Assignmet_Week3_4_SK

March 9, 2021

1 Assignment Week 3-4

Two similar applications employing a scalable 3D ResNet architecture learn to predict the subject's age (regression) or the subject's sex (classification) from T1–weighted brain MR images from the IXI database. The main difference between this applications is the loss function: While we train the regression network to predict the age as a continuous variable with a L2-loss (the mean squared differences between the predicted age and the real age), we use a categorical cross-entropy loss to predict the class of the sex.

Downloading data for the example IXI_HH_sex_classification_resnet

[2]:
```
"""Download and extract the IXI Hammersmith Hospital 3T dataset

url: http://brain-development.org/ixi-dataset/
ref: IXI - Information eXtraction from Images (EPSRC GR/S21533/02)

"""
from __future__ import unicode_literals
from __future__ import print_function
from __future__ import division
from __future__ import absolute_import
from future.standard_library import install_aliases  # py 2/3 compatability
install_aliases()

from urllib.request import FancyURLopener

import os.path
import tarfile
import pandas as pd
import glob
import SimpleITK as sitk
import numpy as np

DOWNLOAD_IMAGES = True
EXTRACT_IMAGES = True
PROCESS_OTHER = True
RESAMPLE_IMAGES = True
CLEAN_UP = True
```

[ ]:
```
def resample_image(itk_image, out_spacing=(1.0, 1.0, 1.0), is_label=False):
    original_spacing = itk_image.GetSpacing()
    original_size = itk_image.GetSize()

    out_size = [int(np.round(original_size[0] * (original_spacing[0] /␣
 ↪out_spacing[0]))),
                int(np.round(original_size[1] * (original_spacing[1] /␣
 ↪out_spacing[1]))),
                int(np.round(original_size[2] * (original_spacing[2] /␣
 ↪out_spacing[2])))]

    resample = sitk.ResampleImageFilter()
    resample.SetOutputSpacing(out_spacing)
    resample.SetSize(out_size)
    resample.SetOutputDirection(itk_image.GetDirection())
    resample.SetOutputOrigin(itk_image.GetOrigin())
    resample.SetTransform(sitk.Transform())
    resample.SetDefaultPixelValue(itk_image.GetPixelIDValue())

    if is_label:
        resample.SetInterpolator(sitk.sitkNearestNeighbor)
    else:
        resample.SetInterpolator(sitk.sitkBSpline)

    return resample.Execute(itk_image)


def reslice_image(itk_image, itk_ref, is_label=False):
    resample = sitk.ResampleImageFilter()
    resample.SetReferenceImage(itk_ref)

    if is_label:
        resample.SetInterpolator(sitk.sitkNearestNeighbor)
    else:
        resample.SetInterpolator(sitk.sitkBSpline)

    return resample.Execute(itk_image)


urls = {}
urls['t1'] = 'http://biomedic.doc.ic.ac.uk/brain-development/downloads/IXI/
 ↪IXI-T1.tar'
urls['t2'] = 'http://biomedic.doc.ic.ac.uk/brain-development/downloads/IXI/
 ↪IXI-T2.tar'
urls['pd'] = 'http://biomedic.doc.ic.ac.uk/brain-development/downloads/IXI/
 ↪IXI-PD.tar'
```

```
urls['mra'] = 'http://biomedic.doc.ic.ac.uk/brain-development/downloads/IXI/
 ↪IXI-MRA.tar'
urls['demographic'] = 'http://biomedic.doc.ic.ac.uk/brain-development/downloads/
 ↪IXI/IXI.xls'

fnames = {}
fnames['t1'] = 't1.tar'
fnames['t2'] = 't2.tar'
fnames['pd'] = 'pd.tar'
fnames['mra'] = 'mra.tar'
fnames['demographic'] = 'demographic.xls'


if DOWNLOAD_IMAGES:
    # Download all IXI data
    for key, url in urls.items():

        if not os.path.isfile(fnames[key]):
            print('Downloading {} from {}'.format(fnames[key], url))
            curr_file = FancyURLopener()
            curr_file.retrieve(url, fnames[key])
        else:
            print('File {} already exists. Skipping download.'.format(
                fnames[key]))

if EXTRACT_IMAGES:
    # Extract the HH subset of IXI
    for key, fname in fnames.items():

        if (fname.endswith('.tar')):
            print('Extracting IXI HH data from {}.'.format(fnames[key]))
            output_dir = os.path.join('./orig/', key)

            if not os.path.exists(output_dir):
                os.makedirs(output_dir)

            t = tarfile.open(fname, 'r')
            for member in t.getmembers():
                if '-HH-' in member.name:
                    t.extract(member, output_dir)


if PROCESS_OTHER:
    # Process the demographic xls data and save to csv
    xls = pd.ExcelFile('demographic.xls')
    print(xls.sheet_names)
```

```
    df = xls.parse('Table')
    for index, row in df.iterrows():
        IXI_id = 'IXI{:03d}'.format(row['IXI_ID'])
        df.loc[index, 'IXI_ID'] = IXI_id

        t1_exists = len(glob.glob('./orig/t1/{}*.nii.gz'.format(IXI_id)))
        t2_exists = len(glob.glob('./orig/t2/{}*.nii.gz'.format(IXI_id)))
        pd_exists = len(glob.glob('./orig/pd/{}*.nii.gz'.format(IXI_id)))
        mra_exists = len(glob.glob('./orig/mra/{}*.nii.gz'.format(IXI_id)))

        # Check if each entry is complete and drop if not
        # if not t1_exists and not t2_exists and not pd_exists and not mra
        # exists:
        if not (t1_exists and t2_exists and pd_exists and mra_exists):
            df.drop(index, inplace=True)

    # Write to csv file
    df.to_csv('demographic_HH.csv', index=False)

if RESAMPLE_IMAGES:
    # Resample the IXI HH T2 images to 1mm isotropic and reslice all
    # others to it
    df = pd.read_csv('demographic_HH.csv', dtype=object, keep_default_na=False,
                     na_values=[]).values

    for i in df:
        IXI_id = i[0]
        print('Resampling {}'.format(IXI_id))

        t1_fn = glob.glob('./orig/t1/{}*.nii.gz'.format(IXI_id))[0]
        t2_fn = glob.glob('./orig/t2/{}*.nii.gz'.format(IXI_id))[0]
        pd_fn = glob.glob('./orig/pd/{}*.nii.gz'.format(IXI_id))[0]
        mra_fn = glob.glob('./orig/mra/{}*.nii.gz'.format(IXI_id))[0]

        t1 = sitk.ReadImage(t1_fn)
        t2 = sitk.ReadImage(t2_fn)
        pd = sitk.ReadImage(pd_fn)
        mra = sitk.ReadImage(mra_fn)

        # Resample to 1mm isotropic resolution
        t2_1mm = resample_image(t2)
        t1_1mm = reslice_image(t1, t2_1mm)
        pd_1mm = reslice_image(pd, t2_1mm)
        mra_1mm = reslice_image(mra, t2_1mm)

        output_dir = os.path.join('./1mm/', IXI_id)
        if not os.path.exists(output_dir):
```

```
            os.makedirs(output_dir)

        print('T1: {} {}'.format(t1_1mm.GetSize(), t1_1mm.GetSpacing()))
        print('T2: {} {}'.format(t2_1mm.GetSize(), t2_1mm.GetSpacing()))
        print('PD: {} {}'.format(pd_1mm.GetSize(), pd_1mm.GetSpacing()))
        print('MRA: {} {}'.format(mra_1mm.GetSize(), mra_1mm.GetSpacing()))

        sitk.WriteImage(t1_1mm, os.path.join(output_dir, 'T1_1mm.nii.gz'))
        sitk.WriteImage(t2_1mm, os.path.join(output_dir, 'T2_1mm.nii.gz'))
        sitk.WriteImage(pd_1mm, os.path.join(output_dir, 'PD_1mm.nii.gz'))
        sitk.WriteImage(mra_1mm, os.path.join(output_dir, 'MRA_1mm.nii.gz'))

        # Resample to 2mm isotropic resolution
        t2_2mm = resample_image(t2, out_spacing=[2.0, 2.0, 2.0])
        t1_2mm = reslice_image(t1, t2_2mm)
        pd_2mm = reslice_image(pd, t2_2mm)
        mra_2mm = reslice_image(mra, t2_2mm)

        output_dir = os.path.join('./2mm/', IXI_id)
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)

        print('T1: {} {}'.format(t2_2mm.GetSize(), t1_2mm.GetSpacing()))
        print('T2: {} {}'.format(t2_2mm.GetSize(), t2_2mm.GetSpacing()))
        print('PD: {} {}'.format(pd_2mm.GetSize(), pd_2mm.GetSpacing()))
        print('MRA: {} {}'.format(mra_2mm.GetSize(), mra_2mm.GetSpacing()))

        sitk.WriteImage(t1_2mm, os.path.join(output_dir, 'T1_2mm.nii.gz'))
        sitk.WriteImage(t2_2mm, os.path.join(output_dir, 'T2_2mm.nii.gz'))
        sitk.WriteImage(pd_2mm, os.path.join(output_dir, 'PD_2mm.nii.gz'))
        sitk.WriteImage(mra_2mm, os.path.join(output_dir, 'MRA_2mm.nii.gz'))


if CLEAN_UP:
    # Remove the .tar files
    for key, fname in fnames.items():
        if (fname.endswith('.tar')):
            os.remove(fname)

    # Remove all data in original resolution
    os.system('rm -rf orig')
```

1.1 training the model

[4]:
```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
from __future__ import print_function
from __future__ import division
from __future__ import absolute_import

import argparse
import os
import pandas as pd
import tensorflow as tf
import numpy as np
import dltk

from dltk.networks.regression_classification.resnet import resnet_3d
from dltk.io.abstract_reader import Reader


EVAL_EVERY_N_STEPS = 100
EVAL_STEPS = 5

NUM_CLASSES = 2
NUM_CHANNELS = 1

BATCH_SIZE = 8
SHUFFLE_CACHE_SIZE = 32

MAX_STEPS = 50000
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-4-f55581a89797> in <module>
     12 import dltk
     13
---> 14 from dltk.networks.regression_classification.resnet import resnet_3d
     15 from dltk.io.abstract_reader import Reader
     16

~\anaconda3\lib\site-packages\dltk\networks\regression_classification\resnet.py
 ↪in <module>
      6 import tensorflow as tf
      7
----> 8 from dltk.core.residual_unit import vanilla_residual_unit_3d
      9
     10
```

```
~\anaconda3\lib\site-packages\dltk\core\residual_unit.py in <module>
     12                                 kernel_size=(3, 3, 3),
     13                                 strides=(1, 1, 1),
---> 14                                 mode=tf.estimator.ModeKeys.EVAL,
     15                                 use_bias=False,
     16                                 activation=tf.nn.relu6,

~\anaconda3\lib\site-packages\tensorflow\python\util\lazy_loader.py in␣
↪__getattr__(self, item)
     60
     61   def __getattr__(self, item):
---> 62     module = self._load()
     63     return getattr(module, item)
     64

~\anaconda3\lib\site-packages\tensorflow\python\util\lazy_loader.py in␣
↪_load(self)
     43     """Load the module and insert it into the parent's globals."""
     44     # Import the target module and insert it into the parent's namespac
---> 45     module = importlib.import_module(self.__name__)
     46     self._parent_module_globals[self._local_name] = module
     47

~\anaconda3\lib\importlib\__init__.py in import_module(name, package)
    125                 break
    126             level += 1
--> 127     return _bootstrap._gcd_import(name[level:], package, level)
    128
    129

~\anaconda3\lib\site-packages\tensorflow_estimator\__init__.py in <module>
      8 import sys as _sys
      9
---> 10 from tensorflow_estimator._api.v1 import estimator
     11
     12 del _print_function

~\anaconda3\lib\site-packages\tensorflow_estimator\_api\v1\estimator\__init__.p␣
↪in <module>
      8 import sys as _sys
      9
---> 10 from tensorflow_estimator._api.v1.estimator import experimental
     11 from tensorflow_estimator._api.v1.estimator import export
     12 from tensorflow_estimator._api.v1.estimator import inputs

~\anaconda3\lib\site-packages\tensorflow_estimator\_api\v1\estimator\experiment␣
↪l\__init__.py in <module>
      8 import sys as _sys
```

```
      9
---> 10 from tensorflow_estimator.python.estimator.canned.dnn import␣
 ↪dnn_logit_fn_builder
     11 from tensorflow_estimator.python.estimator.canned.kmeans import␣
 ↪KMeansClustering as KMeans
     12 from tensorflow_estimator.python.estimator.canned.linear import␣
 ↪LinearSDCA

~\anaconda3\lib\site-packages\tensorflow_estimator\python\estimator\canned\dnn.
 ↪py in <module>
     29 from tensorflow.python.keras.utils import losses_utils
     30 from tensorflow.python.util.tf_export import estimator_export
---> 31 from tensorflow_estimator.python.estimator import estimator
     32 from tensorflow_estimator.python.estimator.canned import head as head_l
 ↪b
     33 from tensorflow_estimator.python.estimator.canned import optimizers

~\anaconda3\lib\site-packages\tensorflow_estimator\python\estimator\estimator.p
 ↪in <module>
     50 from tensorflow.python.util.tf_export import estimator_export
     51 from tensorflow_estimator.python.estimator import model_fn as␣
 ↪model_fn_lib
---> 52 from tensorflow_estimator.python.estimator import run_config
     53 from tensorflow_estimator.python.estimator import util as estimator_uti
 ↪l
     54 from tensorflow_estimator.python.estimator.export import export_lib

~\anaconda3\lib\site-packages\tensorflow_estimator\python\estimator\run_config.
 ↪py in <module>
     28 from tensorflow.core.protobuf import rewriter_config_pb2
     29 from tensorflow.python.distribute import estimator_training as␣
 ↪distribute_coordinator_training
---> 30 from tensorflow.python.distribute import parameter_server_strategy_v2
     31 from tensorflow.python.util import compat_internal
     32 from tensorflow.python.util import function_utils

ImportError: cannot import name 'parameter_server_strategy_v2' from 'tensorflow
 ↪python.distribute' (C:
 ↪\Users\Susheel\anaconda3\lib\site-packages\tensorflow\python\distribute\__ini
 ↪__.py)
```

[ ]:
```
def model_fn(features, labels, mode, params):
    """Model function to construct a tf.estimator.EstimatorSpec. It creates a
        network given input features (e.g. from a dltk.io.abstract_reader) and
        training targets (labels). Further, loss, optimiser, evaluation ops and
        custom tensorboard summary ops can be added. For additional information,
         please refer to https://www.tensorflow.org/api_docs/python/tf/
 ↪estimator/Estimator#model_fn.
```

```
    Args:
        features (tf.Tensor): Tensor of input features to train from. Required
            rank and dimensions are determined by the subsequent ops
            (i.e. the network).
        labels (tf.Tensor): Tensor of training targets or labels. Required rank
            and dimensions are determined by the network output.
        mode (str): One of the tf.estimator.ModeKeys: TRAIN, EVAL or PREDICT
        params (dict, optional): A dictionary to parameterise the model_fn
            (e.g. learning_rate)

    Returns:
        tf.estimator.EstimatorSpec: A custom EstimatorSpec for this experiment
    """

    # 1. create a model and its outputs
    net_output_ops = resnet_3d(
        features['x'],
        num_res_units=2,
        num_classes=NUM_CLASSES,
        filters=(16, 32, 64, 128, 256),
        strides=((1, 1, 1), (2, 2, 2), (2, 2, 2), (2, 2, 2), (2, 2, 2)),
        mode=mode,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(1e-3))

    # 1.1 Generate predictions only (for `ModeKeys.PREDICT`)
    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(
            mode=mode,
            predictions=net_output_ops,
            export_outputs={'out': tf.estimator.export.
↪PredictOutput(net_output_ops)})

    # 2. set up a loss function
    one_hot_labels = tf.reshape(tf.one_hot(labels['y'], depth=NUM_CLASSES),␣
↪[-1, NUM_CLASSES])

    loss = tf.losses.softmax_cross_entropy(
        onehot_labels=one_hot_labels,
        logits=net_output_ops['logits'])

    # 3. define a training op and ops for updating moving averages (i.e. for
    # batch normalisation)
    global_step = tf.train.get_global_step()
    optimiser = tf.train.AdamOptimizer(
        learning_rate=params["learning_rate"],
        epsilon=1e-5)
```

```
    update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
    with tf.control_dependencies(update_ops):
        train_op = optimiser.minimize(loss, global_step=global_step)

    # 4.1 (optional) create custom image summaries for tensorboard
    my_image_summaries = {}
    my_image_summaries['feat_t1'] = features['x'][0, 32, :, :, 0]

    expected_output_size = [1, 96, 96, 1]  # [B, W, H, C]
    [tf.summary.image(name, tf.reshape(image, expected_output_size))
     for name, image in my_image_summaries.items()]

    # 4.2 (optional) track the rmse (scaled back by 100, see reader.py)
    acc = tf.metrics.accuracy
    prec = tf.metrics.precision
    eval_metric_ops = {"accuracy": acc(labels['y'], net_output_ops['y_']),
                       "precision": prec(labels['y'], net_output_ops['y_'])}

    # 5. Return EstimatorSpec object
    return tf.estimator.EstimatorSpec(mode=mode,
                                      predictions=net_output_ops,
                                      loss=loss,
                                      train_op=train_op,
                                      eval_metric_ops=eval_metric_ops)


def train(args):
    np.random.seed(42)
    tf.set_random_seed(42)

    print('Setting up...')

    # Parse csv files for file names
    all_filenames = pd.read_csv(
        args.data_csv,
        dtype=object,
        keep_default_na=False,
        na_values=[]).as_matrix()

    train_filenames = all_filenames[:150]
    val_filenames = all_filenames[150:]

    # Set up a data reader to handle the file i/o.
    reader_params = {'n_examples': 2,
                     'example_size': [64, 96, 96],
                     'extract_examples': True}
```

```
    reader_example_shapes = {'features': {'x': reader_params['example_size'] +␣
↪[NUM_CHANNELS]},
                             'labels': {'y': [1]}}
    reader = Reader(read_fn,
                    {'features': {'x': tf.float32},
                     'labels': {'y': tf.int32}})

    # Get input functions and queue initialisation hooks for training and
    # validation data
    train_input_fn, train_qinit_hook = reader.get_inputs(
        file_references=train_filenames,
        mode=tf.estimator.ModeKeys.TRAIN,
        example_shapes=reader_example_shapes,
        batch_size=BATCH_SIZE,
        shuffle_cache_size=SHUFFLE_CACHE_SIZE,
        params=reader_params)

    val_input_fn, val_qinit_hook = reader.get_inputs(
        file_references=val_filenames,
        mode=tf.estimator.ModeKeys.EVAL,
        example_shapes=reader_example_shapes,
        batch_size=BATCH_SIZE,
        shuffle_cache_size=SHUFFLE_CACHE_SIZE,
        params=reader_params)

    # Instantiate the neural network estimator
    nn = tf.estimator.Estimator(
        model_fn=model_fn,
        model_dir=args.model_path,
        params={"learning_rate": 0.001},
        config=tf.estimator.RunConfig())

    # Hooks for validation summaries
    val_summary_hook = tf.contrib.training.SummaryAtEndHook(
        os.path.join(args.model_path, 'eval'))
    step_cnt_hook = tf.train.StepCounterHook(every_n_steps=EVAL_EVERY_N_STEPS,
                                             output_dir=args.model_path)

    print('Starting training...')
    try:
        for _ in range(MAX_STEPS // EVAL_EVERY_N_STEPS):
            nn.train(
                input_fn=train_input_fn,
                hooks=[train_qinit_hook, step_cnt_hook],
                steps=EVAL_EVERY_N_STEPS)

            if args.run_validation:
```

```
                results_val = nn.evaluate(
                    input_fn=val_input_fn,
                    hooks=[val_qinit_hook, val_summary_hook],
                    steps=EVAL_STEPS)
                print('Step = {}; val loss = {:.5f};'.format(
                    results_val['global_step'],
                    results_val['loss']))

    except KeyboardInterrupt:
        pass

    # When exporting we set the expected input shape to be arbitrary.
    export_dir = nn.export_savedmodel(
        export_dir_base=args.model_path,
        serving_input_receiver_fn=reader.serving_input_receiver_fn(
            {'features': {'x': [None, None, None, NUM_CHANNELS]},
             'labels': {'y': [1]}}))
    print('Model saved to {}.'.format(export_dir))


if __name__ == '__main__':
    # Set up argument parser
    parser = argparse.ArgumentParser(description='Example: IXI HH resnet sex␣
 ↪classification training')
    parser.add_argument('--run_validation', default=True)
    parser.add_argument('--restart', default=False, action='store_true')
    parser.add_argument('--verbose', default=False, action='store_true')
    parser.add_argument('--cuda_devices', '-c', default='0')

    parser.add_argument('--model_path', '-p', default='/tmp/
 ↪IXI_sex_classification/')
    parser.add_argument('--data_csv', default='../../../data/IXI_HH/
 ↪demographic_HH.csv')

    args = parser.parse_args()

    # Set verbosity
    if args.verbose:
        os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
        tf.logging.set_verbosity(tf.logging.INFO)
    else:
        os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
        tf.logging.set_verbosity(tf.logging.ERROR)

    # GPU allocation options
    os.environ["CUDA_VISIBLE_DEVICES"] = args.cuda_devices
```

```
    # Handle restarting and resuming training
    if args.restart:
        print('Restarting training from scratch.')
        os.system('rm -rf {}'.format(args.model_path))

    if not os.path.isdir(args.model_path):
        os.system('mkdir -p {}'.format(args.model_path))
    else:
        print('Resuming training on model_path {}'.format(args.model_path))

    # Call training
    train(args)
```