pip install rasterio

pip install tensorflow

import tensorflow as tf

# Check if TensorFlow is already initialized

if not tf.config.list\_physical\_devices('GPU'):

# If not initialized, manually initialize CUDA libraries

gpus = tf.config.experimental.list\_physical\_devices('GPU')

if gpus:

try:

# Ensure that cuDNN, cuFFT, and cuBLAS are only initialized once

tf.config.experimental.set\_memory\_growth(gpus[0], True)

# Manually initialize cuDNN

tf.keras.backend.set\_floatx('float32') # Ensure that the default data type is float32

# Manually initialize cuFFT

# Manually initialize cuBLAS

print("CUDA libraries initialized successfully.")

except RuntimeError as e:

print(e)

import os

# Disable CUDA initialization by TensorFlow

os.environ['TF\_FORCE\_GPU\_ALLOW\_GROWTH'] = 'true'

# Now import TensorFlow

import tensorflow as tf

#import libraries

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import pandas as pd

import numpy as np

import timeit

import os

import rasterio as rio

import numpy as np

import pandas as pd

import os

import rasterio as rio

from sklearn.preprocessing import StandardScaler

def data\_pre\_process():

print('Running preprocessing script...')

# Sorting training data

Target = '/Users/pnikrou/Documents/fereshte/Data\_nrt/run\_lisflood1/'

inun\_files2 = []

inun\_files2 += [each for each in os.listdir(Target) if each.endswith('.wd')]

inun\_files2.sort()

ls = ['Run1-0000.wd', 'Run1-0001.wd', 'Run1-0002.wd', 'Run1-0003.wd', 'Run1-0004.wd', 'Run1-0005.wd', 'Run1-0006.wd', 'Run1-0007.wd',

'Run2-0000.wd', 'Run2-0001.wd', 'Run2-0002.wd', 'Run2-0003.wd', 'Run2-0004.wd', 'Run2-0005.wd', 'Run2-0006.wd', 'Run2-0007.wd',

'Run3-0000.wd', 'Run3-0001.wd', 'Run3-0002.wd', 'Run3-0003.wd', 'Run3-0004.wd', 'Run3-0005.wd', 'Run3-0006.wd', 'Run3-0007.wd',

'Run4-0000.wd', 'Run4-0001.wd', 'Run4-0002.wd', 'Run4-0003.wd', 'Run4-0004.wd', 'Run4-0005.wd', 'Run4-0006.wd', 'Run4-0007.wd',

'Run5-0000.wd', 'Run5-0001.wd', 'Run5-0002.wd', 'Run5-0003.wd', 'Run5-0004.wd', 'Run5-0005.wd', 'Run5-0006.wd', 'Run5-0007.wd',

'Run6-0000.wd', 'Run6-0001.wd', 'Run6-0002.wd', 'Run6-0003.wd', 'Run6-0004.wd', 'Run6-0005.wd', 'Run6-0006.wd', 'Run6-0007.wd',

'Run7-0000.wd', 'Run7-0001.wd', 'Run7-0002.wd', 'Run7-0003.wd', 'Run7-0004.wd', 'Run7-0005.wd', 'Run7-0006.wd', 'Run7-0007.wd',

'Run8-0000.wd', 'Run8-0001.wd', 'Run8-0002.wd', 'Run8-0003.wd', 'Run8-0004.wd', 'Run8-0005.wd', 'Run8-0006.wd', 'Run8-0007.wd',

'Run9-0000.wd', 'Run9-0001.wd', 'Run9-0002.wd', 'Run9-0003.wd', 'Run9-0004.wd', 'Run9-0005.wd', 'Run9-0006.wd', 'Run9-0007.wd']

for i in ls:

if i in inun\_files2:

inun\_files2.remove(i)

target = []

for i in range(len(inun\_files2)):

data = rio.open(Target + inun\_files2[i])

band = data.read(1)

value = band.flatten()

target.append(value)

Y = np.array(target)

Y[Y < 0.3] = 0

directory1 = '/Users/pnikrou/Documents/fereshte/Data\_nrt/run0/'

inun\_files = []

inun\_files += [each for each in os.listdir(directory1) if each.endswith('.wd')]

inun\_files.sort()

l = ['Run0-0000.wd', 'Run0-0001.wd', 'Run0-0002.wd', 'Run0-0003.wd', 'Run0-0004.wd', 'Run0-0005.wd', 'Run0-0006.wd', 'Run0-0007.wd']

for i in l:

inun\_files.remove(i)

test\_target = []

for i in range(len(inun\_files)):

data = rio.open(directory1 + inun\_files[i])

band = data.read(1)

value = band.flatten()

test\_target.append(value)

Y\_test = np.array(test\_target)

Y\_test[Y\_test < 0.3] = 0

print(Y\_test.shape)

data\_dir = '/Users/pnikrou/Documents/fereshte/Data\_nrt/flows/'

data = []

data += [file for file in os.listdir(data\_dir) if file.endswith('.csv')]

data.sort()

print('Flow data files:', data)

appended\_data = []

for f in os.listdir(data\_dir):

if f.endswith('.csv'):

df = pd.read\_csv(os.path.join(data\_dir, f))

df['Upstream1'] = df['Upstream'].shift(1)

df['Upstream2'] = df['Upstream'].shift(2)

df['Upstream3'] = df['Upstream'].shift(3)

df['Upstream4'] = df['Upstream'].shift(4)

df['Upstream5'] = df['Upstream'].shift(5)

df['Upstream6'] = df['Upstream'].shift(6)

df['Upstream7'] = df['Upstream'].shift(7)

df['Upstream8'] = df['Upstream'].shift(8)

df = df.dropna()

appended\_data.append(df)

appended\_data = pd.concat(appended\_data, ignore\_index=True)

appended\_data.to\_csv('/Users/pnikrou/Documents/fereshte/Data\_nrt/flows/Train/appended.csv', index=False)

df = pd.read\_csv('/Users/pnikrou/Documents/fereshte/Data\_nrt/flows/Test/Upstream\_Flows\_Run0.csv')

df['Upstream1'] = df['Upstream'].shift(1)

df['Upstream2'] = df['Upstream'].shift(2)

df['Upstream3'] = df['Upstream'].shift(3)

df['Upstream4'] = df['Upstream'].shift(4)

df['Upstream5'] = df['Upstream'].shift(5)

df['Upstream6'] = df['Upstream'].shift(6)

df['Upstream7'] = df['Upstream'].shift(7)

df['Upstream8'] = df['Upstream'].shift(8)

df = df.dropna()

all\_data = pd.concat([appended\_data, df], ignore\_index=True)

print('Length of the data:', len(all\_data))

print('Number of data points in the Test part:', df.shape[0])

return all\_data, Y, Y\_test

def preprocess\_data(all\_data, Y, Y\_test):

scaler = StandardScaler()

all\_data = scaler.fit\_transform(all\_data)

X\_Train = all\_data[:2053, :]

X\_Test = all\_data[2053:, :]

x\_train = X\_Train.reshape(X\_Train.shape[0], 1, X\_Train.shape[1])

x\_test = X\_Test.reshape(X\_Test.shape[0], 1, X\_Test.shape[1])

steps = x\_train.shape[1]

features = x\_train.shape[2]

outputs = Y.shape[1]

return X\_Train, X\_Test, x\_train, x\_test, steps, features, outputs

# Example usage:

all\_data, Y, Y\_test = data\_pre\_process()

X\_train, X\_test, x\_train, x\_test, steps, features, outputs = preprocess\_data(all\_data, Y, Y\_test)

Y\_test = Y\_test[:2053] # Adjusting Y\_test size

print(X\_train.shape)

print(X\_test.shape)

print(x\_train.shape)

print(x\_test.shape)

print(Y\_test.shape)

print(steps)

print(features)

print(outputs)

Running preprocessing script...

(185, 1347885)

Flow data files: ['Upstream\_Flows\_Run1.csv', 'Upstream\_Flows\_Run2.csv', 'Upstream\_Flows\_Run3.csv', 'Upstream\_Flows\_Run4.csv', 'Upstream\_Flows\_Run5.csv', 'Upstream\_Flows\_Run6.csv', 'Upstream\_Flows\_Run7.csv', 'Upstream\_Flows\_Run8.csv', 'Upstream\_Flows\_Run9.csv']

Length of the data: 2238

Number of data points in the Test part: 185

(2053, 10)

(185, 10)

(2053, 1, 10)

(185, 1, 10)

(185, 1347885)

1

10

1347885

import tensorflow as tf

gpus = tf.config.experimental.list\_physical\_devices('GPU')

if gpus:

try:

# Currently, memory growth needs to be the same across GPUs

for gpu in gpus:

tf.config.experimental.set\_memory\_growth(gpu, True)

logical\_gpus = tf.config.experimental.list\_logical\_devices('GPU')

print(len(gpus), "Physical GPUs,", len(logical\_gpus), "Logical GPUs")

except RuntimeError as e:

# Memory growth must be set before GPUs have been initialized

print(e)

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import Conv1D, Flatten, Dense, Input, BatchNormalization, Dropout, Activation

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.utils import plot\_model

from tensorflow.keras.activations import relu

import matplotlib.pyplot as plt

import datetime

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import pandas as pd

import numpy as np

import timeit

import os

import rasterio as rio

import tensorflow as tf

print(tf.\_\_version\_\_)

def CNN\_Model(x\_train, Y, x\_test, Y\_test, steps, features, outputs):

'''

Two layered conv network

'''

print('Running the CNN model...')

from keras.optimizers import Adam

model = Sequential()

model.add(Conv1D(32, kernel\_size=1, activation='relu', input\_shape=(steps, features)))

model.add(Conv1D(128, activation='relu', kernel\_size=1))

model.add(Flatten())

model.add(Dense(32, activation='relu'))

model.add(Dense(256, activation='relu'))

model.add(Dense(512, activation='relu'))

model.add(Dense(outputs))

optimizer = Adam(learning\_rate=0.01)

model.compile(loss='mse', metrics=['mse'], optimizer=optimizer)

print(model.summary())

history = model.fit(x\_train, Y, validation\_data=(x\_test, Y\_test), batch\_size=2, verbose=1, epochs=50)

# plot history

plt.plot(history.history['loss'], label='train')

plt.plot(history.history['val\_loss'], label='test')

plt.xlabel('epochs')

plt.ylabel('loss')

plt.legend()

plt.show()

return model

def save\_model(model, name):

# Save the weights

model.save\_weights(name + '.weights.h5')

# Save the model architecture

with open(name+'.json', 'w') as f:

f.write(model.to\_json())

def load\_model(name):

##Loading the model weights

from tensorflow.keras.models import model\_from\_json

# Model reconstruction from JSON file

with open(name+'.json', 'r') as f:

model = model\_from\_json(f.read())

# Load weights into the new model

model.load\_weights(name + '.weights.h5')

return model

#for making predictions

def predict(model, X\_test):

tar\_dir = '/Users/pnikrou/Documents/fereshte/Data\_nrt/run0/CNN\_output' # Directory for the CNN output files

data = rio.open('/Users/pnikrou/Documents/fereshte/Data\_nrt/run0//Run0-0000.wd') # Reference image for fixing raster dimensions

## Make predictions

for i in range(len(X\_test)):

x\_test = X\_test[i]

x\_test = x\_test.reshape(1, 1, X\_test.shape[1])

y\_pred = model.predict(x\_test)

y\_pred.resize(data.height, data.width)

y\_pred[y\_pred < 0.2] = 0

src = data

with rio.Env():

# Write an array as a raster band to a new 8-bit file.

profile = src.profile

# Change the band count to 1, set the dtype to uint8, and specify LZW compression.

profile.update(dtype=str(y\_pred.dtype), count=1, compress='lzw')

with rio.open(tar\_dir + 'CNN\_2016\_{:03}'.format(i + 8) + '.asc', 'w', \*\*profile) as dst:

dst.write(y\_pred, 1)

return model

#Running the models

#Prepare taining and test data

all\_data, Y, Y\_test = data\_pre\_process()

# Call CNN\_Model with all required arguments

model = CNN\_Model(x\_train, Y, x\_test, Y\_test, steps, features, outputs)

# %%

name = '/Users/pnikrou/Documents/fereshte/Data\_nrt/run0/Model/CNN\_Model\_2016'

save\_model(model, name)

# %%

model = load\_model(name)

predict(model, X\_test)