Machine Learning Engineer Nanodegree

Capstone Project -- Fully Convolutional Network for Image Segmentation

Charlio Xu November 2nd, 2017

I. Definition

Project Overview

Computer vision is a popular and fascinating field for deep learning. Common tasks in computer vision include image classification, object detection, object localization and image segmentation. In this project, we are going to implement two fully convolutional networks for image segmentation. In image segmentation, our task is not only to detect and localize common objects which have been learned by some classification neural network, but also need to give a pixel-wise classification. So this task is more involved than object detection and localization in which a box is usually used to capture objects in the image. The following pair of images and the corresponding ground-truth segmentation shows clearly what segmentation is:

The technique we'll use in this project is Fully Convolutional Network, its application to image segmentation is described in the paper: Fully Convolutional Networks for Semantic Segmentation

Problem Statement

The goal of this project is to implement the fully convolutional networks FCN32 and FCN16 fine tuned on VGG16 to give pixel-wise classification of images in order to extract detailed information about the localization and shape of common objects in the images. Tasks involved in the project are the following:

1. Download and preprocess PASCAL VOC2012 data
2. Design and implement FCN32 and FCN16 models in Keras
3. Train FCN32 and FCN16 with VOC segmentation data
4. Demonstrate the predictions from FCN32 and FCN16

Metrics

Dice coefficient is the common choice in object recognition and localization tasks. Though it is not used in the segmentation in the above paper, we are going to use it for our project because later I'll transform segmentation images into binary masks which only contain values of 0 and 1, so they can be analyzed by the Dice coefficient easily.

Dice coefficient is used to measure similarity of two mask images whose value at each pixel is either 0 or 1. It is defined as follows in our project:

```
def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + 1.0) / (K.sum(y_true_f) + K.sum(y_pred_f) + 1.0)
```

We will use it as our evaluation metric instead of accurary which is not suitable for our model because a mask image usually contains far more black background pixels of value 0.

We will also use the corresponding loss function which is simply the negative of Dice coefficient:

```
def dice_coef_loss(y_true, y_pred):
    return -dice_coef(y_true, y_pred)
```

II. Analysis

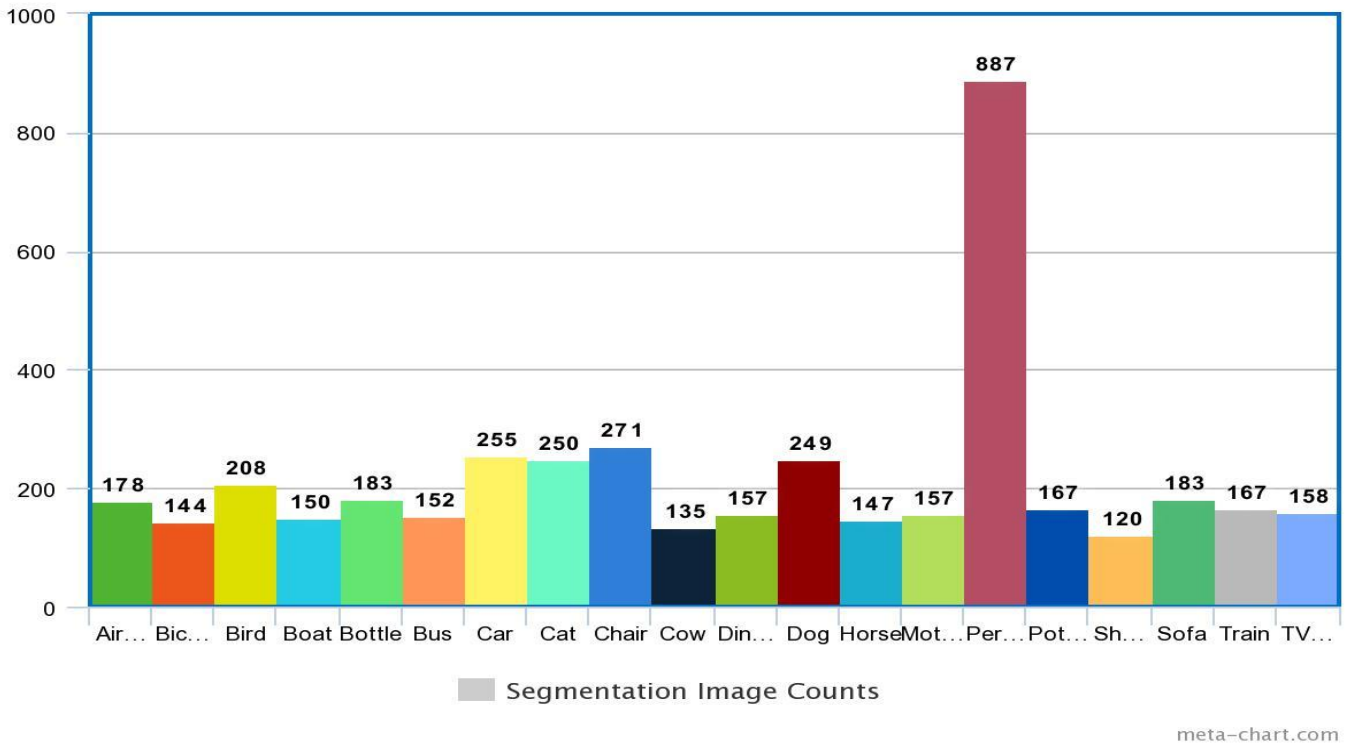
Data Exploration

I am going to use the PASCAL VOC2012 dataset in this project. Notice that this dataset is also used for image classification which is not our purpose. We will only need to use a subset of it. The detailed specification can be found in the Segmentation Image Sets. In summary, there are 2913 pairs of image and segmention in our training and validation dataset. Each image contains some common objects for us to detect. These objects have been learned previously in other classification models. Each image or segmentation has size (500, 332, 3).

Exploratory Visualization

Below is another example pair of image and the corresponding segmentation:


The distribution of 20 classes of objects is shown in the following histogram:



Algorithms and Techniques

Convolutional Neural Network

A convolutional neural network contains several convolutional layers which uses a number of small filters to apply convolution with the input images or intermediate features locally in order to learn 2D-topology-sensitive local patterns. Deep cnn can contain many convolutional layers. For example, the VGG16 classification model has 13 convolutional layers and 3 fully connected layers. Each convolutional layers use 64-512 filters of size (3, 3). Finally, after a softmax, the model will produce a class label for the input image.

Fully Convolutional Neural Network

A cnn classification model is only able to produce a label for the input image. If we want to preduce a pixel-wise classification, which means we'll label every pixel in the image to determine whether the pixel belongs to a human, a cat, a car or a house, etc... Then how can we accomplish this kind of task? The answer is given in this paper Fully Convolutional Networks for Semantic Segmentation. Recall in the VGG16 model, there are three dense layers after the 13 convolutional layers. Now we replace these dense layers with conolutional layers instead, so that at these intermediate steps, we still preserve the 2D structure of the image and not flatten it. In this way, we will get information both about the classes prediction and the localization. Moreover, recall that we use pooling layers in cnn models to reduce the image size. For segmentation, the desired output of a model shall have size (width, height, classes). Then how can we go from smaller reduced images to the original size?

Deconvolutional Layers

The answer is to use deconvolutional layers. Deconvolutional layer is also called convolution transpose. In Keras, it is named Conv2DTranspose. After a deconvolutional operation, we often need a cropping to further adjust the image size.

Skip Structure

The FCN32 model simply upsamples reduced images 32 time to the original size which is (224, 224). Since the model uses result from deep layers which contain more information about global structures of an image instead of local patterns, we will see that prediction from FCN32 is coarse. Though coarse, it still quite successfully captures the main parts of objects in the images. In order to detect smaller patterns like thin and long parts such as wings, legs, arms, antennas, only need to add weights from more shallow layers. This combination of shallow layers containing local patterns with deep layers containing global structures is called a skip structure. In this project, we will use a skip structure in the FCN16. One can go even further to implement the FCN8 model also mentioned in the paper, but we'll stop at FCN16 in this project, since FCN models are even more time-consuming to train than the already time-consuming CNN models.

Transfer Learning on Previous Simpler Models

FCN models are built upon CNN models. After all, in order to conduct image segmentation, one first should know very well about the object classes we are going to find. However, we'll skip the classification training in this project and instead to use weights from the 13 convoluational layers in VGG16 in our FCN32 model. Recall our FCN32 model differs from VGG16 only after the 13 convolutional layers. In addition, we'll use weights of all convolutional layers from FCN32 in training of FCN16. So if you are going to go through the training process in this project, remember to first download the VGG16 weights.

Benchmark

Exact benchmark for our project does not exist, since I simplified the problem by using binary masks instead of colored segmentations. So instead of predicting among say 21 classes, we only need to predict between 2 classes: 0 for black background and 1 for white object mask. However, I still found something that might be helpful as benchmarks: The Berkeley Segmentation Dataset and Benchmark

To have a sense about the final power of the full-ledged FCN series models, below is the evaluation results in the paper:

	pixel acc.	mean acc.	mean IU	f.w. IU
FCN32s-fixed	83.0	59.7	45.4	72.0
FCN32s	89.1	73.3	59.4	81.4
FCN16s	90.0	75.7	62.4	83.0
FCN8s	90.3	75.9	62.7	83.2

III. Methodology

Data Preprocessing

Data preprocessing is done in the data.py file

For each original image, we resize it to (224, 224, 3) which is compatible with the VGG16 model. For segmentation purpose, we also normalize it by substracting the mean and divided by the standard deviation.

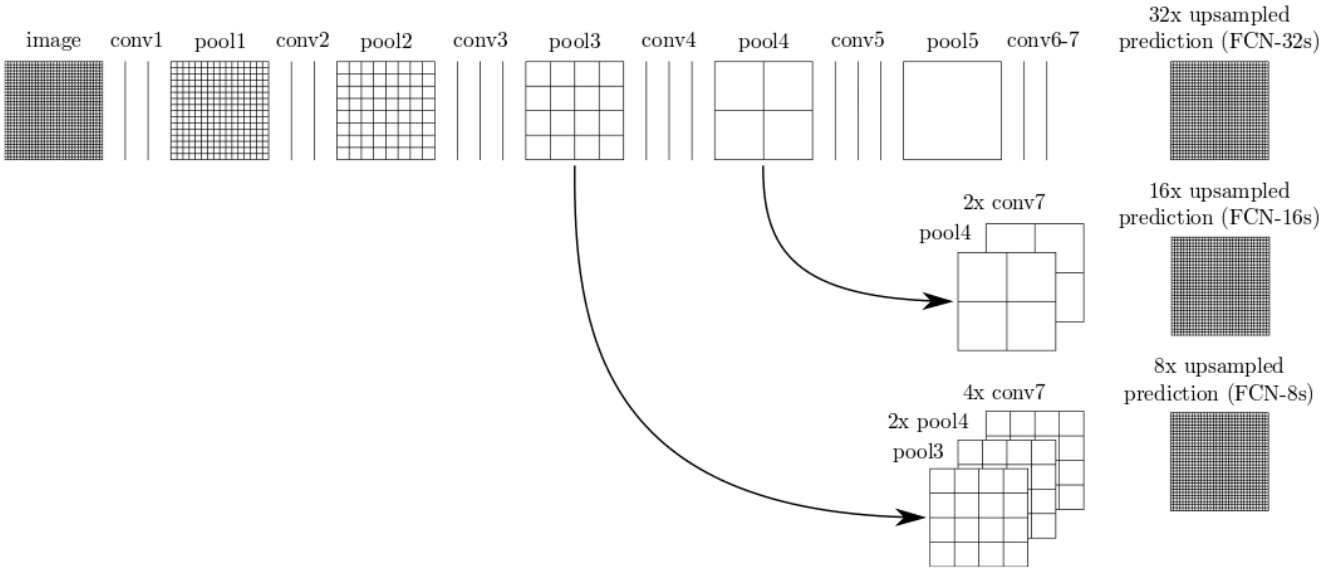
For each segmentation image, we transform it into a white/black mask. So finally our training data will be two numpy arrays of sizes (2913, 224, 224, 3) and (2913, 224, 224, 1). There are 2913 pairs of normalized image and corresponding mask. Pixels in the normalized image have mean 0 and std 1. Each pixel in the mask takes values either 0 or 1. Following shows a mask image in which pixels take values 0 or 255.

Implementation

Implementation consists the following steps.

1. Define and compile VGG16 model and load in its pre-trained weights
2. Define and compile FCN32 model and load weights from convolutional layers of VGG16
3. Train FCN32 model with the preprocessed data
4. Define and compile FCN16 model and load weights from convolutional layers of FCN32
5. Train FCN16 model with preprocessed data

The following figure shows the implementation ideas of the fully convolutional networks.



- In FCN32, the last convolutional layer is upsampled 32 times to get to the original image size of 224x224.
- In FCN16, conv7 is upsampled twice and then fused with pool4, and then upsampled 16 times to get to the original image size of 224x224.
- Difficulty: it is difficult to adjust the upsampling parameters. My solution is to use deconvolutional or conv2dtranpose layers in keras. In this way, one is free to choose the kernel

size, and then after the upsampling, one can apply a cropping layer to adjust the image size to 224x224.

- The loss function we use in the compilation of the modles is the negative of Dice coefficient to measure the discrepancy of the prediction and the original mask.
- Training of convolutional models is usually time-consuming, which takes even longer for fully convolutional networks. In order to get results, one needs to train the models for many hours. For demonstration purpose, I only trained the models for less than two hours with one GPU used. However, the prediction result is still satisfactory for objects with simple shapes like cars, computers, desks, etc.

Refinement

- In training, I used the Adam optimizer. In my initial attempt training, I used the default specifications. However, I found it is ineffective in training the FCN models which contain 13M parameters. So customized the learning rate from the default 1e-3 to 1e-5. See the following comparison:

learning rate	validation loss for FCN32	validation loss for FCN16
1e-3	1.9822	2.2349
1e-5	0.7761	0.7835

- In FCN32 training, I first trained the model for 20 epochs, then I further trained it for another 10 epochs

epochs	validation loss for FCN32
20	0.8235
30	0.7761

- The final Dice coefficient I got for the validation dataset is 0.7761.
- For FCN16 model which is more complicated than FCN32 I first trained it for 10 epochs and then trained it for 5 more epochs to achieve a Dice coefficient of 0.7835 on the validation dataset. The reason I trained FCN16 less than FCN32 is that FCN16 is more time-consuming.

epochs	validation loss for FCN16
10	0.8194
15	0.7835

- Nothing else was modified in the refinement. Since the FCN models come from serious research papers and are well-established. So I know the architectures would work well before training.

IV. Results

Model Evaluation and Validation

1. 20% of the data were used for validation during training. I used all the 2913 pairs of image and mask for training. There is no test dataset, because the number of images and masks is small compared to the huge amount of parameters in the models(~134M).
2. Performance is given by apply the model directly on images to produce masks, and we can visually compare the results as will show later.
3. Since I used binary masks, the Dice coefficient is used as the evaluation metric, while its negative is used as the loss function during training.
4. I tested several images for FCN32 and FCN16. For all images, FCN16 performed better than FCN32 which is expected. The models perform very well to find objects, and are able to give the general shape mask of objects. Models give more accurate masks for objects without many tiny, small, thin, or long parts. Models perform worse on detecting local patterns like the wings of airplanes, legs and arms, etc. The reason is that we didn't exploit the shallow layer information. Though we used one shallow layer in FCN16, it is still not enough. In order to detect local pattern, we should use shallower layer weights. This is done in the FCN8 model mention in the paper.

5. Model robustness 1: the model works well on unseen images which contains the same common classes of objects that were used to train the VGG16 model. As I mentioned before, in order to give pixel-wise segmentation of objects, one has to first train hard on a classification model with these objects.

6. Model robustness 2: because of lack of powerful hardware and limited time, the models were not trained for tens of hours. So the model only works well on simple shapes like those of cars, desks, computers. It fails to capture local patterns, those tiny, thin, long parts of an objects, like arms, legs, plane wings, etc.

7. Model robustness 3: though the model fails to capture local patterns. It indeed works very well to capture the central location of any common objects in an image. This is due to the fact that we applied convolution to deep layers which contain global information in an image.

Justification

1. The FCN series models I trained are very successfully in finding common objects. They also work well in draw the overall shape of objects since we extract a lot of global information from the deeper layers.
2. Models fail to detect local patterns because local information from shallow layers are not used much.
3. Since it is easy to write model definitions in Keras, one can continue to define the FCN8 model, or even FCN4 or FCN2 to get a better result.
4. Due to lack of time, I only trained FCN32 for one hour, and less than 30mins for FCN16. If trained longer, models shall perform better.
5. The segmentation dataset only contains 2913 pairs of image and masks. One can use augmentation to generate more data in order to train the models longer and effectively.

6. Comparison with benchmark models: The score I got on FCN16 is 07835. The following chart shows scores on some previous segmentation algorithms. The FCN16 model I trained outperformed all benchmark models except the first one specifically designed on human detections. This is quite reasonable since as I mentioned, in order to detect local patterns like arms and legs, one needs to fuse more shallow layers. However, in FCN16 model, we stopped at pool layer 4.

Rank	Score	Algorithm
0	0.79	Humans
1	0.71	Ren et al. NIPS2012 (color)
2	0.71	gPb-ucm (color)

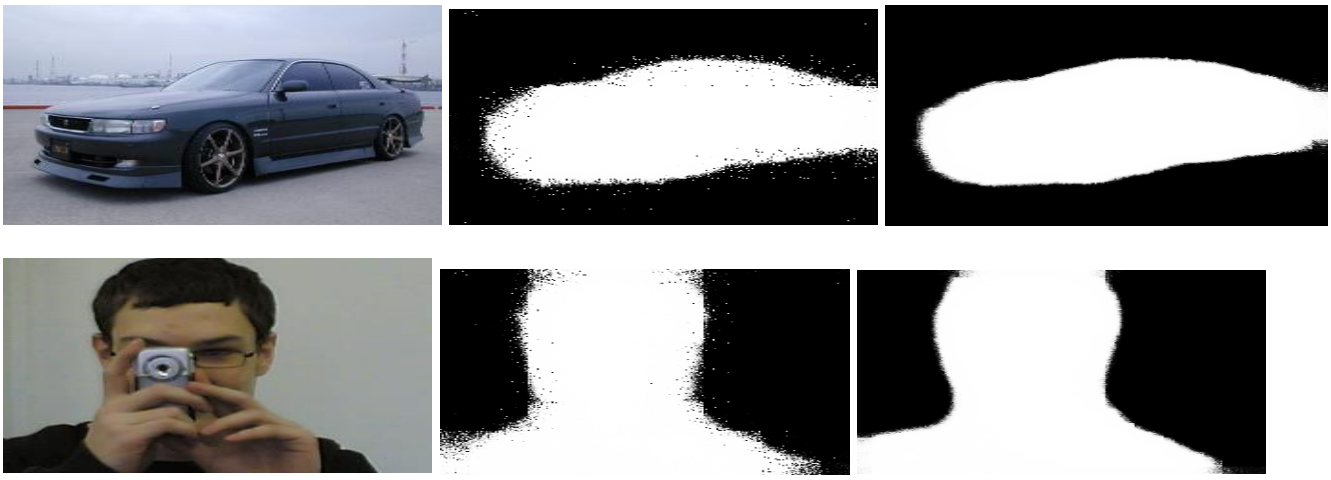
3	0.70	Global Probability of Boundary (color)
4	0.68	xren (color)
5	0.67	Arbelaez POCV2006
6	0.66	Boosted Edge Learning (color)
7	0.65	min-cover
8	0.65	Brightness / Color / Texture Gradients
9	0.63	Brightness / Texture Gradients
10	0.57	Color Gradient
11	0.43	Random

7.

V. Conclusion

Free-Form Visualization

Let us look at two examples of predictions. Left image is the original image, middle one is the prediction from FCN32, right one is the prediction from FCN16.



Reflection

1. The models are well-established in the paper and implemented by the authors. However, I found online resources about fully convolutional networks like blogs, git repos, papers not easy to read. This project shall be the easiset one to understand and implement while still works well up to this time.
2. It takes some time to understand all the differences FCN models make from the original VGG classification models. New concepts include deconvolution, dice coefficient, or cross entropy for logits which are used by other developers for the full-fledged FCN models.
3. These models have huge amount of weights about 130M. The h5 files are about 1.5GB compare to VGG16's 500MB. So in order to get good results, one has to feed models a lot of data. However, even the VOC dataset only has 2913 paired images. If one wants to use it for other more professional purposes, then it will be difficult to collect enough data for training. For example, look at this kaggle competition on ultrasound nerve segmentation
4. Recall we used weights of VGG16 to train our FCN models. In order to do image segmentation, one needs to first train a classification modle on the objects that one wants to detect and localize. The VGG16 model is trained on 1000 classes of common objects. If one wants to detect other uncommon objects, then one will have more work to do.

Improvement

1. This project is a good starting point for one to learn more about image segmentation. I made it easy to understand and implement in Keras.
2. Start from here, one can continue to implement FCN8.
3. Then the next step will be to implement the full-fledged FCN series models, which is able to detect object classes. Original FCN models were trained for 20 object classes, while I simplified it to binary.
4. One runnable implementation in tensorflow is given here
5. Implementation in caffe from the original author is given here