Project of EL-GY9123 Introduction to Machine Learning

Name: Chen Shen, Yaochen Liu

NetID: cs5236, yl4704

Guide to get data

The dataset used in this project is PASCAL Visual Object Classes Challenge 2011 (known as VOC2011).

To download it and make the program work, first you need to download the dataset in tar format.

```
$ curl -L http://host.robots.ox.ac.uk/pascal/VOC/voc2011/VOCtrainval_25-May-2011.tar \
> > VOCtrainval_25-May-2011.tar
```

Then extract it.

```
$ tar -xvzf VOCtrainval_25-May-2011.tar
```

Finally move the data to data folder.

```
$ mv TrainVal/VOCdevkit/VOC2011 /path/to/workspace/data
```

Main files

1. model.py

To build a FCN model based on VGG16.

```
#!usr/bin/python
#-*- coding:utf-8 -*-

'''
Fully Convolutional Neural Network Based on VGG16
'''

'''
Standard libraries
'''
import keras
import keras.backend as K
from keras.models import Model
from keras.layers import Input, Flatten, Activation, Reshape, Dropout, Conv2D, Lambda
from keras.layers.merge import add
from keras.regularizers import l2
from keras.applications.vgg16 import VGG16

'''
Custom functions
'''
from BilinearUpSampling import BilinearUpSampling2D

def fc_vgg16(filters=4096, weight_decay=0., block_name='block5'):
    '''
    Add fully connected layer to the model
    '''
    def f(x):
        x = Conv2D(filters=filters,
                kernel_size=(7, 7),
                activation='relu',
                padding='same',
                dilation_rate=(2, 2),
                kernel_initializer='he_normal',
                kernel_regularizer=l2(weight_decay),
                name='{}_fc6'.format(block_name))(x)
        x = Dropout(0.5)(x)
        x = Conv2D(filters=filters,
                kernel_size=(1, 1),
                activation='relu',
                padding='same',
                dilation_rate=(2, 2),
                kernel_initializer='he_normal',
                kernel_regularizer=l2(weight_decay),
                name='{}_fc7'.format(block_name))(x)
        x = Dropout(0.5)(x)
        return x
    return f

def upsampling_vgg16(classes, target_shape=None, scale=1, weight_decay=0., block_name='featx'):
    '''
    Upsampling layers
    '''
    def f(x, y):
        score = Conv2D(filters=classes,
                kernel_size=(1, 1),
                activation='linear',
                padding='valid',
                kernel_initializer='he_normal',
                kernel_regularizer=l2(weight_decay),
                name='score_{}'.format(block_name))(x)
        if y is not None:
            def scaling(xx, ss=1):
                return xx*ss
            scaled = Lambda(scaling, arguments={'ss': scale},
                    name='scale_{}'.format(block_name))(score)
            score = add([y, scaled])
        upscore = BilinearUpSampling2D(target_shape=target_shape,
                name='upscore_{}'.format(block_name))(score)
        return upscore
    return f

def UpSampler_vgg16(pyramid, scales, classes, weight_decay=0.):
    '''
    Upsampling block
    '''
    blocks = []

    for i in range(len(pyramid) - 1):
        block_name = 'feat{}'.format(i + 1)
        block = upsampling_vgg16(classes=classes,
                target_shape=K.int_shape(pyramid[i+1]),
                scale=scales[i],
                weight_decay=weight_decay,
                block_name=block_name)
        blocks.append(block)

    decoded = None

    for feat, blk in zip(pyramid[:-1], blocks):
        decoded = blk(feat, decoded)

    return decoded

def fcn_vgg16(input_shape, classes, weight_decay=0.,
        trainable_encoder=True, weights='imagenet'):
    '''
    Construct model
    '''
    inputs = Input(shape=input_shape)

    pyrmid_layers = 3

    # Use VGG16 as the encoder
    encoder = VGG16(include_top=False, input_shape=input_shape, weights=weights, classes=classes)

    # Set parameters in VGG16 to be untrainable
    for layer in encoder.layers:
        layer.trainable = False

    first = True
    pyramid = []
    for layer in encoder.layers:
        if first:
            x = layer(inputs)
            first = False
            pyramid.append(x)
        else:
            x = layer(x)
            if x.shape[1:-1] != pyramid[-1].shape[1:-1]:
                pyramid.append(x)

    x = fc_vgg16(filters=4096)(x)
    pyramid[-1] = x

    feat_pyramid = pyramid[::-1][:pyrmid_layers]
    feat_pyramid.append(inputs)

    outputs = UpSampler_vgg16(feat_pyramid,
            scales=[1, 1e-2, 1e-4],
            classes=classes,
            weight_decay=weight_decay)

    scores = Activation('softmax')(outputs)
    return Model(inputs=inputs, outputs=scores)
```

2. train.py

To train the model with VOC2011.

```
#!/usr/bin/python
#-*- coding:utf-8 -*-

'''
Standard libraries
'''
import keras
import keras.backend as K
from keras.optimizers import Adam
from keras.callbacks import (ModelCheckpoint,
        ReduceLROnPlateau,
        EarlyStopping,
        TerminateOnNaN,
        CSVLogger)
from keras import Input
from voc_generator import PascalVocGenerator, ImageSetLoader
import numpy as np

'''
Custom libraries
'''
import model

# Clear session
K.clear_session()

def arg_gen(dataset_name):
    '''
    Generate arguments of dataset generators
    '''
    image_set = 'data/VOC2011/ImageSets/Segmentation/{}.txt'.format(dataset_name)
    image_dir = 'data/VOC2011/JPEGImages/'
    label_dir = 'data/VOC2011/SegmentationClass/'
    target_size = (224, 224)
    return image_set, image_dir, label_dir, target_size

def main():
    '''
    Main function
    '''

    # Define common arguments
    checkpointer = ModelCheckpoint(
            filepath='output/fcn_vgg16_weights_tmp.h5',
            verbose=1,
            save_best_only=True)
    lr_reducer = ReduceLROnPlateau(
            monitor='val_loss',
            factor=np.sqrt(0.1),
            cooldown=0,
            patience=10,
            min_lr=1e-12)
    early_stopper = EarlyStopping(
            monitor='val_loss',
            min_delta=0.001,
            patience=30)
    nan_terminator = TerminateOnNaN()
    csv_logger = CSVLogger('output/tmp_fcn_vgg16.csv')

    # Set data generator
    datagen = PascalVocGenerator(image_shape=[224, 224, 3],
            image_resample=True,
            pixelwise_center=True,
            pixel_mean=[115.85100, 110.50989, 102.16182],
            pixelwise_std_normalization=True,
            pixel_std=[70.30930, 69.41244, 72.60676])

    # Define training set and validation set
    train_loader = ImageSetLoader(*arg_gen('train'))
    val_loader = ImageSetLoader(*arg_gen('val'))

    # Construct model
    fcn_vgg16 = model.fcn_vgg16(input_shape=(224, 224, 3),
            classes=21,
            weight_decay=3e-3,
            weights='imagenet',
            trainable_encoder=False)

    # Set optimizer
    optimizer = Adam(1e-4)

    # Compile model
    fcn_vgg16.compile(
            optimizer=optimizer,
            loss='categorical_crossentropy',
            metrics=['accuracy'])

    # Fit model with the above generators
    fcn_vgg16.fit_generator(
        datagen.flow_from_imageset(
            class_mode='categorical',
            classes=21,
            batch_size=1,
            shuffle=True,
            image_set_loader=train_loader),
        steps_per_epoch=1112,
        epochs=40,
        validation_data=datagen.flow_from_imageset(
            class_mode='categorical',
            classes=21,
            batch_size=1,
            shuffle=True,
            image_set_loader=val_loader),
        validation_steps=1111,
        verbose=1,
        callbacks=[lr_reducer, early_stopper, csv_logger, checkpointer, nan_terminator])

    # Save weights
    fcn_vgg16.save('output/fcn_vgg16.h5')

if __name__ == '__main__':
    main()
```

3. inference.ipynb

The jupyter notebook to test the model.

```
import numpy as np
import keras
import keras.backend as K
from keras.models import load_model
from voc_generator import PascalVocGenerator, ImageSetLoader
from BilinearUpSampling import BilinearUpSampling2D
import matplotlib.pyplot as plt
```

```
/home/yl4704/.local/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the s
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
def arg_gen(dataset_name):
    '''
    Generate arguments of dataset generators
    '''
    image_set = 'data/VOC2011/ImageSets/Segmentation/{}.txt'.format(dataset_name)
    image_dir = 'data/VOC2011/JPEGImages/'
    label_dir = 'data/VOC2011/SegmentationClass/'
    target_size = (224, 224)
    return image_set, image_dir, label_dir, target_size
```

```
datagen = PascalVocGenerator(image_shape=[224, 224, 3],
            image_resample=True,
            pixelwise_center=True,
            pixel_mean=[115.85100, 110.50989, 102.16182],
            pixelwise_std_normalization=True,
            pixel_std=[70.30930, 69.41244, 72.60676])

train_loader = ImageSetLoader(*arg_gen('train'))
val_loader = ImageSetLoader(*arg_gen('val'))

model = load_model('output/fcn_vgg16_weights.h5',
        custom_objects={'BilinearUpSampling2D': BilinearUpSampling2D})
print(model.summary())
```

```
WARNING:tensorflow:From /home/yl4704/.local/lib/python3.6/site-packages/keras/backend/tensorflow_backend
Instructions for updating:
keep_dims is deprecated, use keepdims instead
WARNING:tensorflow:From /home/yl4704/.local/lib/python3.6/site-packages/keras/backend/tensorflow_backend
Instructions for updating:
```

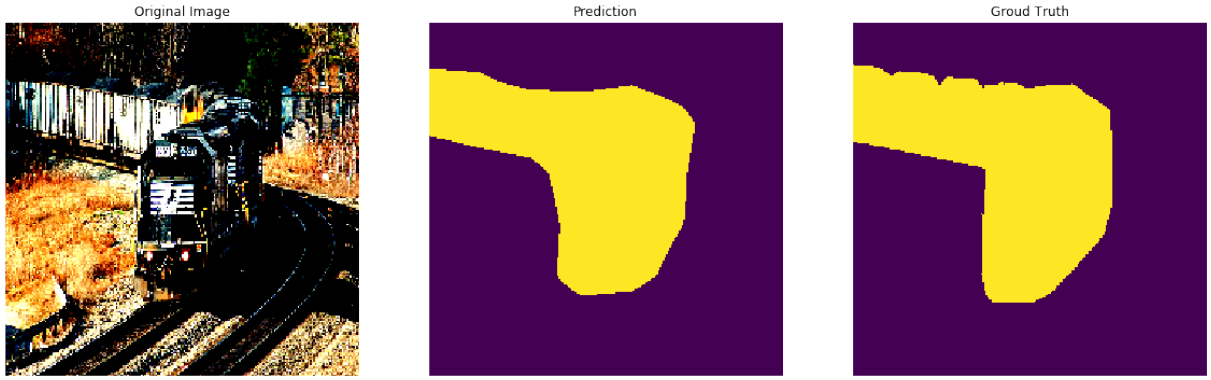
```
keep_dims is deprecated, use keepdims instead
WARNING:tensorflow:From /home/yl4704/.local/lib/python3.6/site-packages/keras/backend/tensorflow_backenc
Instructions for updating:
keep_dims is deprecated, use keepdims instead
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 224, 224, 3)   0
____________________________________________________________________________________________________
block1_conv1 (Conv2D)            (None, 224, 224, 64)  1792        input_1[0][0]
____________________________________________________________________________________________________
block1_conv2 (Conv2D)            (None, 224, 224, 64)  36928       block1_conv1[0][0]
____________________________________________________________________________________________________
block1_pool (MaxPooling2D)       (None, 112, 112, 64)  0           block1_conv2[0][0]
____________________________________________________________________________________________________
block2_conv1 (Conv2D)            (None, 112, 112, 128) 73856       block1_pool[0][0]
____________________________________________________________________________________________________
block2_conv2 (Conv2D)            (None, 112, 112, 128) 147584      block2_conv1[0][0]
____________________________________________________________________________________________________
block2_pool (MaxPooling2D)       (None, 56, 56, 128)   0           block2_conv2[0][0]
____________________________________________________________________________________________________
block3_conv1 (Conv2D)            (None, 56, 56, 256)   295168      block2_pool[0][0]
____________________________________________________________________________________________________
block3_conv2 (Conv2D)            (None, 56, 56, 256)   590080      block3_conv1[0][0]
____________________________________________________________________________________________________
block3_conv3 (Conv2D)            (None, 56, 56, 256)   590080      block3_conv2[0][0]
____________________________________________________________________________________________________
block3_pool (MaxPooling2D)       (None, 28, 28, 256)   0           block3_conv3[0][0]
____________________________________________________________________________________________________
block4_conv1 (Conv2D)            (None, 28, 28, 512)   1180160     block3_pool[0][0]
____________________________________________________________________________________________________
block4_conv2 (Conv2D)            (None, 28, 28, 512)   2359808     block4_conv1[0][0]
____________________________________________________________________________________________________
block4_conv3 (Conv2D)            (None, 28, 28, 512)   2359808     block4_conv2[0][0]
____________________________________________________________________________________________________
block4_pool (MaxPooling2D)       (None, 14, 14, 512)   0           block4_conv3[0][0]
____________________________________________________________________________________________________
block5_conv1 (Conv2D)            (None, 14, 14, 512)   2359808     block4_pool[0][0]
____________________________________________________________________________________________________
block5_conv2 (Conv2D)            (None, 14, 14, 512)   2359808     block5_conv1[0][0]
____________________________________________________________________________________________________
block5_conv3 (Conv2D)            (None, 14, 14, 512)   2359808     block5_conv2[0][0]
____________________________________________________________________________________________________
block5_pool (MaxPooling2D)       (None, 7, 7, 512)     0           block5_conv3[0][0]
____________________________________________________________________________________________________
block5_fc6 (Conv2D)              (None, 7, 7, 4096)    102764544   block5_pool[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 7, 7, 4096)    0           block5_fc6[0][0]
____________________________________________________________________________________________________
block5_fc7 (Conv2D)              (None, 7, 7, 4096)    16781312    dropout_1[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 7, 7, 4096)    0           block5_fc7[0][0]
____________________________________________________________________________________________________
score_feat1 (Conv2D)             (None, 7, 7, 21)      86037       dropout_2[0][0]
```

```
--------------------------------------------------------------------------------------------------
score_feat2 (Conv2D)             (None, 14, 14, 21)    10773       block4_pool[0][0]
--------------------------------------------------------------------------------------------------
upscore_feat1 (BilinearUpSamplin (None, 14, 14, 21)    0           score_feat1[0][0]
--------------------------------------------------------------------------------------------------
scale_feat2 (Lambda)             (None, 14, 14, 21)    0           score_feat2[0][0]
--------------------------------------------------------------------------------------------------
add_1 (Add)                      (None, 14, 14, 21)    0           upscore_feat1[0][0]
                                                                   scale_feat2[0][0]
--------------------------------------------------------------------------------------------------
score_feat3 (Conv2D)             (None, 28, 28, 21)    5397        block3_pool[0][0]
--------------------------------------------------------------------------------------------------
upscore_feat2 (BilinearUpSamplin (None, 28, 28, 21)    0           add_1[0][0]
--------------------------------------------------------------------------------------------------
scale_feat3 (Lambda)             (None, 28, 28, 21)    0           score_feat3[0][0]
--------------------------------------------------------------------------------------------------
add_2 (Add)                      (None, 28, 28, 21)    0           upscore_feat2[0][0]
                                                                   scale_feat3[0][0]
--------------------------------------------------------------------------------------------------
upscore_feat3 (BilinearUpSamplin (None, 224, 224, 21)  0           add_2[0][0]
--------------------------------------------------------------------------------------------------
activation_1 (Activation)        (None, 224, 224, 21)  0           upscore_feat3[0][0]
==================================================================================================
Total params: 134,362,751
Trainable params: 119,648,063
Non-trainable params: 14,714,688
--------------------------------------------------------------------------------------------------
None
```

```
def show_result(n, dataload=train_loader):
    '''
    Show the result of prediction and compare it with the ground truth both in classes and output figur
    '''
    classes = ['background',
               'aeroplane', 'bicycle', 'bird', 'boat',
               'bottle', 'bus', 'car', 'cat',
               'chair', 'sheep', 'dinning table', 'dog',
               'horse', 'motorbike', 'person', 'potted plant',
               'cow', 'sofa', 'train', 'monitor']
    for fn in dataload.filenames[n:n+1]:
        raw = dataload.load_img(fn)
        x = datagen.standardize(raw)
        X = x[np.newaxis, ...]
        label = dataload.load_seg(fn)
        label = np.squeeze(label, axis=-1).astype('int')
        y_enc = np.eye(21)[label]
        y_true = y_enc[np.newaxis, ...]
        result = model.evaluate(X, y_true)

        y_pred = model.predict(X)
        loss = keras.losses.categorical_crossentropy(K.variable(y_true), K.variable(y_pred))
        #print(K.eval(loss))

        pred = np.argmax(y_pred, axis=-1)
        pred = pred[..., np.newaxis]
        pred = np.squeeze(pred, axis=0)
        print('True Labels: ', np.unique(label))
        print('Predicted Labels: ', np.unique(pred))
        print('True Classes: ', end=' ')
        for i in range(1, len(np.unique(label))):
            if i != 1:
                print(end=', ')
            print(classes[np.unique(label)[i]], end='')
        print()
        print('Predicted Classes: ', end=' ')
        for i in range(1, len(np.unique(pred))):
            if i != 1:
                print(end=', ')
            print(classes[np.unique(pred)[i]], end='')
        print()

        plt.figure(figsize=(20,20))
        plt.subplot(1,3,1)
        plt.axis('off')
        plt.imshow(raw)
        plt.title('Original Image')
        plt.subplot(1,3,2)
        plt.axis('off')
        plt.imshow(np.squeeze(pred))
        plt.title('Prediction')
        plt.subplot(1,3,3)
        plt.axis('off')
        plt.imshow(label)
        plt.title('Groud Truth')
        plt.savefig('Result.png')
        plt.show()
```

```
show_result(10)
```

```
1/1 [==============================] - 0s
WARNING:tensorflow:Variable /= will be deprecated. Use variable.assign_div if you want assignment to th


Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integ


True Labels:  [ 0 19]
Predicted Labels:  [ 0 19]
True Classes:  train
Predicted Classes:  train
```



4. BilinearUpSampling.py

Upsampling part, modified from author, JihongJu.

```
#!/usr/bin/python
#-*- coding:utf-8 -*-

import keras.backend as K
import tensorflow as tf
from keras.utils import conv_utils
from keras.engine.topology import Layer
from keras.engine import InputSpec

def resize_images(x, size):
    new_size = tf.convert_to_tensor(size, dtype=tf.int32)
    resized = tf.image.resize_images(x, new_size)
    return resized

class BilinearUpSampling2D(Layer):
    """Upsampling2D with bilinear interpolation."""

    def __init__(self, target_shape=None, data_format=None, **kwargs):
        if data_format is None:
            data_format = K.image_data_format()
        assert data_format in {
            'channels_last', 'channels_first'}
        self.data_format = data_format
        self.input_spec = [InputSpec(ndim=4)]
        self.target_shape = target_shape
        if self.data_format == 'channels_first':
            self.target_size = (target_shape[2], target_shape[3])
        elif self.data_format == 'channels_last':
            self.target_size = (target_shape[1], target_shape[2])
        super(BilinearUpSampling2D, self).__init__(**kwargs)

    def compute_output_shape(self, input_shape):
        if self.data_format == 'channels_last':
            return (input_shape[0], self.target_size[0],
                    self.target_size[1], input_shape[3])
        else:
            return (input_shape[0], input_shape[1],
                    self.target_size[0], self.target_size[1])

    def call(self, inputs):
        return resize_images(inputs, size=self.target_size)
        #return K.resize_images(inputs, self.target_size[0], self.target_size[1], data_format='channels_

    def get_config(self):
        config = {'target_shape': self.target_shape,
                'data_format': self.data_format}
        base_config = super(BilinearUpSampling2D, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))
```

5. printmodel.py

Used to plot the model in .png format.

```
#!/usr/bin/python
#-*- coding:utf-8 -*-

from keras.utils import plot_model
import model

def plot_fcn_vgg16():
    input_shape = (224, 224, 3)
    fcn_vgg16 = model.fcn_vgg16(input_shape=input_shape, classes=21)
    plot_model(fcn_vgg16, to_file='fcn_vgg16.png', show_shapes=True)

if __name__ == '__main__':
    plot_fcn_vgg16()
```

6. voc_generator.py

Data generator for VOC, copied from author, JihongJu.

```
"""Pascal VOC Segmenttion Generator."""
from __future__ import unicode_literals
import os
import numpy as np
from keras import backend as K
from keras.utils.np_utils import to_categorical
from keras.preprocessing.image import (
    ImageDataGenerator,
    Iterator,
    load_img,
    img_to_array,
    pil_image,
    array_to_img)


class PascalVocGenerator(ImageDataGenerator):
    """A real-time data augmentation generator for PASCAL VOC2011."""
```

```
def __init__(self,
             image_shape=(500, 500, 3),
             image_resample=True,
             pixelwise_center=False,
             pixel_mean=(0., 0., 0.),
             pixelwise_std_normalization=False,
             pixel_std=(1., 1., 1.),
             featurewise_center=False,
             samplewise_center=False,
             featurewise_std_normalization=False,
             samplewise_std_normalization=False,
             zca_whitening=False,
             rotation_range=0.,
             width_shift_range=0.,
             height_shift_range=0.,
             shear_range=0.,
             zoom_range=0.,
             channel_shift_range=0.,
             fill_mode='nearest',
             cval=0.,
             horizontal_flip=False,
             vertical_flip=False,
             rescale=None,
             preprocessing_function=None,
             data_format=None):
    """Init."""
    self.image_shape = tuple(image_shape)
    self.image_resample = image_resample
    self.pixelwise_center = pixelwise_center
    self.pixel_mean = np.array(pixel_mean)
    self.pixelwise_std_normalization = pixelwise_std_normalization
    self.pixel_std = np.array(pixel_std)
    super(PascalVocGenerator, self).__init__()

def standardize(self, x):
    """Standardize image."""
    if self.pixelwise_center:
        x -= self.pixel_mean
    if self.pixelwise_std_normalization:
        x /= self.pixel_std
    return super(PascalVocGenerator, self).standardize(x)

def flow_from_imageset(self, image_set_loader,
                       class_mode='categorical', classes=None,
                       batch_size=1, shuffle=True, seed=None):
    """PascalVocGenerator."""
    return IndexIterator(
        image_set_loader, self,
        class_mode=class_mode,
        classes=classes,
        batch_size=batch_size,
        shuffle=shuffle,
        seed=seed)
```

```
class IndexIterator(Iterator):
    """Iterator over index."""

    def __init__(self, image_set_loader, image_data_generator,
                 class_mode='categorical', classes=None,
                 batch_size=1, shuffle=False, seed=None):
        """Init."""
        self.image_set_loader = image_set_loader
        self.image_data_generator = image_data_generator

        self.filenames = image_set_loader.filenames
        self.image_shape = image_set_loader.image_shape

        self.classes = classes
        if class_mode == 'binary':
            label_shape = list(self.image_shape).pop(self.channel_axis - 1)
            self.label_shape = tuple(label_shape)
        elif class_mode == 'categorical':
            label_shape = list(self.image_shape)
            label_shape[self.image_data_generator.channel_axis - 1] \
                = self.classes
            self.label_shape = tuple(label_shape)

        super(IndexIterator, self).__init__(len(self.filenames), batch_size,
                                            shuffle, seed)

    def next(self):
        """Next batch."""
        with self.lock:
            index_array, current_index, current_batch_size = next(
                self.index_generator)
        batch_x = np.zeros(
            (current_batch_size,) + self.image_shape,
            dtype=K.floatx())
        batch_y = np.zeros(
            (current_batch_size,) + self.label_shape,
            dtype=np.int8)
        #batch_y = np.reshape(batch_y, (current_batch_size, -1, self.classes))

        for i, j in enumerate(index_array):
            fn = self.filenames[j]
            x = self.image_set_loader.load_img(fn)
            x = self.image_data_generator.standardize(x)
            batch_x[i] = x
            y = self.image_set_loader.load_seg(fn)
            y = to_categorical(y, self.classes).reshape(self.label_shape)
            #y = np.reshape(y, (-1, self.classes))
            batch_y[i] = y

        # save augmented images to disk for debugging
        #if self.image_set_loader.save_to_dir:
        #    for i in range(current_batch_size):
        #        x = batch_x[i]
        #        y = batch_y[i].argmax(
        #            self.image_data_generator.channel_axis - 1)
        #        if self.image_data_generator.data_format == 'channels_first':
        #            y = y[np.newaxis, ...]
        #        else:
        #            y = y[..., np.newaxis]
        #        self.image_set_loader.save(x, y, current_index + i)

        return batch_x, batch_y


class ImageSetLoader(object):
    """Helper class to load image data into numpy arrays."""

    def __init__(self, image_set, image_dir, label_dir, target_size=(500, 500),
                 image_format='jpg', color_mode='rgb', label_format='png',
                 data_format=None,
                 save_to_dir=None, save_prefix='', save_format='jpg'):
        """Init."""
        if data_format is None:
            data_format = K.image_data_format()
        self.data_format = data_format
        self.target_size = tuple(target_size)

        if not os.path.exists(image_set):
            raise IOError('Image set {} does not exist. Please provide a'
                          'valid file.'.format(image_set))
        self.filenames = np.loadtxt(image_set, dtype=bytes)
        try:
            self.filenames = [fn.decode('utf-8') for fn in self.filenames]
        except AttributeError as e:
            print(str(e), self.filenames[:5])
        if not os.path.exists(image_dir):
            raise IOError('Directory {} does not exist. Please provide a '
                          'valid directory.'.format(image_dir))
        self.image_dir = image_dir
        if label_dir and not os.path.exists(label_dir):
            raise IOError('Directory {} does not exist. Please provide a '
                          'valid directory.'.format(label_dir))
        self.label_dir = label_dir

        white_list_formats = {'png', 'jpg', 'jpeg', 'bmp'}
        self.image_format = image_format
        if self.image_format not in white_list_formats:
            raise ValueError('Invalid image format:', image_format,
                             '; expected "png", "jpg", "jpeg" or "bmp"')
        self.label_format = label_format
        if self.label_format not in white_list_formats:
            raise ValueError('Invalid image format:', label_format,
                             '; expected "png", "jpg", "jpeg" or "bmp"')

        if color_mode not in {'rgb', 'grayscale'}:
            raise ValueError('Invalid color mode:', color_mode,
                             '; expected "rgb" or "grayscale".')
        self.color_mode = color_mode
        if self.color_mode == 'rgb':
            if self.data_format == 'channels_last':
                self.image_shape = self.target_size + (3,)
            else:
                self.image_shape = (3,) + self.target_size
        else:
            if self.data_format == 'channels_last':
                self.image_shape = self.target_size + (1,)
            else:
                self.image_shape = (1,) + self.target_size

        self.grayscale = self.color_mode == 'grayscale'

        self.save_to_dir = save_to_dir
        self.save_prefix = save_prefix
        self.save_format = save_format

    def load_img(self, fn):
        """Image load method.

        # Arguments
            fn: filename of the image (without extension suffix)
        # Returns
            arr: numpy array of shape self.image_shape
        """
        img_path = os.path.join(self.image_dir,
                                '{}.{}'.format(fn,
                                               self.image_format))
        if not os.path.exists(img_path):
            raise IOError('Image {} does not exist.'.format(img_path))
        img = load_img(img_path, self.grayscale, self.target_size)
        x = img_to_array(img, data_format=self.data_format)

        return x

    def load_seg(self, fn):
        """Segmentation load method.

        # Arguments
            fn: filename of the image (without extension suffix)
        # Returns
            arr: numpy array of shape self.target_size
        """
        label_path = os.path.join(self.label_dir,
                                  '{}.{}'.format(fn, self.label_format))
        img = pil_image.open(label_path)
        if self.target_size:
            wh_tuple = (self.target_size[1], self.target_size[0])
        if img.size != wh_tuple:
            img = img.resize(wh_tuple)
        y = img_to_array(img, self.data_format)
        y[y == 255] = 0
        return y

    def save(self, x, y, index):
        """Image save method."""
        img = array_to_img(x, self.data_format, scale=True)
        mask = array_to_img(y, self.data_format, scale=True)
        img.paste(mask, (0, 0), mask)

        fname = 'img_{prefix}_{index}_{hash}.{format}'.format(
            prefix=self.save_prefix,
            index=index,
            hash=np.random.randint(1e4),
            format=self.save_format)
        img.save(os.path.join(self.save_to_dir, fname))
```

Acknowledgements

Some of the codes are implemented from JihongJu's work. These parameters and arguments save us a lot of time to pick good initial values. We would like to express our warmest gratitude to Prof. Yao Wang, for her instructive suggestions when we are confronted with problems of computing power. At the same time, we are also grateful to Mr. Ish Kumar Jain for providing us with valuable advice and instructions concerning Google Cloud Platform.