from \_\_future\_\_ import print\_function

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

import os

import numpy as np

from utils import ADAMLearningRateTracker

import cloud\_net\_model

from losses import jacc\_coef, dice\_coef

from keras.optimizers import Adam

from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, CSVLogger

from generators import mybatch\_generator\_train, mybatch\_generator\_validation, mybatch\_generator\_gan

import pandas as pd

from utils import get\_input\_image\_names

from gan\_model import gan\_arch

def train():

# Define the generator model

generator = cloud\_net\_model.model\_arch(input\_rows=in\_rows,

input\_cols=in\_cols,

num\_of\_channels=num\_of\_channels,

num\_of\_classes=num\_of\_classes)

# Define the discriminator model

discriminator = cloud\_net\_model.model\_arch(input\_rows=in\_rows,

input\_cols=in\_cols,

num\_of\_channels=num\_of\_channels,

num\_of\_classes=num\_of\_classes)

# Define the GAN model

gan = gan\_arch(generator, discriminator)

# Compile the models

generator.compile(optimizer=Adam(lr=starting\_learning\_rate), loss=jacc\_coef, metrics=[jacc\_coef])

discriminator.compile(optimizer=Adam(lr=starting\_learning\_rate), loss=dice\_coef, metrics=[dice\_coef])

gan.compile(optimizer=Adam(lr=starting\_learning\_rate), loss='binary\_crossentropy')

# model.summary()

# Define callbacks

model\_checkpoint = ModelCheckpoint(weights\_path, monitor='val\_loss', save\_best\_only=True)

lr\_reducer = ReduceLROnPlateau(factor=decay\_factor, cooldown=0, patience=patience, min\_lr=end\_learning\_rate, verbose=1)

csv\_logger = CSVLogger(experiment\_name + '\_log\_1.log')

train\_img\_split, val\_img\_split, train\_msk\_split, val\_msk\_split = train\_test\_split(train\_img, train\_msk,

test\_size=val\_ratio,

random\_state=42, shuffle=True)

if train\_resume:

gan.load\_weights(weights\_path)

print("\nTraining resumed...")

else:

print("\nTraining started from scratch... ")

print("Experiment name: ", experiment\_name)

print("Input image size: ", (in\_rows, in\_cols))

print("Number of input spectral bands: ", num\_of\_channels)

print("Learning rate: ", starting\_learning\_rate)

print("Batch size: ", batch\_sz, "\n")

# Train the models

gan.fit\_generator(

generator=mybatch\_generator\_gan(list(zip(train\_img\_split, train\_msk\_split)), generator, discriminator, in\_rows, in\_cols, batch\_sz, max\_bit),

steps\_per\_epoch=np.ceil(len(train\_img\_split) / batch\_sz), epochs=max\_num\_epochs, verbose=1,

validation\_data=mybatch\_generator\_validation(list(zip(val\_img\_split, val\_msk\_split)), in\_rows, in\_cols, batch\_sz, max\_bit),

validation\_steps=np.ceil(len(val\_img\_split) / batch\_sz),

callbacks=[model\_checkpoint, lr\_reducer, ADAMLearningRateTracker(end\_learning\_rate), csv\_logger])

GLOBAL\_PATH = 'G:\cloud dataset\dataset'

TRAIN\_FOLDER = os.path.join(GLOBAL\_PATH, '38-Cloud\_training')

TEST\_FOLDER = os.path.join(GLOBAL\_PATH, '38-Cloud\_test')

in\_rows = 192

in\_cols = 192

num\_of\_channels = 4

num\_of\_classes = 1

starting\_learning\_rate = 1e-4

end\_learning\_rate = 1e-8

max\_num\_epochs = 2000 # just a huge number. The actual training should

.

.

.

.

from keras.layers import Input

from keras.models import Model

def gan\_arch(generator, discriminator):

# Set discriminator weights to non-trainable

for layer in discriminator.layers:

layer.trainable = False

# Define input shape

input\_shape = (generator.input\_shape[1], generator.input\_shape[2], generator.input\_shape[3])

# Define input tensor

input\_tensor = Input(shape=input\_shape)

# Generate images with generator

generated\_images = generator(input\_tensor)

# Output from discriminator

validity = discriminator(generated\_images)

# Create and compile GAN model

gan = Model(input\_tensor, validity)

gan.compile(loss='binary\_crossentropy', optimizer=Adam(lr=0.0002, beta\_1=0.5))

return gan

def train():

# Create generator and discriminator models

generator = cloud\_net\_model.model\_arch(input\_rows=in\_rows,

input\_cols=in\_cols,

num\_of\_channels=num\_of\_channels,

num\_of\_classes=num\_of\_classes)

discriminator = cloud\_net\_model.model\_arch(input\_rows=in\_rows,

input\_cols=in\_cols,

num\_of\_channels=num\_of\_channels,

num\_of\_classes=1)

# Create GAN model by chaining generator and discriminator

discriminator.compile(loss='binary\_crossentropy', optimizer=d\_optimizer)

discriminator.trainable = False

gan = Sequential()

gan.add(generator)

gan.add(discriminator)

gan.compile(loss='binary\_crossentropy', optimizer=g\_optimizer)

# Load weights if training is being resumed

if train\_resume:

gan.load\_weights(weights\_path)

print("\nTraining resumed...")

else:

print("\nTraining started from scratch... ")

# Train the GAN

for epoch in range(max\_num\_epochs):

for batch\_idx in range(num\_batches):

# Generate images with the generator

noise = np.random.normal(0, 1, (batch\_size, noise\_dim))

gen\_imgs = generator.predict(noise)

# Train the discriminator on the generated images

discriminator.trainable = True

discriminator.compile(loss='binary\_crossentropy', optimizer=d\_optimizer)

discriminator.train\_on\_batch(gen\_imgs, np.zeros((batch\_size, 1)))

# Train the discriminator on real images

real\_imgs = train\_images[batch\_idx \* batch\_size:(batch\_idx + 1) \* batch\_size]

discriminator.trainable = True

discriminator.compile(loss='binary\_crossentropy', optimizer=d\_optimizer)

discriminator.train\_on\_batch(real\_imgs, np.ones((batch\_size, 1)))

# Freeze the discriminator and train the generator with adversarial loss

discriminator.trainable = False

gan.compile(loss='binary\_crossentropy', optimizer=g\_optimizer)

gan.train\_on\_batch(noise, np.ones((batch\_size, 1)))

# Plot the progress

print("%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" % (epoch, d\_loss, 100\*d\_acc, g\_loss))

# Save the model after every save\_interval epochs

if epoch % save\_interval == 0:

gan.save\_weights('gan\_%d.h5' % epoch)