به نام خدا

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

گزارش پروژه پایانی

شیما سادات ناصری

۴۰۱۱۲۸۱۴

دکتر مهدی علیاری شوره دلی

تیر ۱۴۰۳

فهرست مطالب

عنوان شماره صفحه

چکیده

در این پروژه مجموعه داده تصویری دستگاه پنتاکم[1] و پتانسیل ممان‌های زرنیکه به عنوان روشی برای استخراج ویژگی‌ها برای طبقه‌بندی تصاویر برای تشخیص بیماری قوزقرنیه بررسی شده‌است. یک مجموعه‌داده از تصاویر برای پیش‌پردازش داده‌ها و استخراج ویژگی‌های زرنیکه استفاده شد که سپس به عنوان ورودی برای انواع مدل‌های یادگیری ماشین، از جمله ماشین‌های بردار پشتیبانی(SVM) ، درختان تصمیم (DT) و جنگل‌های تصادفی (RF) به کار گرفته شد. هدف این پروژه ارزیابی اثربخشی این مدل‌ها در طبقه‌بندی دقیق تصاویر بر اساس ویژگی‌های زرنیکه و یافتن روشی بهینه برای تشخیص بیماری است.

روش‌شناسی به‌کاررفته شامل فرآیند جامعی از آماده‌سازی داده‌ها، استخراج ویژگی از مجموعه‌داده و پس از آن طبقه‌بندی است. ویژگی‌های استخراج‌شده ممان زرنیکه برای آموزش و ارزیابی مدل‌های یادگیری ماشین مذکور استفاده شد. به‌ویژه، مدل MLP با تابع زیان Renyi عملکرد بهینه‌ای از نظر دقت طبقه‌بندی نشان داد.

نتایج با استفاده از ماتریس‌های درهم‌ریختگی و نمودارهای t-SNE نشان داده شده‌اند که بینشی درباره قابلیت جداسازی ویژگی‌ها و عملکرد مدل ارائه می‌دهند. یافته‌های این پروژه نشان‌دهنده پتانسیل ممان‌های زرنیکه به عنوان تکنیکی قوی برای استخراج ویژگی‌ها برای طبقه‌بندی تصاویر است. به طور کلی، این پروژه با ارائه رویکردی سیستماتیک برای استخراج و طبقه‌بندی ویژگی‌ها، به زمینه بینایی کامپیوتر کمک می‌کند و نتایج نشان می‌دهند که ممان‌های زرنیک، همراه با مدل MLP با تابع زیان Renyi ، می‌توانند دقت بالایی در وظایف طبقه‌بندی تصاویر به‌دست آورند، و این رویکرد را برای کاربردهای مشابه در آینده مناسب می‌سازند.

[1] Pentacam

معرفی و پیش‌پردازش داده‌ها

چالش اصلی این بیماری برای پزشکان، تشخیص سریع‌تر بیماری در مراحل اولیه می‌باشد. این بیماری یکی از عوامل منع جراحی انکساری چشم است. به منظور انجام این نوع جراحی، جراح ابتدا باید مطمئن شود که قرنیه چشم بیمار کاملا سالم بوده و حتی مشکوک به قوز قرنیه هم نباشد. در نتیجه تشخیص دقیق این بیماری حتی در مراحل اولیه شکل گیری بشدت حائز اهمیت می‌باشد.

تعریف داده

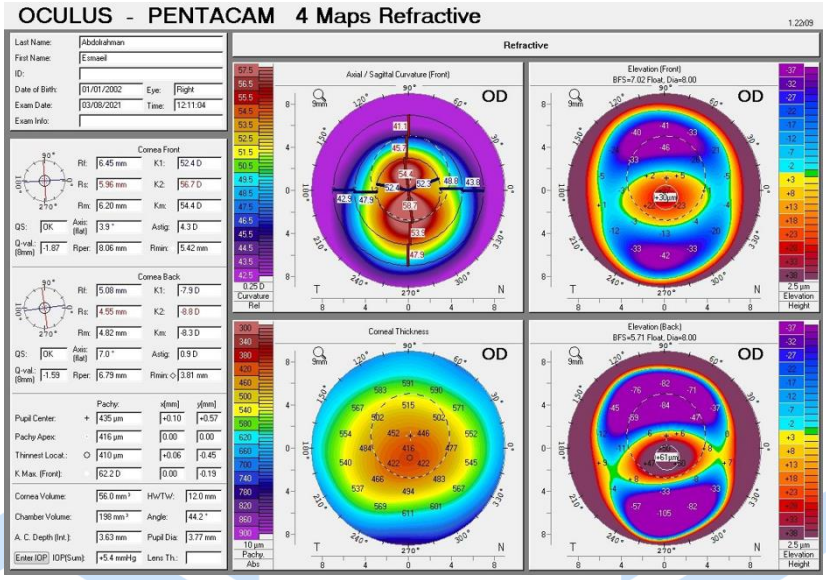
درحال حاضر دستگاه های متنوعی برای تصویر برداری از چشم وجود دارد. پزشکان ، از تصاویر دستگاههای مختلفی مانند سریوس[1] و یا پنتاکم که عملکرد تقریبا مشابه‌ای دارند، برای پی بردن به وجود این بیماری استفاده می‌کنند. در بین این دستگاه ها، دستگاه پنتاکم دارای فراوانی استفاده بیشتری نسبت به سایر دستگاه ها دارد. این دستگاه امکان تشخیص قوز قرنیه خفیف را نیز دارد. یکی از مهم‌ترین نقشه‌ای که این دستگاه ارائه می‌دهد، 4Maps Refractive است که در ادامه از این نقشه برای تحلیل استفاده خواهد شد.

طی همکاری آزمایشگاه ارس با بیمارستان فارابی برای جمع آوری اطلاعات، از بخش لیزیک بیمارستان فارابی استفاده شده‌است. در این پژوهش، از تصاویر مربوط به بیمارانی که در سال های ۲۰۱۹ تا ۲۰۲۳ به بیمارستان مراجعه کرده‌اند، استفاده خواهد شد. با توجه به اینکه وجود بیماری در ژنتیک افراد در حال حاضر در بیمارستان فارابی توسط مصاحبه کلامی با خود بیمار مشخص می‌شود و در بسیاری از مواقع بیمار اطلاع درست و دقیقی در مورد وجود این بیماری در خانواده‌اش ندارد یا به علت عدم شناخت نسبت به بیماری از وجود آن در سایر افراد آگاه نیست؛ در نتیجه پارامتر ژنتیک در این بررسی وارد نشده‌است.

تا کنون ۱۸۷۸ داده برچسب‌دار بدست آمده است که ۸۳۴ داده‌ی آن سالم و ۱۰۴۴ داده‌ی آن قوز قرنیه می‌باشد. از این داده‌ها در ادامه مسیر استفاده خواهد شد.

[1] CSO Sirius Topographer

مطابق شکل ۱، این نقشه‌ها شامل چهار نقشه شیب انحنای قرنیه[1]، برآمدگی قدامی قرنیه[2]، برآمدگی خلفی قرنیه[3] و در نهایت ضخامت قرنیه[4] است که هر پزشک پس از بازنگری تمام ویژگی‌های چهار نقشه بر طبق علم چشم پزشکی، تصمیم‌گیری می‌کند.

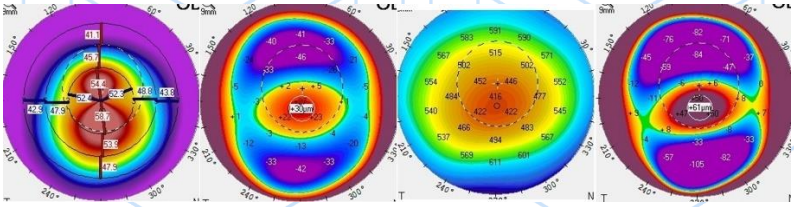


شکل ۱- تصویر **4Maps Refractive**

هدف اصلی ما استفاده از ۴ نقشه اصلی برای ادامه مسیر است اما مسئله مهمی که وجود دارد این است که این تصویر دارای داده‌های زیادی است که به آن نیازی نیست (مانند داده‌های عددی که در کنار نقشه‌ها وجود دارد). به همین دلیل نیاز است تا داده‌ها پیش از ورود به شبکه پیش‌پردازش شوند.

پیش‌پردازش داده‌ها

نقشه‌های اصلی برای هر داده ابتدا نیاز است تا جدا شوند، داده مطابق شکل ۲ خواهد شد:



شکل ۲- نقشه‌های جدا شده از تصویر خام

اگر به هر نقشه توجه کنیم، خواهیم دید که داده‌های عددی ناخواسته زیادی روی نقشه‌ها نشسته که می‌توانند باعث ضعیف شدن عملکرد مدل‌های طبقه بندی کننده در ادامه شود. از طرف دیگر، داده‌های

[1] Sagittal curvature
[2] Anterior (Front) Elevation
[3] Posterior (Back) Elevation
[4] Corneal Thickness

اطراف هر نقشه، داده‌های مفیدی نیستند و می‌توان آن‌ها را حذف کرد. برای این کار، از دو تابع numberRemuving و cutCircle استفاده شده‌است که به صورت زیر عمل می‌کنند:

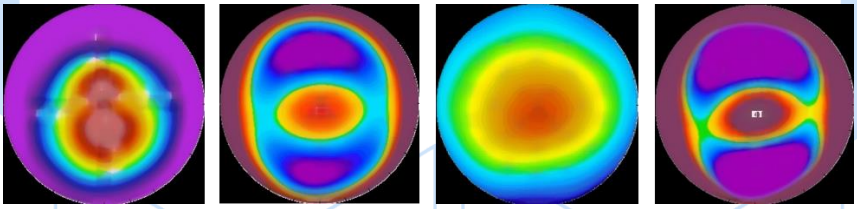
- cutCircle

تابع cutCircle به منظور برش یک دایره از تصویر و حذف نواحی خارج از دایره طراحی شده است. این تابع ابتدا ابعاد تصویر را بدست می‌آورد، سپس یک ماسک دایره‌ای ایجاد می‌کند و آن را به تصویر اعمال می‌کند تا نواحی خارج از دایره با مقدار صفر جایگزین شوند. در نهایت، تصویر ماسک شده به تابع numberRemoving برای پردازش بیشتر ارسال می‌شود.

- numberRemuving

تابع numberRemoving یک تصویر را پردازش می‌کند تا عناصر سیاه (مانند اعداد) را که در یک ناحیه دایره‌ای خاص قرار دارند، شناسایی و حذف کند. در این تابع از تکنیک‌های پردازش تصویر برای شناسایی دایره‌ی نقشه و اجزای سیاه و سفید مانند نوشته‌ها در آن استفاده می‌شود، سپس با تبدیل تصویر از RGB به HSV و استفاده از کانال اول آن، می‌توان پیکسل‌های مربوط به نوشتار را بهتر پیدا کرد. در کنار این، پیکسل‌های ناخواسته درون دایره با مقادیر فیلتر میانگین از ناحیه اطراف به فاصله ۱۰ پیکسل جایگزین می‌گردد. این باعث می‌شود که اجزای حذف شده به صورت یکپارچه با بقیه تصویر ترکیب شوند.

خروجی این دوتابع در نهایت مطابق شکل ۳ خواهد بود:



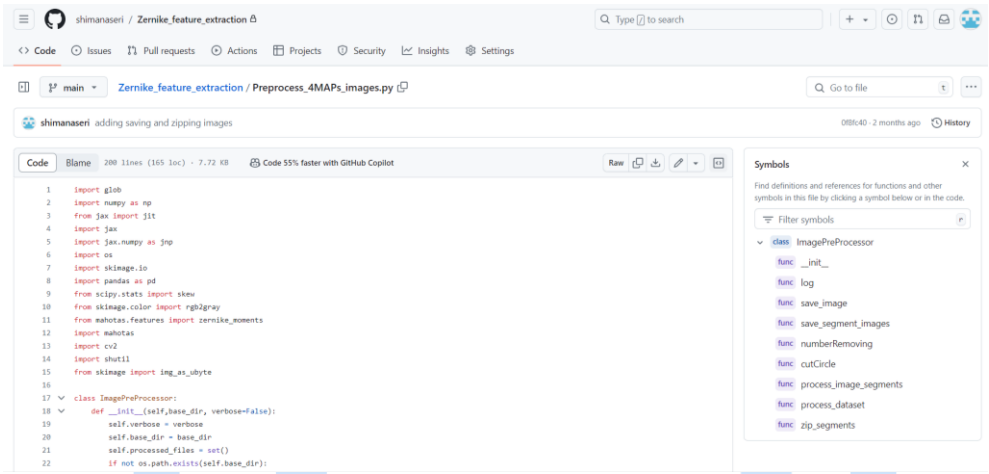
شکل ۳- نقشه‌های پیش‌پردازش شده

لازم به ذکر است که داده‌های مرتبط با چشم چپ و چشم راست نسبت به هم قرینه‌اند و برای یکسان شدن لازم است که یکی تبدیل به دیگری شود که طبق قرارداد، چشم‌های سمت چپ قرینه شدند. در نهایت پس از اجرای توابع بالا، تصاویر در ۴ پوشه مطابق با نقشه آن قرار داده می‌شود و یک فایل dataframe هم در کنار آن برای آدرس دهی به هر فایل ایجاد می‌گردد.

تمامی این فرایندها در یک کد داخل گیت هاب مطابق شکل ۴ قرار داده شده‌است که در صورت نیاز تنها این کد از گیت هاب مطابق کد زیر فراخوانی می‌شود.

```
From Zernike_feature_extraction.Preprocess_4MAPs_images import
ImageProcessor
```

```
processor = ImageProcessor('path_to_image.jpg')
images = processor.process_dataset(target_dir)
```



شکل ۴- نمایی از گیت هاب کد پیش‌پردازش

پس از انجام تمامی فرایندهای لازم، می‌توان از داده‌ها برای طبقه بندی استفاده کرد اما موضوع مهمی که وجود دارد، این است که تصاویر موجود نیاز است تا یک استخراج داده اولیه روی آن‌ها انجام شود تا مراحل طبقه بندی بعدی برای ما آسان‌تر شود.

استخراج ویژگی

در حال حاضر مدل‌های از پیش آموزش دیده و تست شده‌ی بسیاری هستند که می‌توان از آن‌ها برای استخراج ویژگی و یا طبقه بندی استفاده کرد. این مدل‌ها دارای لایه‌های زیاد و پیچیده‌ای هستند که پس از بررسی های زیادی به مدل‌های قابل قبولی تبدیل شده‌اند. اما ایرادی که این مدل‌ها برای داده‌های این پروژه دارند، این است که نقشه‌ها تمرکز دایروی دارند اما روند کاری این مدل‌ها مربعی است و این باعث می‌شود بخشی از ویژگی‌هایی که در هر نقشه وابسته به این خاصیت باشد، ارزش خود را دست بدهد. بدین منظور باید سراغ روش‌هایی که اساس دایروی دارند برویم. استفاده از ممان‌هایی[1] که نسبت به یک مرکز سنجیده می‌شوند، یکی از روش‌هایی است که برای این نقشه‌ها می‌توانند کاربردی باشند که یکی از این ممان‌ها، ممان زرنیکه[2] است که به صورت زیر تعریف می‌شود.

ممان زرنیکه

تبدیل زرنیکه، یک تبدیل ریاضی است که توسط فریدز زرنیکه[3] ، یک فیزیکدان هلندی، در سال ۱۹۳۴ معرفی شد. این تبدیل بر اساس تابع‌های چندجمله‌ای ارتباط داده میان یک تصویر و شکل هندسی آن است. تبدیل زرنیکه برای توصیف ویژگی‌های شکلی تصاویر استفاده می‌شود که از جمله آن‌ها می‌توان به تقارن، نواحی تاریک و روشن، تمامیت و سادگی اشاره کرد.

معادله تبدیل زرنیکه به صورت زیر است:

$$Z_{nm}(\rho,\varphi) = R_{nm}(\rho).e^{im\varphi} \qquad \text{(Eq.1)}$$

در Eq.1، Z_{nm} تابع زرنیکه، $R_{nm}(\rho)$ چندجمله‌ای زرنیکه، ρ شعاع تصویر، φ زاویه، n و m اعداد صحیح نامنفی است. معادله R_{nm} به صورت زیر می‌باشد.

$$R_{nm}(\rho) = \begin{cases} \sum_{i=0}^{\frac{n-m}{2}} \frac{(-1)^i (n-i)!}{i!\,([0.5(n+m)-i]!)^2} & for\ n-m\ even \\ 0 & for\ n-m\ odd \end{cases}$$

روش زرنیکه یک روش قدرتمند برای توصیف ویژگی‌های شکل در تصاویر است. این روش بر اساس تابع‌های زرنیکه از مجموعه توابع چندجمله‌ای است که با استفاده از محدودیت‌ها و ارتباط‌هایی، ویژگی‌های

[1] Momentum
[2] Zernike Moment
[3] Frits Zernike

فضایی و شکلی را به دقت بیان می‌کند. توابع زرنیکه به عنوان یک مجموعه متعامد از توابع پایه بر روی یک دایره واحد در صفحه دو بعدی تعریف شده‌اند و توانایی توصیف شکل‌های مختلف را دارند.

برای استخراج ویژگی‌های شکل با استفاده از روش زرنیکه ، می‌توان از ممان زرنیکه استفاده کرد. ممان زرنیکه معیاری است که با استفاده از توابع زرنیکه و تصویر ورودی، ویژگی‌های شکل را استخراج می‌کند. ممان زرنیکه شامل مجموعه‌ای از ضرایب که توصیف کننده ویژگی‌های شکل هستند است. با اعمال ممان زرنیکه به ویژگی‌های استخراج شده از شبکه پیش‌آموزش دیده، می‌توان ویژگی‌های جدیدی را برای تصاویر استخراج کرده و از آنها در مسائل تشخیص الگو، تشخیص اشیا، پردازش تصویر پزشکی و سایر برنامه‌های بینایی مصنوعی بهره برد.

ممان زرنیکه، معیاری برای توصیف ویژگی‌های شکلی تصویر ارائه می‌دهد و به صورت جمع وزن‌داری از توابع زرنیکه بر روی تصویر محاسبه می‌شود.

معادله محاسبه ممان زرنیکه بر اساس Eq.1 به صورت زیر خواهد بود:

$$M_{nm} = \frac{m+1}{\pi} \int_x \int_y f(x,y)[Z_{mn}(x,y)]^* dxdy \quad where\ x^2 + y^2 \leq 1$$

$$\Rightarrow in\ polar\ form: M_{nm} = \frac{1}{A} \sum_{i=1}^{N} I(x_i, y_i).Z_{nm}(\rho_i, \varphi_i)$$

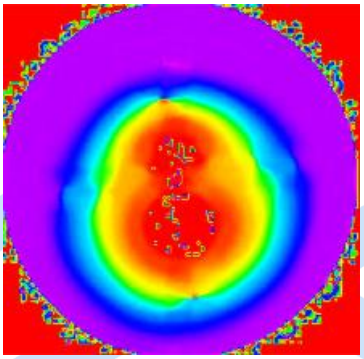
در این معادلات، M_{nm} ممان زرنیکه، $I(\rho,\varphi)$ تصویر ورودی، $Z_{nm}(\rho,\varphi)$ تابع زرنیکه، A مساحت تصویر، (x_i, y_i) مختصات نقاط تصویر، ρ_i شعاع نقطه (x_i, y_i) و N تعداد نقاط تصویر است.

پیاده سازی

برای این کار، از کتابخانه mahotas استفاده شده که در آن به صورت کامل به محاسبه ممان زرنیکه پرداخته است. نکته مهمی که تابع محاسبه‌گر ممان زرنیکه در این کتابخانه دارد این است که نمی‌تواند تصویر با سه کانال را به عنوان ورودی دریافت کند. در روش پیشنهادی خود کتابخانه، تصویر به صورت gray scale استفاده شده‌است؛ اما براساس تصاویر مجموعه داده‌ی این پروژه، باید داده‌های رنگی نیز در فرایند محاسبه حتما قرار داشته باشند. یک راه مناسب برای درگیر کردن رنگ‌ها در این فرایند، تبدیل کردن فرمت داده‌ها از RGB به HSV و استفاده از کانال اول آن است. سه کانال HSV شامل رنگ[1]، اشباع[2]

[1] Hue
[2] Saturation

و مقدار[1] است. به دلیل عدم تغییر محسوس تصویر RGB نسبت به خروجی کانال اول HSV، از این روش استفاده شده‌است. خروجی کانال اول HSV یکی از داده‌ها مطابق شکل ۵ است:



شکل ۵- خروجی کانال اول **HSV**

مشاهده می‌شود که کلیات ترکیب رنگ داده باقی مانده اما اطراف تصویر دارای نویز شده‌است. حال می‌توان با استفاده از این تصویر تک کاناله، ممان های زرنیکه را بدست آورد. کد این قسمت به صورت زیر است:

```
def compute_features(image_path, radius=123, degree=15):
    image = cv2.imread(image_path)
    if image is None:
        return None
    preprocessed_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    gray_image = rgb2hsv(preprocessed_image)[:,:,0]
    zernike_features = zernike_moments(gray_image, radius, degree=degree)
    return zernike_features
```

با استفاده از کانال H از تصویر، ممان‌های زرنیکه محاسبه می‌شوند. این محاسبات با استفاده از یک شعاع مشخص (radius) و درجه‌ای از دقت (degree) انجام می‌شود. در اینجا با در نظر گرفتن شعاع داخل هر نقشه که ۱۲۳ پیکسل است،تنها ممان‌های داخل آن محاسبه می‌شوند و عملا نویزهای به وجود آمده در طی تبدیل شدن به HSV درنظر گرفته نمی‌شوند. درجه دقت نیز با افزایش آن، تعداد ممان‌های خروجی را افزایش می‌دهد. با این وجود باید درنظر داشت که با افزایش آن، میزان محاسبات و زمان آن نیز افزایش پیدا می‌کند.

[1] Value

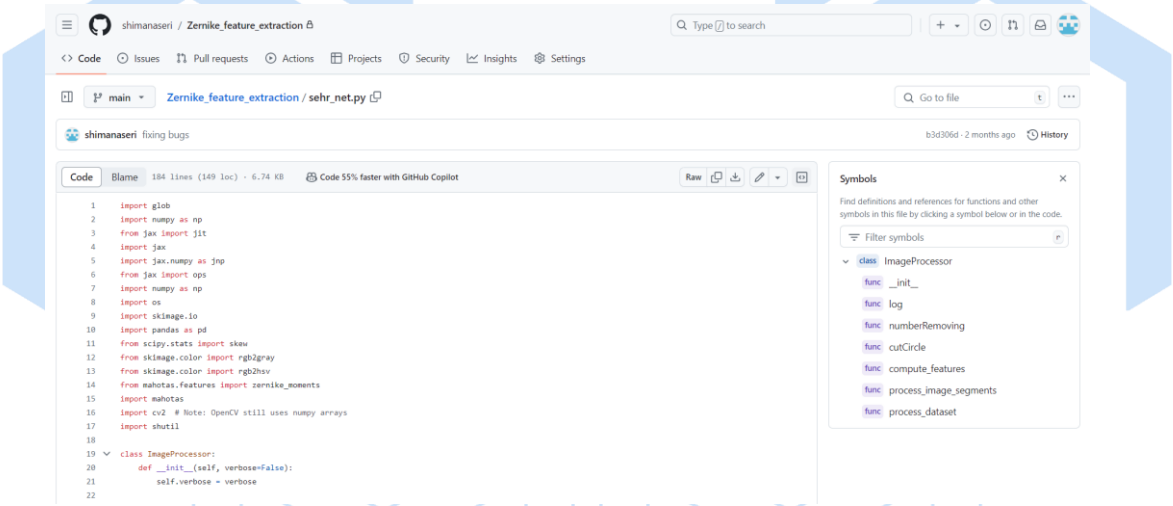
درصورتی که میزان دقت براساس همین تابع باشد، خروجی آن به ازای هر تصویر ورودی، یک بردار ۷۲ تایی از ویژگی است. تمامی این ویژگی‌ها به dataframe اصلی اضافه می‌شوند و برای مثال به صورت جدول ۱ همراه با برچسب‌های آن ذخیره می‌گردد.

جدول ۱-نمونه‌ای از داده‌های ذخیره شده

File Path	Segment	HSV Zernike Moments	File Name	label_2
/content/Dataset/QS_OK_TOTA	AXIAL Curvature	[0.31830989 0.06162201 0.11777852	Gandomi_Hosein_OS_11112019_145343_4 Maps Refr.JPG	1
/content/Dataset/QS_OK_TOTA	Corneal Thickness	[3.18309886e-01 8.95723422e-03	Gandomi_Hosein_OS_11112019_145343_4 Maps Refr.JPG	1
/content/Dataset/QS_OK_TOTA	Elevation (Front)	[0.31830989 0.02538017 0.06059899	Gandomi_Hosein_OS_11112019_145343_4 Maps Refr.JPG	1
/content/Dataset/QS_OK_TOTA	Elevation (Back)	[0.31830989 0.04214225 0.08777779	Gandomi_Hosein_OS_11112019_145343_4 Maps Refr.JPG	1

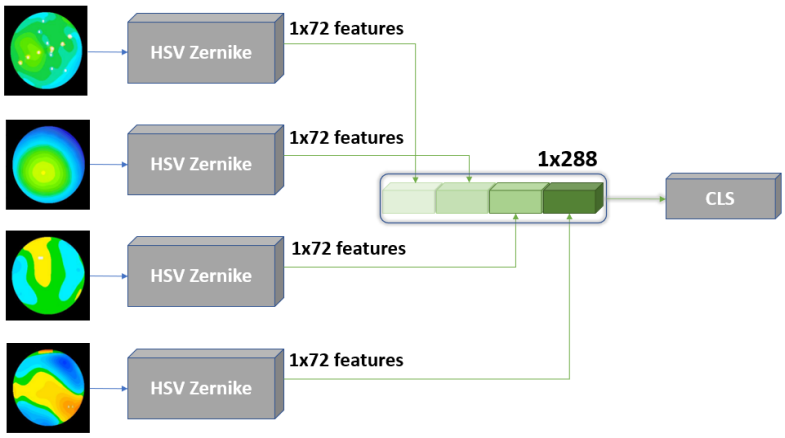
تمامی این فرایندها نیز در گیت هاب مطابق شکل ۶ قرار داده شدند و تنها با یک فراخوانی اجرا شده‌اند که کد آن به صورت زیر است:

```
from Zernike_feature_extraction.sehr_net import ImageProcessor


processor = ImageProcessor('path_to_image.jpg')
image_features = processor.process_dataset(target_dir)
```



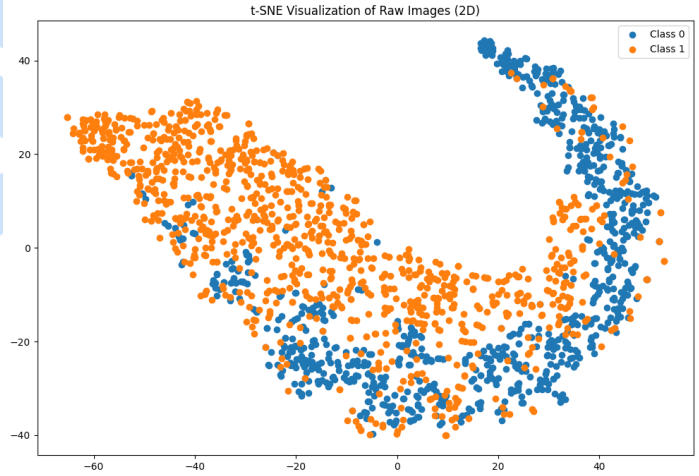
شکل ۶- نمایی از گیت هاب کد استخراج ویژگی

طبق مقاله‌ای که Murat Fırat و همکاران خود برای تشخیص قوزقرنیه ارائه کرده‌اند، ویژگی‌ها را می‌توان به صورت زیر به بخش طبقه بندی داد.



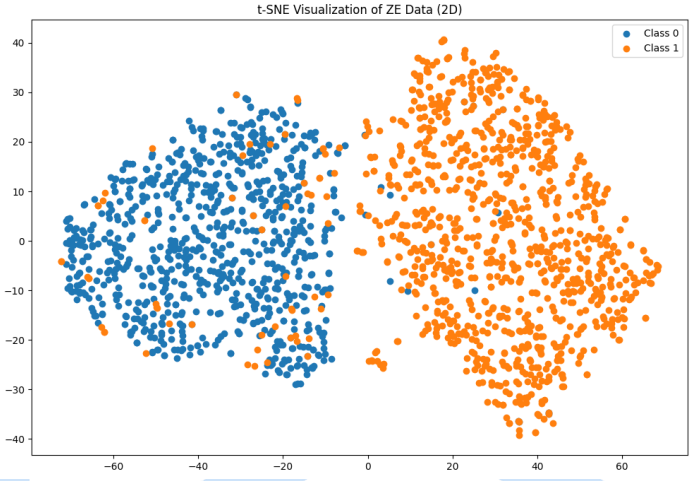
شکل ۷- نمایی از ورودی داده‌ها به شبکه

داده‌های خام در فضای خود با کمک t-SNE مطابق شکل ۸ دیده می‌شوند:



شکل ۸- خروجی **t-SNE** داده‌های خام

همانطور که دیده می‌شود، داده‌ها بشدت درهم تنیده است. حال اگر فضای ویژگی‌های استخراج شده را با کمک t-SNE رسم کنیم، شکل ۹ بدست می‌آید:



شکل ۹- خروجی **t-SNE** ویژگی‌های استخراج شده

همانطور که مشاهده می‌شود، داده‌ها از حالت درهم تنیدگی شدید خارج شده‌اند اما تعدادی از داده‌ها وجود دارند که به نسبت دسته اصلی به دسته مخالف نزدیک‌ترند. این امر می‌تواند به خاطر عدم توانایی ممان‌های زرنیکه برای جداسازی بخشی از داده‌ها و یا اشتباه در برچسب زنی رخ داده باشد.

برای بررسی بهتر داده‌های بدست آمده و مقایسه آن با داده‌های خام، معیارهای Silhouette Score، Calinski-Harabasz Index و Davies-Bouldin Index استفاده می‌کنیم. تحلیل هر معیار به صورت زیر می‌باشد.

معیار Silhouette Score اندازه‌گیری می‌کند که هر نمونه در یک مجموعه داده در مقایسه با خوشه‌های دیگر چقدر در خوشه اختصاص داده شده خود قرار می‌گیرد. از ۱- تا ۱ متغیر است، جایی که مقدار بالاتر نشان دهنده عملکرد بهتر خوشه بندی است. نمره نزدیک به ۱ نشان می‌دهد که نمونه‌ها به خوبی خوشه‌بندی شده‌اند، در حالی که نمرات نزدیک به ۰ نشان‌دهنده همپوشانی خوشه‌ها و نمرات منفی نشان می‌دهد که ممکن است نمونه‌ها به خوشه اشتباهی اختصاص داده شده باشند. امتیاز Silhouette را می توان برای ارزیابی فشرده بودن و جداسازی خوشه ها، ارائه بینشی در مورد کیفیت بازسازی داده ها یا نتایج خوشه بندی استفاده کرد.

شاخص Calinski-Harabasz که به عنوان معیار نسبت واریانس نیز شناخته می‌شود، اندازه‌گیری نسبت بین پراکندگی درون خوشه‌ای و پراکندگی بین خوشه‌ای است. هدف آن به حداکثر رساندن نسبت است که نشان دهنده خوشه‌های به خوبی جدا شده و فشرده است. مقدار بالاتر Calinski-Harabasz Index مربوط به عملکرد خوشه بندی بهتر است. از این شاخص می توان برای ارزیابی کیفیت تخصیص خوشه ها و فشرده بودن داده های بازسازی شده استفاده کرد.

شاخص Davies-Bouldin شباهت بین خوشه ها را با در نظر گرفتن پراکندگی درون خوشه ای و جدایی بین خوشه ای ارزیابی می کند. میانگین شباهت بین هر خوشه و مشابه ترین خوشه را با در نظر گرفتن اندازه آنها اندازه گیری می کند. مقدار پایین‌تر Davies-Bouldin Index نشان‌دهنده عملکرد بهتر خوشه‌بندی است، با مقادیر کوچک‌تر نشان‌دهنده خوشه‌های فشرده‌تر و متمایزتر است. این شاخص می تواند به ارزیابی کیفیت نتایج خوشه بندی یا بازسازی داده ها با در نظر گرفتن تعادل بین فاصله های درون خوشه ای و بین خوشه ای کمک کند.

مقادیر معیارهای توضیح داده شده در بالا به صورت زیر می‌باشد.

جدول ۲- معیارهای محاسبه شده برای داده خام و ویژگی های استخراج شده

	Silhouette Score	Calinski-Harabasz Index	Davies-Bouldin Index
Main data	0.1376	375.3668	2.0950
Features	0.1572	376.9401	2.1566

مقادیر بالا نشان می‌دهند که داده‌ها پس از استخراج ویژگی از آن‌ها، توانسته کمی فاصله بین دو کلاس را بیشتر کند. در بخش بعدی با کمک ویژگی‌های استخراج شده به سراغ طبقه بندی داده‌ها می‌رویم.

طبقه‌بندی داده‌ها

داده‌های بدست آمده در روش‌های MLP، SVM، Decision Tree و Random Forest طبقه بندی و مقایسه می‌شوند. در تمامی روش‌ها، داده‌های تست ۲۰ درصد از داده‌های کل را تشکیل داده و باقی داده‌ها، داده‌های آموزش درنظر گرفته شده‌است.

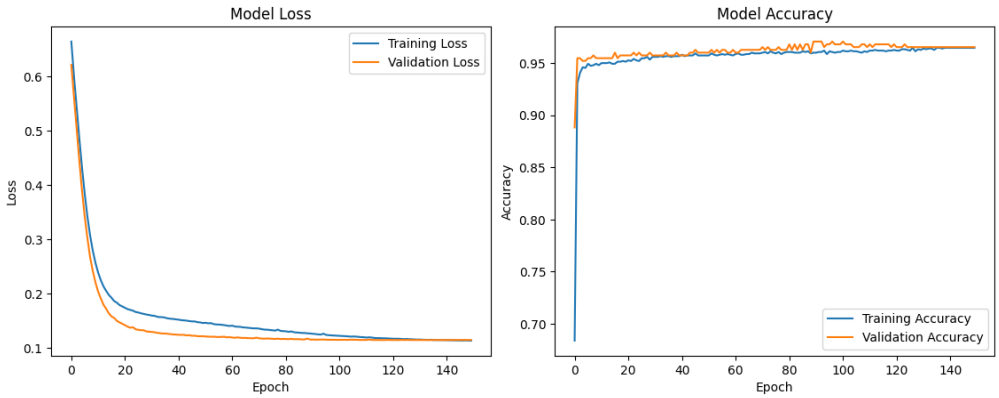
MLP

در این روش بیشتر از قرار دادن لایه های مختلف، بر روی فرایند آموزش توجه شده. کد در این جا به صورت زیر است.

```
model = Sequential()
model.add(Dense(288, input_shape=descriptor_shape, activation='elu'))
model.add(Dense(cls_number, activation='softmax'))

class_weights = compute_class_weight(
                                        class_weight = "balanced",
                                        classes = np.unique(y),
                                        y = y
                                    )
class_weights = dict(zip(np.unique(y), class_weights))

reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.9,
                              patience=3,
                              min_lr=0.000001, verbose=1)

from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=30, verbose=1,
mode='min', restore_best_weights=True)

opt = keras.optimizers.Adam(learning_rate=0.0001)

model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_zm, Y_train_zm,
```

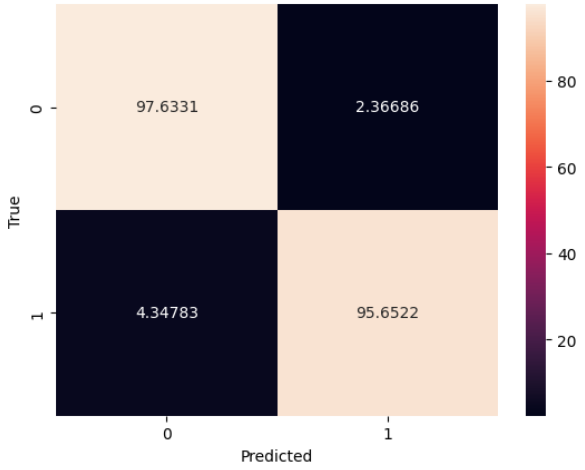
```
batch_size=32,
epochs=150,
validation_data=(X_test_zm, Y_test_zm),
callbacks=[reduce_lr, early_stopping],
class_weight=class_weights
)
```

مدل شامل دو لایه Dense با توابع فعال‌سازی elu و softmax است. وزن‌های کلاس برای متعادل‌سازی داده‌های نابرابر محاسبه شده و بازگشت‌های ReduceLROnPlateau و EarlyStopping برای بهبود فرآیند آموزش و جلوگیری از بیش‌برازش استفاده می‌شوند. مدل با بهینه‌ساز Adam و تابع زیان categorical_crossentropy کامپایل شده و سپس با داده‌های آموزشی آموزش داده می‌شود. نمودارهای آموزش مطابق شکل ۱۰ است:



شکل ۱۰- نمودارهای دقت و زیان در حین آموزش مدل **MLP**

مشخصا هیچ مشکل خاصی در طی آموزش دیده نمی‌شود. ماتریس درهم‌ریختگی به صورت درصدی مطابق شکل ۱۱ است.



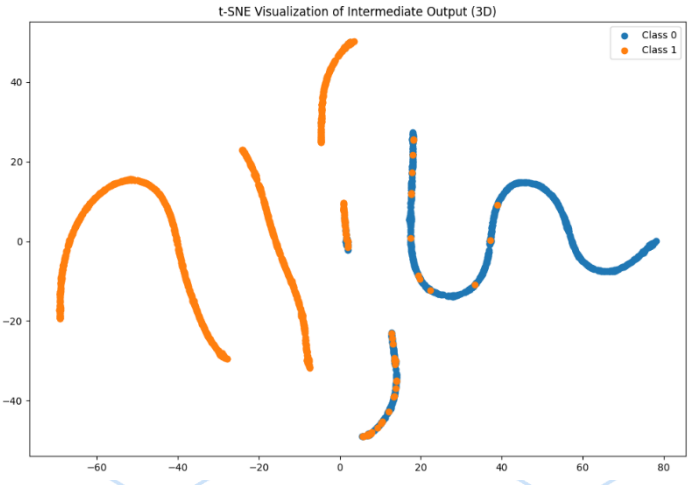
شکل ۱۱- ماتریس درهم‌ریختگی مدل **MLP**

معیارهای بررسی نیز به صورت زیر بدست آمده‌اند.

	precision	recall	f1-score	support
0	0.95	0.98	0.96	169
1	0.98	0.96	0.97	207
accuracy			0.97	376
macro avg	0.96	0.97	0.97	376
weighted avg	0.97	0.97	0.97	376

```
Test AUC: 0.9664265500385902
Test Recall: 0.9654255319148937
Test F1-score: 0.9654255319148937
Test Precision: 0.9654255319148937
```

بر اساس نتایج ماتریس درهم‌ریختگی و معیارهای ارزیابی، مدل دارای عملکرد بسیار خوبی در تشخیص و دسته‌بندی نمونه‌ها است. دقت، بازخوانی، و F1-Score بالا برای هر دو کلاس نشان‌دهنده توانایی مدل در پیش‌بینی دقیق و صحیح برچسب‌ها است. معیار AUC نیز نشان‌دهنده توانایی مدل در تمایز بین کلاس‌ها به طور کلی است. داده‌های آموزش دیده مطابق شکل ۱۲ با t-SNE دیده می‌شود.



شکل ۱۲- داده‌های خروجی با **t-SNE** در مدل **MLP**

این تصویر نیز تأیید می‌کند که مدل دارای عملکرد خوبی در تفکیک و تشخیص کلاس‌ها است و به درستی قادر به یادگیری ویژگی‌های متمایز کننده کلاس‌ها می‌باشد.

استفاده از تابع زیان Renyi

آنتروپی و واگرایی Renyi معیارهایی هستند که از تئوری اطلاعات برای اندازه‌گیری تفاوت بین توزیع‌های احتمال استفاده می‌کنند. این روش می‌تواند به مدل کمک کند تا بهتر و سریع‌تر یاد بگیرد و عملکرد بهتری داشته باشد. به صورت کلی می‌توان گفت که این روش، حالت کلی‌تری از تابع زیان categorical_crossentropy است.

آنتروپی Renyi یک تعمیم از آنتروپی شانون است که با یک پارامتر α تنظیم می‌شود. این آنتروپی اطلاعاتی درباره عدم قطعیت یک توزیع احتمال ارائه می‌دهد.

فرمول محاسبه آنتروپی Renyi به شکل زیر است:

$$H_\alpha(P) = \frac{1}{1-\alpha}\log\sum_i P(i)^\alpha$$

واگرایی Renyi یک معیار برای اندازه‌گیری تفاوت بین دو توزیع احتمال است. این واگرایی نیز با یک پارامتر α تنظیم می‌شود.

فرمول محاسبه واگرایی Renyi به شکل زیر است:

$$D_\alpha(P|Q) = \frac{1}{\alpha-1}\log\sum_i \frac{P(i)^\alpha}{Q(i)^{\alpha-1}}$$

براین اساس، کد تابع زیان به صورت زیر تعریف می‌شود:

```
def renyi_entropy(alpha, y_true, y_pred):
    """Calculates the Renyi entropy of the predicted distribution"""
    term = tf.reduce_sum(tf.pow(y_pred, alpha), axis=-1)
    renyi_ent = (1.0 / (1.0 - alpha)) * tf.math.log(term)
    return renyi_ent

def renyi_divergence(alpha, y_true, y_pred):
    """Calculates the Renyi divergence between true and predicted distributions"""
    p = y_true
    q = y_pred
    p_alpha = tf.pow(p, alpha)
    q_alpha = tf.pow(q, alpha - 1)
    term = tf.reduce_sum(p_alpha / q_alpha, axis=-1)
    renyi_div = (1.0 / (alpha - 1.0)) * tf.math.log(term)
    return renyi_div
```

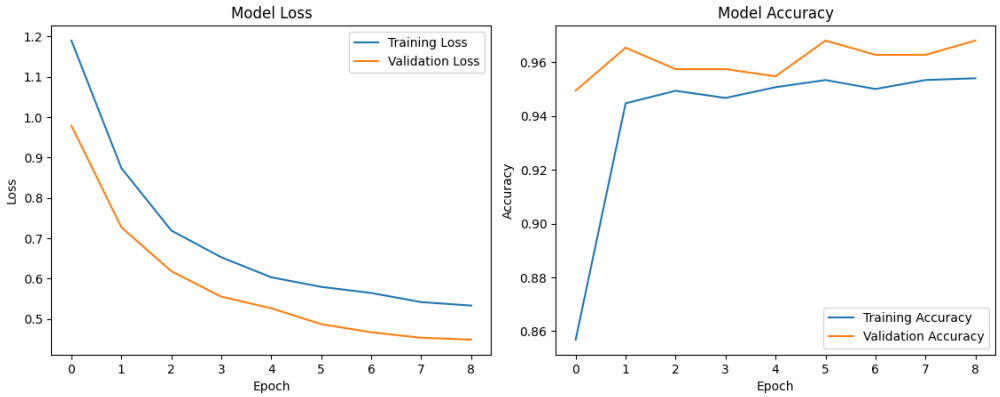
```
def custom_loss(alpha):
    def loss(y_true, y_pred):
        entropy_loss = renyi_entropy(alpha, y_true, y_pred)
        divergence_loss = renyi_divergence(alpha, y_true, y_pred)
        return entropy_loss + divergence_loss
    return loss
```

با تنظیم پارامتر alpha، می‌توان تعادل بین حساسیت به اختلافات کوچک و بزرگ بین توزیع‌های احتمال را کنترل کرد. با تنظیم این پارامتر روی مقدار ۰.۱۵، کد زیر برای آموزش پیاده سازی شده‌است:

```
model_1 = Sequential()
model_1.add(Dense(288, input_shape=descriptor_shape, activation='elu'))
model_1.add(Dense(cls_number, activation='softmax'))

class_weights = compute_class_weight(
                                        class_weight = "balanced",
                                        classes = np.unique(y),
                                        y = y
                                    )
class_weights = dict(zip(np.unique(y), class_weights))

reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.9,
                              patience=3,
                              min_lr=0.000001, verbose=1)

from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=2, verbose=1,
mode='min', restore_best_weights=True)
opt = keras.optimizers.Adam(learning_rate=0.0007)

model_1.compile(optimizer=opt,
                loss=custom_loss(alpha),
                metrics=['accuracy'])

history = model_1.fit(X_train_zm, Y_train_zm,
                      batch_size=32,
                      epochs=9,
```

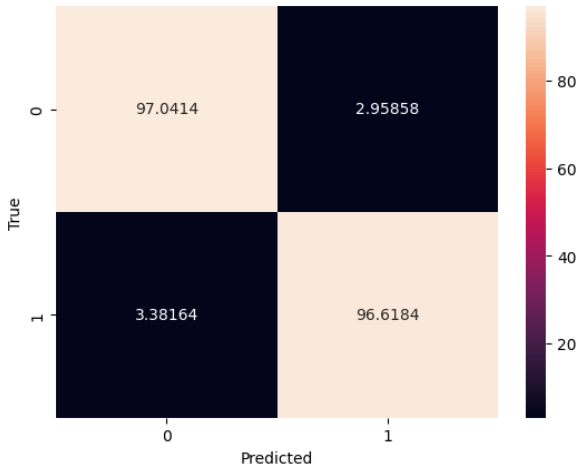
```
validation_data=(X_test_zm, Y_test_zm),
callbacks=[reduce_lr, early_stopping],
class_weight=class_weights
)
```

نمودارهای مربوط به آموزش مطابق شکل ۱۳ است:



شکل ۱۳- نمودار دقت و زیان در مدل **MLP** با تابع زیان **Renyi**

مشخصا هیچ مشکل خاصی در طی آموزش دیده نمی‌شود. ماتریس درهم‌ریختگی به صورت درصدی مطابق شکل ۱۳ است.



شکل ۱۴- ماتریس درهم‌ریختگی مدل **MLP** با تابع زیان **Renyi**

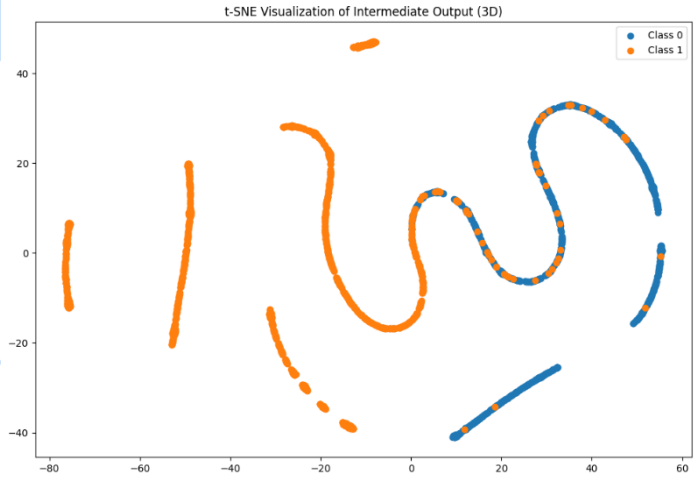
معیارهای بررسی نیز به صورت زیر بدست آمده‌اند.

	precision	recall	f1-score	support
0	0.96	0.97	0.96	169
1	0.98	0.97	0.97	207

```
    accuracy                           0.97       376
   macro avg       0.97      0.97      0.97       376
weighted avg       0.97      0.97      0.97       376

Test AUC: 0.9682988880313295
Test Recall: 0.9680851063829787
Test F1-score: 0.9680851063829787
Test Precision: 0.9680851063829787
```

بر اساس نتایج ماتریس درهم‌ریختگی و معیارهای ارزیابی، مدل دارای عملکرد بسیار خوبی در تشخیص و دسته‌بندی نمونه‌ها است. استفاده از تابع هزینه Renyi باعث افزایش سرعت آموزش و بهبود عملکرد مدل در تشخیص نمونه‌های کلاس ۱ شده است. این بهبود در معیارهای بازخوانی، F1-Score و AUC قابل مشاهده است. نتایج نشان می‌دهد که تابع هزینه Renyi می‌تواند به عنوان یک ابزار موثر برای بهبود یادگیری مدل‌ها در کاربردهای مختلف مورد استفاده قرار گیرد

داده‌های آموزش دیده مطابق شکل ۱۵ با t-SNE دیده می‌شود.



شکل ۱۵- داده‌های خروجی با **t-SNE** در مدل **MLP** با تابع زیان **Renyi**

این تصویر نیز تأیید می‌کند که مدل دارای عملکرد خوبی در تفکیک و تشخیص کلاس‌ها است و به درستی قادر به یادگیری ویژگی‌های متمایز کننده کلاس‌ها می‌باشد.

SVM

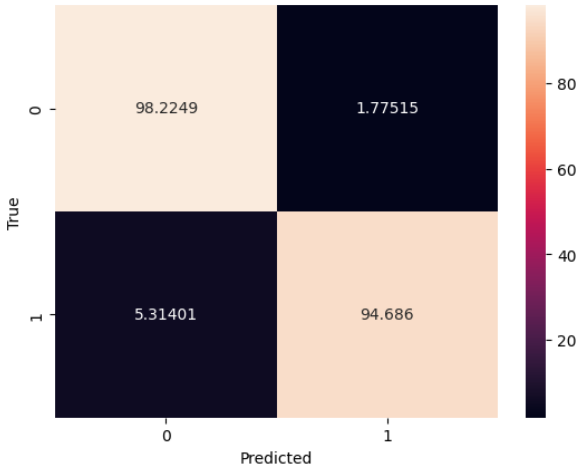
در ابتدا، لازم به ذکر است که هایپرپارامترهای مناسب در این روش باید اول پیدا شود تا بتوان یک طبقه بندی مناسب را داشت؛ درنتیجه نیاز است ابتدا با یک GridSearchCV بهترین هایپرپارامترها را انتخاب کنیم و پس از آن مدل را با آن پارمترها آموزش دهیم. کد این روش به صورت زیر است:

```
# Define parameter grid for GridSearchCV
param_grid = {
    'C': [8.25, 8.5, 8.75, 9],
    'gamma': ['scale'],
    'kernel': ['linear', 'rbf', 'poly']
}

from sklearn.utils.class_weight import compute_class_weight
class_weights = compute_class_weight(
                                        class_weight = "balanced",
                                        classes = np.unique(y),
                                        y = y
                                    )
class_weights = dict(zip(np.unique(y), class_weights))

# Convert one-hot encoded labels to single labels
Y_train_zm_single = np.argmax(Y_train_zm, axis=1)
Y_test_zm_single = np.argmax(Y_test_zm, axis=1)

# Initialize the SVM model with GridSearchCV
svm_model = GridSearchCV(SVC(class_weight=class_weights,
probability=True), param_grid, refit=True, verbose=3)

svm_model.fit(X_train_zm, Y_train_zm_single)
```

در ابتدا، با سعی و خطا، از بازه ۰.۱ تا ۱۰۰، حدود کلی پارامتر c (مجازات) به حالت بالا محدود شد. کرنل sigmoid هم که در ابتدای فرایند بود، به دلیل ضعف کنار گذاشته شد. بهترین هایپرپارامترها به صورت زیر بدست آمدند:

```
Best Parameters: {'C': 8.75, 'gamma': 'scale', 'kernel': 'linear'}
```

ماتریس درهم‌ریختگی به صورت درصدی مطابق شکل ۱۶ است.



شکل ۱۶- ماتریس درهم‌ریختگی مدل **SVM**

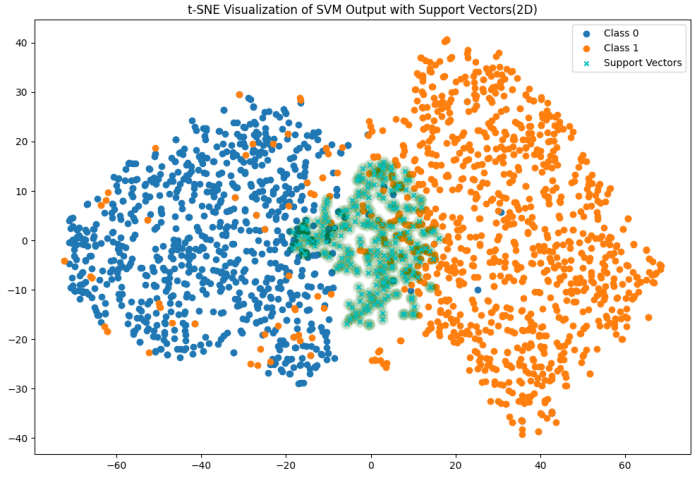
معیارهای بررسی نیز به صورت زیر بدست آمده‌اند.

	precision	recall	f1-score	support
0	0.94	0.98	0.96	169
1	0.98	0.95	0.97	207
accuracy			0.96	376
macro avg	0.96	0.96	0.96	376
weighted avg	0.96	0.96	0.96	376

```
Test AUC: 0.9645542120458508
Test Recall: 0.9627659574468085
Test F1-score: 0.9627659574468085
Test Precision: 0.9627659574468085
```

بر اساس نتایج ماتریس درهم‌ریختگی و معیارهای ارزیابی، مدل دارای عملکرد بسیار خوبی در تشخیص و دسته‌بندی نمونه‌ها است. مدل SVM دقت و بازخوانی کمتری در کلاس ۰ دارد، اما همچنان عملکرد قابل قبولی دارد.

بردارهای پشتیبان بدست آمده مطابق شکل ۱۷ با t-SNE در فضای ویژگی‌ها دیده می‌شود. مرز بین کلاس‌ها توسط بردارهای پشتیبان (نقاط سبز) مشخص شده است. این بردارهای پشتیبان نقاطی هستند که نزدیک به مرز تصمیم‌گیری مدل قرار دارند و تاثیر زیادی در تعیین آن دارند.



شکل ۱۷- ساپورت وکتورهای بدست آمده با **t-SNE** در مدل **SVM**

این تصویر نیز تأیید می‌کند که مدل دارای عملکرد خوبی در تفکیک و تشخیص کلاس‌ها است و به درستی قادر به یادگیری ویژگی‌های متمایز کننده کلاس‌ها می‌باشد. حضور نقاط نارنجی در خوشه آبی و بالعکس نشان‌دهنده مواردی است که مدل ممکن است در تفکیک آن‌ها دچار چالش شده باشد، اما به طور کلی، تعداد این نقاط کم است که نشان‌دهنده عملکرد قابل قبول مدل است.

Decision Tree

برای این روش نیز، نیاز است تا ابتدا هایپرپارمترهای متناسب را پیدا کرد. کد زیر پیاده شده‌است:

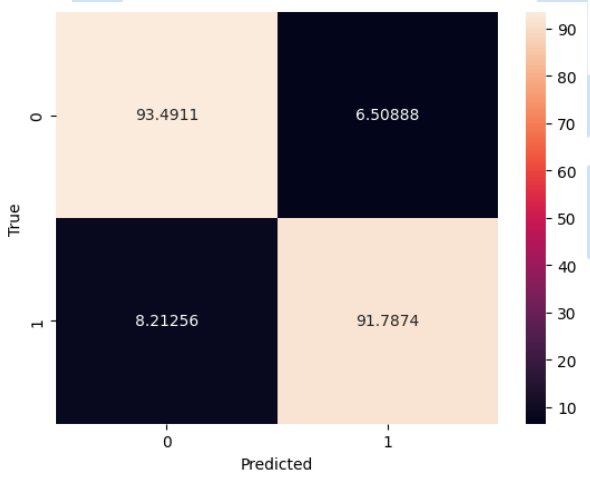
```
class_weights_list = []
for i in range(Y_train_zm.shape[1]):
    class_weights = compute_class_weight(
        class_weight='balanced',
        classes=np.unique(Y_train_zm[:, i]),
        y=Y_train_zm[:, i]
    )
    class_weights_dict = dict(zip(np.unique(Y_train_zm[:, i]),
class_weights))
    class_weights_list.append(class_weights_dict)

model_dt = DecisionTreeClassifier(class_weight=class_weights_list)

param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30],
```

```
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(estimator=model_dt, param_grid=param_grid, cv=3,
n_jobs=-1, verbose=2)
```

پارامترهایی مانند معیار ارزیابی تقسیمات، استراتژی انتخاب تقسیمات، حداکثر عمق درخت، حداقل نمونه‌های مورد نیاز برای تقسیم یک گره داخلی و حداقل نمونه‌های مورد نیاز برای نگه داشتن یک برگ بررسی شدند که بهترین آن‌ها به صورت زیر بدست آمد:

```
Best Parameters: {'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}
```

ماتریس درهم‌ریختگی به صورت درصدی مطابق شکل ۱۸ است.



شکل ۱۸- ماتریس درهم‌ریختگی مدل **DT**

معیارهای بررسی نیز به صورت زیر بدست آمده‌اند.

	precision	recall	f1-score	support
0	0.90	0.93	0.92	169
1	0.95	0.92	0.93	207
accuracy			0.93	376
macro avg	0.92	0.93	0.92	376
weighted avg	0.93	0.93	0.93	376

```
Test AUC: 0.926392819369408
Test Recall: 0.925531914893617
```

```
Test F1-score: 0.925531914893617
Test Precision: 0.925531914893617
```

مدل درخت تصمیم‌گیری دقت و بازخوانی کمتری در هر دو کلاس دارد و AUC آن نیز کمتر از سایر مدل‌هاست. برای انتخاب بهترین مدل، باید به تعادل بین دقت و بازخوانی توجه کرد و همچنین به کاربرد خاص و نیازمندی‌های پروژه توجه نمود.

Random Forest

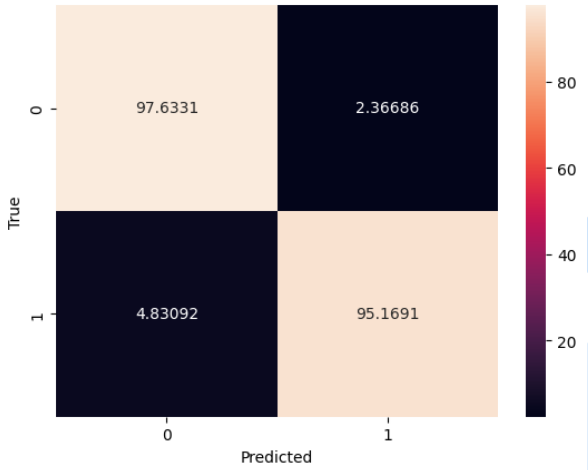
برای این روش نیز، نیاز است تا ابتدا هایپرپارمترهای متناسب را پیدا کرد. کد زیر پیاده شده‌است:

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

class_weights_list = []
for i in range(Y_train_zm.shape[1]):
    class_weights = compute_class_weight(
        class_weight='balanced',
        classes=np.unique(Y_train_zm[:, i]),
        y=Y_train_zm[:, i]
    )
    class_weights_dict = dict(zip(np.unique(Y_train_zm[:, i]),
class_weights))
    class_weights_list.append(class_weights_dict)

model_rf = RandomForestClassifier(class_weight=class_weights_list)

grid_search = GridSearchCV(estimator=model_rf, param_grid=param_grid,
cv=3, n_jobs=-1, verbose=2)
```

یک شبکه پارامتر برای جستجوی GridSearchCV تعریف می‌شود که شامل پارامترهایی مانند تعداد درخت‌ها (n_estimators)، حداکثر عمق درخت (max_depth)، حداقل نمونه‌های مورد نیاز برای تقسیم یک گره داخلی (min_samples_split) و حداقل نمونه‌های مورد نیاز برای نگه داشتن یک برگ (min_samples_leaf) است. بهترین آن‌ها به صورت زیر بدست آمد:

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split':
10, 'n_estimators': 100}
```

ماتریس درهم‌ریختگی به صورت درصدی مطابق شکل ۱۹ است.



شکل ۱۹- ماتریس درهم‌ریختگی مدل **RF**

معیارهای بررسی نیز به صورت زیر بدست آمده‌اند.

	precision	recall	f1-score	support
0	0.94	0.98	0.96	169
1	0.98	0.95	0.97	207
accuracy			0.96	376
macro avg	0.96	0.96	0.96	376
weighted avg	0.96	0.96	0.96	376

```
Test AUC: 0.964011091101392
Test Recall: 0.9627659574468085
Test F1-score: 0.9627659574468085
Test Precision: 0.9627659574468085
```

بر اساس نتایج ماتریس درهم‌ریختگی و معیارهای ارزیابی، مدل دارای عملکرد بسیار خوبی در تشخیص و دسته‌بندی نمونه‌ها است. مدل جنگل تصادفی تقریباً مشابه مدل MLP بدون Renyi Loss عمل می‌کند.

جمع‌بندی

این پروژه با هدف تشخیص بیماری قوزقرنیه با کمک قابلیت ممان‌های زرنیکه به عنوان یک تکنیک استخراج ویژگی برای طبقه‌بندی تصاویر طراحی شد. با استفاده از یک مجموعه‌داده جامع و پیاده‌سازی یک خط‌مشی پیش‌پردازش دقیق، ویژگی‌های زرنیکه از تصاویر استخراج شد.

پس از استخراج ویژگی‌ها از داده و بررسی داده‌ها در شبکه، می‌توان گفت مدل MLP با Renyi Loss بهترین عملکرد را دارد، با کاهش خطاهای مثبت و منفی نادرست و بالاترین مقدار AUC (۰.۹۶۸) را دارد. پس از آن، مدل جنگل تصادفی عملکرد بسیار خوبی دارد و مشابه مدل اولیه بدون Renyi Loss عمل می‌کند. مدل SVM عملکرد قابل قبولی دارد، اما در بازخوانی کلاس ۱ کمی ضعیف‌تر عمل کرده است. مدل درخت تصمیم‌گیری عملکرد ضعیف‌تری نسبت به سایر مدل‌ها دارد، با دقت و بازخوانی پایین‌تر و تعداد بالای خطاهای مثبت و منفی نادرست.

در حین پردازش داده‌ها، معیارهای عملکردی مختلفی نیز مورد بررسی قرار گرفتند که در جدول زیر قابل مشاهده می‌باشند. اولین معیار، معیار زمان صرف شده برای اجرای هر شبکه است. معیار دوم ، تکرار پذیری هر شبکه است؛ بدین صورت که در هر تکرار به صورت نسبی چقدر پاسخ ها به هم نزدیک‌تر بوده‌است که به صورت رتبه بندی نشان داده شده‌است. معیار سوم، معیار زمان صرف شده برای هر روش یک معیار نسبی برای بررسی میزان زمان صرف شده برای توسعه هر شبکه برای رسیدن به بهترین پاسخ خود می‌باشد که به صورت رتبه بندی نشان داده شده‌است. معیار آخر نیز میزان نسبی سادگی شبکه برای رسیدن به بهترین پاسخ خود می‌باشد.

جدول ۲- مقایسه معیارهای عددی و نسبی هر شبکه

	Duration(s)	Repeatability	Time spent on development	Network simplicity
MLP	41.90	4	2	2
MLP with Renyi loss	3.29	5	1	1
SVM	19.74	1	5	3
DT	71.21	1	3	4
RF	417.41	1	3	4

نتیجه‌گیری کلی براساس موارد بالا به صورت زیر است:

۱. **MLP با Renyi Loss:**

- **مزایا :**سریع‌ترین زمان اجرا، بالاترین تکرارپذیری، کمترین زمان توسعه و ساده‌ترین شبکه.
- **مناسب برای :**کاربردهایی که نیاز به دقت بالا، زمان اجرای سریع و سادگی توسعه دارند.

۲. **MLP:**

- **مزایا :**زمان اجرای مناسب، تکرارپذیری خوب، زمان توسعه مناسب و سادگی شبکه خوب.
- **مناسب برای :**کاربردهایی که نیاز به تعادل بین دقت و سادگی توسعه دارند.

۳. **SVM:**

- **مزایا :**زمان اجرای نسبتاً خوب.
- **معایب :**پایین‌ترین تکرارپذیری و بیشترین زمان توسعه.
- **مناسب برای :**کاربردهایی که نیاز به پیچیدگی بیشتر و توانایی تنظیم دقیق پارامترها دارند.

۴. **درخت تصمیم‌گیری:(DT)**

- **مزایا :**سادگی نسبی در مقایسه با جنگل تصادفی.
- **معایب :**زمان اجرای نسبتاً بالا و تکرارپذیری پایین.
- **مناسب برای :**کاربردهایی که نیاز به تصمیم‌گیری سریع با پیچیدگی کم دارند.

۵. **جنگل تصادفی:(RF)**

- **مزایا :**قدرت پیش‌بینی بالا در شرایط پیچیده.
- **معایب :**بیشترین زمان اجرا و تکرارپذیری پایین.
- **مناسب برای :**کاربردهایی که نیاز به دقت بالا و توانایی پردازش داده‌های بزرگ دارند.

درکل ویژگی‌های استخراج‌شده به عنوان ورودی‌های مؤثر برای مدل‌های یادگیری ماشین ثابت شدند، به ویژه مدل MLP با استفاده از تابع زیان Renyi که عملکرد بهینه‌ای از نظر دقت طبقه‌بندی، سرعت اجرا، تکرارپذیری و سادگی ساختار شبکه نشان داد.

به طور خلاصه، یک روش سیستماتیک و مؤثر برای استخراج ویژگی‌ها و طبقه‌بندی تصاویر ارائه شده است که سهم قابل توجهی در زمینه بینایی کامپیوتری دارد. موفقیت لحظات زرنیکه در این زمینه نشان‌دهنده پتانسیل آن‌ها برای کاربردهای گسترده‌تر در تحلیل تصاویر و وظایف مرتبط دیگر است. کارهای آینده می‌تواند با بررسی تکنیک‌های یادگیری عمیق و بهینه‌سازی بیشتر فرایند استخراج ویژگی، این یافته‌ها را گسترش دهد و عملکرد را بهبود بخشد .

پیوست

کدهای مربوط به پروژه در گیت‌هاب در ریپازیتوری زیر که برای کل ترم این درس هست، قرار داده شده‌است:

https://github.com/shimanaseri/ML-course

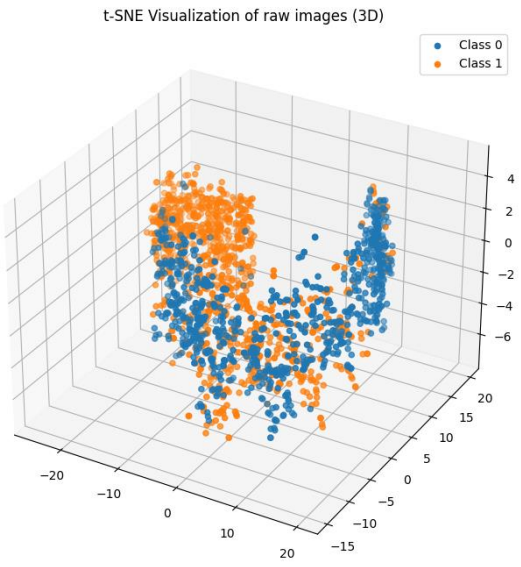
کدهای مرتبط با پیش‌پردازش و استخراج ویژگی در گیت‌هاب به دلیل استفاده و توسعه آن در پروژه‌های آزمایشگاهی غیرقابل دسترسی هستند.

علاوه بر آن، لینک زیر نیز برای اجرای کد در نظر گرفته شده‌است:

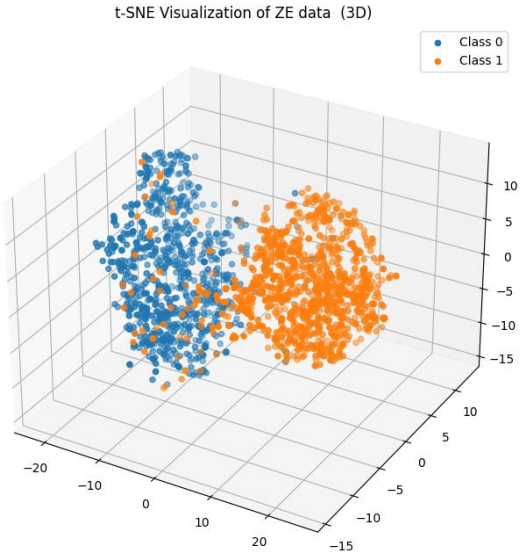
https://colab.research.google.com/drive/1x8nZYjWJiW3_ifpN0OVFBk0uHE6YRbk8?usp=sharing

لازم به ذکر است که پیش‌پردازش و استخراج ویژگی روی داده‌ها از پیش انجام شده و نتایج آن ذخیره شدند تا برای بررسی‌های آتی نیاز به تکرار نباشد. در کد این موارد دانلود می‌شوند و نیازی به اجرای کدهای مربوط به پیش‌پردازش و استخراج ویژگی نیست.

خروجی سه بعدی t-SNE داده‌های خام به صورت زیر است:



خروجی سه بعدی t-SNE ویژگی‌های استخراج شده نیز به صورت زیر است:



برای بررسی داده‌های بدست آمده از مدل MLP، به دلیل اینکه در نهایت خروجی دارای ۲ ویژگی بیانگر احتمال هر کلاس است، امکان رسم سه بعدی برای آن‌ها وجود ندارد.

مراجع

https://onlinelibrary.wiley.com/doi/abs/10.1002/ima.22717

https://www.researchgate.net/publication/349850933_Generalization_of_Cross-Entropy_Loss_Function_for_Image_Classification