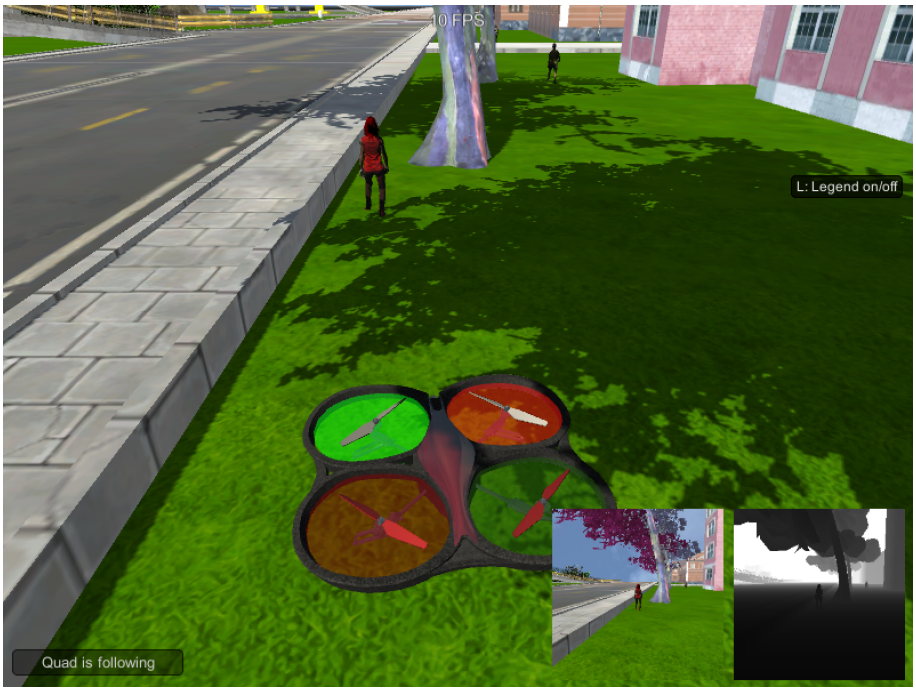
Robotics Nanodegree
Project 4
Follow Me - Semantic Segmentation



November 23, 2017

We train a Deep Neural Network to identify and to follow a target (the *hero*) in a simulator. The goal of the model is to segment/classify every pixels of each image frame from the video camera according to pre-determined classes. In this project, we consider 3 classes: background, people and the hero.

0.1 Technical Overview

Each image frame goes through an **encoder/decoder** model, and the output is an image of the same size as the original image where each pixel is assigned a class.

The **encoder** is a feature extractor and uses a combination of convolution operations and maxpooling operations to reduce the spatial size of the feature maps, while increasing its depth.

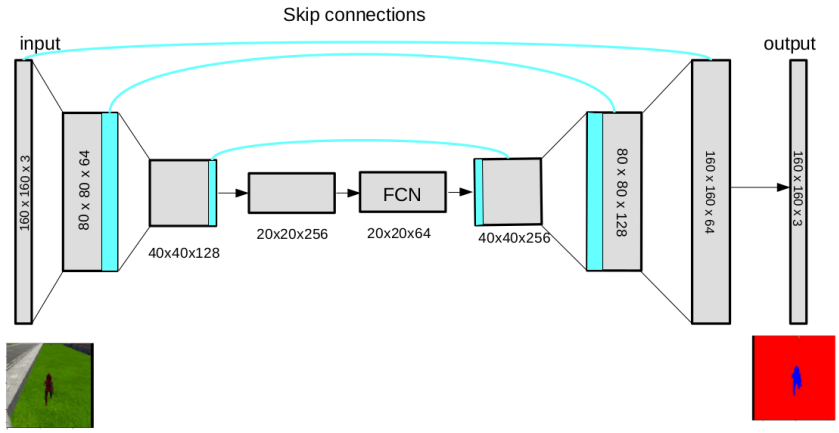
The **decoder** does the opposite work: it builds an image with the same size as the original image using the features extracted by the encoder. The decoder uses a combination of upsampling and convolution operations. The upsampling is done with BilinearUpSampling2D that scales the input image by a factor 2. This layer is then concatenate with the equivalent pooling layer from the encoder (i.e pooling layer with same spatial size as the layer in the decoder). That procedures is known as skip connections and it is aimed at compensating for the loss of resolution from the upsampling operation.

In typical encoder/decoder model a fully connected layer is used to connect the encoder output to the decoder input. However, that approach leads to a loss of spatial information. In ConvNet for semantic segmentation application, the fully connected layer is replaced by a **fully convolutional layer** (FCN), generated using (1×1) convolution.

The output image of the the model is then compared to the ground truth image using cross entropy. The objective is to minimize that error by optimizing the weights and biases variables.

0.1.1 Model architecture

Below is the architecture of our semantic segmentation model.



- input: (?, 160, 160, 3)
- encoder_layer 1 : (?, 80, 80, 64)
- encoder_layer 2 : (?, 40, 40, 128)
- encoder_layer 3 : (?, 20, 20, 256)
- fcn : (?, 20, 20, 64)

- decoder_layer 1 : (?, 40, 40, 256)
- decoder_layer 2 : (?, 80, 80, 128)
- decoder_layer 3 : (?, 160, 160, 64)
- output : (?, 160, 160, 3)

Our model includes a few "tricks" to improve the model performance both in terms of training and speed at inference:

- **Batch Normalization**: With **Batch Normalization**, the output batch at each layer is normalized to have zero mean and unit variance, and then injected into the following layer. That makes it faster for the model to converge, but it comes at a computational cost, as the mean and variance must be computed. Batch Normalization enables to use larger learning rate values, and to some extent also helps prevent overfitting.
- **Separable Convolution layer** We use separable convolution layers as a means of reducing the number of parameters: which is helpful in reducing the need for computational ressources required at training but also it helps make inference computationally faster. It also helps prevent overfitting as it reduces the number of parameters. Note that Separable Convolution layer assumes that there is no correlation between the spatial $(width, height)$ and depth (channels) information.

0.2 Hyper-parameters optimization

The hyper-parameters were optimized via trial and error.

```
learning_rate = 0.002
batch_size = 15
num_epochs = 40
steps_per_epoch = 20 #4131//batch_size
validation_steps = 30 #1184 // batch_size
workers = 10
```

0.2.1 Learning rate

We chose a progressive learning rate of **0.002** down to 0.0005, which results in a more monotonic decrease of the loss. With a higher learning rate (up to 0.005), the loss values would scattered more: more oscillations.

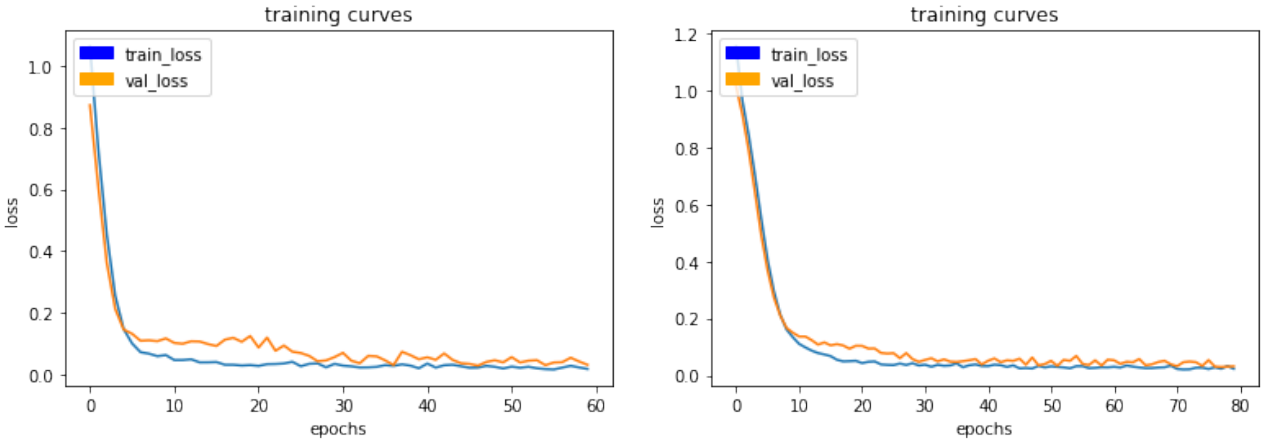


Figure 1: Learning curves when (left) Learning rate = 0.004, (right) Learning rate=0.002. With smaller learning rate, the learning curves are smoother.

From 0 to 40 EPOCHS, we used learning rate= 0.002. From 40 to 55 EPOCHS, learning rate was 0.001, and then from 55 to 90 EPOCHS, the learning rate was 0.0005, and finally for EPOCHS 90 to 190, the learning rate was reduced to 0.0001. As the number of EPOCHS increases and the learning rate decreases, we observed a decrease of the training loss and validation loss.

Training Phase	EPOCHS	learning rate	final train. loss	final valid loss	final score
phase 1	0 to 40	0.002	0.0230	0.0459	0.316
phase 2	40 to 55	0.001	0.0173	0.0318	0.332
phase 3	55 to 90	0.0005	0.0190	0.0272	0.374
phase 4	90 to 150	0.0001	0.0128	0.0192	0.401

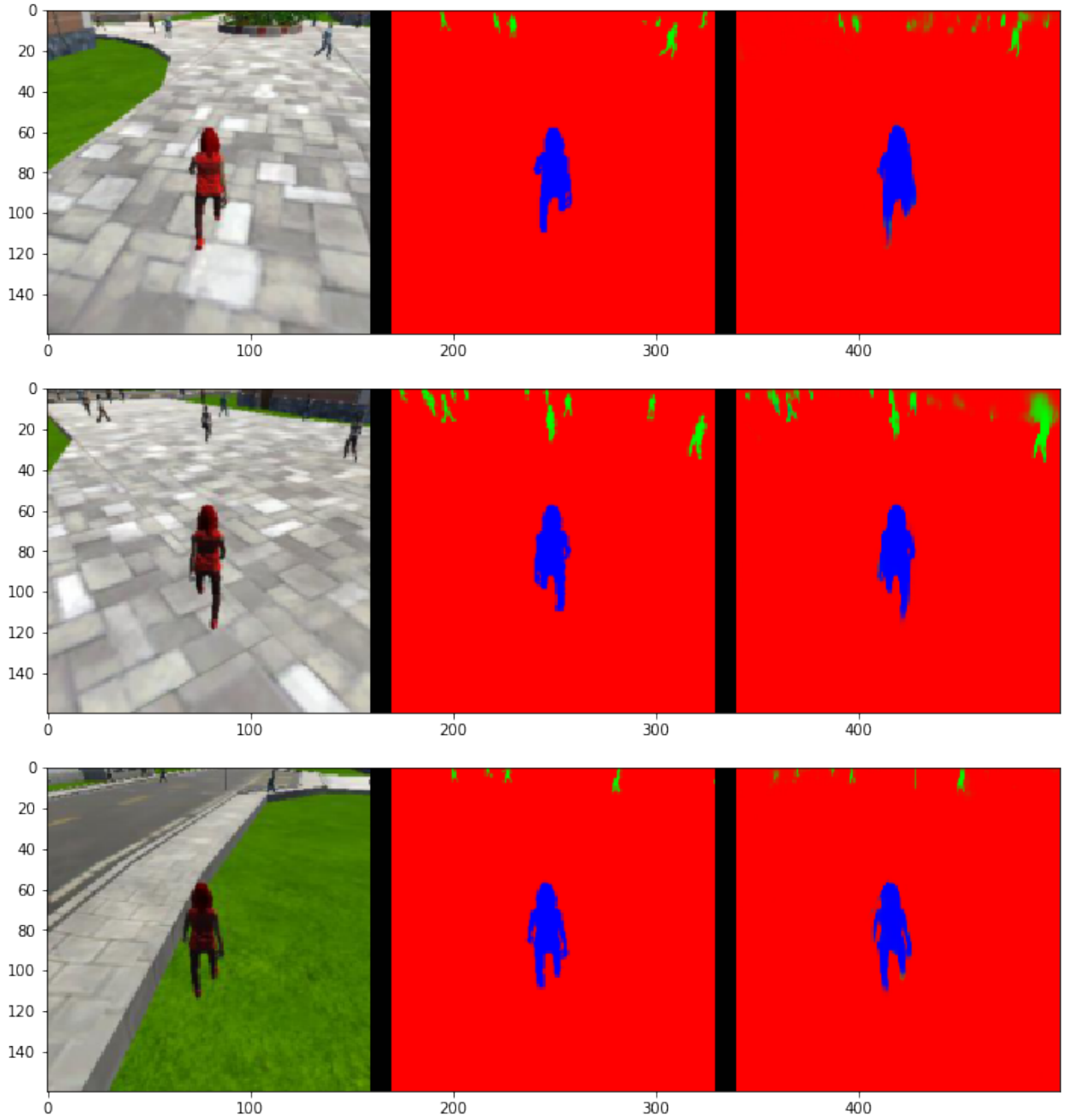


Figure 2: Top figure: Phase1 - 40 EPOCHS. Middle figure: Phase 2 - 55 EPOCHS. Bottom figure: Phase 4 - 150 EPOCHS.

In Phase 2, the model is better able to identify the items at the horizon (top of the images): after phase 1, there are a lot of misclassifications where individuals are classified as background, and background areas are classified as individuals. In phase 4, there are less misclassifications, although the resolution of the segmentation is still poorer than that of the ground truth.

0.2.2 Number of EPOCHS

The number of EP0CHS (*num_epochs*) is increased until the training loss is small and practically constant, and the validation loss does not increase. That is to say that the model does not overfit the training set. We progressively increase the number of EPOCHS to 150, while decreasing the learning rate.

0.2.3 Batch size

The batch size was limited by the GPU memory: 12Gb on my machine (NVidia TitanX). I chose a batch size of 15 examples.

0.2.4 Steps per epoch

I chose a lower value of **steps_per_epoch** than the typical (total_number_of_images / batch_size). This is mainly for conveniency, to get faster updates on the status of the training and validation loss. With Steps_per_epoch = (total_number_of_images / batch_size), one EPOCH takes about 2000sec to complete. Note that reducing the Steps per epoch does not affect the learning since the orginal dataset is shuffled after each EPOCH, and also we are using data augmentation and a high number of num_epochs. In that case, the model is likely to "see" every single samples in the dataset.

0.2.5 Data augmentation

We use 2 data augmentation schemes to increase the number of images, and the variance in the dataset. The image is shifted by a few pixels along x and y direction: that is provided in the initial code. The other data augmentation scheme consists in flipping images and their associated mask wrt the vertical axis. The images to be flipped are randomly selected. The dataset is therefore increased by a factor 2. Note that the data augmentation is performed dynamically, i.e the flipped images are not saved on the drive. The data augmentation can also help limit the overfitting.

The image to be flipped is selected from a random toss (flip_it=np.random.choice([0,1]). The image is flipped only if *flip_it* is **True**.

```
def shift_and_pad_augmentation(image, image_mask):
    shape = image.shape
    new_im = np.zeros(shape)
    new_mask = np.zeros(image_mask.shape)
    new_mask[:,:,0] = 1
    flip_it = np.random.choice([0, 1])

    im_patch, mask_patch = get_patches(image, image_mask)
    patch_shape = im_patch.shape

    ul_y = np.random.randint(0, shape[0]-patch_shape[0])
    ul_x = np.random.randint(0, shape[1]-patch_shape[1])

    new_im[ul_y:ul_y+patch_shape[0],
           ul_x:ul_x+patch_shape[1],
           :] = im_patch

    new_mask[ul_y:ul_y+patch_shape[0],
             ul_x:ul_x+patch_shape[1],
             :] = mask_patch
    #Perform image/mask flip wrt vertical axis
    if flip_it:
```

```
        new_im = np.flip(new_im, 1)
        new_mask = np.flip(new_mask, 1)
    return new_im, new_mask
```

0.3 Results

The training process is slow: it takes about 215 sec to complete 1 EPOCH. The final training loss (after 4 phase learning - 150 EPOCHS) is about 0.0128 and the validation loss is slightly larger at 0.0191. This suggests that our model is marginally overfitting the training set. Adding dropout (on the kernels) or L2 regularizer would help reduce overfitting. The final score is: 0.401.

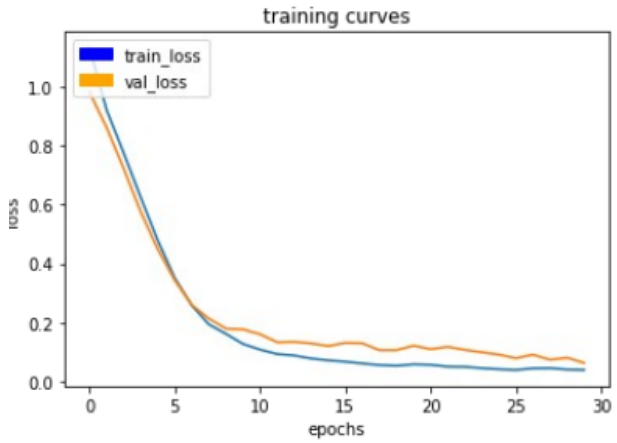
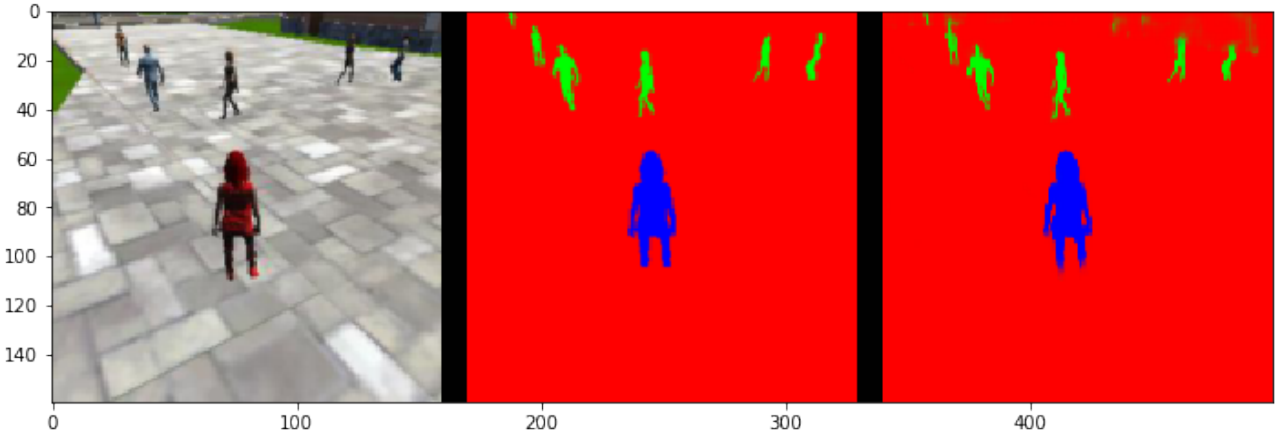


Figure 3: Learning Curves after a few EPOCHS (¡30).

In the image below is one of the segmented images that was output by the model. The blue identifies the hero, the red the background and the green other individuals. Our model identifies fairly well our hero, and people. In fact, the bulk of the pixels related to the hero is well captured, however, the boundaries and fine features (legs) are not reproduced accurately. We also notice that there are some misclassifications: where background pixels are expected, the model identifies a person.



0.4 Future enhancements

1. Increase the model performance by reducing the learning rate at high EPOCHs. When the training loss plateaus, a smaller learner rate might help to get finer update of the weights.

2. Instead of training the model from scratch, we might see better performance by using a pre-trained model for the encoder: GoogleNet, ResNet or VGG.

3. Higher resolution image inputs would help to get better boundary definition. However, it also requires more memory (GPU) resources.

4. Incorporate a regularizer scheme like dropout on the kernel or L2 regularization would help control the overfitting behavior.

5. Data augmentation scheme: the model has troubles classifying properly individuals that are at the horizon. One approach to augment the dataset is to scale down the image, and add 0 padding so that the resulting image has the same size as the original image. Because items at the horizon are typically at the top of the canvas, we would need to translate the scaled image to be in the upper part of the canvas. This additional data augmentation can further help control overfitting.

0.5 Limitations of the models

Our model is very specific to the training dataset. In order for the model to apply to dogs/cats, we would need to add images and masks that includes dogs/cats in the training dataset. So that the model can extract features related to the new items. We also need to add an additional class to account for the new item that needs to be identified, i.e the last layer of the model (the convolution with softmax activation) must be modified.