import os

from PIL import Image

import numpy as np

from keras.models import Sequential

from keras.layers import Conv2D

from keras.layers import AveragePooling2D

from keras.layers import Flatten

from keras.layers import Dense

from keras.models import model\_from\_json

from keras.preprocessing.image import ImageDataGenerator

from scipy.ndimage import imread

from scipy.misc import imresize, imsave

IMG\_SIZE = 24

def collect():

train\_datagen = ImageDataGenerator(

rescale=1./255,

shear\_range=0.2,

horizontal\_flip=True,

)

val\_datagen = ImageDataGenerator(

rescale=1./255,

shear\_range=0.2,

horizontal\_flip=True, )

train\_generator = train\_datagen.flow\_from\_directory(

directory="dataset/train",

target\_size=(IMG\_SIZE, IMG\_SIZE),

color\_mode="grayscale",

batch\_size=32,

class\_mode="binary",

shuffle=True,

seed=42

)

val\_generator = val\_datagen.flow\_from\_directory(

directory="dataset/val",

target\_size=(IMG\_SIZE, IMG\_SIZE),

color\_mode="grayscale",

batch\_size=32,

class\_mode="binary",

shuffle=True,

seed=42

)

return train\_generator, val\_generator

def save\_model(model):

model\_json = model.to\_json()

with open("model.json", "w") as json\_file:

json\_file.write(model\_json)

# serialize weights to HDF5

model.save\_weights("model.h5")

def load\_model():

json\_file = open('model.json', 'r')

loaded\_model\_json = json\_file.read()

json\_file.close()

loaded\_model = model\_from\_json(loaded\_model\_json)

# load weights into new model

loaded\_model.load\_weights("model.h5")

loaded\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

return loaded\_model

def train(train\_generator, val\_generator):

STEP\_SIZE\_TRAIN=train\_generator.n//train\_generator.batch\_size

STEP\_SIZE\_VALID=val\_generator.n//val\_generator.batch\_size

print('[LOG] Intialize Neural Network')

model = Sequential()

model.add(Conv2D(filters=6, kernel\_size=(3, 3), activation='relu', input\_shape=(IMG\_SIZE,IMG\_SIZE,1)))

model.add(AveragePooling2D())

model.add(Conv2D(filters=16, kernel\_size=(3, 3), activation='relu'))

model.add(AveragePooling2D())

model.add(Flatten())

model.add(Dense(units=120, activation='relu'))

model.add(Dense(units=84, activation='relu'))

model.add(Dense(units=1, activation = 'sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit\_generator(generator=train\_generator,

steps\_per\_epoch=STEP\_SIZE\_TRAIN,

validation\_data=val\_generator,

validation\_steps=STEP\_SIZE\_VALID,

epochs=20

)

save\_model(model)

def predict(img, model):

img = Image.fromarray(img, 'RGB').convert('L')

img = imresize(img, (IMG\_SIZE,IMG\_SIZE)).astype('float32')

img /= 255

img = img.reshape(1,IMG\_SIZE,IMG\_SIZE,1)

prediction = model.predict(img)

if prediction < 0.1:

prediction = 'closed'

elif prediction > 0.9:

prediction = 'open'

else:

prediction = 'idk'

return prediction

def evaluate(X\_test, y\_test):

model = load\_model()

print('Evaluate model')

loss, acc = model.evaluate(X\_test, y\_test, verbose = 0)

print(acc \* 100)

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

train\_generator , val\_generator = collect()

train(train\_generator,val\_generator)