

دانشگاه آزاد اسلامی واحد تهران جنوب

گزارش سمینار درس بینایی ماشین

کارشناسی ارشد رشته مهندسی پزشکی گرایش بیوالکتریک

**عنوان مقاله**

**Deep learning based detection of enlarged perivascular spaces on brain MRI**

نگارنده:

**فاطمه خلیلی**

بهار1403

چکیده

یادگیری عمیق در بسیاری از کاربردهای تصویربرداری عصبی موثر نشان داده شده است. با این حال، در بسیاری از سناریوها، تعداد توالی‌های تصویربرداری که اطلاعات مربوط به ضایعات بیماری عروق کوچک را ثبت می‌کنند برای پشتیبانی از تکنیک‌های مبتنی بر داده کافی نیست. علاوه بر این، مطالعات مبتنی بر کوهورت ممکن است همیشه مطلوب یا ضروری نباشد. توالی تصویربرداری برای تشخیص دقیق ضایعه بنابراین، تعیین اینکه کدام توالی تصویربرداری برای تشخیص دقیق ضروری است، ضروری است. این مطالعه یک چارچوب یادگیری عمیق را برای تشخیص فضاهای اطراف عروقی بزرگ (ePVS) معرفی می‌کند و هدف آن یافتن ترکیب بهینه توالی‌های MRI برای کمی‌سازی مبتنی بر یادگیری عمیق است.

ما یک U-Net سبک وزن موثر را که برای تشخیص ePVS تطبیق داده شده است، پیاده سازی کردیم و ترکیب های مختلف اطلاعات از توالی های MRI SWI، FLAIR، T1-weighted (T1W),T2 –weighted(T2W) , را به طور جامع بررسی کردیم. نتایج تجربی نشان داد که MRI ۲TW2 برای تشخیص دقیق ePVS مهم‌ترین است ترکیب SWI، FLAIR و T۱w MRI در شبکه عصبی عمیق بهبودهای جزئی در دقت داشت و منجر به بالاترین حساسیت و دقت (حساسیت = ۰.۸۲، دقت = ۰.۸۳) شد. روش پیشنهادی به دقت قابل مقایسه با حداقل هزینه زمانی در مقایسه با خواندن دستی دست یافت. خط لوله خودکار پیشنهادی خوانش قوی و با زمان کارآمد ePVS از اسکن‌های MR را امکان‌پذیر می‌کند و اهمیت T۲w MRI برای تشخیص ePVS و مزایای بالقوه استفاده از تصاویر چندوجهی را نشان می‌دهد. علاوه بر این، این مدل نقشه‌های کل مغزی از ePVS را ارائه می‌دهد که درک بهتری از همبستگی‌های بالینی آن‌ها را در مقایسه با روش‌های رتبه‌بندی بالینی تنها در چند ناحیه مغزی امکان‌پذیر می‌سازد.

کلید واژه MRI، یادگیری عمیق، فضای اطراف عروقی بزرگ شده است.

**فهرست مطالب**

فصل1:

1-1:دیتاست ها و داده های مقاله:

دیتاست های این مقاله به طورعمومی در دستری بود به حجم 58گیگ.

متاسفانه امکان دانلود دیتاست را نداشتم پس از داده های یکسان و رفرنس های دیگر برای پیاده سازی استفاده کردم به همین دلیل من از چندین دیتا ست متفاوت برای بررسی و پیاده سازی این مقاله استفاده کردم.

1-2:رفرنس ها یا مراجع

با توجه به رفرنس های مقاله دیتاهایی که استفاده شده ترکیبی از چند مقاله است.

بنابراین دیتاست مورد استفاده به شکل یکسان مورد استفاده نیست.

1-3:روش جست و جوی دیتاست ها در گیت هاب:

1\_ابتدا کلمات متناسب با مقاله و عنوان مقاله در گیت هاب سرچ شد.

2\_در مرحله دوم کلمات کلیدی مقاله در گیت هاب سرچ شد.

1)journaltoc\_analysis.py

#!/usr/bin/env python3

# -\*- coding: UTF-8 -\*-

from csv import DictReader, DictWriter

from os import path

import re

from time import sleep

import socket

import sys

from urllib.error import HTTPError, URLError

from urllib.request import urlopen, Request

JTOC\_METADATA\_RES = {

"jtoc\_id": re.compile("journaltocID\:\s\*(?P<jtoc\_id>\d+)"),

"jtoc\_title": re.compile("\<dc\:title\>(?P<jtoc\_title>.\*?)\<\/dc\:title\>"),

"jtoc\_publisher": re.compile("Publisher\:\s\*(?P<jtoc\_publisher>.\*?)\<br\>")

}

JOURNAL\_TYPE\_RES = {

"HYBRID": re.compile('title="Hybrid Journal. It can contain Open Access articles"'),

"OA": re.compile('title="This is an Open Access Journal"'),

"SUBSCRIPTION": re.compile('title="Subscription journal, but a few articles may be freely available"'),

"PARTIALLY\_FREE": re.compile('title="Partially Free or Hybrid journal."'),

"FREE": re.compile('title="Free journal"'),

"ERROR": re.compile("Article not found or there are no recent issues available for this journal")

}

DATA\_FILES = ["../../../data/apc\_de.csv", "../../../data/offsetting/offsetting.csv"]

JOURNALTOC\_RESULTS\_FILE = "journaltoc\_comparison.csv"

RESULTS\_FILE\_FIELDNAMES = ["journal\_full\_title", "publisher", "issns", "is\_hybrid", "in\_jtoc", "jtoc\_publisher", "jtoc\_title", "jtoc\_type"]

ISSN\_TYPES = ["issn", "issn\_print", "issn\_electronic", "issn\_l"]

BATCH\_SIZE = 1000

JTOCS\_USERNAME = "user@example.com"

def main():

analysed\_journals = {}

if path.isfile(JOURNALTOC\_RESULTS\_FILE):

with open(JOURNALTOC\_RESULTS\_FILE) as results:

reader = DictReader(results)

for line in reader:

title = line["journal\_full\_title"]

if title not in analysed\_journals:

analysed\_journals[title] = line

remaining\_journals = {}

for data\_file in DATA\_FILES:

with open(data\_file) as f:

reader = DictReader(f)

for line in reader:

title = line["journal\_full\_title"]

if title in analysed\_journals:

continue

if title not in remaining\_journals:

remaining\_journals[title] = {"journal\_full\_title": line["journal\_full\_title"], "publisher": line["publisher"], "is\_hybrid": line["is\_hybrid"], "issns": []}

for issn\_type in ISSN\_TYPES:

issn = line[issn\_type]

if issn not in remaining\_journals[title]["issns"] and oat.is\_wellformed\_ISSN(issn):

remaining\_journals[title]["issns"].append(issn)

is\_hybrid = line["is\_hybrid"]

if is\_hybrid in ["TRUE", "FALSE"] and is\_hybrid != remaining\_journals[title]["is\_hybrid"]:

remaining\_journals[title]["is\_hybrid"] = "FLIPPED"

msg = "{} unique journals found in OpenAPC and offsetting files, {} already analysed, {} remaining."

oat.print\_g(msg.format(len(remaining\_journals) + len(analysed\_journals), len(analysed\_journals), len(remaining\_journals)))

count = 0

for title, fields in remaining\_journals.items():

count += 1

entry = {field: None for field in RESULTS\_FILE\_FIELDNAMES}

entry["journal\_full\_title"] = title

for key in ["publisher", "is\_hybrid"]:

entry[key] = fields[key]

entry["issns"] = "|".join(fields["issns"])

msg = 'Analysing journal "{}" ({}), OpenAPC hybrid status is {}...'

msg = msg.format(entry["journal\_full\_title"], entry["issns"], entry["is\_hybrid"])

oat.print\_b(msg)

for issn in fields["issns"]:

oat.print\_y("Looking up ISSN " + issn + "...")

jtoc\_metadata = get\_jtoc\_metadata(issn)

if jtoc\_metadata["jtoc\_id"] is not None:

entry["in\_jtoc"] = "TRUE"

for key in ["jtoc\_publisher", "jtoc\_title"]:

entry[key] = jtoc\_metadata[key]

journal\_type = get\_jtoc\_journal\_type(jtoc\_metadata["jtoc\_id"])

entry["jtoc\_type"] = journal\_type

msg = 'Journal found ("{}"), JournalTOCs type is {}'

oat.print\_g(msg.format(entry["jtoc\_title"], entry["jtoc\_type"]))

break

else:

oat.print\_r("None of the associated ISSNS found in JTOCs!")

entry["in\_jtoc"] = "FALSE"

analysed\_journals[title] = entry

if count < BATCH\_SIZE:

sleep(2)

else:

break

with open(JOURNALTOC\_RESULTS\_FILE, "w") as res\_file:

writer = DictWriter(res\_file, fieldnames=RESULTS\_FILE\_FIELDNAMES)

writer.writeheader()

for \_, entry in analysed\_journals.items():

writer.writerow(entry)

def get\_jtoc\_metadata(issn, retries=0):

api\_url = "http://www.journaltocs.ac.uk/api/journals/"

user\_param = "?user=" + JTOCS\_USERNAME

url = api\_url + issn + user\_param

req = Request(url)

try:

response = urlopen(req, timeout=5)

content = response.read().decode("utf8")

results = {}

for key, regex in JTOC\_METADATA\_RES.items():

match = regex.search(content)

if match:

results[key] = match.groupdict()[key]

else:

results[key] = None

return results

except socket.timeout:

if retries <= 3:

print ("Socket timeout, retrying (" + str(retries) + ")")

return get\_jtoc\_metadata(issn, retries + 1)

return None

except ConnectionResetError:

if retries <= 3:

print ("ConnectionResetError, retrying (" + str(retries) + ")")

return get\_jtoc\_metadata(issn, retries + 1)

return None

except HTTPError as httpe:

print("HTTPError: {} - {}".format(httpe.code, httpe.reason))

return None

except URLError as urle:

print("URLError: {}".format(urle.reason))

return None

def get\_jtoc\_journal\_type(jtoc\_id, retries=0):

api\_url = "http://www.journaltocs.ac.uk/index.php?journalID="

url = api\_url + jtoc\_id

req = Request(url)

try:

response = urlopen(req, timeout=5)

content = response.read().decode("utf8")

for journal\_type, re in JOURNAL\_TYPE\_RES.items():

match = re.search(content)

if match:

return journal\_type

raise Exception("Error: No RE matched for journal at " + url)

return None

except socket.timeout:

if retries <= 3:

print ("Socket timeout, retrying (" + str(retries) + ")")

return get\_jtoc\_journal\_type(jtoc\_id, retries + 1)

return None

except ConnectionResetError:

if retries <= 3:

print ("ConnectionResetError, retrying (" + str(retries) + ")")

return get\_jtoc\_journal\_type(jtoc\_id, retries + 1)

return None

except HTTPError as httpe:

print("HTTPError: {} - {}".format(httpe.code, httpe.reason))

return None

except URLError as urle:

print("URLError: {}".format(urle.reason))

return None

if \_\_name\_\_ == '\_\_main\_\_':

sys.path.append(path.dirname(path.dirname(path.dirname(path.abspath(\_\_file\_\_)))))

import openapc\_toolkit as oat

main()

2) gates\_foundation\_preprocessing.py

#!/usr/bin/env python3

# -\*- coding: UTF-8 -\*-

import argparse

import csv

import datetime

import json

from os import path

import sys

from urllib.error import HTTPError, URLError

ARG\_HELP\_STRINGS = {

"source\_file": "The csv input file",

"doi\_file": "The additional report containing a PublicationID -> DOI mapping",

"exchange\_rates\_cache\_file": "An optional cache file for ECB exchange rates",

"no\_decorations": "Do not use ANSI coded colors in console output"

}

EXCHANGE\_RATES\_CACHE = {}

EXCHANGE\_RATES\_CACHE\_FILE = None

DELETE\_REASONS = {}

OUTPUT\_FIELDS = ["Publisher", "Journal title", "Currency", "APC", "Date Payment Completed",

"institution", "period", "euro", "doi", "is\_hybrid"]

NO\_DECORATIONS = False

def delete\_line(line\_dict, reason):

\_print("r", " - " + reason + ", line deleted")

if reason not in DELETE\_REASONS:

DELETE\_REASONS[reason] = 1

else:

DELETE\_REASONS[reason] += 1

for key in line\_dict:

line\_dict[key] = ""

def line\_as\_list(line\_dict):

return [line\_dict[field] for field in FIELDNAMES[FORMAT]]

def is\_zero\_value(string):

number = float(string)

return number == 0.0

def is\_valid\_date(date\_match\_obj):

gd = date\_match\_obj.groupdict()

if gd["year"] is None or gd["month"] is None or gd["day"] is None:

return False

try:

date = datetime.datetime(int(gd["year"]), int(gd["month"]), int(gd["day"]))

if date > datetime.datetime.now():

return False

return True

except ValueError:

return False

def shutdown():

\_print("r", "Updating exchange rates cache...")

with open(EXCHANGE\_RATES\_CACHE\_FILE, "w") as f:

f.write(json.dumps(EXCHANGE\_RATES\_CACHE, sort\_keys=True, indent=4, separators=(',', ': ')))

f.flush()

\_print("r", "Done.")

sys.exit()

def \_print(color, s):

if color in ["r", "y", "g", "b"] and not NO\_DECORATIONS:

getattr(oat, "print\_" + color)(s)

else:

print(s)

def get\_exchange\_rate(currency, date):

if currency not in EXCHANGE\_RATES\_CACHE:

EXCHANGE\_RATES\_CACHE[currency] = {}

if not len(EXCHANGE\_RATES\_CACHE[currency]):

try:

rates = oat.get\_euro\_exchange\_rates(currency)

EXCHANGE\_RATES\_CACHE[currency] = rates

except HTTPError as httpe:

\_print("r", "HTTPError while querying the ECB data warehouse: " + httpe.reason)

shutdown()

except URLError as urle:

\_print("r", "URLError while querying the ECB data warehouse: " + urle.reason)

shutdown()

except ValueError as ve:

\_print("r", "ValueError while querying the ECB data warehouse: " + ve.reason)

shutdown()

# The ECB does not report exchange rate for all dates due to weekends/holidays. We have

# consider some days in advance to find the next possible data in some cases.

for i in range(6):

future\_day = date + datetime.timedelta(days=i)

search\_day = future\_day.strftime("%Y-%m-%d")

if search\_day in EXCHANGE\_RATES\_CACHE[currency]:

\_print("y", " [Exchange rates: Cached value used]")

if i > 0:

msg = " [Exchange rates: No rate found for date {}, used value for {} instead]"

\_print("y", msg.format(date, search\_day))

return EXCHANGE\_RATES\_CACHE[currency][search\_day]

\_print("r", "Error during Exchange rates lookup: No rate for " + date.strftime("%Y-%m-%d") + " or any following day!")

shutdown()

def calculate\_euro\_value(line):

date\_string = line["Date Payment Completed"]

date = datetime.datetime.strptime(date\_string, "%Y-%m-%dT%H:%M:%S.%f%z")

apc\_value = line["APC"]

if not is\_zero\_value(apc\_value):

currency = line["Currency"].strip()

if currency == "EUR":

line["euro"] = apc\_value

msg = " - Created euro field ('{}') by using the value in 'APC' directly since the currency is EUR"

\_print("g", msg.format(apc\_value))

else:

rate = get\_exchange\_rate(currency, date)

euro\_value = round(float(apc\_value) / float(rate), 2)

line["euro"] = str(euro\_value)

msg = " - Created euro field ('{}') by dividing the value in 'APC' ({}) by {} (EUR -> {} conversion rate on {}) [ECB]"

msg = msg.format(euro\_value, apc\_value, rate, currency, date.strftime("%Y-%m-%d"))

\_print("g", msg)

else:

delete\_line(line, "APC value is zero")

def main():

global EXCHANGE\_RATES\_CACHE, EXCHANGE\_RATES\_CACHE\_FILE, NO\_DECORATIONS

parser = argparse.ArgumentParser()

parser.add\_argument("source\_file", help=ARG\_HELP\_STRINGS["source\_file"])

parser.add\_argument("doi\_file", help=ARG\_HELP\_STRINGS["doi\_file"])

parser.add\_argument("-c", "--exchange\_rates\_cache\_file", help=ARG\_HELP\_STRINGS["exchange\_rates\_cache\_file"], default="\_exchange\_rates\_cache.json")

parser.add\_argument("-n", "--no-decorations", help=ARG\_HELP\_STRINGS["no\_decorations"], action="store\_true")

args = parser.parse\_args()

NO\_DECORATIONS = args.no\_decorations

EXCHANGE\_RATES\_CACHE\_FILE = args.exchange\_rates\_cache\_file

if path.isfile(args.exchange\_rates\_cache\_file):

with open(EXCHANGE\_RATES\_CACHE\_FILE, "r") as f:

try:

EXCHANGE\_RATES\_CACHE = json.loads(f.read())

except ValueError:

\_print("r", "Could not decode a cache structure from " + EXCHANGE\_RATES\_CACHE\_FILE + ", starting with an empty cache.")

doi\_mappings = {}

with open(args.doi\_file, "r") as doi\_file:

reader = csv.DictReader(doi\_file)

for line in reader:

pub\_id = line["PublicationID"]

doi\_mappings[pub\_id] = line["DOI"]

f = open(args.source\_file, "r", encoding="utf-8")

reader = csv.DictReader(f)

modified\_content = [list(OUTPUT\_FIELDS)]

line\_num = 1

for line in reader:

line\_num += 1

line["institution"] = "Bill & Melinda Gates Foundation"

line["period"] = ""

line["is\_hybrid"] = ""

line["euro"] = ""

line["doi"] = doi\_mappings.get(line["PublicationID"], "")

\_print("b", "--- Analysing line " + str(line\_num) + " ---")

# DOI check

if len(line["doi"].strip()) == 0:

delete\_line(line, "Empty DOI")

modified\_content.append([line[field] for field in OUTPUT\_FIELDS])

continue

# period field generation

date\_string = line["Date Payment Completed"]

date = datetime.datetime.strptime(date\_string, "%Y-%m-%dT%H:%M:%S.%f%z")

line["period"] = str(date.year)

# euro field generation

calculate\_euro\_value(line)

modified\_content.append([line[field] for field in OUTPUT\_FIELDS])

with open('out.csv', 'w') as out:

writer = oat.OpenAPCUnicodeWriter(out, None, False, True)

writer.write\_rows(modified\_content)

print("\n\nPreprocessing finished, deleted articles overview:")

sorted\_reasons = sorted(DELETE\_REASONS.items(), key=lambda x: x[1])

sorted\_reasons.reverse()

for item in sorted\_reasons:

\_print("r", item[0].ljust(72) + str(item[1]))

\_print("r,", "-------------------------------------------------")

\_print("r", "Total".ljust(72) + str(sum(DELETE\_REASONS.values())))

shutdown()

if \_\_name\_\_ == '\_\_main\_\_' and \_\_package\_\_ is None:

sys.path.append(path.dirname(path.dirname(path.dirname(path.dirname(path.abspath(\_\_file\_\_))))))

import openapc\_toolkit as oat

main()

# **3) jisc\_preprocessing.py**

#!/usr/bin/env python3

# -\*- coding: UTF-8 -\*-

import argparse

import csv

import datetime

import json

from os import path

import re

import sys

from urllib.error import HTTPError, URLError

ARG\_HELP\_STRINGS = {

"source\_file": "The jisc csv file",

"exchange\_rates\_cache\_file": "An optional cache file for ECB exchange rates",

"no\_decorations": "Do not use ANSI coded colors in console output",

"jisc\_file\_format": "The format type of the Jisc input file"

}

FIELDNAMES = {

"2014\_16": {

"article": [

"APC paid (actual currency) including VAT if charged",

"APC paid (£) including VAT (calculated)",

"APC paid (£) including VAT if charged",

"Currency of APC",

"DOI",

"Date of APC payment",

"Date of initial application by author",

"ISSN0",

"Institution",

"Journal",

"Licence",

"PubMed Central (PMC) ID",

"PubMed ID",

"Publisher",

"TCO year",

"Type of publication",

"Drop?",

"Year of publication",

"period",

"is\_hybrid",

"euro"

],

"book": [

"Line number",

"APC paid (actual currency) including VAT if charged",

"APC paid (£) including VAT (calculated)",

"APC paid (£) including VAT if charged",

"Article title",

"Currency of APC",

"DOI",

"Date of APC payment",

"Date of initial application by author",

"Institution",

"Journal",

"Licence",

"Publisher",

"TCO year",

"Type of publication",

"Year of publication",

"period",

"euro",

"ISBN"

]

},

"2017": {

"article": [

"APC paid (£) including VAT if charged",

"DOI",

"Date of APC payment",

"ISSN0",

"Institution",

"Journal",

"Licence",

"PubMed ID",

"Publisher",

"TCO year",

"Type of publication",

"Drop?",

"Period of APC payment",

"period",

"is\_hybrid",

"euro"

],

"book": [

"Line number",

"APC paid (£) including VAT if charged",

"Article title",

"DOI",

"Date of APC payment",

"Institution",

"Journal",

"Licence",

"Publisher",

"TCO year",

"Type of publication",

"Period of APC payment",

"period",

"euro",

"ISBN"

]

},

"2018": {

"article": [

"Institution",

"Date of acceptance",

"PubMed ID",

"DOI",

"Publisher",

"Journal",

"Type of publication",

"Date of publication",

"Date of APC payment",

"APC paid (£) including VAT if charged",

"period",

"is\_hybrid",

"euro"

],

"book": [

"Line number",

"Institution",

"Date of acceptance",

"DOI",

"Publisher",

"Journal",

"Type of publication",

"Article title",

"Date of publication",

"Date of APC payment",

"APC paid (£) including VAT if charged",

"period",

"euro",

"ISBN"

]

}

}

PUBLICATION\_TYPES\_BL = [

"Book chapter",

"Book edited",

"Conference Paper/Proceeding/Abstract",

"Letter"

]

PUBLICATION\_TYPES\_BOOKS = [

"Book",

"Monograph"

]

DATE\_DAY\_RE = {

"2014\_16": re.compile("(?P<year>[0-9]{4})-?(?P<month>[0-9]{2})?-?(?P<day>[0-9]{2})?"),

"2017": re.compile("(?P<year>[0-9]{4})-?(?P<month>[0-9]{2})?-?(?P<day>[0-9]{2})?"),

"2018": re.compile("(?P<month>[0-9]{1,2})/(?P<day>[0-9]{1,2})/(?P<year>[0-9]{4})")

}

DATE\_STRPTIME = {

"2014\_16": "%Y-%m-%d",

"2017": "%Y-%m-%d",

"2018": "%m/%d/%Y"

}

PERIOD\_FIELD\_SOURCE = {

"2014\_16": [

"Date of APC payment",

"Year of publication",

"Date of initial application by author",

"TCO year"

],

"2017": [

"Date of APC payment",

"Period of APC payment",

"TCO year"

],

"2018": [

"Date of APC payment",

"Date of publication",

"Date of acceptance"

]

}

EXCHANGE\_RATES\_CACHE = {}

EXCHANGE\_RATES\_CACHE\_FILE = None

DELETE\_REASONS = {}

CURRENT\_YEAR = 2017

#CURRENT\_YEAR = datetime.datetime.now().year

NO\_DECORATIONS = False

def delete\_line(line\_dict, reason):

\_print("r", " - " + reason + ", line deleted")

if reason not in DELETE\_REASONS:

DELETE\_REASONS[reason] = 1

else:

DELETE\_REASONS[reason] += 1

for key in line\_dict:

line\_dict[key] = ""

def line\_as\_list(line\_dict, pub\_type):

return [line\_dict[field] for field in FIELDNAMES[FORMAT][pub\_type]]

def is\_money\_value(string):

try:

number = float(string)

return number > 0

except ValueError:

return False

def is\_valid\_date(date\_match\_obj):

gd = date\_match\_obj.groupdict()

if gd["year"] is None or gd["month"] is None or gd["day"] is None:

return False

try:

date = datetime.datetime(int(gd["year"]), int(gd["month"]), int(gd["day"]))

if date > datetime.datetime.now():

return False

return True

except ValueError:

return False

def shutdown():

\_print("r", "Updating exchange rates cache...")

with open(EXCHANGE\_RATES\_CACHE\_FILE, "w") as f:

f.write(json.dumps(EXCHANGE\_RATES\_CACHE, sort\_keys=True, indent=4, separators=(',', ': ')))

f.flush()

\_print("r", "Done.")

sys.exit()

def \_print(color, s):

if color in ["r", "y", "g", "b"] and not NO\_DECORATIONS:

getattr(oat, "print\_" + color)(s)

else:

print(s)

def get\_exchange\_rate(currency, frequency, date, jisc\_format):

if currency not in EXCHANGE\_RATES\_CACHE:

EXCHANGE\_RATES\_CACHE[currency] = {}

if frequency not in EXCHANGE\_RATES\_CACHE[currency]:

EXCHANGE\_RATES\_CACHE[currency][frequency] = {}

if not len(EXCHANGE\_RATES\_CACHE[currency][frequency]):

try:

rates = oat.get\_euro\_exchange\_rates(currency, frequency)

EXCHANGE\_RATES\_CACHE[currency][frequency] = rates

except HTTPError as httpe:

\_print("r", "HTTPError while querying the ECB data warehouse: " + httpe.reason)

shutdown()

except URLError as urle:

\_print("r", "URLError while querying the ECB data warehouse: " + urle.reason)

shutdown()

except ValueError as ve:

\_print("r", "ValueError while querying the ECB data warehouse: " + ve.reason)

shutdown()

if frequency == "D":

# The ECB does not report exchange rate for all dates due to weekends/holidays. We have

# consider some days in advance to find the next possible data in some cases.

day = datetime.datetime.strptime(date, DATE\_STRPTIME[jisc\_format])

for i in range(6):

future\_day = day + datetime.timedelta(days=i)

search\_day = future\_day.strftime("%Y-%m-%d")

if search\_day in EXCHANGE\_RATES\_CACHE[currency][frequency]:

\_print("y", " [Exchange rates: Cached value used]")

if i > 0:

msg = " [Exchange rates: No rate found for date {}, used value for {} instead]"

\_print("y", msg.format(date, search\_day))

return EXCHANGE\_RATES\_CACHE[currency][frequency][search\_day]

\_print("r", "Error during Exchange rates lookup: No rate for " + date + " or any following day!")

shutdown()

else:

return EXCHANGE\_RATES\_CACHE[currency][frequency][date]

def calculate\_euro\_value(line, jisc\_format):

payment\_date = line["Date of APC payment"]

date\_match = DATE\_DAY\_RE[jisc\_format].match(payment\_date)

if jisc\_format in ["2017", "2018"]:

apc\_pound = line["APC paid (£) including VAT if charged"]

field\_used\_for\_pound\_value = "APC paid (£) including VAT if charged"

elif jisc\_format == "2014\_16":

apc\_orig = line["APC paid (actual currency) including VAT if charged"]

apc\_pound = ""

field\_used\_for\_pound\_value = ""

for field in ["APC paid (£) including VAT (calculated)", "APC paid (£) including VAT if charged"]:

if is\_money\_value(line[field]):

apc\_pound = line[field]

field\_used\_for\_pound\_value = field

break

if is\_money\_value(apc\_orig):

currency = line["Currency of APC"].strip()

if currency == "EUR":

line["euro"] = apc\_orig

msg = " - Created euro field ('{}') by using the value in 'APC paid (actual currency) including VAT if charged' directly since the currency is EUR"

\_print("g", msg.format(apc\_orig))

elif len(currency) == 3:

if date\_match and is\_valid\_date(date\_match):

rate = get\_exchange\_rate(currency, "D", payment\_date, jisc\_format)

euro\_value = round(float(apc\_orig) / float(rate), 2)

line["euro"] = str(euro\_value)

msg = " - Created euro field ('{}') by dividing the value in 'APC paid (actual currency) including VAT if charged' ({}) by {} (EUR -> {} conversion rate on {}) [ECB]"

msg = msg.format(euro\_value, apc\_orig, rate, currency, payment\_date)

\_print("g", msg)

else:

year = line["period"]

if int(year) >= CURRENT\_YEAR:

del\_msg = "period ({}) too recent to determine average yearly conversion rate".format(year)

delete\_line(line, del\_msg)

return

try:

rate = get\_exchange\_rate(currency, "A", year, jisc\_format)

except KeyError:

\_print("r", "KeyError: An average yearly conversion rate is missing (" + currency + ", " + year + ")")

shutdown()

euro\_value = round(float(apc\_orig) / float(rate), 2)

line["euro"] = str(euro\_value)

msg = " - Created euro field ('{}') by dividing the value in 'APC paid (actual currency) including VAT if charged' ({}) by {} (avg EUR -> {} conversion rate in {}) [ECB]"

msg = msg.format(euro\_value, apc\_orig, rate, currency, year)

\_print("g", msg)

if line["euro"] == "" and is\_money\_value(apc\_pound):

if date\_match and is\_valid\_date(date\_match):

rate = get\_exchange\_rate("GBP", "D", payment\_date, jisc\_format)

euro\_value = round(float(apc\_pound) / float(rate), 2)

line["euro"] = str(euro\_value)

msg = " - Created euro field ('{}') by dividing the value in '{}' ({}) by {} (EUR -> GBP conversion rate on {}) [ECB]"

msg = msg.format(euro\_value, field\_used\_for\_pound\_value, apc\_pound, rate, payment\_date)

\_print("g", msg)

else:

year = line["period"]

if int(year) > CURRENT\_YEAR:

del\_msg = "period ({}) too recent to determine average yearly conversion rate".format(year)

delete\_line(line, del\_msg)

return

try:

rate = get\_exchange\_rate("GBP", "A", year, jisc\_format)

except KeyError:

\_print("r", "KeyError: An average yearly conversion rate is missing (GBP, " + year + ")")

shutdown()

euro\_value = round(float(apc\_pound) / float(rate), 2)

line["euro"] = str(euro\_value)

msg = " - Created euro field ('{}') by dividing the value in '{}' ({}) by {} (avg EUR -> GBP conversion rate in {}) [ECB]"

msg = msg.format(euro\_value, field\_used\_for\_pound\_value, apc\_pound, rate, year)

\_print("g", msg)

if line["euro"] == "":

delete\_line(line, "Unable to properly calculate a converted euro value")

def main():

global EXCHANGE\_RATES\_CACHE, EXCHANGE\_RATES\_CACHE\_FILE, NO\_DECORATIONS, FORMAT

parser = argparse.ArgumentParser()

parser.add\_argument("source\_file", help=ARG\_HELP\_STRINGS["source\_file"])

parser.add\_argument("jisc\_file\_format", choices=list(FIELDNAMES), help=ARG\_HELP\_STRINGS["jisc\_file\_format"])

parser.add\_argument("-c", "--exchange\_rates\_cache\_file", help=ARG\_HELP\_STRINGS["exchange\_rates\_cache\_file"], default="\_exchange\_rates\_cache.json")

parser.add\_argument("-n", "--no-decorations", help=ARG\_HELP\_STRINGS["no\_decorations"], action="store\_true")

args = parser.parse\_args()

NO\_DECORATIONS = args.no\_decorations

EXCHANGE\_RATES\_CACHE\_FILE = args.exchange\_rates\_cache\_file

FORMAT = args.jisc\_file\_format

if path.isfile(args.exchange\_rates\_cache\_file):

with open(EXCHANGE\_RATES\_CACHE\_FILE, "r") as f:

try:

EXCHANGE\_RATES\_CACHE = json.loads(f.read())

except ValueError:

\_print("r", "Could not decode a cache structure from " + EXCHANGE\_RATES\_CACHE\_FILE + ", starting with an empty cache.")

f = open(args.source\_file, "r", encoding="utf-8")

reader = csv.DictReader(f)

article\_content = [list(FIELDNAMES[FORMAT]["article"])]

book\_content = [list(FIELDNAMES[FORMAT]["book"])]

empty\_article\_line = ["" for i in range(len(FIELDNAMES[FORMAT]["article"]))]

empty\_book\_line = ["" for i in range(len(FIELDNAMES[FORMAT]["book"]))]

for line in reader:

line["period"] = ""

line["euro"] = ""

line["Journal"] = line["Journal"].replace("\n", " ")

\_print("b", "--- Analysing line " + str(reader.line\_num) + " ---")

is\_book = False

pub\_type = line["Type of publication"]

if pub\_type in PUBLICATION\_TYPES\_BOOKS:

line["Line number"] = str(reader.line\_num)

line["ISBN"] = ""

is\_book = True

else:

line["is\_hybrid"] = ""

# Publication blacklist checking

if pub\_type in PUBLICATION\_TYPES\_BL and not is\_book:

delete\_line(line, "Blacklisted pub type ('" + pub\_type + "')")

article\_content.append(list(empty\_article\_line))

continue

# DOI checking

if len(line["DOI"].strip()) == 0 and not is\_book:

delete\_line(line, "Empty DOI")

article\_content.append(list(empty\_article\_line))

continue

# Drop checking

if "Drop?" in FIELDNAMES[FORMAT]["article"] and line["Drop?"] == "1":

delete\_line(line, "Drop mark found")

article\_content.append(list(empty\_article\_line))

continue

# period field generation

for source\_field in PERIOD\_FIELD\_SOURCE[FORMAT]:

content = line[source\_field].strip()

match = DATE\_DAY\_RE[FORMAT].match(content)

if match:

year = match.groupdict()["year"]

if int(year) > CURRENT\_YEAR:

continue

line["period"] = year

msg = " - Created period field ('{}') by parsing value '{}' in column '{}'".format(year, content, source\_field)

\_print("g", msg)

break

else:

delete\_line(line, "Unable to determine payment date for period column")

article\_content.append(list(empty\_article\_line))

continue

# euro field generation

calculate\_euro\_value(line, FORMAT)

if is\_book:

if line["Line number"] != "":

book\_content.append(line\_as\_list(line, "book"))

delete\_line(line, "Book content (extracted to separate file)")

article\_content.append(list(empty\_article\_line))

else:

article\_content.append(line\_as\_list(line, "article"))

with open('out.csv', 'w') as out:

writer = oat.OpenAPCUnicodeWriter(out, None, False, True)

writer.write\_rows(article\_content)

with open('out\_books.csv', 'w') as out:

writer = oat.OpenAPCUnicodeWriter(out, None, False, True)

writer.write\_rows(book\_content)

print("\n\nPreprocessing finished, deleted articles overview:")

sorted\_reasons = sorted(DELETE\_REASONS.items(), key=lambda x: x[1])

sorted\_reasons.reverse()

for item in sorted\_reasons:

\_print("r", item[0].ljust(72) + str(item[1]))

\_print("r,", "-------------------------------------------------")

\_print("r", "Total".ljust(72) + str(sum(DELETE\_REASONS.values())))

shutdown()

if \_\_name\_\_ == '\_\_main\_\_' and \_\_package\_\_ is None:

sys.path.append(path.dirname(path.dirname(path.dirname(path.dirname(path.abspath(\_\_file\_\_))))))

import openapc\_toolkit as oat

main()

**4) norway\_preprocessing.py**

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

import argparse

from os import path

import sys

AVG\_YEARLY\_CONVERSION\_RATES = {

"2015": 0.1119,

"2016": 0.1077

}

def main():

parser = argparse.ArgumentParser()

parser.add\_argument("source\_file")

args = parser.parse\_args()

result = oat.analyze\_csv\_file(args.source\_file, 500)

if result["success"]:

csv\_analysis = result["data"]

print csv\_analysis

else:

print result["error\_msg"]

sys.exit()

dialect = csv\_analysis.dialect

csv\_file = open(args.source\_file, "r")

reader = oat.UnicodeDictReader(csv\_file, dialect=dialect)

fieldnames = reader.reader.fieldnames

modified\_content = [fieldnames]

for line in reader:

rate = AVG\_YEARLY\_CONVERSION\_RATES[line["Year"]]

euro\_value = float(line["APC in NOK"]) \* rate

line["APC in NOK"] = str(round(euro\_value, 2))

line\_as\_list = [line[field] for field in fieldnames]

modified\_content.append(line\_as\_list)

csv\_file.close()

with open('out.csv', 'w') as out:

quotemask = [False, True, True, True, True, True, False, True, False]

writer = oat.OpenAPCUnicodeWriter(out, quotemask, False, True)

writer.write\_rows(modified\_content)

if \_\_name\_\_ == '\_\_main\_\_' and \_\_package\_\_ is None:

sys.path.append(path.dirname(path.dirname(path.dirname(path.dirname(path.abspath(\_\_file\_\_))))))

import openapc\_toolkit as oat

main()

**5) openapc-se\_preprocessing.py**

#!/usr/bin/env python3

# -\*- coding: UTF-8 -\*-

import argparse

from os import path

import sys

ARG\_HELP\_STRINGS = {

"apc\_se\_file": 'Path to the openapc-se core data file (usually named "apc\_se.csv")',

"org\_acronym\_file": 'Path to the openapc-se institutional acronym mapping table (usually ' +

'named "org\_acronym\_name\_map.tsv")',

"transagree\_file": 'Path to the openapc-de transformative agreements file',

"duplicates\_file": 'Path to the openapc-de unresolved duplicates file'

}

INSTITUTIONAL\_MAPPINGS = {}

TRANSAGREE\_DOIS = []

DUPLICATES\_DOIS = []

EMPTY\_LINE = ["" for x in range(18)]

QUOTE\_MASK = [True, False, False, True, True, True, True, True, True, True, True, True, True,

True, True, True, True, True]

def main():

parser = argparse.ArgumentParser()

parser.add\_argument("apc\_se\_file", help=ARG\_HELP\_STRINGS["apc\_se\_file"])

parser.add\_argument("org\_acronym\_file", help=ARG\_HELP\_STRINGS["org\_acronym\_file"])

parser.add\_argument("transagree\_file", help=ARG\_HELP\_STRINGS["transagree\_file"])

parser.add\_argument("duplicates\_file", help=ARG\_HELP\_STRINGS["duplicates\_file"])

args = parser.parse\_args()

\_, acronyms = oat.get\_csv\_file\_content(args.org\_acronym\_file, "utf-8", True)

for line in acronyms:

INSTITUTIONAL\_MAPPINGS[line[0]] = line[1]

oat.print\_b("Loading transformative agreements file...")

\_, transagree\_content = oat.get\_csv\_file\_content(args.transagree\_file, "utf-8", True)

for line in transagree\_content:

doi = line[3]

if oat.has\_value(doi):

TRANSAGREE\_DOIS.append(line[3])

oat.print\_b("Done, " + str(len(TRANSAGREE\_DOIS)) + " DOIs extracted from transformative agreements file.")

oat.print\_b("Loading unresolved duplicates file...")

\_, duplicates\_content = oat.get\_csv\_file\_content(args.duplicates\_file, "utf-8", True)

for line in duplicates\_content:

doi = line[3]

if oat.has\_value(doi) and doi not in DUPLICATES\_DOIS:

DUPLICATES\_DOIS.append(doi)

oat.print\_b("Done, " + str(len(DUPLICATES\_DOIS)) + " unique DOIs extracted from unresolved duplicates file.")

apc\_se\_header, apc\_se\_content = oat.get\_csv\_file\_content(args.apc\_se\_file, "latin1", True)

stats = {

"offsetting\_duplicates": 0,

"unresolved\_duplicates": 0,

"no\_positive\_euro\_value": 0

}

modified\_content = []

for line in apc\_se\_content:

if line[3] in TRANSAGREE\_DOIS:

modified\_content.append(list(EMPTY\_LINE))

stats["offsetting\_duplicates"] += 1

continue

if line[3] in DUPLICATES\_DOIS:

modified\_content.append(list(EMPTY\_LINE))

stats["unresolved\_duplicates"] += 1

continue

euro = float(line[2])

if not euro > 0:

modified\_content.append(list(EMPTY\_LINE))

stats["no\_positive\_euro\_value"] += 1

continue

try:

line[0] = INSTITUTIONAL\_MAPPINGS[line[0]]

modified\_content.append(line)

except KeyError:

oat.print\_r("Error: No mapping found for institutional acronym '" + line[0] + "'!")

sys.exit()

print(modified\_content[0:100])

with open("apc\_se\_preprocessed.csv", "w") as f:

writer = oat.OpenAPCUnicodeWriter(f, QUOTE\_MASK, True, True)

writer.write\_rows(apc\_se\_header + modified\_content)

msg = "Preprocessed file written, {} offsetting duplicates removed, {} zero-cost articles removed, {} articles marked as unresolved duplicates removed."

oat.print\_g(msg.format(stats["offsetting\_duplicates"], stats["no\_positive\_euro\_value"], stats["unresolved\_duplicates"]))

if \_\_name\_\_ == '\_\_main\_\_' and \_\_package\_\_ is None:

sys.path.append(path.dirname(path.dirname(path.dirname(path.dirname(path.abspath(\_\_file\_\_))))))

import openapc\_toolkit as oat

main()