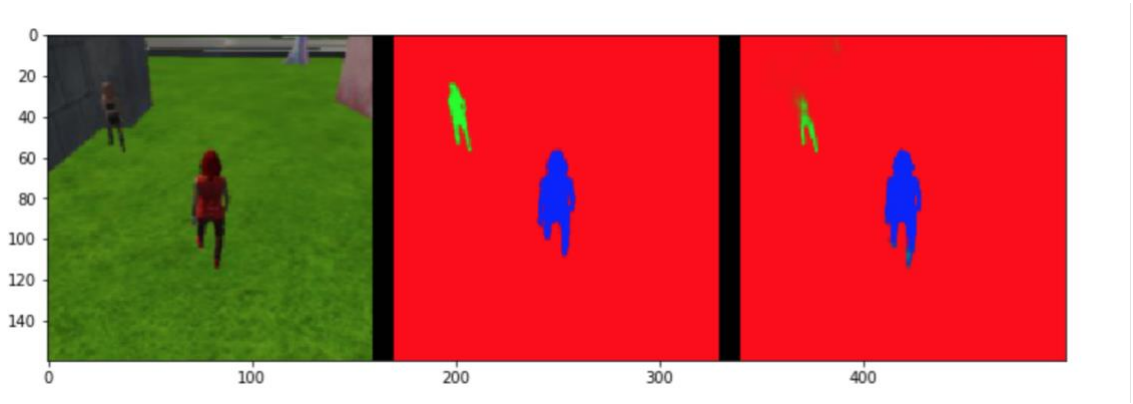
Follow Me Project

By Patrick Wellins

Overview

A trained fully convolutional network used for semantic segmentation will be explained in the following report. The fully convolutional network will enable a quadcopter to follow the hero in a simulation. With semantic segmentation, the goal is to classify each pixel of an input image to a specific class. The objective is to classify each input pixel to one of three classes, these classes consist of the hero, other people, and the background. Building a FCN that effectively implements semantic segmentation will allow the quadcopter to follow the hero. Image 1 illustrates the conversion from an input image to an output image where each pixel is designated to one of three classes. The red pixels represent the background, green pixels represent other people, and blue pixels represent the hero. The middle image of Image 1. is the ground truth labeling of the input image. The closer the output of the FCN is to the ground truth the better, accurate perception allows robots to function in their environments more effectively just as a 20-20 vision will enhance the capabilities of a pilot. For the fully convolutional network to be useful it must effectively classify pixels of images that are not in the training set. The scoring metric for model effectiveness in this project is the intersection over union score.

Image 1.

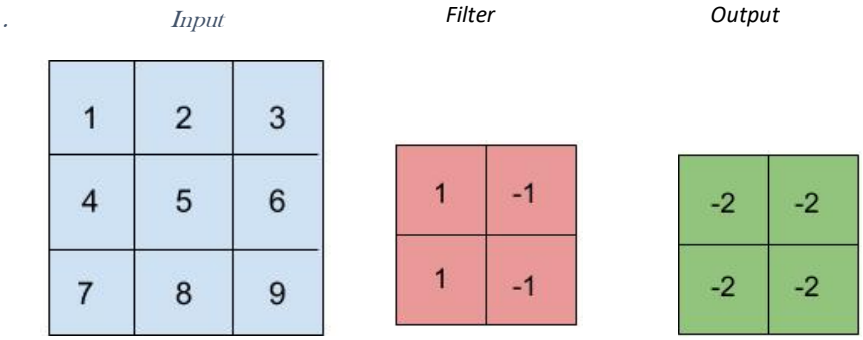


FCN Model Architecture Components

In this section, a brief description of the components of the FCN is given. This background information will make the description of the trained model clearer.

Convolutional Layer

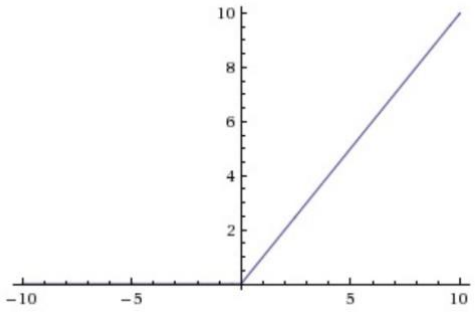
Input layers are traversed with filters to generate output layers. These filters are made up of weights that are optimized with the gradient descent algorithm.



For example, traversing the 3x3 blue matrix above with the 2x2 red filter with a stride of one will create the 2x2 green matrix as output. Multiple filters can be used in convolutional layers. For example, if three filters were used for the above example, there would be three 2x2 matrices as output. After filters are used to traverse input layers, an activation function is applied to the outputs. Generally, in image recognition the 'relu' activation function is applied. If a 'relu' activation function is applied to the green matrix, all entries will go to zero. Below is a graph of the 'relu' function.

Relu Activation Function

Relu activation function.

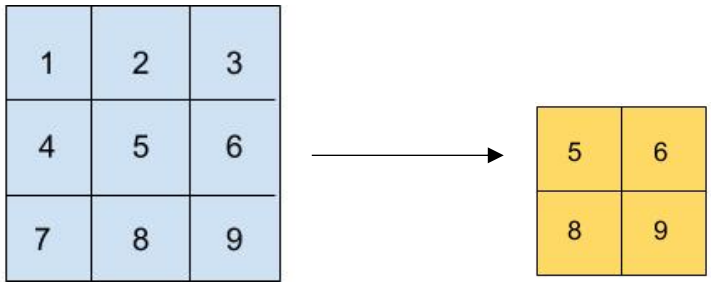


$$f(x) = \begin{cases} x \; if \; x > 0 \\ 0 \; otherwise \end{cases}$$

Pooling

Pooling layers reduce the dimensionality of convolutional layers in a neural network. An example of max pooling is shown below. Max pooling can reduce the number of parameters in a network while still retaining important information.

Max Pooling Operation



In the example above a 2x2 window with a stride of 1 for max pooling transforms the blue matrix to the yellow matrix.

Normalization

Normalization takes entries of a convolutional layer and sets the mean of the values to zero, and sets the variance of the values to 1. If X represents the values in a convolutional layer, $X_{normalized}$ can be represented by the equation below. Where μ_x is the mean of the X, and σ_x is the variance of X.

$$X_{normalized} = \frac{X - \mu_x}{\sigma_x}$$

Normalizing the features allows for gradient descent to work more efficiently by making the cost function more symmetric.

1x1 Convolutional Layer

The 1x1 convolutional layer allows for the network to retain spatial information about the input image and reduce the dimensionality of features. In contrast, fully connected layers do not retain spatial information about the input image. In semantic segmentation, it is important to keep this spatial information for correct pixel classification.

Bilinear up-sampling

Bilinear up-sampling increases the dimension of the input layer. Since the input image's size gets reduced by the encoder layers in the network, it is necessary to increase the dimension of the layers until the output is the same dimension as the ground truth label image.

Concatenation

Concatenating layers in the fully convolutional network allows non-adjacent layers to be used as inputs to deeper layers in the network. Concatenation performs a function that is similar to skip connections. Skip connections in fully convolutional networks can improve the resolution for the task of semantic segmentation. During the process of down sampling some information is lost, concatenating layers in the network counters the loss of information due to down sampling.

Cross Entropy Loss Function

$$\boldsymbol{D(S,L) = -\sum_i L_i\, log(S_i)}$$

To make this equation more concrete I will provide an example with actual numbers. Suppose there are three classes that the model is trained to predict and the correct label is $L = \begin{bmatrix}1.0\\0.0\\0.0\end{bmatrix}$ and the prediction is $S = \begin{bmatrix}0.7\\0.2\\0.1\end{bmatrix}$. The loss of this prediction would be $-\ (1 * \log(0.7) + \ 0.0 * \log(0.2) + \ 0.0(0.1)) = 0.356$. With an understanding of the natural log function, it is known that the closer the first entry in the vector S comes to 1 lower the loss will be. In this instance, each pixel belongs to one of three classes, and the objective is to find parameters in the model to minimize this loss function.

Learning Parameters with Gradient Descent

$$W := W - \alpha \nabla E$$

W the vector of weights in the FCN.

α is the learning rate.

∇E is the gradient of the error function with respect to the weights.

An intuitive way to understand what is happening with gradient descent is to imagine being on a mountain, high up on the mountain the error is high. And to minimize the error steps are taken in the direction that take you lower the fastest. The size of the step corresponds to the learning rate. For a detailed explanation, Michael Nielsen's online book of deep learning is the best written explanation of backpropagation I have seen.

Description of Actual FCN

The fully convolutional network consists of three encoder layers, a 1x1 convolutional layer, and three decoder layers followed by a soft max output. The input to this network is an image with dimensions 160x160x3, as the input passes through the network the eventual output has the same dimensions as the input. The first encoder consists of a convolutional layer with a filter size of 3, a stride of 1, and 'same' padding, which sets the padding to 1. Padding surrounds the image with zeros, the number of zeros added to the perimeter of the input depends on the padding value. Without padding, information on the edges of images is not used to the same extent as information in the middle of the image. Padding the image solves the problem of lost image information. The output of this layer has the dimension 160x160x32, this dimension is the result of 32 filters being applied to the input layer. The entries of this layer are normalized and then max pooled. The max pooling operation down samples the height and width of the input by half, which creates a layer with the dimension 80x80x32. This down sampling is achieved with a 2x2 pooling window, using a pooling window of this size down sampled the input by one half. The process of convolution, normalization, and max pooling is repeated another two times. After the initial three encoder layers of the network, the layer dimension is 20x20x128. At this point a 1x1 convolutional layer with 128 filters is used instead of a fully connected layer. After the 1x1 convolution layer, a series of decoder blocks are employed. The decoder blocks up sample the previous layers by

doubling the width and height of the previous layer. In addition to increasing the dimension of the input, layers are concatenated to the up sampled layer. Concretely, the 20x20x128 dimension layer is up sampled to have dimension 40x40x128, and a 40x40x64 layer is concatenated to create a layer that has the dimension 40x40x194. This layer then gets convolved with 128 filters, which leads to a layer with the dimension 40x40x128. The entries of the layer then get normalized and the process repeats another two times until the dimension on the layer becomes 160x160x32. This layer then becomes the input to the soft max function. The soft max function outputs a probability distribution that corresponds to the three classes for each pixel. The final prediction corresponds to the entry with the highest probability.

Minimizing Error Function

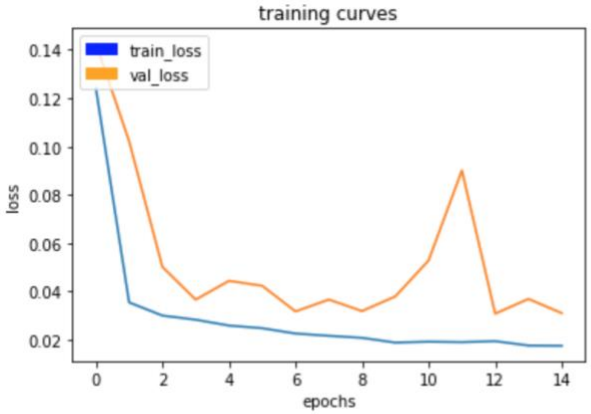
After 15 epochs of training the FCN had a loss score of 0.0176 on the training set and 0.0311 on the validation set. It is critical that during training the validation set error drops along with the training set error. If the validation loss is not dropping with the training loss, this is an indication that the model is over fitting the training data and will not generalize well to new data.

Final IoU Score

The final evaluation of the FCN model is the intersection over union score. The intersection over union is a measure of how well the output image matches the ground truth labeled image. If the output and the ground truth perfectly overlap then the intersection over union score is 100 percent. The final intersection over union score of the model is 0.4143.

Training Curves

The chart below shows the loss function for the training set and validation set throughout training. The training loss drops smoothly and the validation loss drops but with more volatility than the training loss. This is likely due to the images in the validation set having different distributions than the batches used in training.




```
200/200 [==============================] - 155s - loss: 0.0176 - val_loss: 0.0311
```

FCN Model Summary

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 160, 160, 3)	0
separable_conv2d_keras_1 (Se	(None, 160, 160, 32)	155
batch_normalization_1 (Batch	(None, 160, 160, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 80, 80, 32)	0
separable_conv2d_keras_2 (Se	(None, 80, 80, 64)	2400
batch_normalization_2 (Batch	(None, 80, 80, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 40, 40, 64)	0
separable_conv2d_keras_3 (Se	(None, 40, 40, 128)	8896
batch_normalization_3 (Batch	(None, 40, 40, 128)	512
max_pooling2d_3 (MaxPooling2	(None, 20, 20, 128)	0
conv2d_1 (Conv2D)	(None, 20, 20, 128)	16512
batch_normalization_4 (Batch	(None, 20, 20, 128)	512
bilinear_up_sampling2d_1 (Bi	(None, 40, 40, 128)	0
concatenate_1 (Concatenate)	(None, 40, 40, 192)	0
separable_conv2d_keras_4 (Se	(None, 40, 40, 128)	26432
batch_normalization_5 (Batch	(None, 40, 40, 128)	512
bilinear_up_sampling2d_2 (Bi	(None, 80, 80, 128)	0
concatenate_2 (Concatenate)	(None, 80, 80, 160)	0
separable_conv2d_keras_5 (Se	(None, 80, 80, 64)	11744
batch_normalization_6 (Batch	(None, 80, 80, 64)	256
bilinear_up_sampling2d_3 (Bi	(None, 160, 160, 64)	0
concatenate_3 (Concatenate)	(None, 160, 160, 67)	0
separable_conv2d_keras_6 (Se	(None, 160, 160, 32)	2779
batch_normalization_7 (Batch	(None, 160, 160, 32)	128
conv2d_2 (Conv2D)	(None, 160, 160, 3)	867

Total params: 72,089
Trainable params: 70,937
Non-trainable params: 1,152

The table above is a representation of the fully convolutional network that was described in the previous paragraphs. This network is what allows the quadcopter to follow the hero.

1x1 Convolutions and Fully Connected Layers

An example of fully connected layers in a network is seen in the LeNet-5 convolutional neural network. The LeNet-5 has been used to recognize hand written digits. In this network, there are two fully connected layers that precede the soft max output. The reason the fully connected layers can be used for this task is due to the objective of the network. An objective of the LeNet-5 convolutional neural network is to classify handwritten digits correctly, with this objective there is no regard for spatial prediction in the network architecture (LeNet – Convolutional Neural Network in Python 2016).
In contrast, the objective of the FCN for the follow me project is to effectively perform semantic segmentation. As previously stated, with the objective of semantic segmentation, the FCN must account for the spatial features of an object when making a prediction. Since fully connected layers lose spatial information, they cannot be used for semantic segmentation. Therefore, the 1x1 convolution is used and not fully connected layers, with 1x1 convolutions more depth and parameters can be added to the model while retaining spatial information.

Encoding and Decoding

The encoding segment of the FCN is responsible for extracting features from the image. Images are unstructured data, and do not have labeled features. Deep neural networks are effective for learning which features of an image correspond to a certain label. When looking at what the filters of a convolutional neural network learn after training there is a hierarchy of patterns from the shallow layers to the deep layers. In the shallow layers, simple patterns are seen in the filters that correspond to things like lines, deeper layers identify more complex patterns. Specific labels correspond to certain patterns of feature combinations. For example, if a CNN is being trained to recognize dogs, the combination of fur, paws, two eyes, and other attributes that belong to a dog could indicate that the image is a dog. The 1x1 convolutional layer retains information regarding what is in the image and where things are in the image. The decoder then scales this information up in size to construct a layer with the same width and height dimensions as the input. This final scaled up layer is the input to the soft max function.

Hyper Parameter Selection

The hyper parameters of the FCN model are variables that are tuned by the engineer and not by the gradient descent algorithm. Whether hyper parameter values are intelligently selected or not can determine whether a model is useful or not. The value of the learning rate is a great example of a hyper parameter that needs to be intelligently set. If the learning rate is too high there is the potential that the weights never converge to a global or local minima because weight updates always overshoot the minima. After a few experiments, the learning rate for the FCN was ultimately set to 0.009. The initial learning rate was set to 0.1, this number was selected due to prior experience with building other machine learning algorithms. The learning rate of 0.1 was not used is due to the observation that the loss function was not dropping after around the eighth epoch of training. This indicated that the minima was overshot due to parameter updates that were too large. The batch size hyper parameter controls how many images are used to measure the error and adjust the model weights. When using batches, the entire training set is not used when adjusting model weights with gradient descent. The batch size that was used for the model was 30. The batch sizes of 32, 64, 128, and 256 have been tested by researchers when training convolutional neural nets (Mishkin, D., Sergievskiy, N., & J. M 2016). I started with a batch size of 30, and after adjusting the

learning rate a few times I eventually surpassed the minimum intersection over union score of 0.40.

An epoch of training is when the entire training set is propagated through the network. The more epochs of training, the more the model's weights can be tuned to minimize error. It is important to acknowledge that training the network with excessive epochs can lead to an over fit model that does not generalize well to out of sample data. I experimented with 8 epochs of training and observed that the intersection over union score was less than 0.40. I increased the number of epochs to 15 and this was enough to reach an intersection over union score of 0.4143. The steps per epoch and validation steps were kept at default settings. The steps per epoch is set to 200, this is the number of batches that go through the network in one epoch. The validation steps variable is set to 50, this is the number of batches of validation images that runs through the network to calculate the validation loss.

Limitations of Fully Convolutional Network

The fully convolutional network would not be able to follow a cat, dog, or car effectively without making changes to training set and retraining the model. The training set would need to have images with dogs, cats and cars and corresponding labels such as Image 1. If the FCN was trained on an updated training set with additional classes then the FCN would be able to segment the image and the quadcopter would be able to follow different objects. This indicates the value of data in deep learning, there are so many tasks deep neural networks can do but they depend on having the correct labeled data to be achieved.

Future Improvements

There are several ways to improve the FCN. The first obvious choice is to collect more training data, particularly when the hero is far away. The intersection over union score for the images when the hero far away is low relative to when the quad is behind the hero, improving this score will improve the final score as well as enable the quad to find the hero at longer ranges more effectively. Several different FCN architectures can be built that vary in complexity, and grid search can be used on each architecture to optimize hyper parameters. For example, a total of 8 FCN's with different architectures can be designed. The number of encoder decoder blocks could range from two to five, two of the FCN's would have two encoder decoder blocks, another two FCN's with three encoder decoder blocks and so on. The number of filters for the models can also be set to different values. For example, one of the FCN's with two encoder decoder blocks could have a 128 channel 1x1 convolutional layer and the other could have a 256 channel 1x1 convultional layer. A grid of hyper parameter values that include learning rate, batch size, and epochs can be created and used for all the different FCN architectures. After training the eight different architectures with grid search to find optimal hyper parameters, a FCN with a final score that is greater than 0.4143 would likely emerge.

References

LeNet - Convolutional Neural Network in Python. (2016, June 17). Retrieved March 20, 2018, from https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/

Mishkin, D., Sergievskiy, N., & J. M. (2016, June 13). *Systematic evaluation of CNN advances on the ImageNet*[Scholarly project]. Retrieved March 20, 2018, from https://arxiv.org/pdf/1606.02228.pdf