A

PROJECT REPORT ON

"IDENTIFY AND TRACK THE TARGET"

SUBMITTED FOR THE COURSE
EGEE-529 : PRINCIPLES OF NEURAL SYSTEMS & IMPLEMENTATION

BY

VARDA DEEPAK PAROPKARI **CWID: 888239266**
OMKAR TOLARAM JAGTAP **CWID: 888253887**

UNDER THE GUIDANCE OF
PROF: DR. LAN NGUYEN

DEPARTMENT OF ELECTRICAL ENGINEERING
CALIFORNIA STATE UNIVERSITY, FULLERTON
SPRING 2019

ABSTRACT

IDENTIFY AND TRACK THE TARGET

Goal: The goal of the project is to identify and track the target in simulation.

Brief Description: The project aims to identify a target in an environment which is simulated and then track that target. One of the main steps would be to train a deep neural network to understand that simulated scene and also understand the target. This project has many applications in Robotics, Unmanned Aerial Vehicle and can also be used automotive industry for simple applications such as advanced cruise control. The simulated environment scene has a target moving and UAV that is trying to follow the person. The aim is to train the neural network in to understand the scene and understand the target, location of the target, maintain the distance with the target means to move the UAV closer or farther away from the target as necessary. This project is part of our project in robotics course that we are undertaking online.

CONTENTS

PART ONE

INTRODUCTION

The report presents an application for Identifying and Tracking a specified target. The application is fit to use in an environment where the target has to be identified from among large crowd. The application uses an Unmanned Aerial Vehicle (quadcopter) to track the target continuously in the simulator. The quadcopter follows the target wherever it goes and keeps a track of it by filming a live video stream using the front facing camera.

In this project the simulation environment is built by using Unity. In the simulation environment the quadcopter follows a person in Red colored clothing, let's call it as 'Hero'. This is the target that the quadcopter follows and along with the target there are other spawn people, used to test the ability of the quadcopter to identify and track the target from the crowd.

1.1. Objective

The objectives of this project are first to generate images in the purpose of data collection and pre-process the images from the simulator. Secondly the important part that comes in frame, which involves designing the neural network architecture. After building the neural network, the only thing remaining is training it and performing testing & evaluation.

1.2 Scope

This project has wide range of scope in the field of Robotics. It can also be used in automotive industries for simple applications such as advanced cruise control. This application can also be used by government to identify & track criminal activities.

PART TWO
BLOCK DIAGRAM

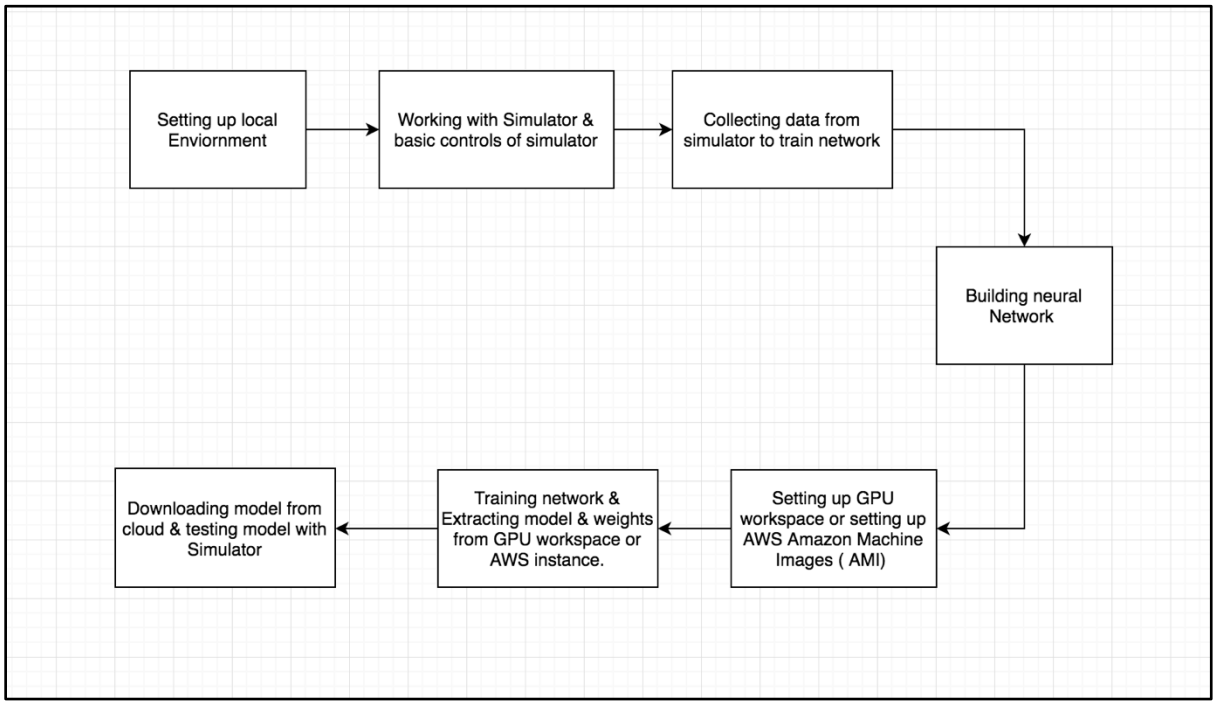

Fig. 2.1

2.1 Block Diagram Description

1. Setting up local Environment : This is the phase where initial network design is test to make check error and design is error free.
2. Working with Simulator : Here we learn about our simulation environment and its basic control with some commands.
3. Data Collection : It is task where we collect data from the simulator to train network.
4. Building Neural Network : This involves designing and building neural network for this application by tuning parameters.
5. Setting up workspace : We used GPU workspace for this task. One can also simply use AWS Amazon
6. Training : This phase includes training of network and extracting final model & weights from GPU workspace.
7. Downloading : Downloading model from cloud and testing model with simulator.

PART THREE

UNITY SIMULATOR

The environment requires Python 3.5 64-bit and Anaconda. As per requirement we have install python and anaconda and started working out the simulation environment.

1. To interface neural net with the QuadSim simulator, we installed QuadSim that has been custom tailored for this project. We are exploring this now to understand fully which then allows us to work flowless.

2. We also taken Data sets into three categories as per requirement of our project:
   a) training data
   b) Validation data
   c) Sample Evaluation Data

3. Simulation Environment looks like the following picture.

Fig. 3

PART FOUR
MODEL DEVELOPMENT

4.1 DATA

The project works on Unity environment. The quadcopter follows the target in Unity environment created. Therefore, the dataset had to be created by training the quadcopter. Some part of the data was given by Udacity, while the rest of the dataset was created while training the quadcopter. The quadcopter was trained in two ways, i.e. once, while only patrolling without target in the environment and secondly, with the target in sight.

Managing Data Collection:

There are 3 major aspects to the data collection process that you can control in order determine the type of data you collect. These are as follows:

1. The path the quad will take while on patrol.
2. The path the hero will walk.
3. The locations of distractor spawns.

- **Setting Patrol Points**

Press the 'P' key to set a patrol point. A green patrol point will appear at the quads position.

- **Setting Hero Path Points**

The hero path points are very similar to the patrol points, except they are always placed at ground level. When reaching the end, the hero will despawn before reappearing at the beginning of the path.

Fig. 4.1

- **Setting Spawn Points**

One can set a spawn point at the quads current x,y position by pressing the 'I' key. Blue markers will appear at the spawn locations.

4.2 BUILDING THE MODEL

The interest of this project is to classify if the person/target is present or not in the input image and also where that person/target is located. This information helps the quadcopter in taking necessary actions like moving away or moving closer or turning if target is off the center of the image.
Semantic segmentation [3] is used in applications like this.

Semantic segmentation:

- It is the task of assigning meaning to a part of an object.
- It is done at pixel level, where we assign each pixel to target class such as road, person, etc.
- It helps in getting information of every pixel in image rather than just slicing sections into bounding boxes.

The project requires a model that is able to segment objects within the video stream. At first, we thought to use many full-scaled convolutional layers one after another. But we soon found this generates too many parameters and becomes computationally very expensive.
Instead, we'll use a method of first extracting important features through **encoding**, then upsample into an output image in **decoding**, finally assigning each pixel to one of the classes. Thus, we are using Fully Convolutional Networks (FCN) in our project.

In this project, FCN model is built for the purpose of training model to detect and to locate the hero within an image.

The steps for this are as follows:

- To create an Encoder Block
- To create 1x1 convolution
- To create Decoder Block
- To build a reliable **FCN model** which comprises of encoder block, 1x1 convolution block and decoder block.

WHY FCN?

- A Fully Convolutional Network is more specialized, and efficient, than a fully connected layer.

- Fully connected neural networks are good for classifying , whereas their performance is not good when it comes to feature extraction. However, Fully Convolutional Networks are trained to identify and extract features from the images.

- Fully connected layers do not preserve the spatial information, whereas Fully Convolutional Networks preserve spatial information.

- Fully connected layers are expensive in terms of memory and computations because each neuron in one layer is connected to every neuron in the previous layer, and each connection has its own weight.

- In contrast, in a convolutional layer each neuron is only connected to a few nearby neurons in the previous layer, and the same set of weights is used for every neuron. This pattern is used for cases where the data can be interpreted as spatial with the features to be extracted.

FCN have achieved a state of art in computer vision techniques such as semantic segmentation. FCN takes advantage of three special techniques:

1. They replace fully connected layers by 1x1 Convolutional layers.
2. Up-sampling using Transposed convolutional layers or Bilinear up-sampling.
3. Skip connections.

4.3 NETWORK ARCHITECTURE

According to recent research on networks like '*ResNet'*, more layers are better, as long as they are not plain layers. But, also as networks become deeper, the training time increases.
After trial and error with different number of layers, the project gave optimal result using five layers.

The project uses FCN model which has an architecture of two Encoders, one 1x1 Convolutional network and two Decoders. [4]

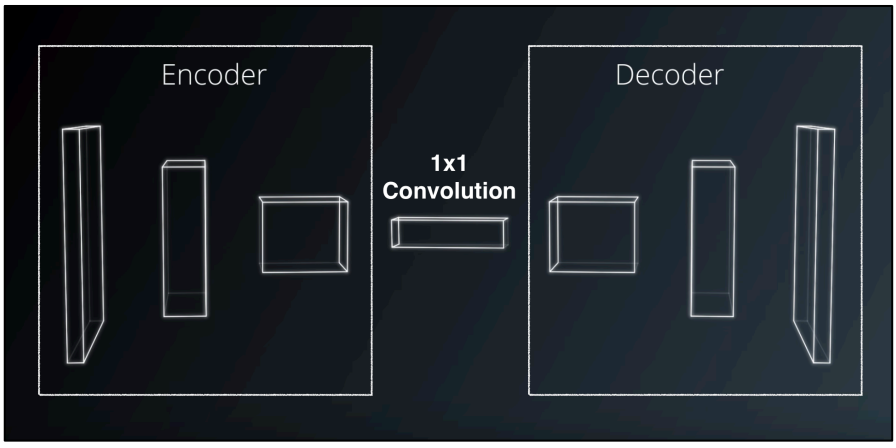

Fig. 5.3 *Photo Courtesy : Udacity*

4.4. SEPARABLE CONVOLUTION LAYERS

- It is a technique that reduces the number of parameters needed unlike normal convolutions, thus increasing efficiency for the encoder.
- They are also called as depthwise separable convolutions.
- They also have the added benefit of reducing overfitting to an extent, because of the fewer parameters.

CODE FOR SEPARABLE CONVOLUTIONS:

It is based on the *tf.contrib.keras* and can be implemented as follows

```
output = SeparableConv2DKeras(filters, kernel_size, strides, padding, activation)(input)
```

```
def separable_conv2d_batchnorm(input_layer, filters, strides=1):
    output_layer = SeparableConv2DKeras(filters=filters,kernel_size=3, strides=strides,
                            padding='same', activation='relu')(input_layer)

    output_layer = layers.BatchNormalization()(output_layer)
    return output_layer

def conv2d_batchnorm(input_layer, filters, kernel_size=3, strides=1):
    output_layer = layers.Conv2D(filters=filters, kernel_size=kernel_size,
strides=strides,
                      padding='same', activation='relu')(input_layer)

    output_layer = layers.BatchNormalization()(output_layer)
    return output_layer
```

Batch Normalization :

While training the model, when normalize each layer's input by using the mean & variance of the values in the current mini-batch is the process of Batch Normalization.
Some advantages of batch normalization are as follows :

- It trains network faster
- It allows high learning rate
- Provides a bit of regularization
- It simplifies the creation of deeper network.

4.5 ENCODER

- The encoder section comprises of two encoder blocks, each of which includes separable convolutional layers.

- The first thing that our network should do is detect features capable of transforming the input into semantic segmentation. These feature detectors are nothing but series of separable convolution layers. Feature extraction is the main function of our encoder.

- Each encoder layer allows the model to gain a better understanding of the shapes in the image. For instance, the first layer is able to extract characteristics like lines, hues and brightness. The second layer is able to extract shapes such as squares, circles and curves.

MATHEMATICAL CALCULATIONS:

To determine the height/width of our subsequent layers we use the formula

$$[(N - F + 2*P) / S] + 1$$

GoogleNet and Resnet had 64, 128, and 256 layers of depth came to notice. This is the number of kernels we opted for in the encoder, 1x1 convolution, and decoder. So, the network, from a depth point of view, is structured in the following manner: Input > 64 > 128 > 256 > 128 > 64 > Output

We can then calculate what the size of our first layer will be:
layer1_height_width = (160-3 + 2*1)/2 +1;
therefore, layer1 = 80x80x64 since we chose 64 filters (or kernels) of 3x3 filter dimension (which is a default setting). For the second layer, we chose 2^7=128 filters of 3x3 size.
The encoder block harnesses the separable_conv2d_batchnorm (), which defaults to a kernel size of 3x3, and same padding (zeros at the edges).

```
def encoder_block(input_layer, filters, strides):
    output_layer = separable_conv2d_batchnorm(input_layer, filters, strides)
    return output_layer
```

The start of our model looks like this:

def fcn_model (inputs, num_classes):

#ENCODER

```
  layer1 = encoder_block (inputs, 64, 2)
  layer2 = encoder_block (layer1, 128, 2)
```

4.6. 1x1 Convolutional Layer

The 1x1 convolution layer is simply a regular convolution, with a kernel and stride of 1.Using 1x1 convolutional layers instead of fully connected layers, provides us the following advantages:

1. It makes our network more flexible by allowing different sized input images, instead of being fixed to one size
2. It decreases dimensions, while preserving spatial information of the image, which allows us to output a segmented image
3. It adds depth to our model and increases parameters at a fairly low computational price.

New 1x1 convolution looks like this:

We went all the way to 2^8=256 filters for the middle 1x1 convolutional layer, so we wouldn't compromise spatial data from our image.

The next portion of our `fcn_model` function is:
def fcn_model(inputs, num_classes):

```
# See encoder portion from above
# 1x1 Convolution layer using conv2d_batchnorm()
layer3 = conv2d_batchnorm(layer2, 256, kernel_size=1, strides=1)
```

4.7. DECODER SECTION:

- The decoder upscales the output of encoder, such that it is same size as original image.
- The decoder section of the model can either be composed of transposed convolution layers or bilinear up-sampling layers.
- The transposed convolution layers reverse the regular convolution layers, multiplying each pixel of the input with the kernel.
- Therefore, the project uses Bilinear up-sampling with layer concatenation.
- Layer concatenation is used because of flexibility i.e. depth of input layers need not match unlike elementwise addition.
- Layer concatenation is used to concatenate the up-sample layer and a layer with more spatial information than up-sampled layer.

BILINEAR UP-SAMPLING

It is a sampling technique that utilizes the weighted average of the four nearest known pixels, located diagonally to given pixel, to estimate a new pixel intensity.

```
def bilinear_upsample(input_layer):
    output_layer = BilinearUpSampling2D((2,2))(input_layer)

    return output_layer
```

In addition to bilinear_upsample, we'll use skip connections for concatenation with layers. Concatenate, which will help us implement skip connections, as well as the separable_conv2d_batchnorm function. These functions will make up our **decoder block** function:

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

    # Upsample the small input layer using the bilinear_upsample() function.
    upsample = bilinear_upsample(small_ip_layer)

    # Concatenate the upsampled and large input layers using layers.concatenate
    concat_layer = layers.concatenate([upsample, large_ip_layer])

    # Add separable convolution layers
    temp_layer = separable_conv2d_batchnorm(concat_layer, filters)
    output_layer = separable_conv2d_batchnorm(temp_layer, filters)


    return output_layer
```

Activation function :

- After the output from the decoder is obtained, we apply an activation function.
- The activation function used is SoftMax function.
- SoftMax function is used to generate the probability for each of the pixels.

$$\sigma\,(Xj) = \frac{e^{Xj}}{\sum_i e^{Xj}}$$

code:

```
return layers.Conv2D(num_classes, 1, activation= 'softmax' , padding='same')(layer5)
```

The complete FCN model looks like:

```
def fcn_model(inputs, num_classes):

    # Add Encoder Blocks.
    # Remember that with each encoder layer, the depth of your model (the number of
filters) increases.
    layer1 = encoder_block(inputs, 64, 2)
    layer2 = encoder_block(layer1, 128, 2)


    # Add 1x1 Convolution layer using conv2d_batchnorm().
    layer3 = conv2d_batchnorm(layer2, 256, kernel_size=1, strides=1)

    # Add the same number of Decoder Blocks as the number of Encoder Blocks
    layer4 = decoder_block(layer3, layer1, 128)
    layer5 = decoder_block(layer4, inputs, 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')(layer5)
```

This final layer will be the same height and width of the input image.

PART FIVE

TRAINING

Model has been trained on Udacity GPU workspace. It took almost three hours for training.

5.1 HYPERPARAMETRS

There are some hyperparameters are necessary to train the model:

- learning_rate = 0.001
- batch_size = 64
- num_epochs = 50
- steps_per_epoch = 65
- validation_steps = 50
- workers = 120
- steps_per_epoch_check = 4132/batch_size

```
Training Curves :


Epoch 1/50
64/65 [============================>.] - ETA: 1s - loss: 0.8473
```

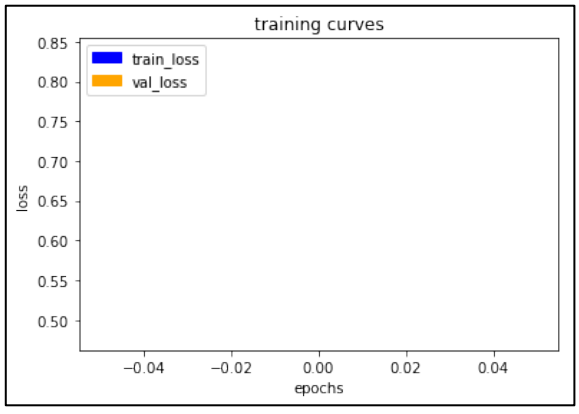

Fig. 5.1.1

```
65/65 [==============================] - 171s - loss: 0.8420 - val_loss: 0.4797

Epoch 2/50
64/65 [============================>.] - ETA: 1s - loss: 0.2854
```

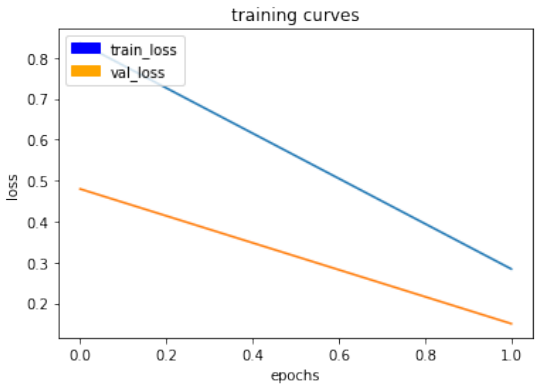

Fig. 5.1.2

```
65/65 [==============================] - 164s - loss: 0.2833 - val_loss: 0.1504

Epoch 49/50
64/65 [==============================>.] - ETA: 1s - loss: 0.0183
```

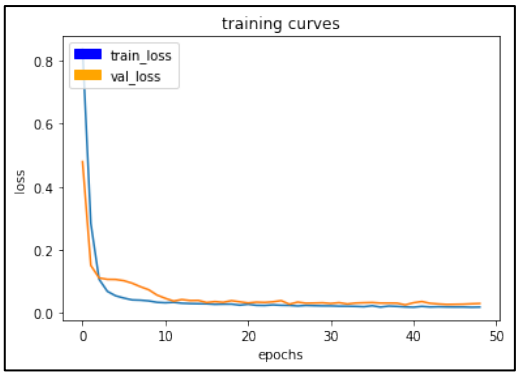

Fig. 5.1.3

```
65/65 [==============================] - 154s - loss: 0.0184 - val_loss: 0.0297


Epoch 50/50
64/65 [==============================>.] - ETA: 1s - loss: 0.0166
```

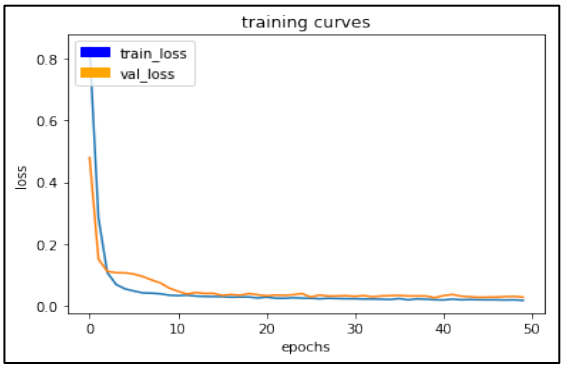

Fig. 5.1.4

```
65/65 [==============================] - 163s - loss: 0.0166 - val_loss: 0.0269
```

5.2. PREDICTION

The predictions can be compared to the ground truth labels, to evaluate how well model is doing under different conditions.

We are going to predict three different scenarios where

1. Patrolling with target : In this , it tests how well the network can detect the hero from a distance.
2. Patrolling without target : Where it tests how much reliable our network is and is it makes mistake and identifies the wrong target.
3. Following images : It tests how well the network can identify the target while following them.

The following cell will write predictions to files and return paths to the appropriate directories. The 'run_num' parameter is used to define or group all the data for a particular model run.

```
run_num = 'run_1'

val_with_targ, pred_with_targ =
model_tools.write_predictions_grade_set(model,
                                        run_num,'patrol_with_targ',
'sample_evaluation_data')

val_no_targ, pred_no_targ =
model_tools.write_predictions_grade_set(model,
                                        run_num,'patrol_non_targ',
'sample_evaluation_data')

val_following, pred_following =
model_tools.write_predictions_grade_set(model,
                                        run_num,'following_images',
'sample_evaluation_data')
```

Now lets look at predictions, and compare them to the ground truth labels and original images.

1. Images while following the target

```
im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','following_images',
run_num)
for i in range(3):
    im_tuple = plotting_tools.load_images(im_files[i])
    plotting_tools.show_images(im_tuple)
```

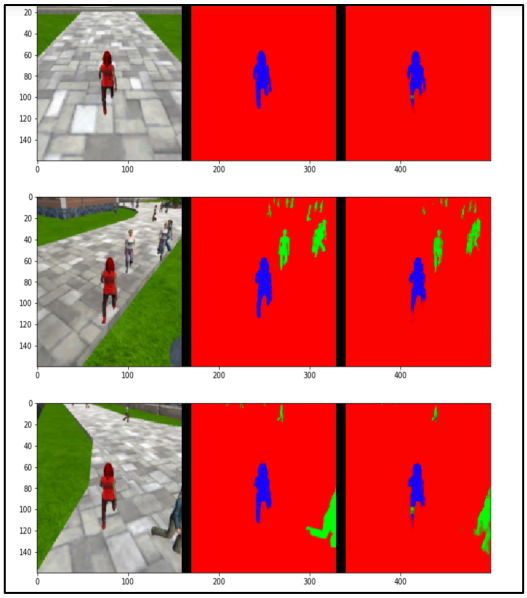

Fig.5.2.1

2. Images while at patrol without target

```
im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','patrol_non_targ',
run_num)
for i in range(3):
    im_tuple = plotting_tools.load_images(im_files[i])
    plotting_tools.show_images(im_tuple)
```

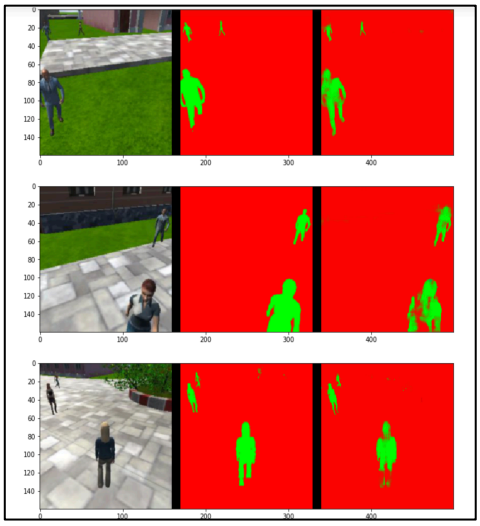

Fig.5.2.2

3. Images while at patrol with target

```
im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','patrol_with_targ',
run_num)
for i in range(3):
    im_tuple = plotting_tools.load_images(im_files[i])
    plotting_tools.show_images(im_tuple)
```

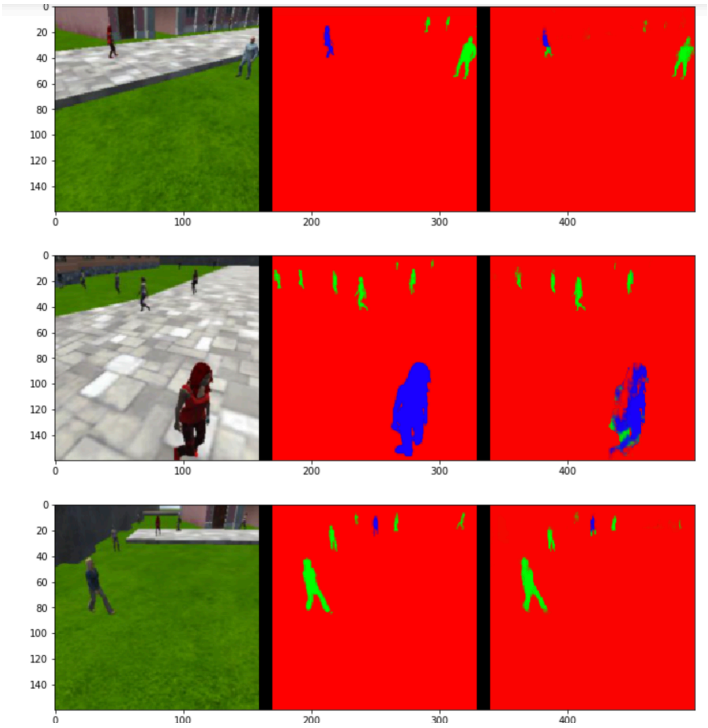

Fig.5.2.3

5.3. EVALUATION

The following cells include several different scores to help to evaluate model under different conditions.

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 0.9952974070911688
average intersection over union for other people is 0.33418126749734256
average intersection over union for the hero is 0.8692177391004331
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 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 0.9859718706663971
average intersection over union for other people is 0.7102342846296277
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 57, number false negatives: 0
```

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 0.996074780985695
average intersection over union for other people is 0.4290915871110559
average intersection over union for the hero is 0.21699848309305347
number true positives: 146, number false positives: 2, number false negatives: 155
```

# 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)

```
0.7619577308120133
```

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

```
0.543108111097
```

And the **final grade score** is
final_score = final_IoU * weight
print(final_score)

```
0.413825423917
```

PART SIX

CONCLUSION & FUTURE ENHANCEMENT

Additional data and training time would allow the network to classify the target more accurately, by showing it more facets and features of our target, repeatedly. In order for this Deep Neural Network to be used to follow another target: such as a cat or a dog, it would just need to be trained on a new set of data. Also, the encoder and decoder layer dimensions may have to adjusted depending on the overall depth of the network.

With increasing encoder and decoder blocks, we will be able to achieve higher accuracy and efficiency. Collecting good data that covers all scenarios that the model needs to perform well in real life is as challenging as architecting and training the model. This application can also be used as classifier on the next level which will lead to an another additional feature. This project can also be used by government agencies in the purpose of identify and tracking any kind of criminal activities. With additional features ,this can also be used in Robotics industry, self-driving vehicle industry.

PART SEVEN

REFERENCES

1) *Udacity*, classroom.udacity.com/nanodegrees/nd209/parts/c199593e-1e9a-4830-8e29-2c86f70f489e/modules/cac27683-d5f4-40b4-82ce-d708de8f5373/lessons/197a058e-44f6-47df-8229-0ce633e0a2d0/concepts/a509e167-7370-41ed-ac2d-834076117008.

2) L. Wang, W. Ouyang, X. Wang and H. Lu, "Visual Tracking with Fully Convolutional Networks," *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, 2015, pp. 3119-3127.doi: 10.1109/ICCV.2015.357
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7410714&isnumber=7410356

3) T. Wang, "Context Propagation from Proposals for Semantic Video Object Segmentation," *2018 25th IEEE International Conference on Image Processing (ICIP)*, Athens, 2018, pp. 256-260.
doi: 10.1109/ICIP.2018.8451215
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8451215&isnumber=8451009

4) J. Xu, L. Song and R. Xie, "Two-stream deep encoder-decoder architecture for fully automatic video object segmentation," *2017 IEEE Visual Communications and Image Processing (VCIP)*, St. Petersburg, FL, 2017, pp. 1-4.
doi: 10.1109/VCIP.2017.8305089
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8305089&isnumber=8305018

5) D. Kang, Z. Ma and A. B. Chan, "Beyond Counting: Comparisons of Density Maps for Crowd Analysis Tasks—Counting, Detection, and Tracking," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 5, pp. 1408-1422, May 2019.doi: 10.1109/TCSVT.2018.2837153
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8360001&isnumber=8705610