Follow-Me: A Deep Learning Project

January 30, 2018

Fully Convolutional Network Architecture and Parameter Tuning

In this project we train a fully convolutional network to track the presence of a hero in an image frame so that a drone will search for and follow camera views that contain the hero. The network architecture of the fully convolutional network used is based on a depth shortened version of the Xception [1] and the Resnet [2] architectures. In fact, the encoder block in our model mimics the downsampled convolution blocks in the entry flow Xception model but instead of taking a maxpool to reduce feature map size, we take lesson from the architecture of the Resnet model and downsample by using a stride 2 convolution. Our decoder block then reverses the actions taken in the encoder block and upsamples an input image while concatenating it with the dimensionally matched encoder block. The resulting architecture consists of four layers of encoder blocks, a 1x1 convolutional connector, and four layers of decoder blocks each skip connected to the corresponding dimension consistent encoder layer.

Taking a closer look, a fully convolutional network architecture consists of an encoder block, decoder block, an intermediate 1x1 convolution block that transitions between the encoder and decoder blocks and a 1x1 convolution that reduces filtration depth to the number of output classes. The encoder block consists of a sequence of N single stride, same padding convolutional layers with some fixed number of filters followed by either a double stride, same padding convolutional layer or pooling filter that simultaneously reduces the size of the feature map and increases the size of filters. The decoder block consists of a sequence of convolutional layers created by upsampling a previous layer and concatenating the upsample with the same dimension feature map of the corresponding encoding block layer. The 1x1 convolution block acts as a replacement of a flattened fully connected layer in the middle of the more conventional convolutional neural network to preserve the spatial dimension of the feature map in link from encoding to decoding. The decoding layer acts as a replacement of fully connected layers at the end of convolutional neural networks to build upon the layers of segmented spatial information decomposed by the filters in the encoder layers by upsampling back to the original image dimensions for a display of the resulting image segmentation. The final layer of the FCN is a 1x1 convolution that replaces the final fully connected layer of conventional convoultional neural networks that works to reduce the filtration size depth to the number of classes required for output classification but also has the added component of preserving the location of objects classified on the spatial information stored

in the feature map. In addition, for each convolution layer, any 2 dimensional output image convolved by a nxn kernel can be modified to include activation, batch normalization, and separability to decrease compute time, increase accuracy, and decrease loss while maintaining generalizability in classification.

Taking a step back to look at the overall picture of what we are doing here, we find that the purpose of the encoder part is to break down the image into parts that realize distinguishing features of the objects in the image. Each layer selects different categories of distinguishing features of the image for classification. On the other hand, the purpose of the decoder part is to reassemble the image deconstructed by the encoder part into a semantically segmented decomposition of the image. While typical convolutional neural networks use fully connected layers to output only a classification of objects, the resurrection of spatial information in fully convolutional networks through the use of 1x1 convolutional layers and decoder blocks outputs not only a classification of objects but also outputs where in the image the object is located. Here, we use fully convolutional networks to find where a hero is within a given camera shot that the drone takes as it attempts to locate her.

The basic architecture of our fully convolutional network is illustrated by the following diagram 1. Each encoder and decoder block of the network has components illustrated by the diagram 2.

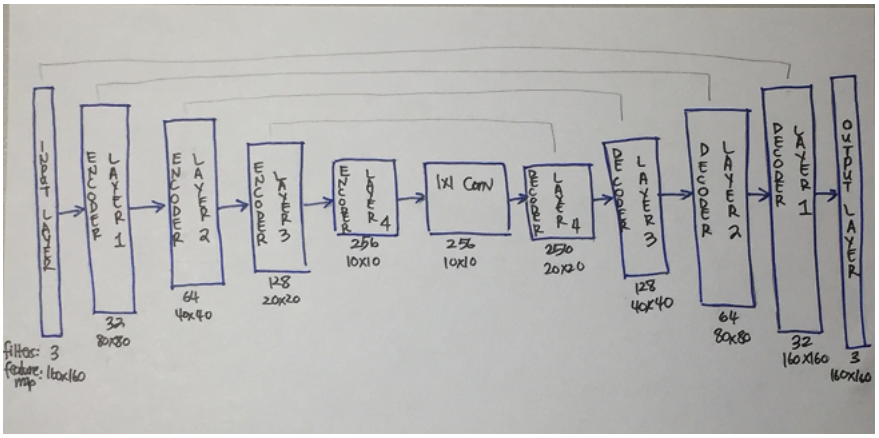

Figure 1

Explicitly, the code corresponding to these diagrams is given by:

```
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,
```

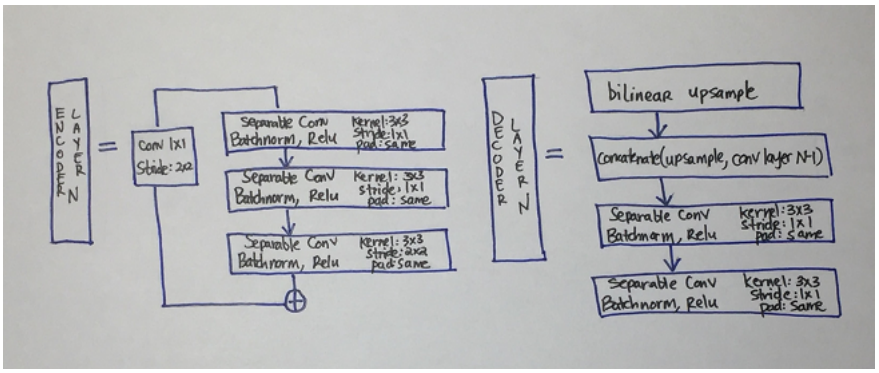

Figure 2

```
                        padding='same', activation='relu')(
    input_layer)

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

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

def encoder_block(input_layer, filters):

    # TODO Create a separable convolution layer using the
    separable_conv2d_batchnorm() function.
    residual = conv2d_batchnorm(input_layer, filters, kernel_size =
     1, strides=2)
    x = separable_conv2d_batchnorm(input_layer, filters, strides=1)
    x = separable_conv2d_batchnorm(x, filters, strides=1)
    x = separable_conv2d_batchnorm(x, filters, strides=2)
    x = layers.add([x, residual])
    return x

def decoder_block(small_ip_layer, large_ip_layer, filters):

    # TODO Upsample the small input layer using the
    bilinear_upsample() function.
    upsampled = bilinear_upsample(small_ip_layer)
    # TODO Concatenate the upsampled and large input layers using
    layers.concatenate
    input_layer = layers.concatenate([upsampled, large_ip_layer])
    # TODO Add some number of separable convolution layers
    x = separable_conv2d_batchnorm(input_layer, filters, strides=1)
    x = separable_conv2d_batchnorm(x, filters, strides=1)
    return x

def fcn_model(inputs, num_classes):

    # TODO Add Encoder Blocks.
    filters = 32
    conv_layer1 = encoder_block(inputs, filters)
    conv_layer2 = encoder_block(conv_layer1, filters*2)
    conv_layer3 = encoder_block(conv_layer2, filters*4)
    conv_layer4 = encoder_block(conv_layer3, filters*8)
    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
```

```
    conv_1x1 = conv2d_batchnorm(conv_layer4, filters*8 ,
    kernel_size =1, strides = 1)
    # TODO: Add the same number of Decoder Blocks as the number of
    Encoder Blocks
    deconv_layer4 = decoder_block(conv_1x1, conv_layer3, filters*8)
    deconv_layer3 = decoder_block(deconv_layer4, conv_layer2,
    filters*4)
    deconv_layer2 = decoder_block(deconv_layer3, conv_layer1,
    filters*2)
    deconv_layer1 = decoder_block(deconv_layer2, inputs, filters)
    # 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')(deconv_layer1)
```

Observe that the beginning of our fully convolutional network consists of a sequence of four encoder blocks that take in as input the previous encoder output and starts with the original image as input. We also set filters for each encoder block to start at 32 and increase by a multiple of two with each additional block. Each encoder block consists of two 3x3 convolutional layers of stride 1 followed by one 3x3 convolutional layer of stride 2 whose output tensor is added to the encoder input convolved with a 1x1 stride 2 convolution. As an alternative to maxpooling or average pooling, a stride 2 convolution dimensionally reduces the width and height of the feature map by 1/2 so the ouput of the encoder block has a feature map with side length half the size of its input side length. To compensate for the loss of resolution due to this downsampling, the filters are increased by a multiplicative factor of 2. In addition, the more layers that are evaluated, the more complex detail is captured by the feature space. To represent the increased complexity at each level, one increases the output depth by increasing the number of filters. However, the limitation here is that increasing the number of filters also increases the memory needed to store parameters and the time needed to perform computation. To accomodate these limitations, we start with a low number of filters. Now consider the reasoning behind the build of each 3x3 convolutional layer. Each 3x3 convolutional layer is separable which means the kernel can be expressed as the outer product of 3 vectors so by implementing this reduction, the runtime and parameter storage space can be reduced from $O(w^3)$ to $O(3w)$ where w represents the elements in each dimension of the kernel [3]. Next, a Relu activation is applied so that non-linear functions can be learned and convergence greatly accelerated due to its piecewise linearity and nonsaturating form. However, if the learning rate is too high, a large gradient flow could cause the weight updates to reduce to zero, thereby killing any further updates, but typically this is not a problem when low learning rates are chosen [4]. Lastly, batch normalization is applied to speed up convergence by admiting the use of higher learning rates without the risk of divergence and regularizing the model for better generalization [5]. Continuing the analysis of the FCN, the middle of the network is a regular 1x1 convolution followed by Relu activation and batchnorm. And the end of the network consists of a sequence of four decoder layers followed by a 1x1 convolution with output classes as filters and softmax activation. The decoder layers take in as inputs the previous layer output, the encoder block corresponding to the same dimension as the previous layer output upsampled by 2, and the number of filters corresponding to the respective encoder block. Within the decoder block is a bilinear upsampling of the previous layer followed by a concatenation of

the upsampling with the N-1 encoder block to the Nth decoder block and two 3x3 separable convolution layers. Together, the decoder sequence transforms the encoder sequence into spatially segmented objects coming from the original image and ends with an output classification.

To train the neural network, we used the following set of parameters to achieve our goal of obtaining a minimum final IOU score of .4:

```
learning_rate = 0.01
batch_size = 81
num_epochs = 20
steps_per_epoch = 4131/81
validation_steps = 50
workers = 2
```

A rule for choosing learning rate is to start with a large learning rate ($< .2$) and continue to reduce the learning rate over time when the loss hits a plateau. In general, the loss decreases with decreasing learning rate but also slows down gradient descent so choosing a small learning rate to begin with will slowly decrease the loss over an increasing number of epochs but will make sure that the loss function converges to a minimum. In the case of large learning rates it is not gauranteed that the loss will converge to a minimum. In fact, gradient descent may overshoot the minimum so that the loss will fail to converge or may even diverge. To be on the safe side, we choose a small learning rate which is the starting rate chosen in the Xception model to increase the likelihood for convergence to a local minimum over a long number of iterations. As for the choice of number of epochs, typically increasing the number of epochs lowers the loss during training time and increases the accuracy for prediction unless saturation is reached or learning stops because the limitations of the training set do not generalize. Starting with 10 epochs, the model makes a .39 final IOU score. All else being equal, we estimate that doubling the number of epochs to 20 epochs will allow us to reach the minimum IOU score of .4. When implemented, this turns out to be the case. In general practice, the batch size is set to conform with the limits of memory requirements of the computer being used such as 32, 64, 128, 256. Small batch sizes converge more quickly but can accumulate noise over time while large batch sizes converge more slowly but with more accurate estimates of the error gradient. We take a batch size that is small enough to fit within the memory parameters of our device and also divides the total number of images in our sample so that the steps per epoch is even for each epoch. Lastly, the choice of validation steps and workers are set to be unchanged from their recommended values.

Results

During training, overall loss per epoch on training set and validation set is given by the following trend 3. The loss on the training set drops sharply after the first epoch and continues a downward trend culminating at 0.0161 at the end of 20 epochs. On the other hand, the loss on the validation set at first increases until the fourth epoch and starts to decrease until the eighth. After the eighth epoch validation loss oscillates with small average amplitude and ends at 0.0652 in the twentieth epoch.

The model is evaluated on 542 images taken from the quadrotor following behind the target, 270 images while the quadrotor is on patrol and the target

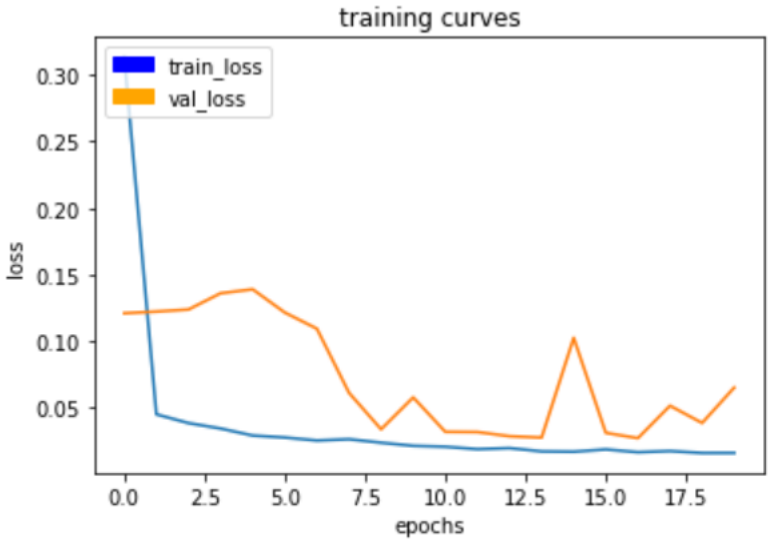

Figure 3

is not visible, and 322 images of the target taken from far away. The scores calculated to detect the hero, background, and other people for images taken while following the target is given by 4.

```
number of validation samples intersection over the union evaulated on 542
average intersection over union for background is 0.9867031966018991
average intersection over union for other people is 0.25793394516376944
average intersection over union for the hero is 0.8439465866385055
number true positives: 539, number false positives: 1, number false negatives: 0
```

Figure 4

The scores calculated to detect the hero, background, and other people for images taken while the quadrotor is on patrol without the target is given by 5.

```
number of validation samples intersection over the union evaulated on 270
average intersection over union for background is 0.9833299829472373
average intersection over union for other people is 0.6470363892850577
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 193, number false negatives: 0
```

Figure 5

The scores calculated to detect the hero, background, and other people for images taken when the target is far away is given by 6.

The final IOU score is given by an average of the IOU scores for the two sets of images containing the hero and is calculated to be 0.59. The final score which is the final IOU score multiplied by the weight of true positives over the sum of true positives, false negatives, and false positives is calculated to be 0.42.

```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.9893059362316634
average intersection over union for other people is 0.3221076379573407
average intersection over union for the hero is 0.3357872687485601
number true positives: 206, number false positives: 10, number false negatives: 95
```

Figure 6

Future Enhancements and Extensions

In order to improve image recognition of the hero, one suggestion is to collect more example states of which the hero could be in when detected in an image frame within her environment so that there is a more likely chance that a prediction state would coincide with a similar trained state. Training on as many example states as possible reduces the leap the model has to take to generalize to an unseen state of the hero. By training on multiple images where the hero is in a dense crowd, the model has a better chance at distinguishing between the characteristic of other people and the hero. By training on images of the hero in different backgrounds and angles, the model will be able to identify the hero in more varied environments and will be more likely to classify correctly due to the wide collection of similar images.

Further, we could upgrade our memory capacity on our computers to train on a depth expanded architecture by increasing the number of filters and convolutional layers in the fully convolutional network. To avoid saturation and degradation of accuracy as a byproduct of increasing depth, we can mimic the Resnet architecture by introducing residual blocks to our network [2]. To regulate overfitting we can employ dropout and L2 regularization [4]. We could also implement a consistent decrease in learning rate and an increase in batch size every time the validation accuracy plateaus [6]. Other adaptive learning rate methods that can be used are Adagrad, RMSprop, and Adam [4].

The fully convolutional network used here can certainly be reused to train on different data sets for the task of following other objects such as dogs, cats, or cars but the initial data set to train the model should be replaced with new data taken from following the new object so the new model can capture the new object's idiosyncratic characteristics. However, if the new data has larger or smaller feature maps, the network would need to be rescaled to accomodate the size disparity. A limitation of the model is that it is made to identify the specific characteristics of the hero and is not meant to generalize to following other people that do not look like the hero. However, turning off the color channels and running the model on training examples of identifying the hero could generalize to a model that could follow people of average body types similar to the hero. In fact, by the high number of false positives found in the predictions for images where there was no hero, this already suggests that our model classifies the hero's body type as the same as other people's body types. Alternatively, in the case for following dogs or cats, training on following a set of images containing a dog could generalize to following cats as their gaits and body types are similar. In the case of a car, following a specific car could boil down to identifying the license plate within the scene which in this case a sharp, translation invariant classifier could do the job whereas following a general version of a car would require learning the general features of the back

of a car.

Another limitation to our model is that it is hard for our classifier to identify the hero when she is far away. This may be either due to inadequate sampling of the hero from a distance or it may be due to the fact that nondescript characteristics of the hero from far away appear more like a featureless clump of noise whose configuration is hard to come by again and does not generalize well with a coarse feature map. To overcome this obstacle a deeper network could be used to train for finer patterns but there is always a limit to what we can detect correctly.

References

[1] F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv preprint arXiv: 1610.02357*, 2017

[2] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv: 1512.03385*, 2015

[3] I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. MIT Press. *http://www.deeplearningbook.org*, 2016.

[4] Stanford. CS231n Convolutional Neural Networks for Visual Recognition. *http://cs231n.github.io*, 2017.

[5] S. Ioffe, C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv: 1502.03167*, 2015

[6] A. Devarakonda, M. Naumov, M. Garland. AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks. *arXiv preprint arXiv: 1712.02029*