بسم تعالی

تمرین پیاده سازی مقاله
Medical image classification for Alzheimer's using a deep learning approach
با زبان برنامه نویسی پایتون

مقاله مد نظر، تشخیص و پیش بینی آلزایمر بر اساس کلاس بندی تصاویر MRI از مغز میباشد. در راستای این هدف از دیتاست **OASIS3** که شامل **۲۱۶۰** تصویر یکتا از MRI مغز است.

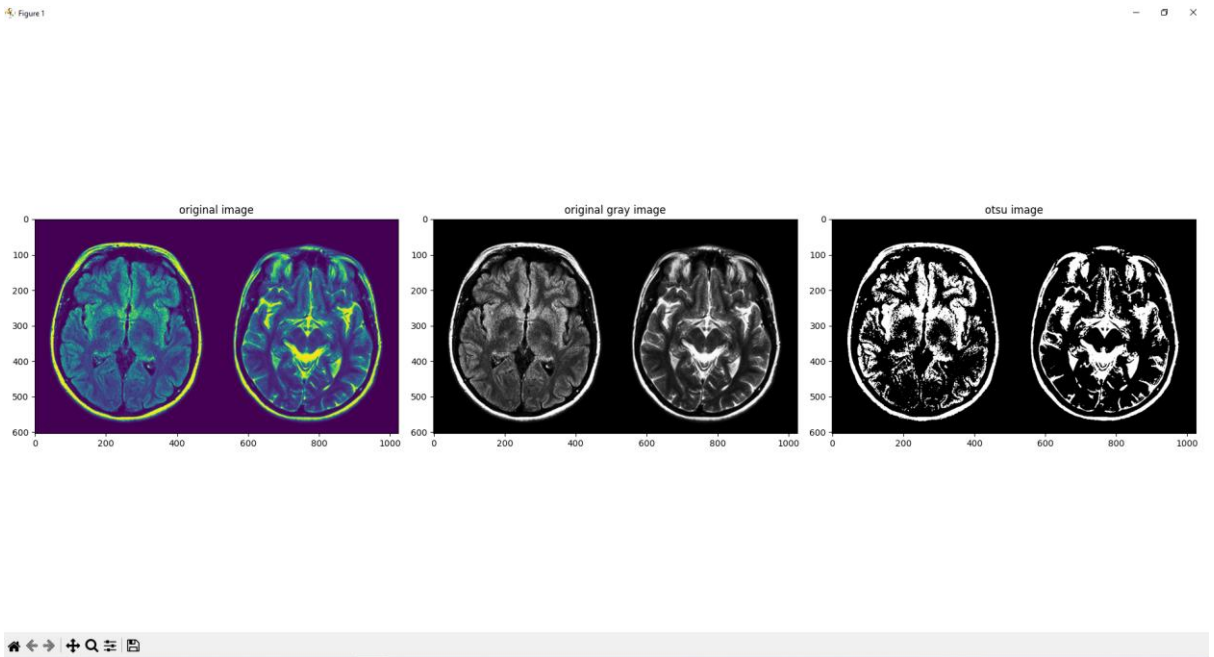
نکته ای که در این مقاله وجود دارد استفاده از یک متد هوشمندانه در جهت پیش پردازش داده ها برای بالا بردن دقت آموزش و جدا سازی بهتر داده ها میباشد و آن تکنیکOTSU میباشد.

الگوریتم OTSU

این الگوریتم در بینایی ماشین برای آستانه بندی خودکار تصاویر مورد استفاده قرار میگیرد. به صورت کلی این الگوریتم با بهره گیری از thresholding یا آستانه گذاری تصاویر را به دو بخش پس زمینه و پیش زمینه تقسیم میکند. با این کار contrast تصاویر بالا رفته و دقت مدل در یادگیری افزایش پیدا میکند. همچنین نویز ها نیز حذف میشوند.

به جهت پیاده سازی مراحل تکنیک OTSU یک برنامه با نام **otsu.py** ایجاد شده است که روند تغییرات مراحل پیش پردازش داده را با تکنیک آستانه گذاری OTSU به نمایش در آورده است.

خروجی فایل به صورت زیر میباشد:



روند کار

روند کار به این صورت است که ابتدا طبق گفته مقاله تمامی تصاویر باید resize شده و به **ابعاد ۱۹۶ در ۱۹۶ پیکسل** درآیند. سپس تصاویراز RGB به **Grayscale** یا سطح خاکستری تبدیل میشوند و ۸ بیتی میشوند.

پس از آن با استفاده از الگوریتم OTSU تصاویر با سطوح خاکستری مختلف تبدیل به دو سطح شده و پس زمینه و پیش زمینه تفکیک میشوند.

در این پیاده سازی فایلی با نام Sement_all_images.py وجود دارد که دایرکتوری دیتاست را میگیرد و به صورت خودکار تمامی عملیات پیش پردازش داده را روی تمامی تصاویر اعمال میکند و در آخر در فولدر جدید با همان دسته بندی ها ذخیره میکند.

در زیر کد این برنامه را میبینیم.

```
import cv2 as cv
import glob

Directories = ["Mild Dementia","Moderate Dementia","Non Demented","Very mild
Dementia"]
for Dname in Directories:

   # Find all JPG files in the current directory
   images = glob.glob(f"F:\\tech\hadiSarab\GitHub\Data\input\{Dname}\*.jpg")
   print("\n\n\n\n\n\n\n\n\n Directory : ",Dname)

   for image in images:
      img = cv.imread(image, cv.IMREAD_GRAYSCALE)
      assert img is not None, "file could not be read, check with
os.path.exists()"
      new_width = 196
      new_height = 196
      img = cv.resize(img, (new_width, new_height))
      img = cv.medianBlur(img,5)

      th = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,\
      cv.THRESH_BINARY,11,2)
      name = image.split("\\")[-1]
      # print(name)

      cv.imwrite(f"F:\\tech\hadiSarab\GitHub\Data\Alzimer after
preprocessing\{Dname}\{name}",th)
```

برای این کار من در این پیاده سازی از کتابخانه OPENCV و از فانکشن adaptiveThresholding استفاده کردم. در این کتابخانه دو نوع thresholding وجود دارد یکی mean که با استفاده از میانگین توابع توذیع آستانه گذاری را انجام میدهد و دیگری Gaussian که با استفاده از تکنیک آستانه گذاری تطبیقی گوسی این کار را انجام میدهد . روش کار به این صورت است که ابتدا **kernel size** را مشخص میکند و روی تصویر اعمال میکند که باعث حذف نویز از تصاویر میشود. سپس **میانگین محلی** را بر اساس سایز کرنل در همسایگی پیکسل محاسبه میکند و **آستانه گذاری** را انجام میدهد.

این روش دقت بالاتری نسبت به صرفا میانگین گیری دارد .از مزایای دیگر این روش مقاومت در برابر نویز و قابلیت انطباق را میتوان نام برد.

پس از اعمال مراحل پیش پردازش بر روی تمامی تصاویرو ذخیره در فولدر های جدید نوبت به ایجاد مدل deep learning با استفاده از لایه های کانولوشن میرسد.

طبق مقاله مدلی که باید پیاده سازی شود شامل لایه های زیر میباشد.

Table 1 Parameters of proposed approach

Layer (type)	Output shape	Param
conv2d (Conv2D)	(None, 196, 196, 16)	160
conv2d_ (Conv2D)	(None, 196, 196, 16)	2320
conv20_2 (Conv20)	(None, 196, 196, 32)	4640
conv2d_3 (Conv2D)	(None, 196, 196, 64)	18,496
conv20_4 (Conv20)	(None, 196, 196, 128)	0
dropout (Dropout)	(None, 196, 196, 128)	73,856
conv20_5 (Conv20)	(None, 196, 196, 256)	295,168
dropout_1 (Dropout)	(None, 196, 196, 256)	0
batch_normalization (BatchNormalization)	(None, 196, 196, 256)	1024
max_pooling2d (MaxPooling2D)	(None, 98, 98, 256)	0
flatten (Flatten)	(None, 2,458,624)	0
dense (Dense	(None, 64)	157,352,000
dropout_2 (Dropout)	(None, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 64)	256
dense_1 (Dense)	(None, 4)	260
activation (Activation)	(None, 4)	0
Total params	157,748, 180	
Trainable params	157,747,540	
Non-trainable params	640	

این مدل شامل ۶ لایه کانولوشن، pooling ، dropout و fullyconnect میباشد.

جهت پیاده سازی این مدل از کتابخانه keras که بر پایه Tensorflow نوشته شده است استفاده شد. همچنین برای تقسیم داده های به داده های test و train و validation از کتابخانه sikitlearn

استفاده گردید. جهت وارد کردن داده های تصاویر به برنامه نیاز به یک جدول شامل آدرس تصویر و لیبل داشتم که از کتابخانه Numpy برای ایجاد Dataframe استفاده کردم.

برای ایجاد دیتافریم برنامه دایرکتوری ذخیره دیتاست را میگیرد و به صورت خودکار تمامی تصاویر موجود در دایرکتوری را در هر چهار دسته بندی خوانده و نام فولدرآن را به عنوان **لیبل** تصویر درج میکند. در کل چهار کلاس

```
["Mild Dementia","Moderate Dementia","Non Demented","Very mild Dementia"]
```

وجود دارد.

پس از ایجاد دیتا فریم داده ها طبق گفته مقاله به صورت **۸۰ درصد داده آموزش و ۲۰ درصد داده تست تقسیم بندی میشوند. از آن ۸۰ درصد داده آموزشی ۳۰ درصد برای Validation** جداشد.

حال به ایجاد مدل اصلی میرسیم.

مدل اصلی را از نوع **sequential** انتخال میکنیم و به ترتیب لایه هارا برای آن تعریف میکنیم.

تمامی مراحل ایجاد مدل در فایل **Model.py** ذخیره شده که کد آن در زیر قابل ملاحظه است

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout, BatchNormalization, Activation
import os
import numpy as np
from PIL import Image
from sklearn.model_selection import train_test_split
import pandas as pd


data_dir = "F:\\tech\GitHub\Data\Alzeimer After Preprocessing"


categories = [d for d in os.listdir(data_dir) if
os.path.isdir(os.path.join(data_dir, d))]


image_data = {}


for category in categories:
    category_path = os.path.join(data_dir, category)
    for image_file in os.listdir(category_path):
        image = Image.open(os.path.join(category_path, image_file))
        image_data[os.path.join(category_path, image_file)] = category
```

```
image_paths = list(image_data.keys())
image_labels = np.array([image_data[path] for path in image_paths])


data = {'image_path': image_paths, 'label': image_labels}
df = pd.DataFrame(data)


print(df.head())


X_train1, X_test, y_train1, y_test = train_test_split(image_paths,
image_labels, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train1, y_train1,
test_size=0.3, random_state=42)


# Initialize the model
model = Sequential()

# Add Convolutional and Pooling layers
model.add(Conv2D(16, kernel_size=(3,3), input_shape=(196, 196, 3),
activation='relu'))
model.add(Conv2D(16, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(128, kernel_size=(3,3), activation='relu'))
model.add(Dropout(0.2))
model.add(Conv2D(256, kernel_size=(3,3), activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())

# Add Dense layers
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(4))
model.add(Activation('softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_val,
y_val))
```

پس از ایجاد مدل همانطور که مقاله اشاره کرده است تعداد تکرار یا **epoch** را **۱۰۰** درنظر میگیریم.

به جهت کم حجم شدن فایل نهایی دیتاست را که بیش از ۱ گیگ فضا نیاز داشت، در فایل ارائه قرار داده نشده است. صرفا تعدادی از هر فولدر برای تست کارکرد برنامه قرار داده شده است.

همچنین لینک منابع مورد استفاده جهت اجرا این تمرین در فایل **links.txt** موجود میباشد.