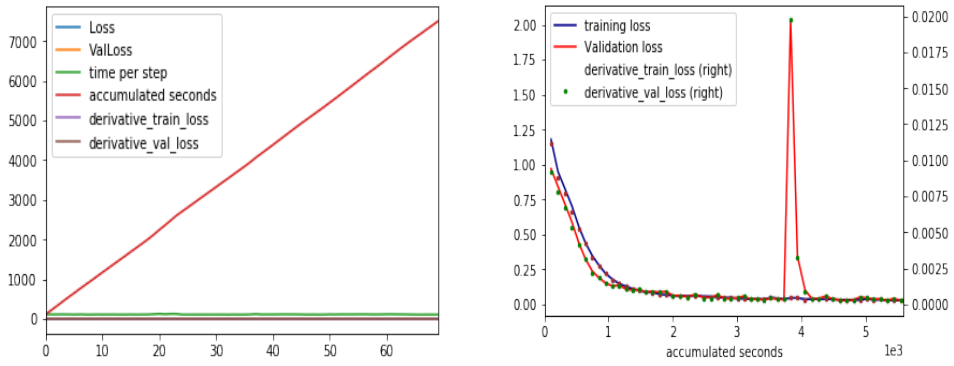
PARAMETERS In this run, the convergence to a local minima was faster than Run 3 and more stable than Run 0 to run 2. However the lower loss at 0.025 did not provide an accurate performance, instead it seems to be overfitted to the data. There is a sharp rise in loss at epoch 37 on the log graph for run 3. It does not mean that a weight set before epoch 37 may perform better and objective than the ones after epoch 37 - it could be less trained and the loss spike can be random near the local maxima. This implies the model was well overfitted to a part of the training set and it output a huge loss when it encountered the part of validation set which is totally different from the training set it was heavily overfitted. A dropout is needed to reduce such overfitting. Figure 3.6 shows a gradual faster learning curve than run 3 but it is more gradual than run 2. It is notable that a spike in the validation and train loss, which occured after first potential convergence to a local minima. This might be explained that the random sampling of the input was not well dispersed and be trained the neural network with only with biasedly sampled training batches. Figure 5.7 shows much more cleaner output of scene segmentation than using the weights of run 3.

3.4 CONCLUSION

The selection of hyper parameter was the most important part. It can take a long time to verify the right parameters and optimise them further. As each training session of a neural network until outputing a meaningful performance takes long time and computatively intensive, parameter matrix method used in the last project to train support vector machine (Figure 3.7) is not applicable unless owning a massive computation resource. The important take way from the training section is to start from a resonable hyper parameters and infer the

parameter	run4	run4.1	run4.2
learning rate	0.0005	0.0005	0.0005
batch size	16	16	8
num of epoch	50	20	20
accumulated trained epoch	50	70	90
steps per epoch	32	32	32
validation steps	16	16	16
workers	2	4	4
number of training sets	4131	4131	4131
input weights	new	4	4_1
output weights	4	4_1	4_2
iou hero close	0.623	0.875	0.808
iou hero distant	0.009	0.091	0.032
final score	0.202	0.318	0.275

Table 3.3: The parameters and validation results of Run 4



(a) Analysis 1 of run4

(b) Analysis 2 of run4

Figure 3.6: Analysis for run4

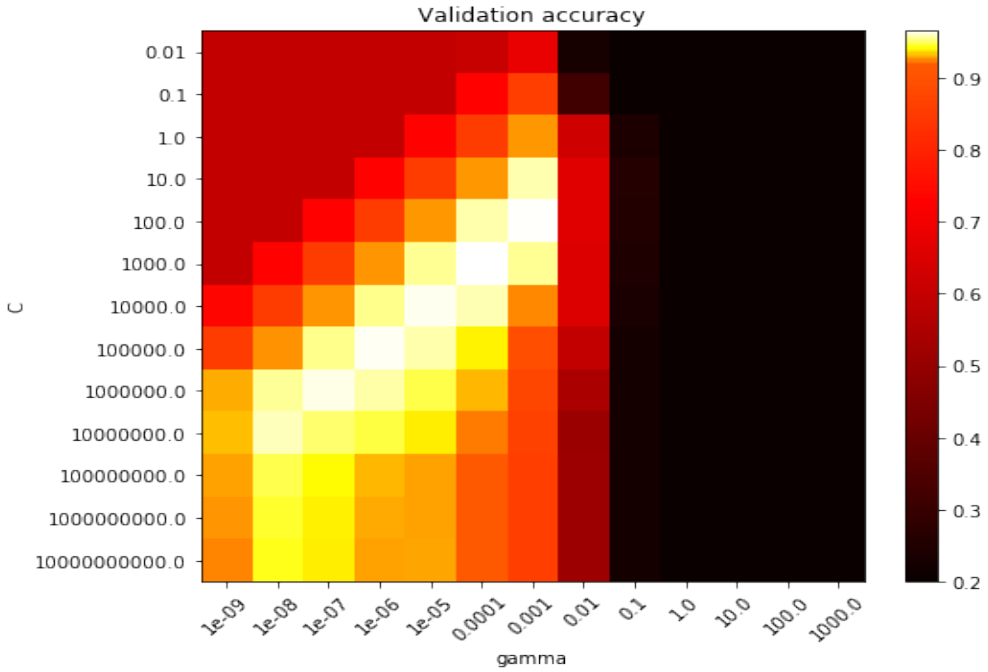


Figure 3.7: Analysis for run4

effectiveness from the result.

3.5 Future Enhancements

Different architecture One of my thought based on reviewing the scoring method of this project, the it might be more plausible to have two different streams of FCN to identify hero in distant and close ranges because mostly the model suffers inaccuracy in identification of distant objects. This is because of the single stream architecture FCN is more likely to specialised object recognition in proximity as the the first few layers of weights recognise the simple features of the heroin and the overall shape is recognised by the later layers. However, the shape of heroin does not have much details and size of is as small as the simple features can be detected by weights in the first layer of the network. To circumvent this, train another stream of FCN specialised for recognising heroin in distance by setting smaller filter sizes and feeding only training data sets with heroin in distance with other various objects and environment. By merging it carefully, actually this is the challenging part to do so and have incepeted the solution yet, the dual stream FCN may be able to recognise obejects both in proximity, and distance. Another recommended arthitectural change is to embrace an autoencoder before the 1x1 convoluion layer. With an autoencoder, model can learn without supervision, providing that a well sorted inputs are only fed into neural networks, preprocessing will not be neccessary.

MORE DATA COLLECTION Due to the data deficiency in the provided data, especially the variety of scenes with a hero in distance, a well planned data acquisition is needed as shown in the data collection failure case of run 2.4 and run 2.5.

DIFFERENT PARAMETERS At the end of writing this report, I realised that filter was 32 for all the experiments. I believe this could be one of the reasons why small objects, tiny image of hero in distance, could not be detected as good as the big objects. By increasing this filter size to 32, or even 64, small hero images might be detected better.

4 Analysis

4.1 Advantage of the model

With parameters supplied, this model can converge into a reasonable local minima within 1 hour to get the 43% accuracy detection performance of he test data. This is a light and simple FCN also many rooms to improve too.

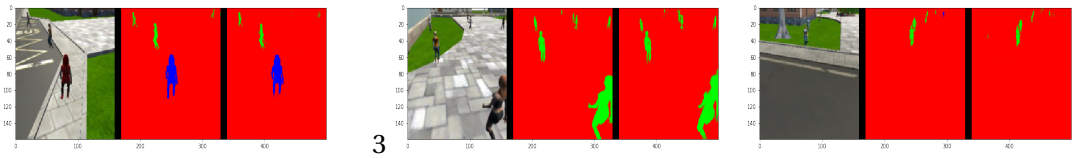
4.2 Limitation of the model

General purpose? Definately not enough training for many different objects, the training was limited to the object spawned in the simulated area. There is no animals, no cars. To make this recognised, more data sets are required. NVIDIA is conducting such general purpose data collection in smart cities by using cameras installed in many locations in there. In addition, the size of neural network should become larger than this project to classify many more objects and this requires massive cloud system infrastructure to process the collected data. Many people trying to make a general purpose classifer, however, only neural network itself might not be a model to do so becuase of its training time, model requirement, and the sensitivity of model to training data. There must be some more components to be added to create the novel generalised classifier.

Other style of processing Data collection never represents the reality perfectly. The conditions for operation changes with demands, and surroundings. That is a model with a fixed weights cannot adapt such dynamic environment in the realy world and performs well. Instead of setting the weights in a batch methods at once, an online learning is highly recommended, which learns the target and modify the model at operation.

5 APPENDIX

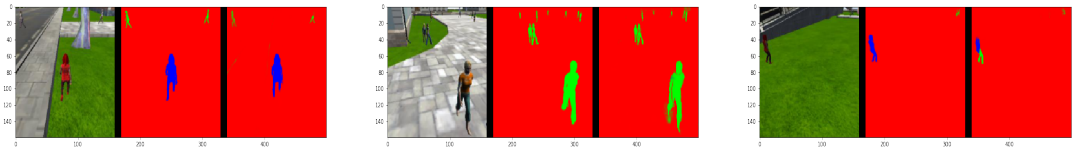
5.1 IMAGES



3

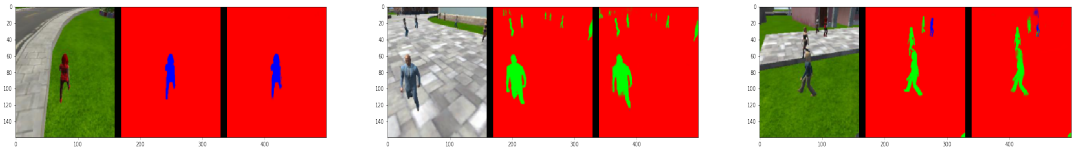
(a) following images of run2.3 (b) patrol without target of run2.3 (c) patrol with target of run2.3

Figure 5.1: validation images of run2.3



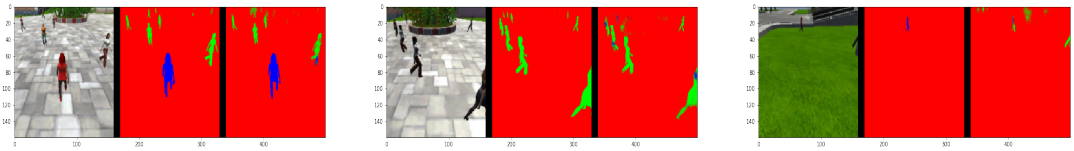
(a) following images of run2.4 (b) patrol without target of run2.4 (c) patrol with target of run2.4

Figure 5.2: validation images of run2.4



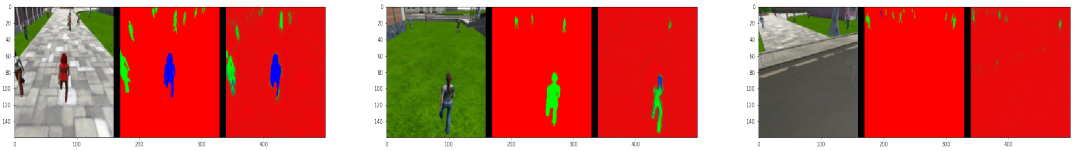
(a) following images of run2.5 (b) patrol without target of run2.5 (c) patrol with target of run2.5

Figure 5.3: validation images of run2.5



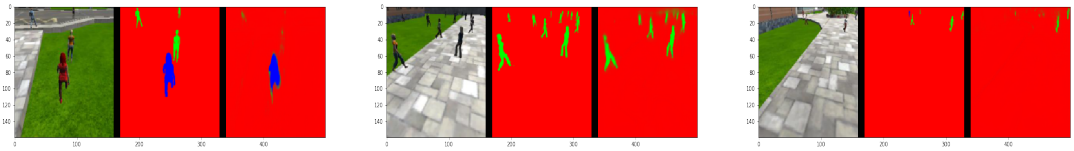
(a) following images of run2.6 (b) patrol without target of run2.6 (c) patrol with target of run2.6

Figure 5.4: validation images of run2.6



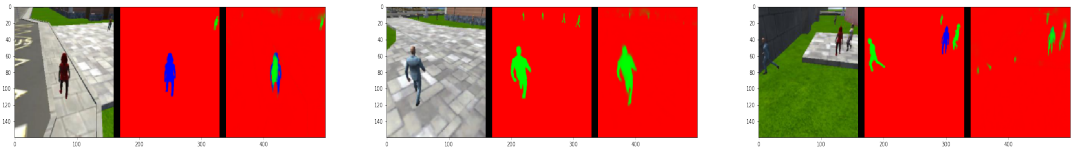
(a) following images of run3 (b) patrol without target of run3 (c) patrol with target of run3

Figure 5.5: validation images of run3



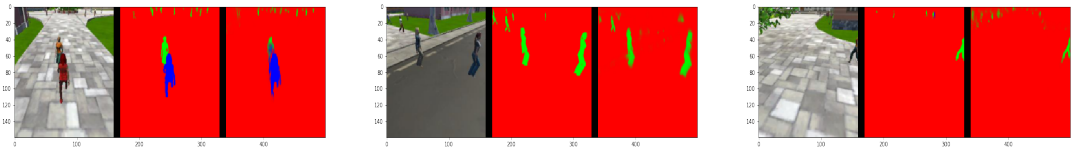
(a) following images of run3 (b) patrol without target of run3 (c) patrol with target of run3

Figure 5.6: validation images of run3



(a) following images of run4 (b) patrol without target of run4 (c) patrol with target of run4

Figure 5.7: validation images of run4



(a) following images of run41 (b) patrol without target of run41 (c) patrol with target of run41

Figure 5.8: validation images of run41