به نام خدا

گزارش پروژه پایانی مبانی بینایی کامپیوتر

توسعه الگوریتم Anti Spoofing برای تشخیص زنده بودن چهره در ویدئو

اعضای گروه: هانیه اسعدی ۹۹۵۲۱۰۵۵، نیایش خانی ۹۹۵۲۱۲۳۵

استاد درس: دکتر محمدرضا محمدی

نیم سال دوم ۱۴۰۳-۱۴۰۲

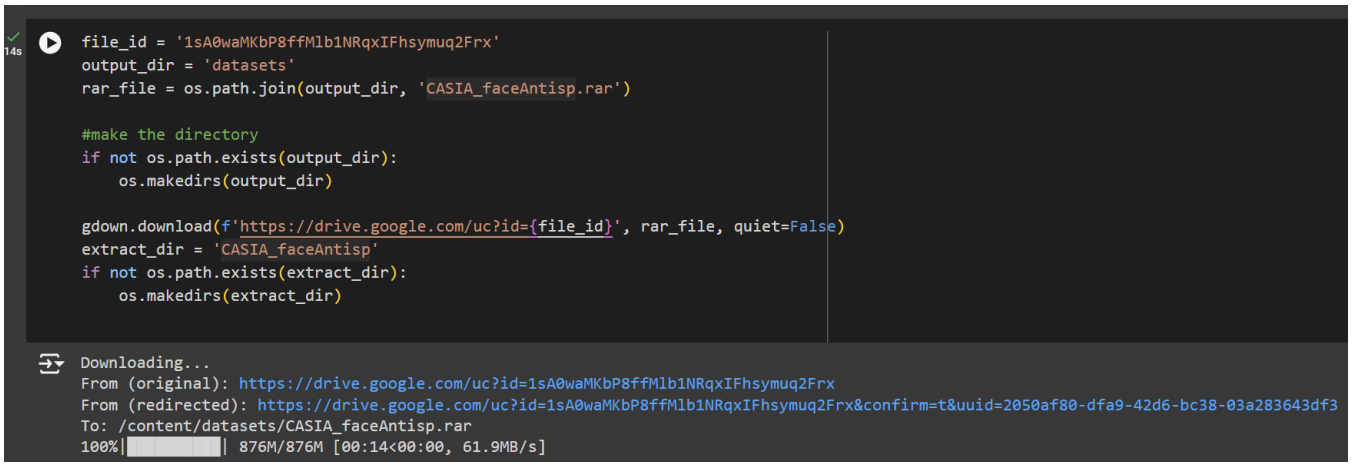
برای شروع پروژه ابتدا به دنبال دیتاست مناسبی بودیم که در آخر با جستجوی فراوان دیتاست CASIA-FASD در لینک زیر را پیدا کردیم.

https://github.com/mnikitin/Learn-Convolutional-Neural-Network-for-Face-Anti-Spoofing/issues/5

https://mega.nz/file/9BFDiKqT#VLexsuFDZjoA97c1J_h9hInm8AG75h6kG-TUfm3hYwg

با دریافت دیتاست مورد نظر آن را در گوگل درایو آپلود کردیم.

حال به سراغ توضیحات کد می‌رویم :



```
file_id = '1sA0waMKbP8ffMlb1NRqxIFhsymuq2Frx'
output_dir = 'datasets'
rar_file = os.path.join(output_dir, 'CASIA_faceAntisp.rar')

#make the directory
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

gdown.download(f'https://drive.google.com/uc?id={file_id}', rar_file, quiet=False)
extract_dir = 'CASIA_faceAntisp'
if not os.path.exists(extract_dir):
    os.makedirs(extract_dir)
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1sA0waMKbP8ffMlb1NRqxIFhsymuq2Frx
From (redirected): https://drive.google.com/uc?id=1sA0waMKbP8ffMlb1NRqxIFhsymuq2Frx&confirm=t&uuid=2050af80-dfa9-42d6-bc38-03a283643df3
To: /content/datasets/CASIA_faceAntisp.rar
100%| | 876M/876M [00:14<00:00, 61.9MB/s]
```

پس از نصب dependency های لازم، دیتاست را دانلود می کنیم.

```
train_folder_path = 'CASIA_faceAntisp/train_release'
test_folder_path = 'CASIA_faceAntisp/test_release'
```

پوشه هایی که حاوی داده های آموزشی و تست هستند را نام‌گذاری می کنیم. هر پوشه حاوی ۲۰ پوشه دیگر است که در هر کدام ۱۲ ویدئو قرار دارد. بنابراین در مجموع ۲۴۰ ویدئو برای تست داریم.

```
with rarfile.RarFile(rar_file) as rf:
  rf.extractall(extract_dir)
```

سپس با استفاده از خط بالا، فایل دریافت شده را extract می‌کنیم.

به دلیل وجود نداشتن لیبل برای ویدئوهای موجود در دیتاست، با استفاده از این دیکشنری آنها را لیبل گذاری می کنیم.

```
labels = {
    'HR_1': 1,
    'HR_2': 0,
    'HR_3': 0,
    'HR_4': 1,
    '1': 1,
    '2': 1,
    '3': 0,
    '4': 0,
    '5': 0,
    '6': 0,
    '7': 0,
    '8': 0
}
```

استخراج فریم از تصویر

```
def create_frame_labels_dict(folder_path, labels):
    frame_labels_dict = {}

    for folder in os.listdir(folder_path):
        path = os.path.join(folder_path, folder)
        if os.path.isdir(path):
            # Iterate through each file in the folder
            for file in os.listdir(path):
                file_name, file_extension = os.path.splitext(file)
                # if file_name in labels:
                label = labels[file_name]
                frame_labels_dict[f"{folder}_{file_name}"] = label

    return frame_labels_dict
```

سپس برای دسترسی راحتتر به هر فایل و لیبل آن، با استفاده از تابع، نام پوشه به همراه نام فایل را با لیبل آن در یک دیکشنری ذخیره می کنیم.

```
train_frame_labels_dict = create_frame_labels_dict(train_folder_path, labels)
test_frame_labels_dict = create_frame_labels_dict(test_folder_path, labels)
```

این کار را برای دیتاهای train و تست انجام می دهیم.

از آنجایی که نیاز است تا از هر ویدئو چند فریم تهیه کنیم، با استفاده از تابع make_directory ، یک پوشه برای نگهداری از فریم ها ترتیب می دهیم.

حال با استفاده از تابع extract_random_frames ، از هر ویدئو چند فریم را به صورت رندوم استخراج می کنیم؛ به این صورت که با دریافت مسیر پوشه، مکان ذخیره کردن خروجی و تعداد فریم های مورد استخراج، به ازای هر فایل موجود در پوشه، با استفاده از تابع cv2.VideoCapture فریم های هر ویدئو را استخراج می کنیم.

سپس از میان فریم های دریافت شده، به تعدادی که در ورودی تابع گرفته ایم، چند فریم انتخاب می کنیم. در آخر آنها را در خروجی مورد نظر با نام پوشه و فایل خود ذخیره می کنیم. برای جلوگیری از خطاهای احتمالی مانند ارور در خواندن فریم هم، پیام مناسب چاپ می شود.

```
train_frames_dir = 'train_frames'
make_directory(train_frames_dir)

extract_random_frames(train_folder_path, train_frames_dir)
```

```
Warning: Could not read frame 105 from video HR_3.avi
Warning: Could not read frame 313 from video HR_2.avi
Warning: Could not read frame 131 from video HR_3.avi
```

برای استخراج فریم از داده آموزشی، ابتدا یک پوشه برای آنها می‌سازیم. سپس با استفاده از تابعی که پیش تر ذکر کردیم، به ازای هر ویدئو موجود در آرگومان اول، ۱۰ فریم استخراج کرده و آنها را در دایرکتوری ذکر شده ذخیره می کنیم. همانطور که مشاهده می شود، تعدادی از فریم ها قابل خواندن نبوده اند.

برش ناحیه چهره

با توجه به مورد ذکر شده در داکیومنت سوال پروژه، نیاز به برش ناحیه چهره از هر فریم خواهیم داشت بنابراین برای انجام این کار تابع extract_faces_from_frames را تهیه کردیم.

برای تشخیص چهره از تابع cv2.CascadeClassifier استفاده شده است. از آنجا که این تابع برای تصاویری که چهره شخص از کنار مشخص است، دقت بالایی ندارد ، از متد های دیگر نیز برای این کار استفاده کردیم و با مقایسه خروجی ها و تعداد شکست ها در تشخیص چهره در هر متد، تابع مورد نظر را انتخاب کردیم.

توابع دیگری که برای ارزیابی مورد استفاده قرار گرفتند cv2.FaceDetectorYN (مناسب برای تشخیص چهره از هر زاویه) و MTCNN() بودند. علی رغم فرض ما در مورد عملکرد بهتر این توابع، متد CascadeClassifier نتیجه بهتری داشت و تعداد چهره های بیشتری را تشخیص داد.

منبع : https://medium.com/@itsuki.enjoy/frontal-profile-face-detection-in-python-6bed8d196edf

برای تشخیص چهره در سایز های مختلف از تابع detectMultiScale استفاده می کنیم که نیازمند دریافت تصویر خاکستری است. بنابراین هر فریم را به سطح خاکستری برده و تابع مورد نظر را بر روی آن اجرا می کنیم.

در آخر نیز چهره های تشخیص داده شده را در فولدر خروجی داده شده ذخیره می کنیم.

```
# make directory for store the detected faces
train_faces_dir = 'train_faces'
make_directory(train_faces_dir)

extract_faces_from_frames(train_frames_dir, train_faces_dir)
```

همانطور که مشاهده می شود، با استفاده از تابع مذکور تشخیص چهره را در داده های train انجام داده ایم.

این کار را برای داده های train انجام می دهیم.

```
#dictionary to store frame names and corresponding features
features_dict = create_frequency_frame_dict(train_faces_dir)
```

استخراج فرکانس تصویر

با استفاده از تابع extract_frequency_features و کتابخانه numpy ، فرکانس تصویر را استخراج می کنیم.

سپس با استفاده از تابع create_frequency_frame_dict ، فرکانس هر فریم از ویدئوهای موجود در مسیر ورودی را استخراج کرده و آنها را در یک دیکشنری ذخیره می کنیم.

این کار را برای داده های train و test و همچنین استخراج فریم و برش چهره را برای داده تست انجام می دهیم.

```
#dictionary to store frame names and corresponding features
features_dict = create_frequency_frame_dict(train_faces_dir)

test_frames_dir = 'test_frames'
make_directory(test_frames_dir)

test_faces_dir = 'test_faces'
make_directory(test_faces_dir)

extract_random_frames(test_folder_path, test_frames_dir)

extract_faces_from_frames(test_frames_dir, test_faces_dir)
```

```
Warning: Could not read frame 131 from video HR_3.avi
Warning: Could not read frame 118 from video HR_3.avi
Warning: Could not read frame 235 from video HR_2.avi
Warning: Could not read frame 105 from video HR_3.avi
```

```
features_dict_test = create_frequency_frame_dict(test_faces_dir)
```

سپس با استفاده از تابع load_frames_and_labels_from_dict ، هر فریم با لیبل های خود را در یک لیست قرار داده و آنها را برای بررسی دقت مدل آماده می کند.

در تابع load_frequency_features_and_labels_from_dict نیز همین کار را برای فرکانس انجام می دهیم.

```
train_frames, train_labels = load_frames_and_labels_from_dict(train_frame_labels_dict, train_frames_dir)
test_frames, test_labels = load_frames_and_labels_from_dict(test_frame_labels_dict, test_frames_dir)

train_faces_frame, train_faces_label = load_frames_and_labels_from_dict(train_frame_labels_dict, train_frames_dir)
test_faces_frame, test_faces_label = load_frames_and_labels_from_dict(test_frame_labels_dict, test_frames_dir)

train_frequency_frame, train_frequency_label = load_frequency_features_and_labels_from_dict(features_dict, train_frames_dir)
test_frequency_frame, test_frequency_label = load_frequency_features_and_labels_from_dict(features_dict, test_frames_dir)
```

فریم ها، تصاویر چهره و فرکانس های داده های train و test را به همراه لیبل با استفاده از توابع ذکر شده استخراج می کنیم.

فرض ما برای این پروژه در کنار هم قرار دادن مدل های آموزش دیده شده بر اساس فریم تصویر، چهره برش داده شده و فرکانس است. به همین دلیل راه زیر را در پیش گرفتیم.

ابتدا مدل ها را با شبکه MobileNet با وزن های آموزش دیده از قبل در imagenet می سازیم. سپس برای لایه های آخر از Flatten ، Dropout و Dense استفاده می کنیم تا در نهایت خروجی باینری داشته باشیم.

سپس هر کدام از این خروجی ها را با یکدیگر concat کرده و آن را مدل می دهیم تا مدل ما بر اساس هر سه ویژگی آموزش ببیند.

پس از کامپایل کردن مدل، روی داده آموزش و تست مدل را فیت می کنیم. (از داده تست برای validation استفاده شده است.)

همچنین به منظور جلوگیری از overfit شدن مدل، از early stopping استفاده کرده ایم تا در صورت تغییر نکردن accuracy پس از گذشت ۳ دوره، آموزش متوقف شود.

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
Epoch 1/5
8/8 [==============================] - 97s 12s/step - loss: 0.6626 - accuracy: 0.6417 - val_loss: 0.6375 - val_accuracy: 0.6676
Epoch 2/5
8/8 [==============================] - 84s 11s/step - loss: 0.6417 - accuracy: 0.6667 - val_loss: 0.6387 - val_accuracy: 0.6676
Epoch 3/5
8/8 [==============================] - 84s 11s/step - loss: 0.6289 - accuracy: 0.6667 - val_loss: 0.6395 - val_accuracy: 0.6676
Epoch 4/5
8/8 [==============================] - 84s 11s/step - loss: 0.6332 - accuracy: 0.6667 - val_loss: 0.6378 - val_accuracy: 0.6676
Epoch 5/5
8/8 [==============================] - 85s 11s/step - loss: 0.6263 - accuracy: 0.6708 - val_loss: 0.5920 - val_accuracy: 0.6676
12/12 [==============================] - 34s 3s/step - loss: 0.5920 - accuracy: 0.6676
Test Accuracy: 66.76%
```