Prepared By
Aram, Greg, Sami

Assignment 2: Segmenting Images

CS 699: Rep. Learning
Due: Sat'10-5 @ Midnight

Introduction

The purpose of the assignment is to give you practical experience training vision networks in TensorFlow[1]. You will train Convolutional Neural Network (CNN) for image segmentation & detection. Given a test image $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$, where W and H are the spatial dimensions, and C is the channel dimension (usually $C = 3$, for RGB images), image segmentation models we consider take the form:

$$h_\theta : \mathbb{R}^{H \times W \times C} \rightarrow [0, 1]^{H \times W \times M} \tag{1}$$

where M is the number of labels. Specifically, running the model on image $\mathbf{x}$ yields $h(\mathbf{x})$ as a Tensor containing confidences per label for every input pixel. The labels correspond to visual entities including 'person', 'bike', 'road', 'background', etc. Some dataset entries:

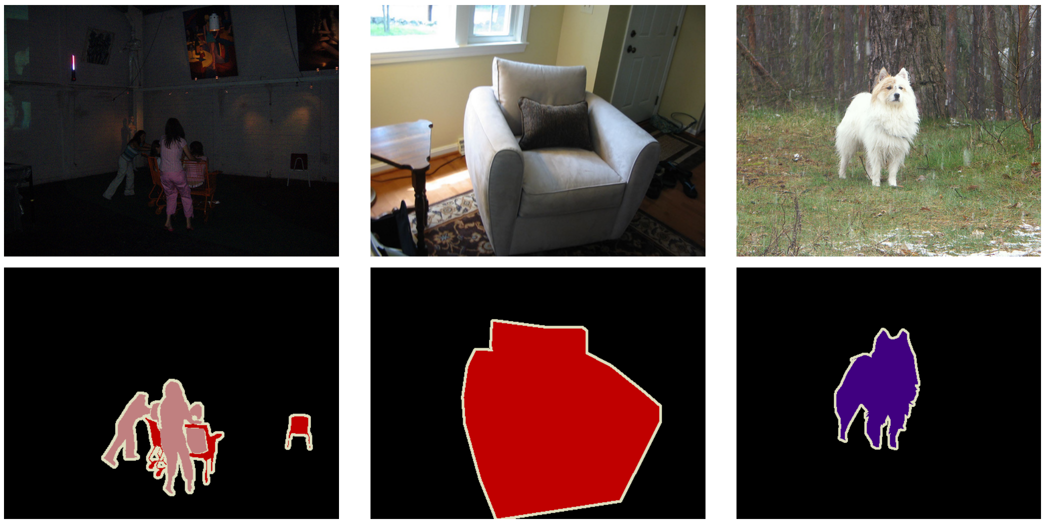

Figure 1: Examples from dataset. Top: Images. Bottom: Segmentation masks.

Dataset

You should download and untar the dataset locally (and/or onto vocareum):
`https://drive.google.com/file/d/17sGsnFrsjDkJUaQfyP82toBH1oBoZBrG`

The dataset tar consists of:

- Directory `images`, with 2330 JPG images (named `0000.jpg` thru `2329.jpg`). Images differ in size, but the larger spatial dimension is $\leq$ 500px. These are the *training images*.

[1]You can use PyTorch, though teaching staff is focusing on, and are able to help more with, TensorFlow.

- Directory `tf_segmentation`. It contains one PNG image for every image in `images`. The PNGs are grayscale (all channels are the same value i.e. you can decode just one channel). The spatial dimensions of a PNG equal its corresponding JPG. The pixel values are discrete (`uint8`) and take-on integer values $\{0, 255\} \cup \{1, 2, \ldots, M\}$, where 1 thru M refer to *entity pixels*, 0 refers to background, and 255 refer to object boundaries.
- Directory `segmentation`, similar to `tf_segmentation`, contains one label PNG mask per example, but its colors are chosen for visual distinction rather than being $\{0, 255\} \cup \{1, 2, \ldots, M\}$. It is left for debugging. We computed `tf_segmentation` from `segmentation`.

Tasks

You are to implement a segmentation model, using python (any framework, so long it works on vocareum). The starter code contains SegmentationModel, which you are expected to implement. Our auto-grader code will construct your model with no arguments, and will call predict on a batch of images.

```python
from PIL import Image
model = SegmentationModel()
model.load('model_file')
test_image_files = ...  # array of file names
images = [zeropad(np.array(Image.open(f))) for f in test_image_files]
predictions = model.predict(images)
```

In addition to implementing `SegmentationModel` (any framework), you are asked to implement helper functions (in TensorFlow). **Before continuing, untar the data and modify flag `dataset_dir` of code to point to untarred data path**.

[40 points] Task 1: Reading Data

Often times (like in this assignment), the training data cannot be directly plugged-into your learning model. Specifically, the data requires pre-processing (at the very least, reading the images into image tensors). In this task, you will implement 2 functions (headers below).

The first one accepts path to a text file, which contains image IDs (one per line). It should return a tf.data.Dataset object that when iterated, gives the full path of the training images (in `images`; one path per image ID). In addition, it can optionally (if second argument is set) return an additional Dataset that is capable of iterating over ground-truth images (in `tf_segmentation`). Both datasets must maintain their order i.e. zipping them should give (image, segmentation) pairs.

For the ease of programming with batches, we would like to make all images have the same spatial size. The second function should accept a string tensor (image filename). It builds a TensorFlow subgraph that decodes the image, and pads its spatial dimensions with zeros.

```
def get_filename_data_readers(image_ids_file, get_labels=False,
                              x_jpg_dir=None, y_png_dir=None):
  """Given image IDs file, returns Datasets, which generate image paths.

  The goal of this function is to convert from image IDs to image paths.
  Specifically, the return type should be:
    if get_labels == False, return type should be tf.data.Dataset.
    Otherwise, return type should be pair (tf.data.Dataset, tf.data.Dataset).
  In both cases, the Dataset objects should be "not batched".

  For example, if the file contains 2 lines: "0000\n0001", then the returned
  dataset should give an iterator that when its tensor is ran, gives "0000" the
  first time and gives "0001" the second time.

  Args:
    image_ids_file: text with one image ID per line.
    get_labels: If set, returns 2 Datasets: the containing the image files (x)
      and the second containing the segmentation labels (y). If not, returns
      only the first argument.
    x_jpg_dir: Directory where each image lives. Specifically, image with
      ID "image1" will live on "x_jpg_dir/image1.jpg".
    y_png_dir: Directory where each segmentation mask lives. Specifically,
      image with ID "image1" will live on "x_png_dir/image1.png".

  Returns:
    instance of tf.data.Dataset, or pair of instances (if get_labels == True).
  """
  x_jpg_dir = x_jpg_dir or os.path.join(FLAGS.data_dir, 'images')
  y_png_dir = y_png_dir or os.path.join(FLAGS.data_dir, 'tf_segmentation')

def decode_image_with_padding(im_file, decode_fn=tf.image.decode_jpeg,
                              channels=3, pad_upto=500):
  """Reads an image, decodes, and pads its spatial dimensions, all in TensorFlow

  Args:
    im_file: tf.string tensor, containing path to image file.
    decode_fn: Tensorflow function for converting
    channels: Image channels to decode. For data (x), set to 3 channels (i.e. RGB).
      For labels (segmentation masks), set to 1, because other 2 channels contain
      identical information.
    pad_upto: Number of pixels to pad to.

  Returns:
    Pair of Tensors:
      The first must be tf.int vector with 2 entries: containing the original height
        and width of the image.
      The second must be a tf.int matrix with size (pad_upto, pad_upto, 3)
        i.e. the contents of the image, with zero-padding.
  """
```

[20 points] Task 2: Ground-Truth Masks

The Segmentation model that you will implement (next task) should output predicted segmentation i.e. $[0, 1]^{H\times W\times M}$, where $H = W = 500$. However, we are aware that our images were padded Implement a function, that will most-likely be used in your model. The function header is copied:

```
def make_loss_mask(shapes):
  """Given tf.int Tensor matrix with shape [N, 2], make N 2D binary masks.

  These binary masks will be used "to mask the loss". Specifically, if the
  image is shaped as (300 x 400) and therefore so its labels, we only want
  to penalize the model for misclassifying within the image boundary (300 x 400)
  and ignore values outside (e.g. at pixel [350, 380]).

  Args:
    shapes: tf.int Tensor with shape [N, 2]. Entry shapes[i] will be a vector:
      [image height, image width].

  Returns:
    tf.float32 mask of shape [N, 500, 500], with mask[i, h, w] set to 1.0
    iff shapes[i, 0] < h and shapes[i, 1] < w.
  """
```

Hint for tasks 1 & 2 – You might find the functions useful: `tf.shape`, `tf.set_shape`, and `tf.sequence_mask`,

[100 points] Task 3: Segmentation Model

Implement the class `SegmentationModel`. Specifically, your class must implement '`train()`', '`save()`' for training, and '`predict()`', '`load()`' for testing. Our auto-grader for testing will **only** call '`load('model_file')`' (once) and '`predict()`' (once or more). Make sure you pre-train your model, creating file `model_file` on vocareum. Nonetheless, the save() and train() **will be called** later-on (after submission period is over), just to do spot-checking that your trainer indeed generates a model of similar performance.

Evaluation

Your model is graded based on its accuracy in predicting non-boundary (we ignore your model's output, at locations where ground-truth is at an object-object boundary pixel). We will be running your model on unseen test samples (outside the training set).

TODO: Add target accuracies