Udacity Robotics Nano-Degree

Follow Me Project

Fergal Toohey

Abstract

The objective of the Follow Me project was to create, and train, a fully convolutional network model, such that a target “Hero” person could be identified in simulated images of a city, via image segmentation, with an IOU of greater than 0.40.

The model and weights would then be fed to a simulated drone controller, to allow it to identify and follow the target.

This report outlines a successful implementation of the project, with discussion of the additional topics required in the rubric. The final Jupyter Notebook, and resulting weights file, are included alongside this submission.

Table of Contents

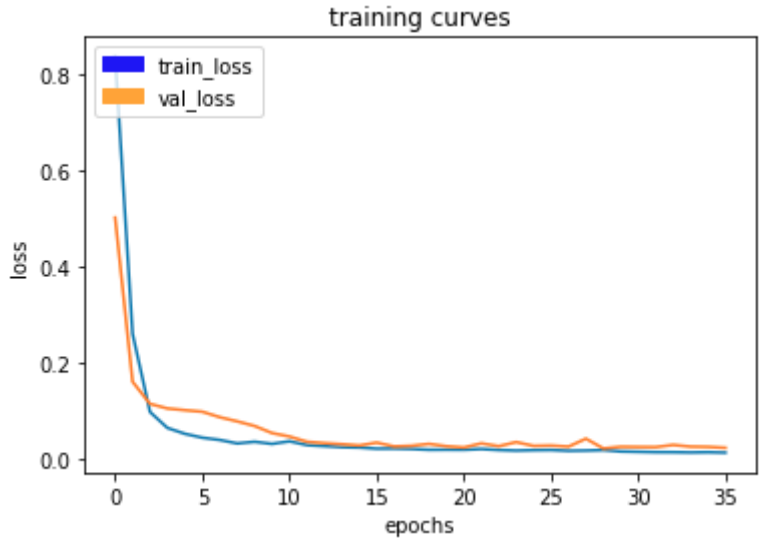
Result

Final score

0.435.

Detailed score breakdown

Model 35, epoch 35: (Four-deep Encoder/Decoder, dual separable convolutions in Decoder)



Training Loss

0.0147

Validation Loss

0.0248

Scores for while the quad is following behind the target

Number of validation samples intersection over the union evaluated on: 542
Average intersection over union for background: 0.995310339307041
Average intersection over union for other people: 0.36041863777158206
Average intersection over union for the hero: 0.909416627440122
Number true positives: 539, Number false positives: 0, Number false negatives: 0

Scores for images while the quad is on patrol and the target is not visible

Number of validation samples intersection over the union evaluated on: 270
Average intersection over union for background: 0.9870851211041798

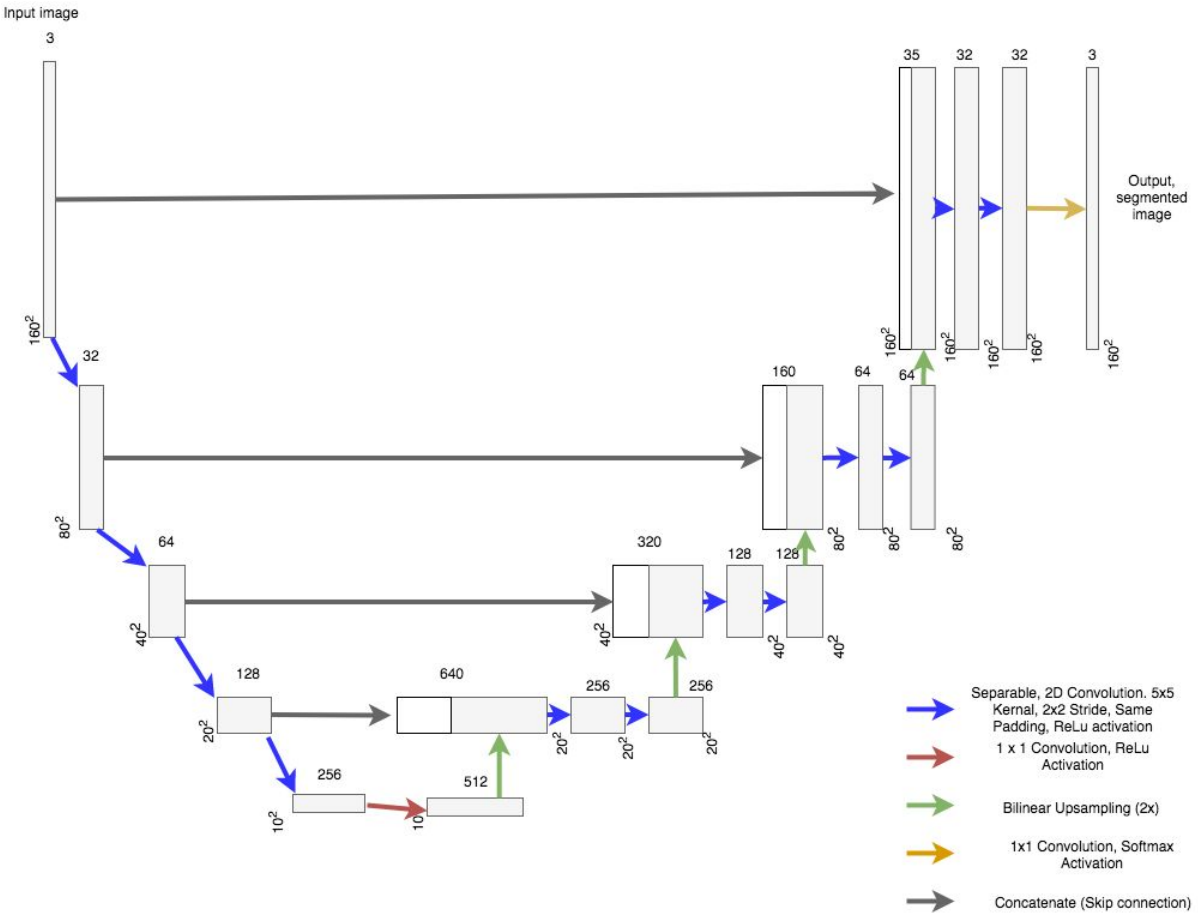
Average intersection over union for other people: 0.760285338640684
Average intersection over union for the hero: 0.0
Number true positives: 0, Number false positives: 21, Number false negatives: 0

Score while the target is far away

Number of validation samples intersection over the union evaluated on: 322
Average intersection over union for background: 0.9963182760568456
Average intersection over union for other people: 0.44708361969628296
Average intersection over union for the hero: 0.2212266893764725
Number true positives: 125, Number False positives: 1, Number false negatives: 176

Architecture

The final network architecture was based on the Encoder-Decoder approach of image segmentation, with a depth of four blocks, with skip connections:



Inputs and outputs

The inputs were 160 x 160 images, with three colour channels.

The output map was similar: 160 x 160 images, with three segmentation classes.

Fully Convolutional Network

The Encoders in a Encoder-Decoder FCN model serve to reduce the spatial complexity of the image, whilst greatly increasing its semantic complexity. The Decoders then increase the spatial resolution of the resulting output, whilst also simplifying the semantic representation.

Encoder

The input was first processed with four encoder blocks.
Each encoder block consisted of a single two-dimensional, Separable Convolution, with a Kernel Size of 5x5, a Stride of 2x2, Same padding, and Relu activation. Output depth was 32, 64, 128, 256, respectively.

The output of the encoders was then passed through a 1 x 1 regular convolution layer (both Kernel and Stride of 1), with a depth of 512. The 1x1 convolution at this point serves as a 512 layer, 10x10 size, encoding of the features of the image. At this point the spatial information has been squeezed from 160x160 to 10x10, by all the proceeding encoders.

(Note: at this point, if categorisation, and not segmentation was desired, a fully connected layer could have been used to remove all remaining spatial information and output probabilities of classes, skipping the following Decoder section.)

Decoder

Four blocks of decoding layers were then applied. Each decoder block received the output of the preceding layer, and applied 2x bilinear upsampling. The result was then concatenated with the output of the equivalent Encoder Layer, as illustrated by the skip connections in the diagram. The combination of upsampling and skip connections serve to increase the spatial resolution of the features. The combined outputs were then passed through two separable convolutional layers, again with Kernel Size of 5x5, a Stride of 2x2, Same padding, and Relu activation. The output depth was 256, 128, 64, and 32, respectively.

Output

The output from the final decoder layer was passed through a final 1x1 convolutional layer, with a depth of 3 and same padding, and finally Softmax activation to produce the pixelwise classification of the image.

This final convolution condenses the feature information from 32 feature maps, to 3 probablistic maps of the features that need to be identified (background, other people, hero).

Convolutions used

Depthwise separable convolutions are used in the Encoder and Decoder layers primarily for efficiency reasons, as they have vastly fewer parameters. Taking the first encoding layer as an example:

The image would have three input channels, traversed by 32 kernels of size 5x5, resulting in 2400 (5x5x3x32) parameters using a regular convolution.
In contrast, a separable convolutions first perform convolution on each input layer, and then combines these outputs with a 1x1 convolution. Each of the 3 input channels each get traversed by a 5x5 kernel, and these 3 feature maps get traversed by 32 1x1 convolutions, resulting in 171

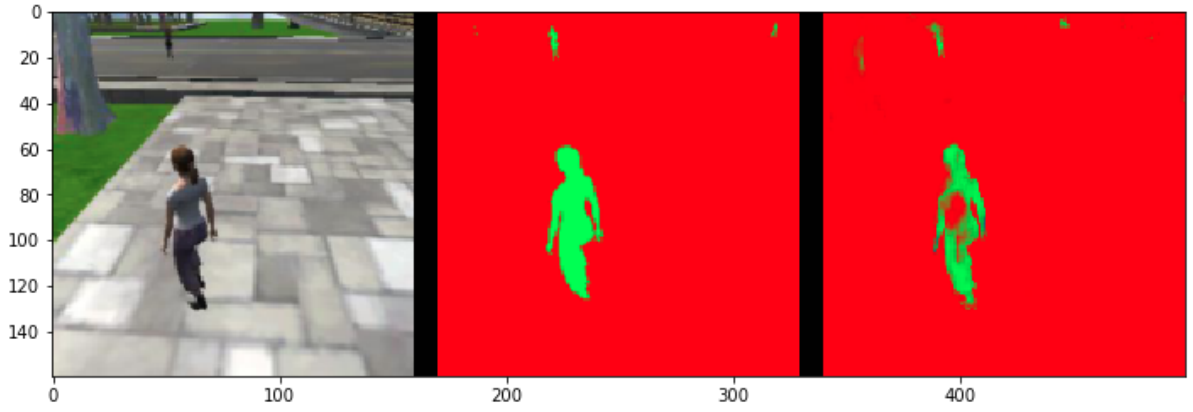
parameters (5x5x3 + 32x3).

Batch Normalisation was applied to the outputs of each convolution in the model, with the exception of the final output segmentation map, to ensure the weights were kept within their optimum range.

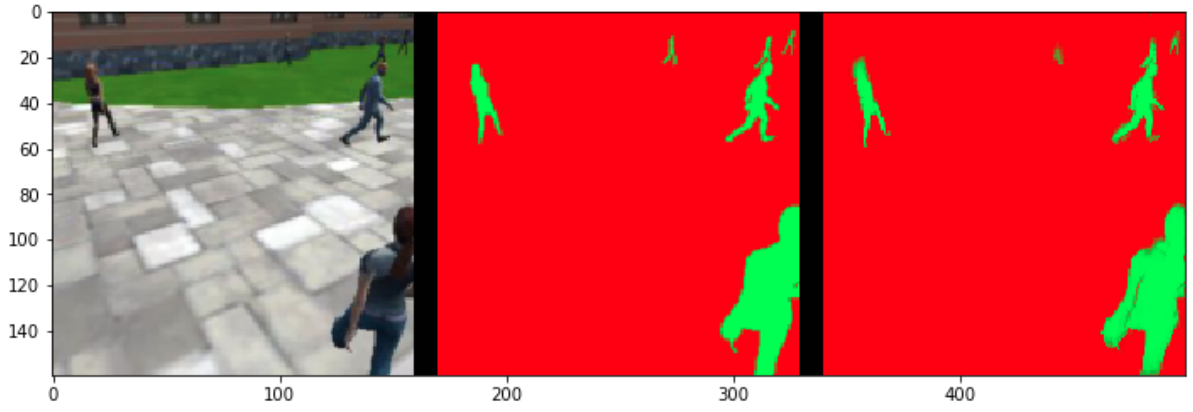
Model parameters

Kernel Size

A Kernel size of 5 was chosen to increase the field of view of the filters: with a smaller kernel size, the segmentation had a tendency to misinterpret the scene particularly when the decoy characters in the image were wearing solid colours: it would often detect the character's edges very well (mapping them as green), but would encode the centre of the character as background (red). The improvements between kernels of 3 and 5 can be seen in the sample images below from otherwise similar training runs:



Kernel_size = 3



Kernel_size = 5

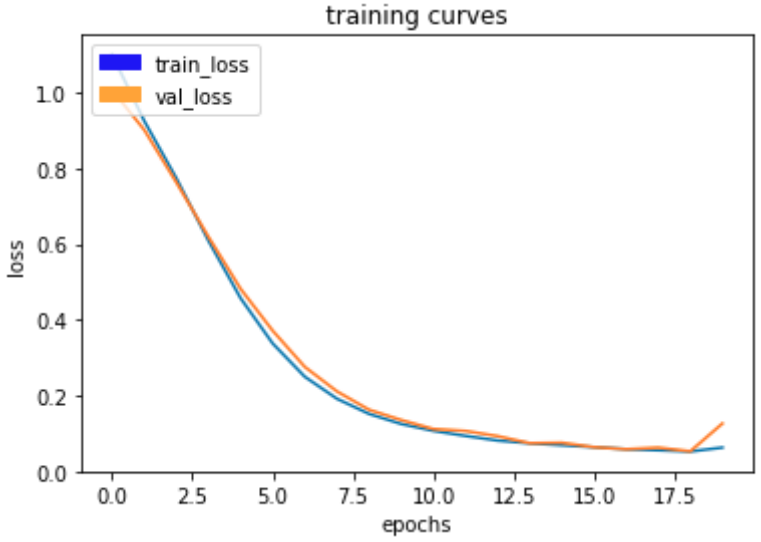
Encoder/Decoder depth

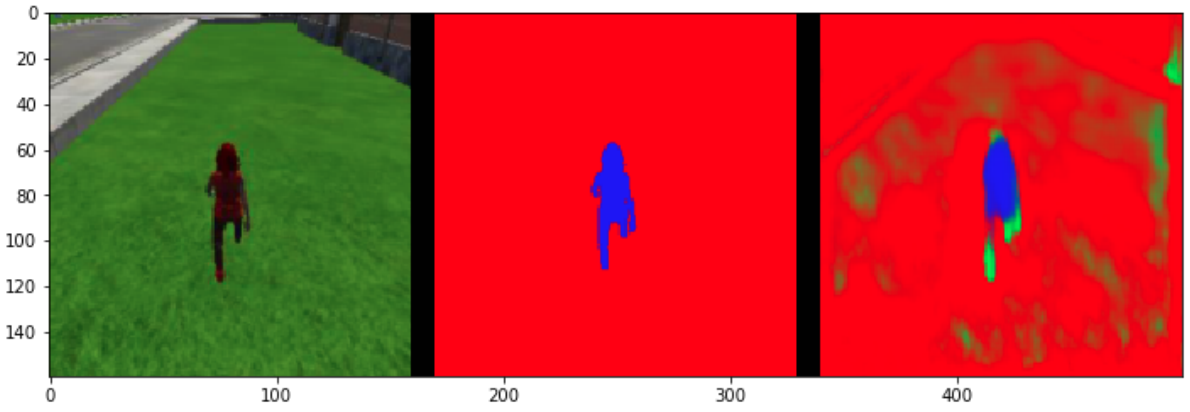
The Encoder/decoder depth of 4, with double separable convolution in the decoders, was settled on after attempting smaller models initially. The depth was progressively increased as testing showed that the depth, whilst keeping all other parameters equal, was a major factor in the performance of the model:

Two examples which clearly demonstrate this are shown below:

- "Model 7" - Two-deep Encoder/Decoder filter count: 32, 64, single separable convolution in the decoder blocks

Loss: 0.0634
Val Loss: 0.1264
Final Score: 0.244

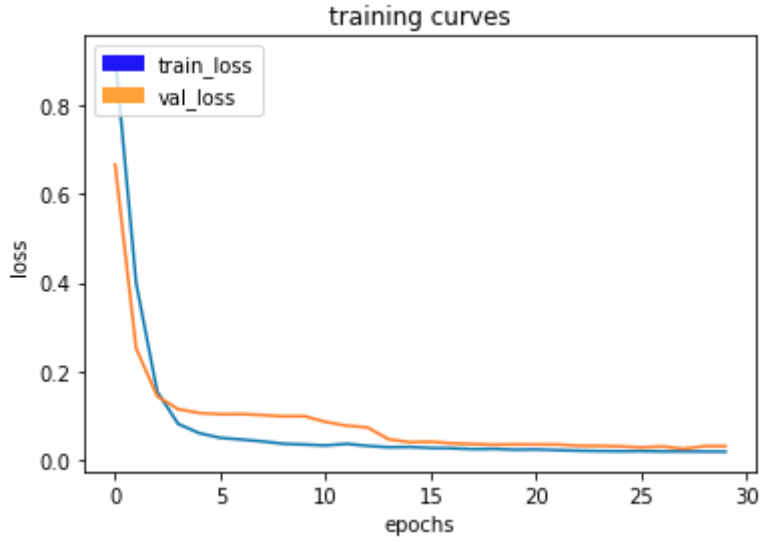


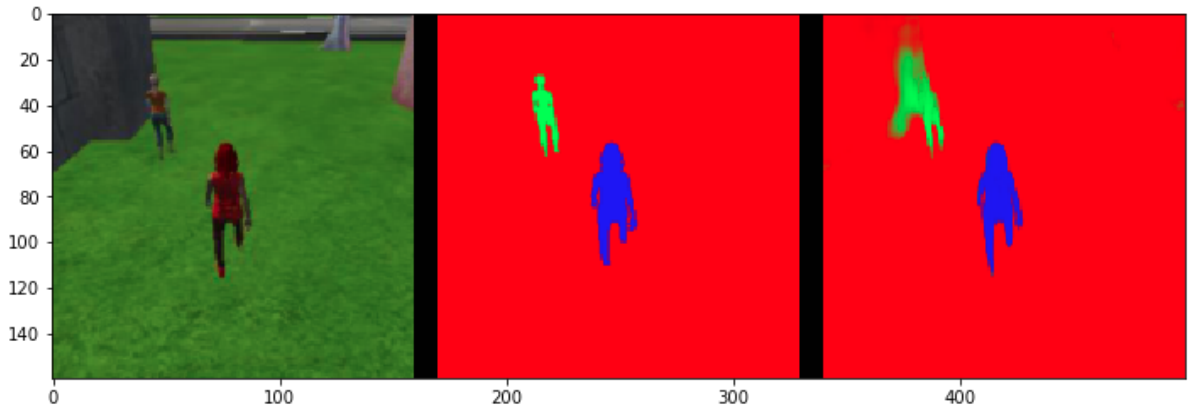


Increasing this model to three deep increased performance markedly:

- “Model 14”: Three deep encoder/decoder: Filter count: 32, 64, 128, single separable convolution in the decoder blocks

Loss: 0.0177
Val Loss: 0.0299
Final Score: 0.341





Training

Hyperparameters

Hyperparameter tuning was approached systematically, with reference to the Keras documentation, other research, and the experiences of other students with this data-set.

Epoch

The majority of runs ran to 20-30 epochs. Initial longer runs showed that the validation-loss plateaued whilst training loss continued to decrease, resulting in overfitting. To avoid this, the Early Stopping callback was implemented to terminate the training after 10 epochs with no Val-Loss improvement. The Model Checkpoint callback was also implemented, with corresponding helper functions added to the notebook, to ensure the best set of weights produced in a run could be isolated. The final set of weights was obtained at Epoch 35 of a model that "Early Stopping" terminated at Epoch 38.

Learning Rate: 0.002

On the suggestion of other students on the Slack channel, the Nadam optimizer in Keras was utilized, which is the provided Adam optimiser, but with Nesterov momentum. Not only does this approach have the theoretical benefit of significantly reducing training and validation loss, it removed learning rate as a hyper-parameter to tune: Keras documentation strongly recommend leaving learning rate, at 0.002 for their implementation of this algorithm.

Batch size: 64

Initial attempts at training the network involved a batch size of 128. However, this led to consistently poor results: Validation Loss typically 50-100% higher than training loss. On research, it became clear that larger batch sizes increase the "confidence" or "peakiness" of the network in relation to the features it detects, and on a small sample of images such as this, it leads to overfitting. The final network was trained on a batch size of 64, which gave significantly better correlation between Training and Validation Loss.

Steps per epoch: 60

Until the later stages of the project, the number of steps per epoch was equal to the Number of Images/Batch Size. This figure was reduced to 60 to check validation loss more often, to make the Early Stopping callback more effective.

Validation steps: 37

Set to 37, based on the number of images in the validation set / batch size, to ensure that validation was completed on all images in the validation set. (In practice, this was twice as high as needed, due to an error counting the images in the validation set. However, this would not have led to any significant difference apart from an increase in validation time.)

Workers: 4

Set to 4, based on the resources available on the AWS p2.xlarge instance.

Training Data

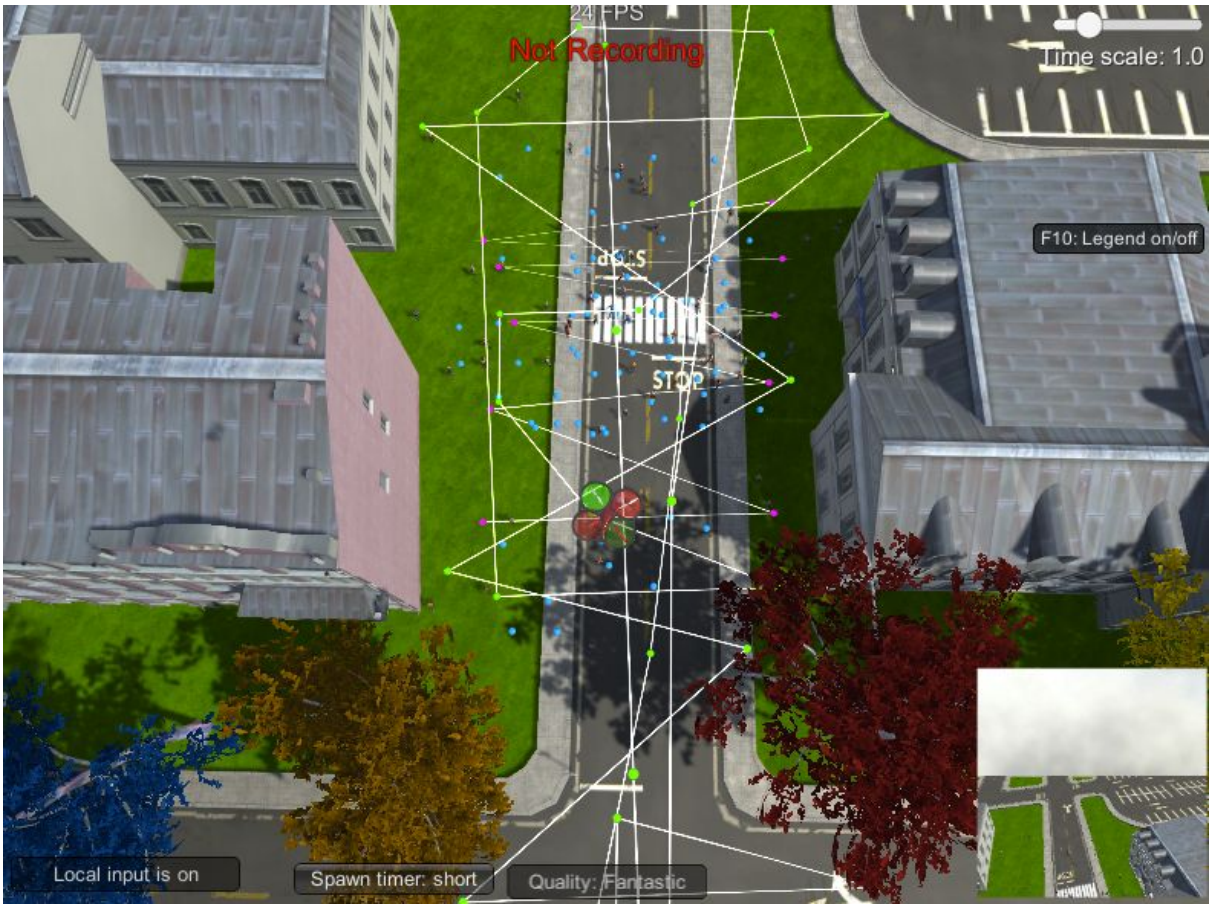
Roughly 1000 additional training pairs featuring Hero were added to the stock data. Using the stock data, the model had consistently poor performance at distance, with detection rates below one in three during the “target far away” evaluation.

Adding more data of this type biased the model towards suggesting a particular feature was the hero.

To achieve this, the patrol path and hero path were configured to ensure the drone imaged the hero at a reasonable distance, through a dense crowd. A script shared on the Slack Channel by Roy Veshoda was then applied to isolate all images featuring Hero within the newly recorded data.

Finally, the model continued to show poor performance when decoy characters were near large white road marking, particularly if they were wearing light clothing. An additional data run was performed, focused intensively on collecting data to aid the model on distinguishing these cases.

“Distant hero” training path



“White road markings” training map

Portability to other scenarios

Cats/Dogs

This model is likely quite adaptable to following other objects, provided they are in the same scale as humans. This task was, however relatively simple: the “Hero” was very clearly identifiable by being essentially a solid red figure. The model may not be as robust at telling the difference between different animals of similar scales without this heuristic - of particular concern is that the model showed poor performance at distinguishing between brickwork and decoy people at larger distances. This suggests that subtle anatomical differences between cats and dogs would likely be lost.

An increase in image resolution, and possibly the addition of point-cloud information would most likely help with this.

Cars

In contrast, the model would likely show good performance with cars as it stands, with the larger shape and colour differences between cars providing more obvious features for the network to learn than is the case with humans. The other advantage when training for cars is that, with the exception of the front wheels whose angle varies, cars are essentially rigid objects. This is in stark contrast to humans or animals whose shape and features change as they walk and move, vastly increasing the complexity of the problem.

Data reuse

In any scenario where the model is used to identify different classes, new training data will be required: it contains only segmentation maps for three classes “background, other people, and target/hero”. The addition of a new class, such as cats, dogs, or cars, will need require new training sets: without supplying training data, the model won’t know what a “car” is.

However, the existing data is not necessarily of no use to other segmentation tasks:

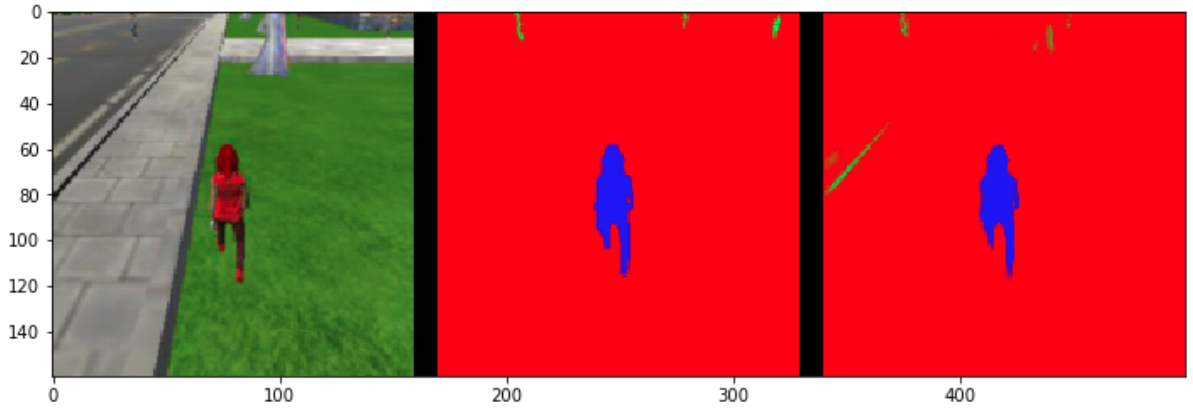
An approach frequently used in state-of-the-art computer vision is to freeze the weights within some of the first encoder layers of a model, thereby transferring the model’s understanding of low level features (simple curves, lines, etc.) to a different segmentation task. Models do not necessarily need to be fully re-trained from scratch, but instead the later layers can be trained less intensively with a smaller set of new data for the classes to be segmented/categorised. Pre-trained VGG16 and ResNet models are commonly used in this way.

In this context of this question, a model trained on the Follow Me data could, in theory, be re-used, if later layers were initialised and retrained with data for classifying dogs, cats, or cars. Better results would be obtained with a full set of new data, however.

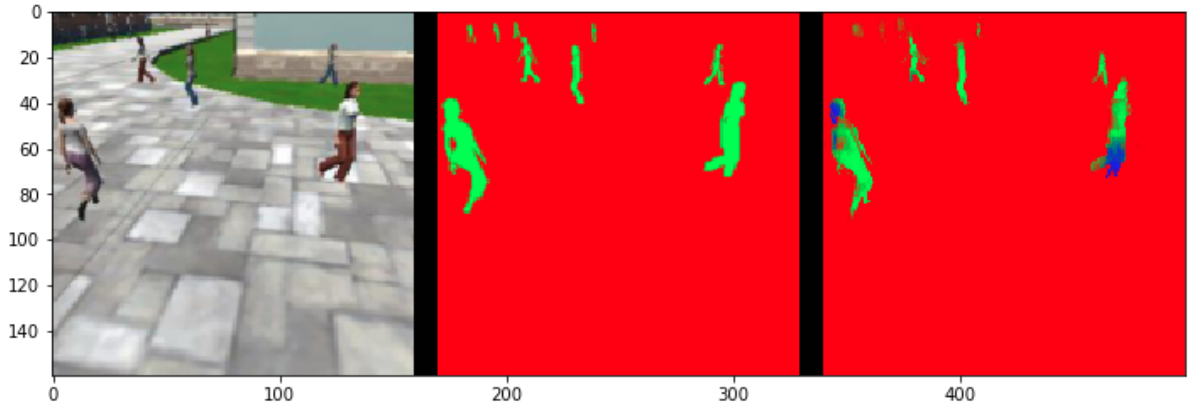
Future Enhancements:

Atrous/dilated convolutions:

A potential enhancement could be the utilisation of atrous/dilated convolutions. This would further increase the field of view of the model, without negatively affecting performance. The model suffers significantly from poor context awareness, evident particularly where it often labels very long segments of the edge of the road in green for “other people”. Similarly, it occasionally labels another woman’s reddish trousers as “Hero”. It is possible that increasing the field of view at all stages would lead the model to avoid these mistakes. An inception layer, with of a variety of different kernel sizes, could also be added, which could provide benefit in this regard.



Road identified as “Other People”



Red trousers identified as “Hero”

Data Augmentation:

Another enhancement to increase the effective amount of training data, could be data augmentation. No augmentation of the data is implemented by the the training iterator at

present (the feature appears to be commented out of the BatchIteratorSimple class), however a great many options are available, both within Keras/Tensorflow.

At the minimum, image flipping would likely be of immediate benefit, by doubling the amount of training data, but scaling and skewing would also be valuable, so long as kept within reasonable limits.