دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

گزارش پروژه کارشناسی

ساخت سیستم هوشمند خوشه بندی تصاویر افراد

نگارش

محمد مظفری

استاد راهنما

دکتر رضا صفابخش

شهریور ۱۴۰۰

به نام خدا

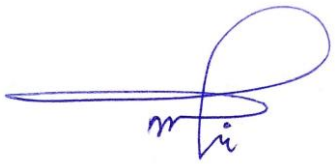
تاریخ: ۳ شهریور ۱۴۰۰

اینجانب محمد مظفری متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

محمد مظفری



تقدیم به

پدر و مادر عزیز و مهربانم

که در سختی‌ها و دشواری‌های زندگی، همواره یاوری دلسوز و فداکار

و پشتیبانی محکم و مطمئن برایم بوده‌اند.

تقدیر و تشکر

از تمامی معلمان و اساتیدی که توفیق دانش آموزی و دانشجویی در محضرشان را داشتم سپاسگزارم.

از دوستان عزیزم که همراهان همیشگی من بوده‌اند و اوقات خوشی را در کنار هم سپری کرده‌ایم، تقدیر و تشکر دارم.

و در پایان از تمامی عزیزانی که در طول انجام این پروژه مرا یاری کرده‌اند کمال تشکر و قدردانی را ابراز می‌نمایم.

چکیده

هدف این پروژه ساخت یک سیستم هوشمند است که عمل خوشه‌بندی را بر روی تصاویر چهره انجام می‌دهد. خوشه‌بندی چهره در گوشی‌های موبایل برای دسته‌بندی تصاویر گالری کاربرد دارد. همچنین از چنین سیستمی در بررسی ویدیوهای دوربین‌های مدار بسته نیز می‌توان استفاده کرد. این سیستم در قالب یک نرم افزار کاربردی پیاده سازی می‌شود. این نرم افزار دارای رابط گرافیکی مناسب برای ارتباط با کاربر است و از چند بخش اصلی تشکیل می‌شود. بخش اول سیستم مربوط به دریافت ورودی است. کاربر پوشه‌ای را که در آن تعدادی تصویر وجود دارد مشخص می‌کند و به برنامه می‌دهد. برنامه تمام تصویرهای موجود در این پوشه را پیدا کرده و به کاربر نشان می‌دهد. در قسمت بعد برنامه باید بر روی هر کدام از تصاویر پردازش‌هایی را انجام داده و قسمت چهره‌ها را مشخص کند. این کار با الگوریتم‌های متفاوتی می‌تواند انجام گردد که در این پروژه از الگوریتم MTCNN برای این کار استفاده شده است. پس از این مرحله با استفاده از یک الگوریتم مناسب یک بازنمایی برای هر چهره بدست آورده می‌شود. این بازنمایی ویژگی‌های چهره را در خود دارد و برای خوشه‌بندی مورد استفاده قرار می‌گیرد. بخش بعدی پروژه مربوط به پیاده‌سازی یک الگوریتم خوشه‌بندی مناسب است تا مشخص شود کدام تصاویر مربوط به افراد یکسانی هستند به این ترتیب تصاویر مربوط به افراد مختلف در خوشه‌های مختلف قرار می‌گیرند. در آخر و پس از مشخص شدن خوشه‌ها برنامه این امکان را به کاربر می‌دهد تا تصاویر مربوط به افراد مختلف را در پوشه‌های مختلف ذخیره کند.

واژه‌های کلیدی:

خوشه‌بندی چهره، تشخیص چهره، شبکه‌های عصبی، یادگیری عمیق، شبکه کانولوشنی گرافی

فهرست مطالب

صفحه

فهرست اشکال

صفحه

فهرست جداول

صفحه

فصل اول مقدمه

فصل ۱ مقدمه

با پیشرفت تکنولوژی روز به روز تعداد دوربین‌های اطراف ما بیشتر می‌شود و در نتیجه با افزایش تعداد دوربین‌ها حجم زیادی از داده‌های تصویری تولید می‌شود. برای مثال دوربین‌های مدار بسته - که استفاده از آن‌ها یکی از مهم‌ترین اقدامات امنیتی است - روزانه ساعت‌ها ویدیو ضبط می‌کنند که تصاویر افراد زیادی در آن‌ها دیده می‌شود. یکی از مهم‌ترین پردازش‌ها بر روی این ویدیوها اینست که هویت افراد مختلف دیده شده در تصاویر شناسایی شود. به این کار شناسایی چهره[1] گفته می‌شود. واضح است که انجام این کار به صورت دستی بسیار وقت‌گیر و پرهزینه است. به همین دلیل توسعه سیستم‌های هوشمند که توانایی انجام چنین پردازش‌هایی را دارند بسیار ارزشمند است.

ساخت سیستم‌هایی که قابلیت شناسایی هویت افراد از روی چهره را دارند نیاز به حجم زیادی از تصاویر چهره دارد که برچسب گذاری شده باشند. از این رو مسئله شناسایی چهره، یک مسئله نظارت شده[2] است و جمع‌آوری داده برای این مسائل هزینه‌بر است. بنابراین به جای ساخت یک سیستم شناسایی چهره می‌توان سیستمی ساخت که خوشه‌بندی چهره‌ها[3] را به صورت خودکار انجام دهد.

خوشه‌بندی در حالت کلی به این معنی است که تعدادی از اشیا را در گروه‌های مختلف تقسیم کنیم، به گونه‌ای که اشیای موجود در یک گروه ویژگی‌های یکسان یا نزدیک به هم داشته باشند. بنابراین منظور از خوشه‌بندی چهره این است که بر روی تعداد زیادی از تصاویر چهره افراد مختلف پردازش شود و این تصاویر در گروه‌های مختلف تقسیم شوند بگونه‌ای که هر گروه مربوط به تصاویر یک فرد خاص باشد. حل این مسئله نیازمند نوعی الگوریتم هوشمند است، زیرا تصاویر می‌توانند تغییرات زیادی داشته باشند اما هم‌چنان مربوط به یک شخص باشند، تغییراتی نظیر نور محیط، حالت چهره، سن و غیره. مسئله خوشه‌بندی چهره یک مسئله بدون نظارت[4] است و از این جهت جمع‌آوری داده‌های آموزشی برای آن آسان‌تر است.

---

[1] Face Recognition

[2] Supervised

[3] Face Clustering

[4] Unsupervised

خوشه‌بندی چهره کاربردهای زیادی در زمینه‌های مختلف دارد. برای مثال می‌توان از چنین سیستمی در گوشی‌های موبایل استفاده کرد تا تصاویر گالری افراد را با توجه به اشخاص موجود در آن‌ها دسته‌بندی کرد. همچنین از این سیستم می‌توان در دوربین‌های مداربسته استفاده کرد تا افراد مختلفی که در منطقه‌ای خاص تردد کرده‌اند را تشخیص داد و سپس به صورت دستی هویت آن‌ها را شناسایی کرد به این ترتیب دیگر نیازی به مشاهده و بررسی تمام ویدیوهای ضبط شده نیست.

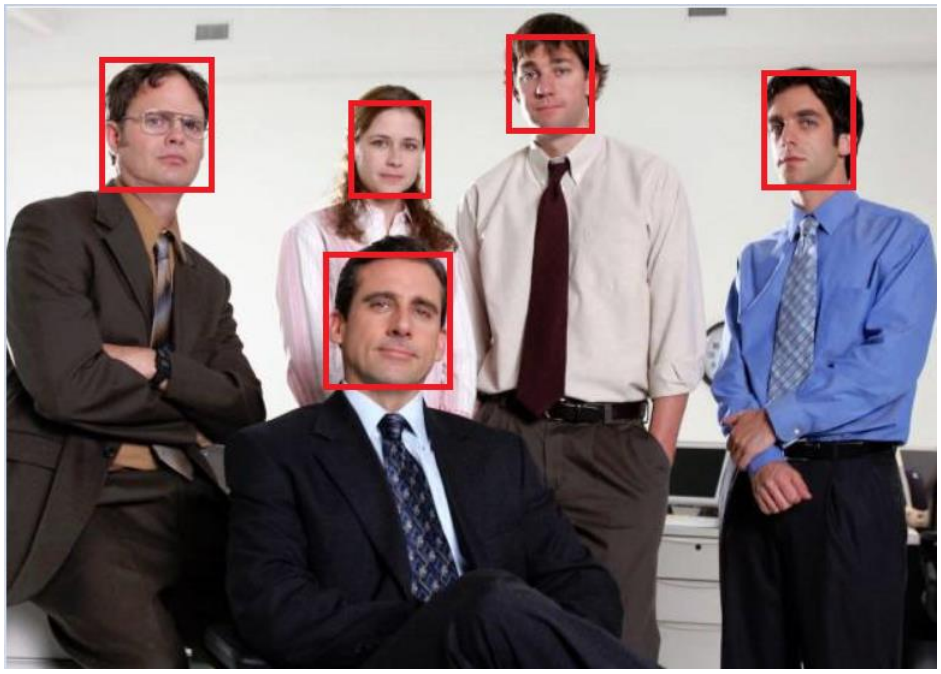
هدف این پروژه ساخت یک برنامه است که به کمک آن بتوانیم تعدادی تصویر را بر اساس افراد مختلف موجود در آنها دسته‌بندی کنیم. برای مثال فرض کنید دو تصویر شکل ۱-۱ به برنامه داده شوند. در این دو تصویر ۵ فرد مختلف وجود دارند. آنچه در نهایت انتظار داریم این است که برنامه تصویر افراد مختلف را در پوشه‌های مختلف ذخیره کند.

شکل ۱-۱: دو تصویر از ۵ فرد مختلف

برای انجام چنین کاری ۳ مرحله اصلی وجود دارد که هر کدام یک مسئله است که باید حل شود.

۱- در قدم اول باید قسمت‌های مربوط به چهره را در تصاویر مختلف تشخیص دهیم و جدا کنیم. به این کار تشخیص چهره[1] گفته می‌شود در شکل ۱-۲ می‌بینیم که قسمت‌های مربوط به چهره در تصویر تشخیص داده شده‌اند و با کادر قرمز رنگ مشخص شده‌اند. در فصل ۲ به بررسی این مسئله و راه‌حل پیشنهادی برای حل آن می‌پردازیم.

[1] Face Detection



شکل ۱-۲: تشخیص چهره

۲- پس از اینکه قسمت چهره را در تصاویر یافتیم باید یک بازنمایی[1] از این تصاویر بدست آوریم. هدف از این بازنمایی این است که هر تصویر با یک بردار چگال[2] نمایش داده شود و این بردارها بگونه‌ای باشند که بازنمایی‌های مربوط به افراد یکسان فاصله کم و بازنمایی‌های مربوط به افراد مختلف فاصله زیادی داشته باشند. اگر بتوانیم این کار را انجام دهیم آنگاه برای تشخیص یکی بودن دو تصویر می‌توانیم از فاصله برداری بازنمایی‌های آن‌ها استفاده کنیم. در فصل ۳ به بررسی بیشتر این مسئله می‌پردازیم.

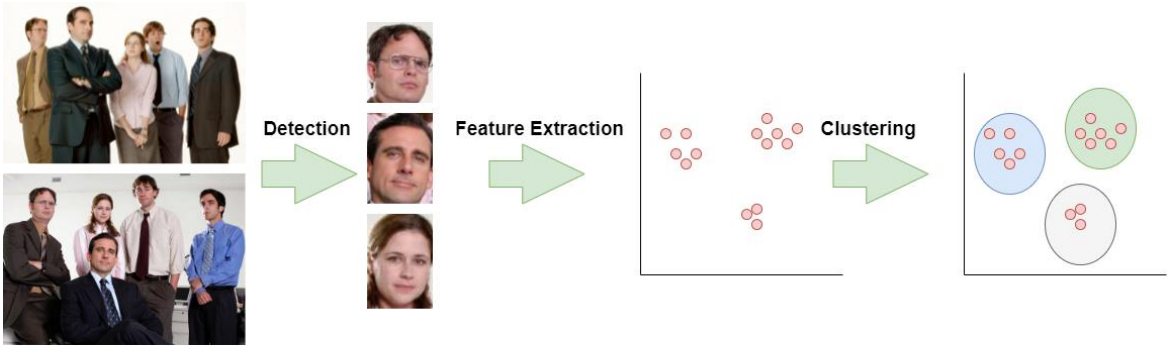
۳- هر بازنمایی برداری یک نقطه در فضای برداری با ابعاد مشخص است اگر بتوانیم این بازنمایی‌ها را در یک دستگاه مختصات رسم کنیم (در عمل نمی‌توانیم این کار را انجام دهیم زیرا ابعاد بردارها در مرتبه چند صد هستند و رسم آنها امکان‌پذیر نیست.) می‌بینیم که تصاویر مربوط به افراد مختلف نزدیک به هم و نسبت به سایر تصاویر فاصله دارند. هدف ما در مرحله سوم این است که

[1] Representation

[2] Dense

بر روی این نقاط عمل خوشه‌بندی را انجام دهیم تا بتوانیم تصاویر را دسته‌بندی کنیم. در فصل ۴ به روش‌های ممکن برای خوشه‌بندی می‌پردازیم و با این مسئله بیشتر آشنا می‌شویم.

شکل ۱-۳ چارچوب کلی برنامه را نشان می‌دهد.



شکل ۱-۳: چارچوب کلی آنچه در این پروژه انجام می‌شود.

تشخیص چهره: برای انجام تشخیص چهره در این پروژه از یک شبکه عصبی پیچشی با نام MTCNN[1] استفاده خواهیم کرد. این شبکه‌ی عصبی یک تصویر را از ورودی دریافت می‌کند و در خروجی مکان چهره‌ها و نشانه‌های صورت را مشخص می‌کند. پس از انجام این بخش تشخیص چهره را بر روی تعداد زیادی تصویر انجام می‌دهیم تا عملکرد این الگوریتم را از نظر سرعت ارزیابی کنیم.

استخراج ویژگی: برای این کار از یک شبکه‌ی عصبی عمیق به نام Inception-ResNet-v1[2] استفاده می‌کنیم. این شبکه عصبی برای یک مسئله دسته‌بندی چهره‌ها آموزش دیده است و از خروجی لایه‌ی یکی به آخر آن به عنوان بازنمایی استفاده می‌کنیم. برای ارزیابی کیفیت این ماژول دقت[1] و سطح زیر نمودار ROC[2] آن را برای مجموعه داده LFW محاسبه می‌کنیم.

خوشه‌بندی: الگوریتم‌های زیادی برای خوشه‌بندی موجود هستند اما تصاویر چهره ویژگی‌های خاصی دارند که استفاده از این روش‌ها ممکن است دقت پایینی داشته باشد. به همین منظور از یک الگوریتم

[1] Accuracy

[2] AUC (Area Under the Curve)

مبتنی بر شبکه‌های عصبی گرافی برای خوشه‌بندی استفاده می‌کنیم. برای ارزیابی نیز معیارهای Precision و Recall و F را گزارش می‌کنیم.

با توجه به آنچه در این فصل گفته شد در فصل ۲ به طور خلاصه پیشینه هرکدام از این موضوعات معرفی می‌شوند سپس در فصل ۳ به بررسی روش MTCNN برای تشخیص چهره می‌پردازیم. در فصل ۳ توضیح می‌دهیم که چگونه می‌توان با استفاده از شبکه عصبی Inception-ResNet-v1 برای چهره‌ها بازنمایی مناسب یافت. پس از آن در فصل ۴ با الگوریتم خوشه‌بندی تصاویر چهره آشنا می‌شویم. نهایتا در فصل ۵ نگاهی اجمالی به رابط کاربری گرافیکی ساخته شده برای برنامه خواهیم داشت.

فصل دوم مرور پیشینه

فصل ۲ مرور پیشینه

همانطور که در فصل قبل گفتیم برای ساختن یک سیستم خوشه‌بندی سه قسمت اصلی باید وجود داشته باشند. این سه قسمت عبارتند از:

۱- تشخیص چهره
۲- بازنمایی چهره
۳- خوشه‌بندی بازنمایی‌های بدست آمده

برای انجام هرکدام از این بخش‌ها کارهای زیادی انجام شده است که در ادامه به آنها می‌پردازیم.

تشخیص چهره: تشخیص چهره یک زیر مجموعه از مسئله تشخیص شی[1] است در مسئله تشخیص شی در یک تصویر تمام اشیا موجود پیدا می‌شوند در حالی که در تشخیص چهره اشیا همان چهره‌های صورت هستند. برای انجام این کار تاکنون چند رویکرد اصلی وجود داشته است[۳]:

۱- روش‌های مبتنی بر دانش[2]: این روش‌ها بر اساس مجموعه‌ای از قوانین نوشته می‌شوند. برای مثال اگر در تصویر چشم، بینی و دهان به فاصله مشخصی از هم وجود داشته باشند یک چهره را تشکیل می‌دهند. یکی از مشکلات استفاده از این روش‌ها این است که تبدیل دانش انسانی به مجموعه‌ای از قوانین کار سختی است. اگر قوانین خیلی جزیی باشند آنگاه ممکن است بسیاری از تصاویر چهره را تشخیص ندهند و اگر خیلی کلی باشند ممکن است تصاویری که چهره نباشند را به اشتباه چهره تشخیص دهند. به عنوان نمونه یانگ و هوانگ از چنین رویکردی برای تشخیص چهره استفاده کردند[۴].

۲- روش‌های مبتنی بر ویژگی[3]: برخلاف رویکرد مبتنی بر دانش، محققان در تلاش بوده‌اند تا ویژگی‌های ثابت چهره را برای تشخیص پیدا کنند. فرض اساسی بر این استدلال آن است که انسان می تواند بدون زحمت چهره ها و اشیاء را در حالت‌ها و شرایط نوری مختلف تشخیص دهد

[1] Object Detection

[2] Knowledge-based Methods

[3] Feature-based Methods

و بنابراین، باید ویژگی‌هایی وجود داشته باشد که در این تغییرات ثابت هستند. روش‌های متعددی برای تشخیص ابتدا ویژگی‌های صورت و سپس استنباط وجود یک چهره ارائه شده است. معمولاً ویژگی‌های صورت مانند ابرو، چشم، بینی و دهان با استفاده از آشکارسازهای لبه استخراج می‌شوند. سپس بر اساس ویژگی‌های استخراج شده، یک مدل آماری برای توصیف روابط آن‌ها و تأیید وجود یک چهره ساخته می‌شود.

۳- تطبیق قالب[1]: در این روش‌ها از یک قالب از پیش تعیین شده برای چهره استفاده می‌شود این قالب می‌تواند به صورت دستی یا با استفاده از یک تابع تعریف شود. با داشتن یک تصویر ورودی میزان تطابق آن با قالب‌ها بدست می‌آید و سپس با استفاده از این میزان تطابق می‌توان وجود چهره را تشخیص داد. از نکات مثبت این رویکرد می‌توان به سادگی پیاده‌سازی آن اشاره کرد. اما تجربه نشان داده که این روش معمولا دقت خوبی را بدست نمی‌آورد و این موضوع به خاطر تغییرات زیاد در حالت چهره‌هاست. به عنوان مثال ساکای از چنین رویکردی برای تشخیص چهره استفاده کرده است[۵].

۴- روش‌های مبتنی بر ظاهر[2]: بر خلاف روش‌های تطبیق قالب که قالب‌ها از قبل ساخته می‌شوند در این روش‌ها قالب‌ها با استفاده از داده‌های آموزشی یاد گرفته می‌شوند. به طور کلی این روش‌ها از تکنیک‌های تحلیل آماری و یادگیری ماشین استفاده می‌کنند. آنچه در این پروژه برای تشخیص چهره انجام می‌دهیم از چنین رویکردی بهره می‌برد.

بازنمایی چهره: برای بدست آوردن بازنمایی برای تصاویر چهره، روش‌های متفاوتی در طی سال‌ها معرفی شده‌اند. این روش‌ها را در حالت کلی می‌توانیم به دو بخش تقسیم کنیم:

۱- روش‌های سنتی: این روش‌ها قدیمی‌تر هستند و برای یافتن بازنمایی برای چهره‌ها از روش‌هایی مانند تطبیق قالب استفاده می‌کنند. تطبیق قالب برای بازنمایی چهره تقریبا همانند تشخیص

[1] Template Matching

[2] Appearance-based Methods

چهره است که یک قالب از پیش تعیین شده داریم و با استفاده از آن یک بازنمایی برای چهره می‌یابیم. دقت این روش‌ها نسبتا پایین است و رفته رفته جای خود را به روش‌ها دسته دوم دادند.

۲- روش‌های مبتنی بر یادگیری عمیق[1]: در این دسته از روش‌ها از یک شبکه عصبی با تعداد لایه‌های زیاد استفاده می‌شود. این شبکه‌ها معمولا بر روی یک مسئله دسته‌بندی آموزش می‌بینند و سپس از خروجی یکی از لایه‌ها نزدیک به انتها به عنوان بازنمایی استفاده می‌شود. در این پروژه برای بدست آوردن بازنمایی از این رویکرد استفاده می‌شود.

خوشه‌بندی: پس از بدست آوردن بازنمایی برای تصاویر، باید داده‌های موجود را خوشه‌بندی کنیم. الگوریتم‌های زیادی برای خوشه‌بندی وجود دارند اما این الگوریتم‌ها لزوما برای خوشه‌بندی تصاویر چهره مناسب نیستند. در ادامه چند روش معروف خوشه‌بندی نام برده و مشکلات آن‌ها تشریح می‌شوند.

۱- الگوریتم K-means Clustering: مشکل استفاده از این الگوریتم این است که خوشه‌ها باید محدب[2] باشند.

۲- الگوریتم Spectral Clustering: خوشه‌ها در این الگوریتم باید از نظر تعداد داده متوازن باشند.

۳- الگوریتم DBSCAN[6]: در این الگوریتم فرض می‌شود خوشه‌های مختلف چگالی داده یکسانی دارند.

اما آزمایش‌ها نشان داده که بدلیل اینکه چهره‌ها تحت حالات مختلف می‌توانند قرار بگیرند، خوشه‌های حاصل ویژگی‌های گفته شده در بالا را ندارند و اگر از این الگوریتم‌ها برای خوشه‌بندی استفاده کنیم معمولا کارایی مورد نظر را بدست نخواهیم آورد. در این پروژه الگوریتم دیگری معرفی می‌شود که شرط‌های محدود کننده بالا را نیاز ندارد و عملکرد بهتری خواهد داشت.

در این فصل به طور خلاصه با پیشنیه سه موضوع اصلی مورد بحث آشنا شدیم. در فصل‌های آتی به بررسی روش‌های مورد استفاده در این پروژه می‌پردازیم و هرکدام از مسائل را با جزییات بیشتر تشریح می‌کنیم.

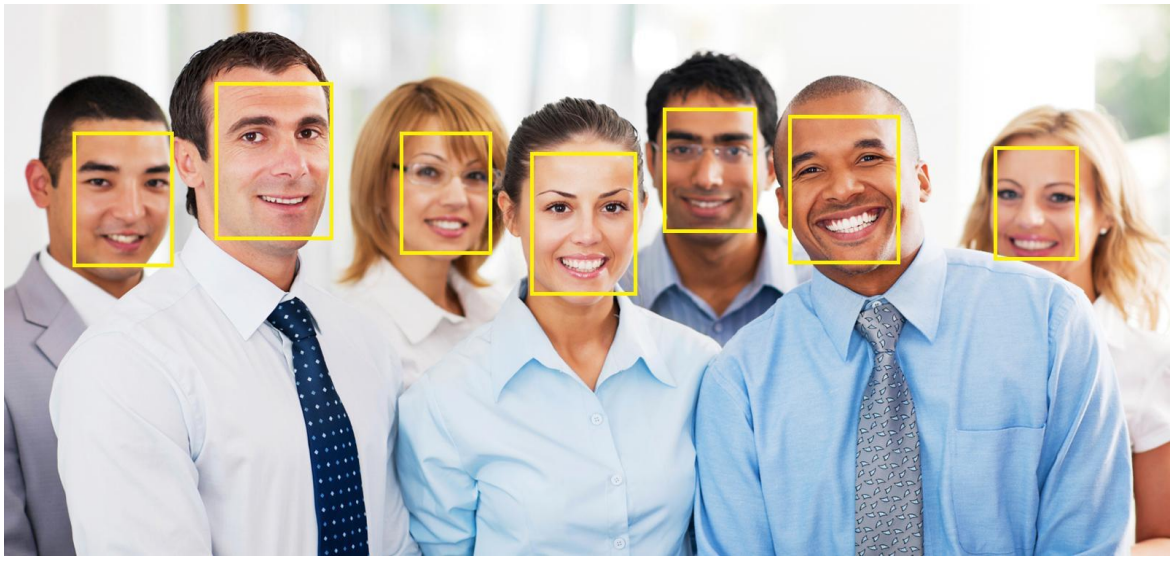
[1] Deep Learning

[2] Convex

فصل سوم تشخیص چهره

فصل ۳ تشخیص چهره[1]

برای اینکه بتوانیم تصاویر را با توجه به افراد موجود در آنها دسته بندی کنیم، در ابتدا نیاز داریم چهره اشخاص موجود در تصاویر را پیدا کنیم. بنابراین اولین مرحله کار این است که برنامه بتواند در تصاویر داده شده قسمت‌های مربوط به چهره را تشخیص دهد. برای مثال به شکل ۳-۱ دقت کنید همانطور که در این شکل می‌بینیم بخش‌هایی از تصویر که مربوط به چهره افراد است تشخیص داده شده و با یک کادر مربعی مشخص شده‌اند. پیدا کردن این بخش‌ها در تصاویر تشخیص چهره نامیده می‌شود. در این فصل به روش‌ها و چالش‌های انجام اینکار پرداخته می‌شود.



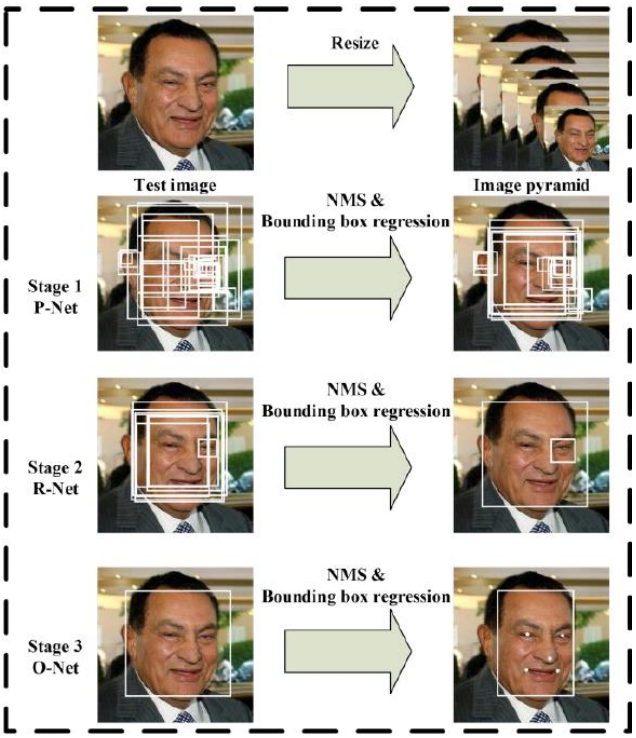
شکل ۳-۱: تشخیص چهره افراد در یک تصویر

تشخیص چهره با روش‌های مختلفی می‌تواند انجام شود که در زیرفصل‌های آتی به توضیح یکی از این روش‌ها می‌پردازیم.

[1] Face Detection

۳ – ۱ الگوریتم MTCNN۱[1]

الگوریتم MTCNN یکی از معروف ترین الگوریتم‌ها برای تشخیص مکان چهره‌ها[۲] و نشانه‌های چهره[۳] است. این الگوریتم چندوظیفه‌ای[۴] است به این معنی که کار تشخیص چهره و پیداکردن نشانه‌های چهره را همزمان انجام می‌دهد و آبشاری[۵] است به این معنی که عملیات تشخیص چهره در چند مرحله انجام می‌شود. شکل ۳-۲ خط لوله[۶] این الگوریتم را نشان می‌دهد که در ادامه به توضیح قسمت‌های مختلف آن می‌پردازیم.



شکل ۳-۲: خط لوله الگوریتم MTCNN برای تشخیص مکان چهره و نشانه‌های صورت[۱]

[1] Multi-task Cascaded Convolutional Neural Networks

[2] Face Detection

[3] Facial Landmarks

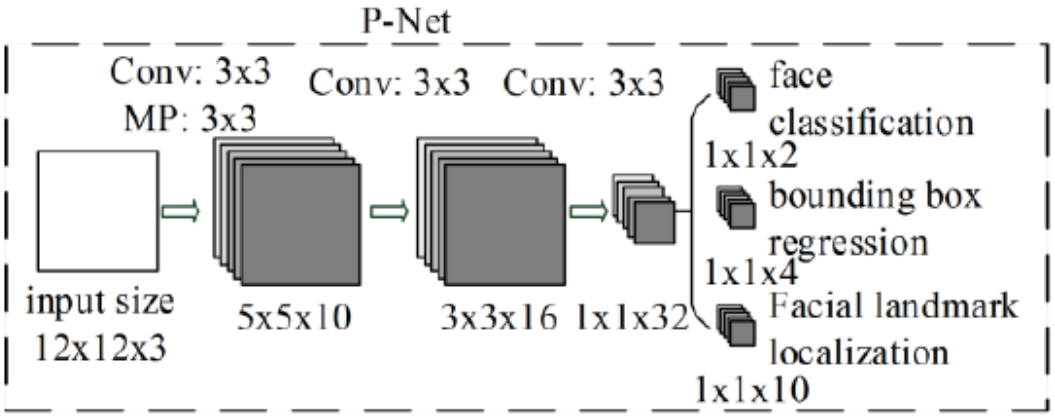
[4] Multi-task

[5] Cascaded

[6] pipeline

همانطور که در شکل ۳-۲ می‌بینیم در ابتدا تصویر ورودی را در اندازه‌های مختلف در می‌آوریم. هدف این‌کار اینست که بتوانیم چهره‌ها با اندازه‌های مختلف را پیدا کنیم. تصاویر بدست آمده به عنوان ورودی به مرحله ۱ داده می‌شوند.

مرحله ۱: در این مرحله از یک شبکه‌ی عصبی پیچشی (CNN) استفاده می‌شود تا تعدادی پنجره[1] به عنوان کاندید انتخاب شوند. شبکه‌ی عصبی استفاده شده در این مرحله نسبت به شبکه‌های مراحل بعد ساده‌تر است. این شبکه را P-Net (Proposal Network) می‌نامیم، به این معنی که این شبکه تعدادی کاندید را برای ما مشخص می‌کند و با استفاده از NMS[2] کاندیدهایی که باهم اشتراک زیادی دارند را تلفیق می‌کند. NMS تکنیکی است که از بین چند موجودیت که باهم اشتراک دارند (برای مثال کادرها) یکی را انتخاب میکند. شکل ۳-۳ معماری این شبکه را نشان می‌دهد.



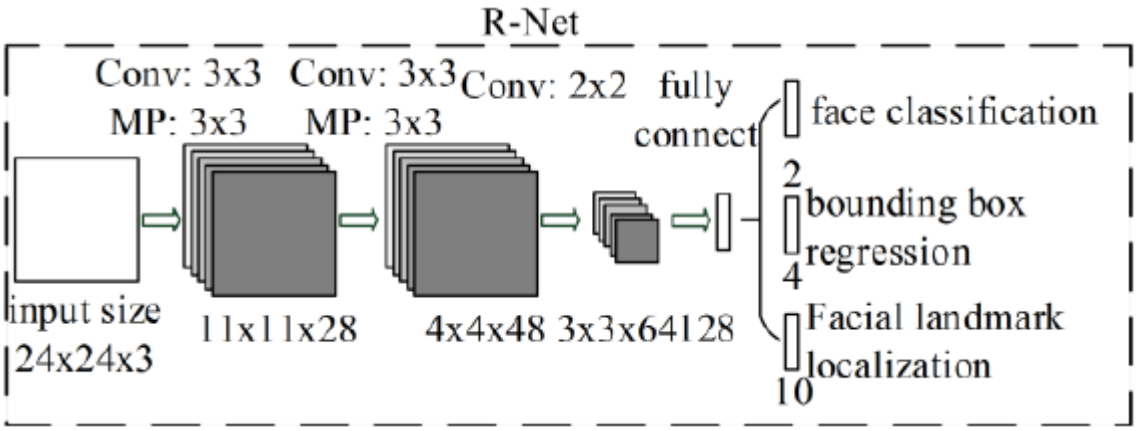
شکل ۳-۳: معماری شبکه P-Net[۱]

مرحله ۲: کاندیدهای بدست آمده از مرحله قبل زیاد هستند و فقط تعداد کمی از آنها صحیح هستند. در این مرحله کاندیدهای بدست آمده قبل را به یک شبکه CNN جدید به نام R-Net می‌دهیم. مسئولیت این شبکه اینست که برخی از کاندیدهای اشتباه بدست آمده در مرحله ۱ را حذف کند و دقت پنجره‌های

[1] Bounding Box

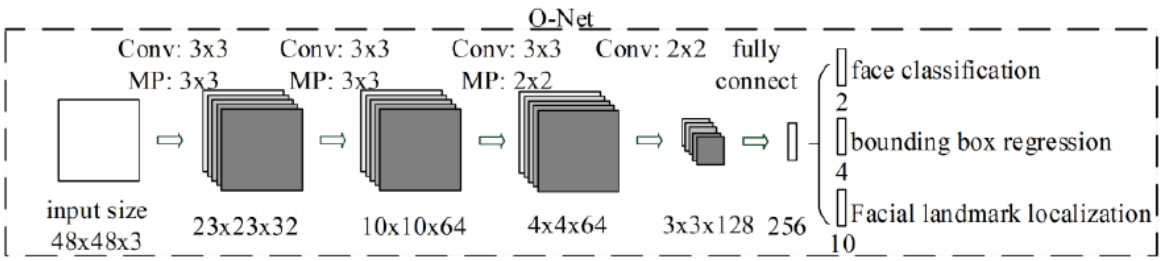
[2] Non Maximal Suppression

چهره‌ها را افزایش دهد. در نهایت در این مرحله با استفاده از تکنیک NMS پنجره‌هایی که با هم اشتراک زیادی دارند را یکی می‌کنیم. شکل ۳-۴ معماری شبکه R-Net (Refine Network) را نشان میدهد.



شکل ۳-۴: معماری شبکه R-Net[۱]

مرحله ۳: هدف این مرحله نیز تقریبا مانند مرحله قبل است. کاندیدهای بدست آمده از مرحله ۲ به مرحله ۳ داده می‌شوند، تعدادی از کاندیدهای اشتباه حذف می‌شوند، دقت پنجره‌ها افزایش پیدا می‌کند و در نهایت در خروجی پنجره چهره‌ها و مکان ۵ نشانه صورت بدست می‌آید. همانطور که در شکل ۳-۵ میبینیم این شبکه نسبت به شبکه‌های قبل عمیق‌تر است و آن را O-Net (Output Network) می‌نامیم.



شکل ۳-۵: معماری شبکه O-Net[۱]

۳ – ۱ – ۱ نحوه آموزش شبکه‌ها

برای آموزش شبکه‌ها باید یک تابع هزینه تعریف شود. پس از تعریف تابع هزینه پارامترهای مدل باید به گونه‌ای انتخاب شوند که مقدار هزینه کمینه شود. اگر به شکل ۳-۵ دقت کنیم می‌بینیم که خروجی مدل شامل ۳ بردار است:

- یک بردار ۲ تایی که نشان می‌دهد آیا تصویر مورد نظر مربوط به یک چهره است یا نیست.
- برداری ۴ تایی که مکان چهره (در صورت وجود) در تصویر را نشان می‌دهد. (با یک بردار ۴ تایی میتوان مکان و اندازه یک کادر مربعی را مشخص کرد)
- برداری ۱۰ تایی که مکان ۵ نشانه صورت را نشان می‌دهند. (برای هر نقطه دو مولفه لازم است بنابراین به ۱۰ مولفه نیاز داریم.)

به این ترتیب برای آموزش مدل باید برای هرکدام از این قسمت‌ها یک تابع هزینه معرفی شود.

۱- **تابع هزینه دسته‌بندی چهره**[1]: با توجه به اینکه مسئله مربوط به این قسمت یک مسئله دسته‌بندی است تابع هزینه مناسب برای آن نیز تابع هزینه cross-entropy است که به صورت زیر تعریف می‌شود:

$$L_i^{\det} = -(y_i^{\det} \log(p_i) + (1 - y_i^{\det})(1 - \log(p_i)) \qquad (۱ - ۳)$$

در رابطه بالا $L_i^{\det}$ هزینه دسته‌بندی داده i ام و $y_i^{\det}$ برچسب دسته‌بندی داده i ام را نشان می‌دهند. همچنین p_i خروجی مدل است که احتمال چهره بودن یا نبودن تصویر i ام را نشان می‌دهد.

۲- **تابع هزینه رگرسیون کادر چهره**[2]: هدف از این تابع هزینه اینست که مدل به گونه ای آموزش ببیند که بردار دوم خروجی (بردار ۴ تایی) مکان چهره را به صورت یک کادر مربعی پیش‌بینی کند. بنابراین این مسئله بر خلاف مسئله قسمت قبل یک مسئله رگرسیون است، زیرا میخواهیم

[1] Face Classification

[2] Bounding Box Regression

یک خروجی پیوسته را پیشبینی کنیم. ساده ترین تابعی که برای این قسمت به ذهن می‌رسد تابع هزینه MSE است که به صورت زیر تعریف می‌شود.

$$L_i^{box} = \left|\left|\hat{y}_i^{box} - y_i^{box}\right|\right|_2^2 \qquad (۲-۳)$$

برای هر داده آموزشی y_i^{box} نشان دهنده کادری است که برای مکان چهره پیشبینی کرده ایم و $\hat{y}_i^{box}$ برچسب داده آموزشی است. به این ترتیب L_i^{box} که هزینه مربوط به پیشبینی کادر برای داده‌ی i ام است را به صورت فاصله اقلیدسی y_i^{box} و $\hat{y}_i^{box}$ تعریف می‌کنیم.

۳- **تابع هزینه محل‌یابی نشانه‌های چهره**[1]: این تابع هزینه نیز همانند تابع هزینه قبل به صورت یک مسئله رگرسیون فرمول می‌شود و بنابراین تابع هزینه به صورت زیر تعریف می‌شود:

$$L_i^{landmark} = \left|\left|\hat{y}_i^{landmark} - y_i^{landmark}\right|\right|_2^2 \qquad (۳-۳)$$

در رابطه بالا $\hat{y}_i^{landmark}$ برچسب داده آموزشی i ام، $y_i^{landmark}$ مکان پیشبینی شده برای نشانه‌های صورت در داده‌ی i ام و $L_i^{landmark}$ مقدار هزینه محل یابی نشانه‌های چهره‌ی داده i ام هستند.

برای داده‌های مختلف ممکن است نتوانیم از هر سه تابع هزینه استفاده کنیم. برای مثال برای یک تصویر که هیچ چهره‌ای در آن وجود ندارد فقط از تابع هزینه اول استفاده می‌شود. برای حل این موضوع از یک sample type indicator استفاده می‌کنیم: به این ترتیب که $\beta_i^j \in \{0,1\}$ اگر ۰ باشد یعنی تابع هزینه j ام برای داده‌ی i ام استفاده نمی‌شود و اگر ۱ باشد استفاده می‌شود. تابع هزینه کلی مدل با توجه به ۳ تابع معرفی شده و پارامترهای α و β به صورت زیر تعریف می‌شود:

$$J = \sum_{i=1}^{N} \sum_{j \in <det,box,landmark>} \alpha_j \beta_i^j L_i^j \qquad (۴-۳)$$

[1] Facial Landmarks Localization

در رابطه بالا N تعداد کل داده‌های آموزشی است. همچنین پارامتر α_j نیز اهمیت تابع هزینه j ام را نشان می‌دهد. برای مثال در شبکه O-Net که خروجی مدل است آلفای مربوط به تابع هزینه سوم را نسبت به سایر شبکه‌ها بزرگتر انتخاب می‌کنیم.

۳ – ۲ پیاده‌سازی و چالش‌ها

در این قسمت به بررسی آنچه در پروژه پیاده‌سازی شده است پرداخته می‌شود، همچنین برخی از چالش‌هایی که در پیاده سازی با آن‌ها روبرو شدیم و راه‌حل آنها بررسی می‌شود.

در این پروژه هیچ مدلی آموزش داده نشده است و آنچه انجام شده اینست که از یک کتابخانه موجود و مدل از پیش آموزش داده شده استفاده شده است. با استفاده از این کتابخانه میتوانیم کار تشخیص چهره در تصاویر را انجام دهیم بدون آنکه نیاز داشته باشیم مدلی را از ابتدا آموزش دهیم.

کتابخانه مورد استفاده در این پروژه facenet_pytorch[7] است که می‌توان آن را به سادگی با استفاده از pip نصب کرد. در شکل ۳-۶ روش‌های نصب این چارچوب[1] دیده می‌شود.

```
# With pip:
pip install facenet-pytorch

# or clone this repo, removing the '-' to allow python imports:
git clone https://github.com/timesler/facenet-pytorch.git facenet_pytorch

# or use a docker container (see https://github.com/timesler/docker-jupyter-dl-gpu):
docker run -it --rm timesler/jupyter-dl-gpu pip install facenet-pytorch && ipython
```

شکل ۳-۶: روش‌های نصب کتابخانه facenet-pytorch

ماژول facenet_pytorch کلاسی با نام MTCNN دارد که می‌توان با استفاده از آن چهره‌های موجود در یک تصویر را یافت. در ادامه با برخی از چالش‌هایی که در استفاده از این کلاس وجود دارد آشنا می‌شویم.

[1] Framework

۳ – ۲ – ۱ یک اندازه کردن چهره‌ها برای قابلیت بسته‌بندی[1]

فرض کنید می‌خواهیم چهره‌های موجود در ۱۰۰ تصویر را بیابیم. یک راه که وجود دارد اینست که ابتدا تصویر اول را به مدل بدهیم و مدل با یکبار حرکت به جلو[2] چهره‌های موجود در آن را بیابد سپس تصویر دوم و ... این کار بسیار کند است زیرا هر اجرای رو به جلو برای یک شبکه عصبی بزرگ ممکن است زمانگیر باشد. به جای آن می‌توانیم یک بسته[3] از ۲۰ تصویر بسازیم و همه آن ۲۰ تصویر را به یکباره به مدل دهیم. انجام عملیات تشخیص چهره وقتی که از قابلیت بسته‌بندی استفاده می‌کنیم سریع‌تر است، زیرا بسیاری از عملیات‌ها می‌توانند به صورت موازی انجام شوند. (مخصوصا زمانی که از GPU استفاده می‌کنیم.) علاوه بر اینها اندازه بسته‌ها نمی‌تواند به اندازه دلخواه بزرگ باشد، زیرا حافظه زیادی اشغال خواهد کرد. با این تفاسیر استفاده از بسته با اندازه حدودا ۳۲ معقول است و سرعت کار را افزایش زیادی می‌دهد.

تمام تصاویری که در یک بسته به شبکه داده می‌شوند باید اندازه یکسان داشته باشند. – مثلا همگی ۱۰۰۰ در ۱۰۰۰ باشند. به همین دلیل باید تمام تصاویر را به یک اندازه یکسان ببریم و سپس میتوانیم از بسته‌بندی استفاده کنیم. اما انجام این کار می‌تواند مشکل‌زا باشد برای مثال برخی تصاویر به صورت عمودی[4] هستند و تغییر اندازه آن‌ها ممکن است باعث تغییر زیاد نسبت طول به عرض شود و کیفیت تصویر را کاهش دهد به گونه‌ای که چهره‌های موجود در آن قابل تشخیص نباشند. این موضوع در برنامه کاربردی محدودیت ایجاد می‌کند، زیرا نمی‌توانیم تضمین کنیم که اندازه تمام تصاویر ورودی یکیست و اگر نتوانیم این تضمین را داشته باشیم نمی‌توانیم از بسته‌بندی استفاده کنیم. بنابراین باید تصاویر یک به یک بررسی شوند که باعث افت سرعت می‌شود.

نکته قابل توجه اینست که مشکل تغییر اندازه فقط زمانی وجود دارد که نسبت طول به عرض تصاویر متفاوت باشند. اگر این نسبت برای تصاویر یکی باشد (حتی اگر اندازه آنها متفاوت باشند) میتوانیم همه‌ی

[1] Batching

[2] Forward Pass

[3] Batch

[4] portrait

آنها را به یک اندازه یکسان ببریم بدون اینکه در تصاویر اعوجاج[1] ایجاد شود. از این موضوع می‌توانیم استفاده کنیم تا مشکل را حل کنیم. برای این کار با توجه به اندازه تصاویر می‌توانیم به اطراف آنها پیکسل سیاه اضافه کنیم تا نسبت طول به عرض همه‌ی آن‌ها یکی شود. به شکل ۳-۷ دقت کنید. این تصویر ابتدا یک تصویر عمودی بوده است اما با اضافه کردن پیکسل سیاه در سمت چپ و راست توانستیم آن را به شکل یک تصویر مربعی در بیاوریم حال اگر یک تصویر افقی داشته باشیم برای تبدیل کردن آن به تصویر مربعی باید به بالا و پایین آن پیکسل سیاه اضافه کنیم. پس از این کار می‌توانیم همه‌ی تصاویر را به یک اندازه مربعی یکسان ببریم و هیچ اعوجاجی نخواهیم داشت.

شکل ۳-۷: تبدیل یک تصویر عمودی به تصویر مربعی با اضافه کردن پیکسل سیاه به چپ و راست آن

۳ – ۲ – ۲ خواندن سریع تصاویر از دیسک

با توجه به اینکه ابعاد تصویر ورودی می‌تواند بالا باشد، بنابراین امکان اینکه تمام تصاویر یکجا از دیسک به حافظه اصلی خوانده شوند و سپس پردازش بر روی آن‌ها انجام شود وجود ندارد. (یا حتی اگر امکان پذیر

[1] Distortion

باشد قطعا مقیاس‌پذیر نخواهد بود.) به همین دلیل هر بار یک بسته از تصاویر را از دیسک به حافظه اصلی منتقل می‌کنیم، مدل را برای این بسته اجرا و نتایج را ذخیره می‌کنیم و سپس یک بسته دیگر را از دیسک به حافظه اصلی می آوریم و همین کار را تکرار می‌کنیم تا همه‌ی تصاویر پردازش شوند.

با دقت در روند گفته شده در میابیم که برای هر بار اجرای مدل، یک خواندن از دیسک هم داریم. این موضوع باعث می‌شود اگر سرعت انتقال از دیسک به حافظه اصلی پایین باشد گلوگاه[1] فرایند تشخیص چهره همین انتقال داده از دیسک به حافظه شود و نه اجرای شبکه عصبی! به همین دلیل ضرورت دارد این انتقال با سرعت زیاد انجام شود وگرنه سرعت اجرا افت خواهد کرد.

برای حل این مشکل از DataLoader در فریمورک پایتورچ استفاده می‌شود. استفاده از DataLoader به برنامه ما اجازه می‌دهد که عمل خواندن از دیسک را بتوانیم در چند نخ[2] مجزا انجام دهیم. اینکار سرعت خواندن داده‌ها را افزایش می‌دهد، زیرا استفاده از چند نخ هنگام انجام اعمال IO باعث می‌شود پردازنده بیشتر مشغول باشد و بهره‌وری آن افزایش یابد. در غیر این صورت ممکن است در برخی از زمان‌ها پردازنده منتظر یک عمل ورودی/خروجی بماند و در نتیجه زمان اجرای برنامه زیاد شود.

۳ – ۲ – ۳ مرتب کردن چهره‌ها پس از تشخیص

بسته به نیاز گاهی ممکن است بخواهیم فقط یک چهره موجود در یک تصویر را بیابیم و گاهی همه‌ی چهره‌ها را. اگر هم بخواهیم تمام چهره‌های موجود را پیدا کنیم ممکن است گاهی بخواهیم این تصاویر را بر اساس یک پارامتر مرتب کنیم. برای اینکار سه حالت در نظر گرفته شده است:

۱- امتیاز چهره: کلاس MTCNN در کنار هر چهره‌ای که تشخیص داده یک عدد بین ۰ و ۱ نیز به ما می‌دهد. این عدد که امتیاز آن چهره است به ما می‌گوید که احتمال آنکه کادر تشخیص داده شده واقعا چهره باشد چقدر است. هر چه این عدد به ۱ نزدیک‌تر باشد یعنی چهره واضح‌تر است. میتوانیم چهره‌های تشخیص داده شده را با استفاده از این عدد مرتب کنیم و به این ترتیب اگر

[1] Bottleneck

[2] Thread

فقط یک چهره را بخواهیم، اولین چهره، چهره‌ای است که بیشترین امتیاز را دارد. برای مثال به شکل ۳-۸ توجه کنید. همانطور که می‌بینید در این تصویر چند چهره وجود دارد، اما چون یکی از چهره‌ها واضح تر است امتیاز آن بیشتر خواهد بود.

شکل ۳-۸

۲- اندازه چهره: گاهی وقت‌ها در تصویر یک چهره دورتر از دیگری است، بنابراین اندازه یکی از دیگری بزرگتر می‌شود. به این ترتیب می‌توانیم چهره‌های تشخیص داده شده را با توجه به اندازه‌ی آن‌ها مرتب کنیم و با این کار چهره‌های نزدیک‌تر را اول و چهره‌های دورتر را آخر می‌آوریم. برای انجام این‌کار نیز می‌توانیم مساحت کادرهای تشخیص داده شده را مرتب کنیم.

۳- مرکزیت چهره‌ها: برخی چهره‌ها در تصاویر به مرکز تصویر نزدیک‌ترند و برخی دورتر. می‌توانیم چهره‌ها را بر اساس فاصله آن‌ها از مرکز تصویر نیز مرتب کنیم. این گزینه بیشتر برای ارزیابی استفاده می‌شود و در عمل کاربرد چندانی ندارد.

به این ترتیب با حل مشکلات گفته شده، استفاده از MTCNN امکان‌پذیر می‌شود. در پیاده‌سازی انجام شده برای پروژه کلاس MTCNNDetection نوشته شده این کلاس آدرس تصاویر را به عنوان ورودی در قالب یک فایل CSV دریافت میکند. با استفاده از روش گفته شده تصاویر را در قالب بسته‌ها با اندازه معین

به MTCNN می‌دهد. پس از تشخیص چهره‌های موجود در تصاویر چهره‌ها را با توجه به یکی از معیارهای گفته شده در قسمت ۳-۲-۳ مرتب می‌کند و در نهایت کادرهای مربوط به چهره‌های بدست آمده را در یک فایل CSV ذخیره می‌کند تا در مراحل بعد مورد استفاده قرار گیرد. شکل ۳-۹ API کلاس MTCNNDetection را نشان می‌دهد.

```
class MTCNNDetection:
    """
    A module for detecting the bounding box
    position of faces in images.
    """

    def __init__(self, csv_files, save_folder, batch_size, size, one_face=False, device='cpu', mode='prob', same=False):

    def detect_faces(self, thresh=0.99, num_workers=0, gui_params=None):···

    def select_box(self, bbox_img):···

    def save_in_csv(self, faces):···
```

شکل ۳-۹: API پیاده‌سازی شده برای تشخیص چهره

۳ – ۳ ارزیابی

حال پس از آنکه با موفقیت توانستیم از MTCNN در برنامه خود استفاده کنیم وقت آن است که به ارزیابی عملکرد آن بپردازیم. ارزیابی الگوریتم تشخیص چهره به دو دلیل اهمیت دارد:

۱- با توجه به نتایجی که از ارزیابی بدست می‌آوریم عملکرد سیستم خود را از لحاظ سرعت و دقت خواهیم دید و این به ما کمک می‌کند تا بفهمیم که چه چیزی را می‌توانیم از سیستم انتظار داشته باشیم و چه چیزی را نه.

۲- اگر در آینده بخواهیم از سیستم تشخیص چهره دیگری به جای MTCNN استفاده کنیم آنگاه با استفاده از ارزیابی‌های کمی که در این قسمت انجام می‌دهیم راهی برای مقایسه خواهیم داشت.

در این قسمت به ارزیابی الگوریتم MTCNN از نظر سرعت می‌پردازیم و چند نمونه تشخیص چهره انجام شده توسط برنامه را مشاهده می‌کنیم.

۳ – ۳ – ۱ ارزیابی سرعت

برای ارزیابی سرعت اجرای الگوریتم از دیتاست LFW[1] استفاده میکنیم[۸]. این دیتاست متشکل از ۱۳۲۳۳ تصویر از ۵۷۴۹ شخص مختلف است. اندازه‌ی تمام تصاویر ۲۵۰ در ۲۵۰ است. در بعضی از تصاویر بیش از یک چهره وجود دارد، اما چهره مورد نظر آن چهره‌ای است که به مرکز نزدیک‌تر است.

از آنجایی که نسبت طول به عرض در تمام تصاویر یکسان است نیازی به اضافه کردن پیکسل سیاه به تصاویر نیست. در جدول ۳-۱ زمان اجرای الگوریتم برای پردازش ۱۳۲۳۳ تصویر آورده شده است. همانطور که می‌بینیم و انتظار داریم هر چه اندازه تصویرهای ورودی بیشتر باشد، زمان اجرای الگوریتم بیشتر می‌شود. در این آزمایش از کارت گرافیک NVIDIA GeForce 1050Ti استفاده شده است.

جدول ۳-۱: زمان اجرای الگوریتم MTCNN بر روی دیتاست LFW. در این آزمایش از GPU استفاده شده، اندازه هر بسته برابر با ۳۲ قرار داده شده و فقط چهره‌ای که به مرکز نزدیکتر بوده انتخاب شده است

اندازه ورودی	زمان کل	تعداد تصاویر پردازش شده در ثانیه
۲۵۰ در ۲۵۰	۱۷۰ ثانیه	تقریبا ۷۸
۵۴۰ در ۶۴۸	۲۸۷ ثانیه	تقریبا ۴۶
۷۲۰ در ۸۶۴	۵۰۰ ثانیه	تقریبا ۲۶

با توجه به اینکه اندازه اولیه تصاویر در دیتاست LFW ۲۵۰ در ۲۵۰ است، بنابراین انتخاب این اندازه برای ورودی شبکه امکان پذیر است. اما در حالت کلی اندازه تصاویر معمولا بیشتر از این مقدار است و تغییر اندازه تصاویر به این اندازه ممکن است بسیاری از جزییات تصویر را از بین ببرد. اما اندازه ۵۴۰ در ۶۴۸ اندازه خوبی است و حتی اگر تصاویر بزرگتر از این اندازه را نیز داشته باشیم، با کوچک کردن آن‌ها به این

[1] Labeled Faces in the Wild

اندازه اطلاعات زیادی از دست نمی‌رود. پردازش تقریبا ۴۶ تصویر با این اندازه کارایی قابل قبولی برای فاز تشخیص چهره می‌دهد.

۳ – ۳ – ۲ چند نمونه تشخیص چهره

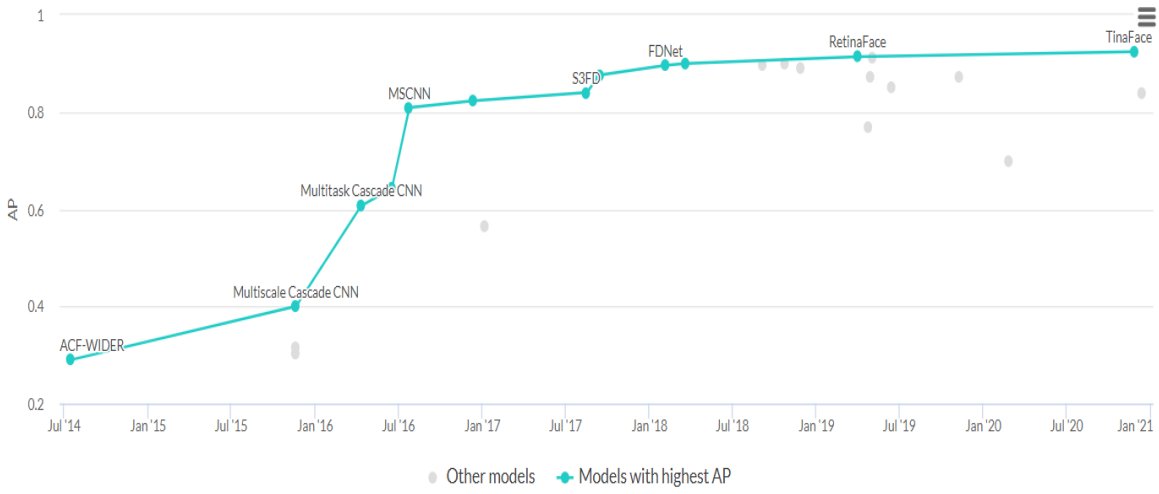
در این پروژه بخش مربوط به تشخیص چهره به صورت کمی ارزیابی نشده است. و ارزیابی آن تنها به صورت کیفی و با مشاهده چند مورد از نتایج تشخیص چهره انجام شده است. شکل ۳-۱۰ چند مورد تصویر را نشان می‌دهد و شکل ۳-۱۱ نتایج تشخیص چهره توسط الگوریتم MTCNN را نشان می‌دهد.

شکل ۳-۱۰: تصویری شامل چهره ۵ شخص مختلف

شکل ۳-۱۱: چهره‌های تشخیص داده شده در شکل ۳-۱۰

۳ – ۴ کارهای آینده

الگوریتم MTCNN یکی از معروف‌ترین الگوریتم‌های تشخیص چهره است، اما تنها الگوریتم موجود برای این کار نیست. این الگوریتم در سال ۲۰۱۶ معرفی شده و پس از آن الگوریتم‌های بسیار دیگری نیز معرفی شده‌اند شکل زیر که از سایت paperswithcode.com گرفته شده است نشان می‌دهد که الگوریتم‌های خوب دیگری نیز برای تشخیص چهره وجود دارند.



شکل ۳-۱۲: عملکرد الگوریتم‌های مختلف تشخیص چهره بر روی دیتاست WIDER Face (Hard) [۹]

دلیل اینکه برای پروژه از الگوریتم MTCNN استفاده کردیم این بود که نسبت به سایر الگوریتم‌ها ساده‌تر است و دقت نسبتا خوبی نیز دارد، اما الگوریتم‌های جدیدتر مانند RetinaFace[10] و ... وجود دارند که عملکرد بهتری را به ما می‌دهند. یکی از کارهایی که در آینده برای بهبود برنامه می‌توان انجام داد این است که به جای MTCNN از روش‌های جدیدتر استفاده شود اینکار کیفیت فاز تشخیص چهره را بهبود می‌بخشد که منجر به بهبود عملکرد کل برنامه می‌شود.

۳ – ۵ جمع‌بندی

در این فصل به معرفی مسئله تشخیص چهره پرداختیم. سپس جزییات الگوریتم MTCNN برای تشخیص چهره را بررسی کردیم. در انتها عملکرد آن را از نظر سرعت ارزیابی کردیم. پس از بدست آوردن چهره‌ها

از تصاویر نوبت به استخراج ویژگی از آن‌ها می‌رسد تا بتوانیم برای چهره‌ها بازنمایی مناسب به دست آوریم. در فصل بعد به بررسی این مسئله می‌پردازیم.

فصل چهارم بازنمایی چهره

فصل ۴ بازنمایی چهره[1]

در فصل قبل با تشخیص چهره آشنا شدیم. الگوریتم MTCNN به ما این اجازه را می‌دهد تا قسمت‌های چهره را بر روی یک تصویر تشخیص دهیم. پس از انجام این‌کار نوبت به آن می‌رسد که چهره‌های مربوط به افراد مختلف را از هم جدا کنیم. در این فصل و فصل بعد به تشریح دقیق چگونگی انجام این کار می‌پردازیم.

فرض کنید هر تصویر چهره دارای ابعاد ۱۰۰ در ۱۰۰ باشد. بنابراین هر تصویر یک آرایه ۳ در ۱۰۰ در ۱۰۰ است (۳ به خاطر وجود ۳ کانال رنگ است). به این ترتیب هر چهره را می‌توانیم یک نقطه در یک فضای برداری ۳۰ هزارتایی در نظر بگیریم. این فضای برداری بسیار بزرگ است و تصاویر زیادی را در خود دارد. همچنین استفاده از پیکسل‌ها به عنوان ویژگی[2] برای بازنمایی برداری کار درستی نیست. علاوه بر این‌ها این فضای برداری تصاویر زیادی که چهره نیستند را در نیز در بر دارد. به همین دلایل خوشه‌بندی تصاویر در چنین فضای برداری دقت لازم را نخواهد داشت و برای ما مفید نخواهد بود.

هدف ما در این فصل اینست که هر تصویر را (که یک نقطه در فضای برداری ۳۰ هزار بعدی است) به یک فضای برداری جدید ببریم که ویژگی‌های زیر را داشته باشد:

۱- ابعاد این فضای برداری جدید باید بسیار کمتر از فضای برداری اولیه باشند.

۲- در فضای برداری جدید، نقاط مربوط به تصاویر چهره یک شخص فاصله کمی باهم داشته باشند و نقاط مربوط به تصاویر چهره اشخاص متفاوت فاصله زیاد.

انجام این کار مزیت‌های زیر را دارد:

۱- اجرای یک الگوریتم خوشه بندی در فضای برداری جدید دقت بیشتری خواهد داشت.

۲- اجرای الگوریتم خوشه‌بندی در این فضای برداری بسیار سریعتر از فضای برداری اولیه خواهد بود، زیرا ابعاد این فضا از فضای اولیه بسیار کوچک تر است.

[1] Face Representation

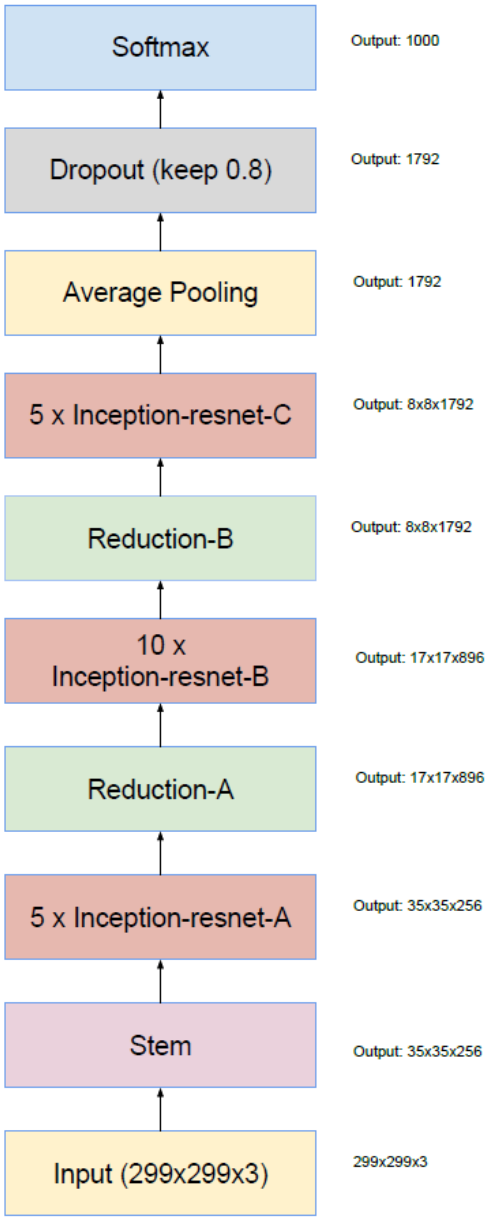
[2] Feature

بنابراین هدف ما در این فصل آن است که یک تابع (تبدیل) بیابیم که هر تصویر چهره را به یک فضای برداری جدید با ویژگی‌های گفته شده ببرد. همانطور که می‌دانیم شبکه‌های عصبی در واقع یک تابع هستند. به همین دلیل تابع مورد نظر خود را به صورت یک شبکه عصبی مدل می‌کنیم و با آموزش آن به هدف خود می‌رسیم.

در ادامه این فصل به چگونگی ساختن این شبکه عصبی و آموزش آن خواهیم پرداخت.

۴ – ۱ شبکه‌ی عصبی Inception-ResNet-v1

در این قسمت با معماری شبکه عصبی Inception-ResNet-v1[2] و چگونگی آموزش آن آشنا می‌شویم. در شکل ۴-۱ ساختار کلی این شبکه نشان داده شده است. در ادامه هر یک از قسمت‌های آن به صورت دقیقتر بررسی و تشریح می‌شوند. البته آنچه در این تصویر دیده می‌شود معماری معرفی شده در مقاله مربوط به آن است و با توجه به کاربرد آن در اینجا تفاوت‌های جزیی در آن ایجاد شده که در ادامه این تفاوت‌ها نیز توضیح داده می‌شوند، اما ساختار کلی شبکه همان چیزی است که در شکل ۴-۱ دیده می‌شود.



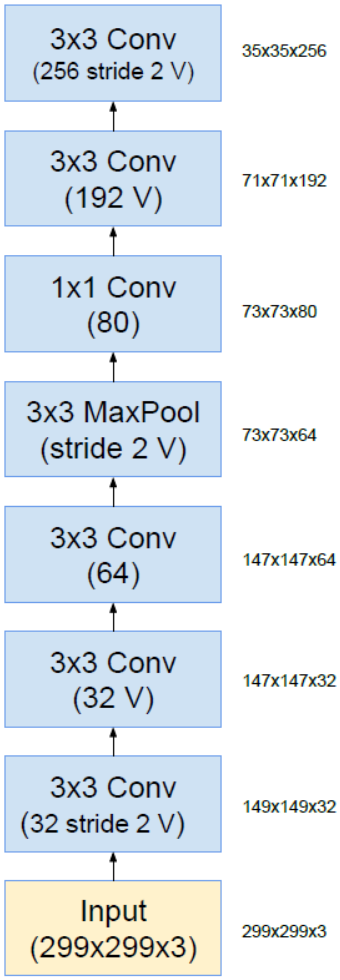
شکل ۴-۱ : معماری شبکه‌ی عصبی Inception-ResNet-v1[۲]

در شکل ۴-۱ می‌بینیم که ابعاد تصویر ورودی باید ۲۹۹ در ۲۹۹ باشد اما این موضوع لازم نیست، زیرا همانطور که می‌بینیم در لایه‌های انتهایی یک لایه Average Pooling داریم. این لایه به ما این اجازه را می‌دهد که اندازه تصویر ورودی هر چقدر که می‌خواهیم باشد و نیازی نیست حتما ۲۹۹ در ۲۹۹ باشد. در

واقع در این برنامه نیز ورودی شبکه تصویری با اندازه ۱۱۲ در ۱۱۲ است. (اندازه ۱۱۲ در ۱۱۲ برای کاربرد مورد نظر مناسب است، زیرا ورودی شبکه قرار است تصویری از یک چهره باشد و این اندازه کفایت می‌کند.)

۴ – ۱ – ۱ بلاک Stem

دومین بلاک در شکل ۴-۱، Stem نام گذاری شده است. Stem خود شامل چند لایه از Convolution است. ساختار Stem را در شکل ۴-۲ می‌بینیم.



شکل ۴-۲: ساختار Stem در شبکه Inception-ResNet-v1 [۲]

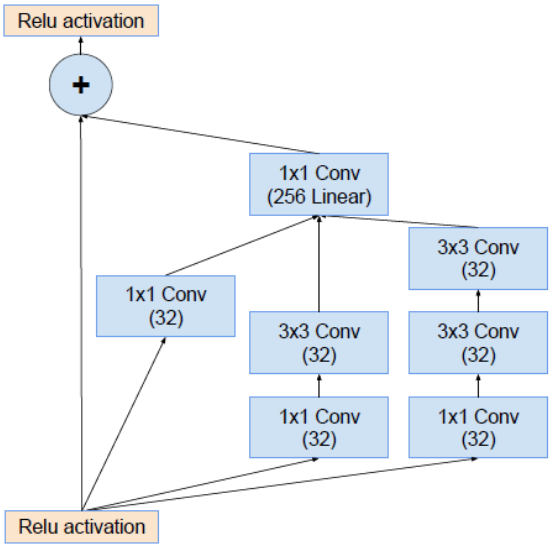
در شکل ۴-۲ دو نوع ماژول دیده می‌شود:

۱- ماژول Conv: این ماژول شامل سه لایه‌ی Convolution، سپس BatchNorm و در نهایت تابع فعالیت ReLU است.

۲- ماژول MaxPool: این ماژول با توجه به اندازه کرنل و stride عمل max pooling را انجام می‌دهد. (این ماژول هیچ پارامتری ندارد)

۴ – ۱ – ۲ بلاک‌های Inception-ResNet-A، Inception-ResNet-B و Inception-ResNet-C

در شکل ۴-۱ سه بلاک به نام‌های Inception-ResNet-A، Inception-Resnet-B و Inception-ResNet-C دیده می‌شود. معماری هر یک از این ماژول‌ها در شکل‌های ۴-۳، ۴-۴ و ۴-۵ آورده شده است.

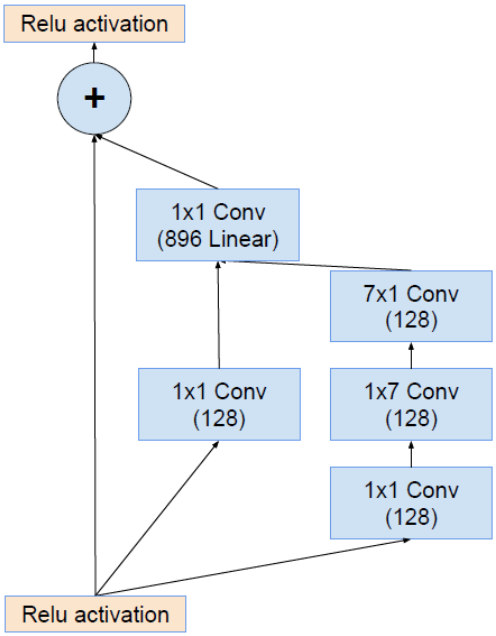


شکل ۴-۳: ماژول Inception-ResNet-A [۲]

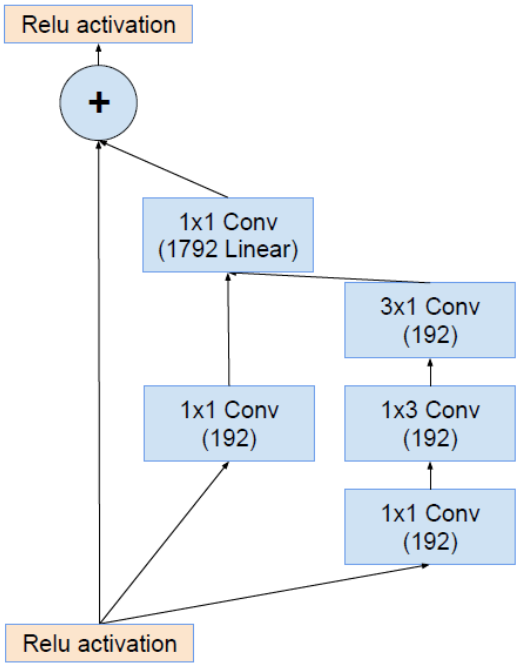
همانطور که دیده می‌شود این سه ماژول مشابه یکدیگرند با برخی تفاوت‌های جزئی. ماژول اصلی سازنده این سه، ماژول Conv است. در مورد این ماژول‌ها نکات زیر قابل توجه هستند:

۱- در هر سه این ماژول‌ها یک مسیر مستقیم از ورودی به خروجی وجود دارد که به آن Residual Connection گفته می‌شود. استفاده از این اتصال آموزش مدل عمیق را آسانتر می‌کند، زیرا گرادیان‌ها به راحتی از این اتصالات عبور می‌کنند.

۲- از کانولوشن های ۱ در ۱ برای تغییر تعداد کانال‌ها استفاده می‌شود. این موضوع به کاهش تعداد پارامترهای مدل کمک می‌کند.



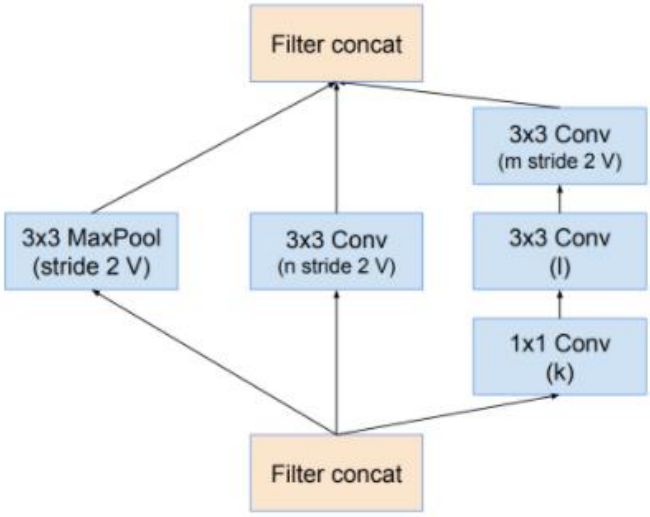
شکل ۴-۴: ماژول **Inception-ResNet-B** [۲]



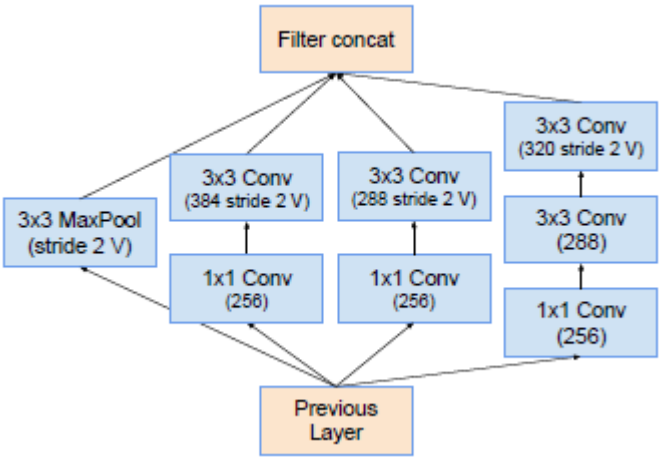
شکل ۴-۵: ماژول **Inception-ResNet-C** [۲]

۴ – ۱ – ۳ بلاک‌های Reduction-A و Reduction-B

دو بلاک با نام Reduction-A و Reduction-B در شکل ۴-۱ دیده می‌شوند. معماری این دو بلاک در شکل‌های ۴-۶ و ۴-۷ دیده می‌شوند. هدف اصلی این دو بلاک کاهش ابعاد و افزایش عمق ورودی است.



شکل ۴-۶: معماری Reduction-A برای کاهش ابعاد ورودی از 35x35 به 17x17 [۲]



شکل ۴-۷ :معماری Reduction-B برای کاهش ابعاد ورودی از 17x17 به 8x8 [۲]

۴ – ۱ – ۴ سایر بلاک‌ها

علاوه بر چند ماژول گفته شده در قسمت‌های قبل در شکل ۴-۱ ماژول‌های دیگری نیز دیده می‌شود که در این قسمت به طور خلاصه به توضیح هر یک می‌پردازیم:

۱- Average Pooling: این ماژول همانند ماژول MaxPool عمل می‌کند. با این تفاوت که به جای محاسبه Maximum از Average مقدارها استفاده می‌کند. در پیاده سازی انجام شده به جای استفاده از این ماژول از ماژول Adaptive Average Pool استفاده شده. استفاده از این ماژول به شبکه اجازه می‌دهد که ورودی با هر سایز دلخواه را قبول کند.

۲- [11]Dropout: این لایه به صورت رندم تعدادی از نورون‌های ورودی را ۰ می‌کند این کار باعث می‌شود که شبکه به صورت تعمیم‌یافته آموزش ببیند و نوعی تنظیم کننده[1] محسوب می‌شود.

تا به اینجا با جزییات معماری شبکه عصبی Inception-ResNet-v1 آشنا شدیم. علاوه بر این شبکه نسخه‌های دیگر Inception-ResNet نیز وجود دارند که می‌توانیم از آنها نیز استفاده کنیم. در این پروژه به دلیل اینکه با استفاده از نسخه ۱ آن به دقت قابل قبولی رسیدیم به بررسی نسخه‌های دیگر نپرداختیم. در ادامه بررسی می‌کنیم که چگونه می‌توانیم این شبکه را برای هدف خود آموزش دهیم.

۴ – ۲ آموزش مدل

قبل از استفاده، شبکه عصبی معرفی شده در بخش قبل باید آموزش داده شود. برای آموزش به این صورت عمل می‌کنیم که شبکه را به عنوان یک دسته‌بند[2] آموزش می‌دهیم. اما می‌دانیم که هدف نهایی ما دسته بندی نیست. بلکه می‌خواهیم یک بازنمایی از تصاویر بدست آوریم. به همین دلیل پس از آموزش مدل لایه آخر شبکه که امتیاز هر کلاس را مشخص می‌کند را حذف می‌کنیم و از خروجی لایه‌ی یکی به آخر آن به عنوان یک بازنمایی از تصویر ورودی استفاده می‌کنیم.

[1] Regularizer

[2] Classifier

بنابراین باید شبکه عصبی را به عنوان یک دسته بند آموزش دهیم. برای این‌کار از یک تابع هزینه مناسب برای دسته‌بندی استفاده می‌کنیم. ساده‌ترین گزینه استفاده از تابع هزینه cross-entropy است که به صورت زیر تعریف می‌شود.

$$L_i = -\sum_{j=1}^{output\ size} y_{ij} \log \widehat{y_{ij}} \qquad (۱-۴)$$

در رابطه بالا L_i هزینه بدست آمده از داده i ام را نشان می‌دهد. y_{ij} مشخص می‌کند که آیا داده‌ی i ام متعلق به کلاس j ام است یا نه (۱ و ۰) و $\widehat{y_{ij}}$ امتیاز داده‌ی i ام برای کلاس j ام است. به این ترتیب هزینه کل برابر است با مجموع هزینه‌های مربوط به هر داده.

آخرین قدم برای آموزش مدل انتخاب یک مجموعه آموزشی است. مجموعه‌های آموزشی بسیاری به صورت رایگان موجود هستند. در زیر به دو مورد از این مجموعه ها اشاره می‌کنیم.

۱- مجموعه آموزشی VGGFace2[12]: این دیتاست از ۳ میلیون و ۳۱۰ هزار تصویر از ۹۱۳۱ فرد مختلف تشکلیل شده است.

۲- مجموعه آموزشی CASIA-WebFace[13]: این دیتاست متشکل از ۴۹۴ هزار و ۴۱۴ تصویر از ۱۰۵۷۵ شخص مختلف است.

با توجه به داده‌های آموزشی و تابع هزینه داده شده می‌توان شبکه مورد نظر را آموزش داد.

در این پروژه از آموزش این شبکه صرف نظر شده، اما این شبکه قبلا توسط افراد دیگر آموزش داده شده و وزن‌های آموزش داده شده آن در اختیار عموم قرار گرفته. بنابراین با دانلود این وزن‌ها می‌توان بدون نیاز به آموزش شبکه، یک مدل آموزش داده شده داشت.

۴ – ۳ ارزیابی عملکرد

بعد از اینکه با استفاده از شبکه‌ی عصبی توانستیم یک بازنمایی برای تصاویر پیدا کنیم وقت آن می‌رسد که با استفاده از این بازنمایی‌ها در فضای برداری جدید خوشه‌بندی را انجام دهیم. همان طور که انتظار داریم کیفیت خوشه‌بندی نهایی به کیفیت بازنمایی بدست آمده در این مرحله وابسته است. در این قسمت به ارزیابی عملکرد شبکه عصبی از نظر کیفیت و سرعت می‌پردازیم.

۴ – ۳ – ۱ ارزیابی کیفیت

برای ارزیابی شبکه‌ی عصبی از نظر دقت باید از یک دیتاست که دارای زوج داده است استفاده کنیم. دیتاست LFW دارای یک مجموعه زوج داده می‌باشد. این مجموعه داده به این صورت است که هر داده آن شامل یک زوج تصویر است. اگر این دو تصویر مربوط به یک شخص باشند، برچسب مربوط به آن ۱ و اگر مربوط به افراد مختلف باشند برچسب آن ۰ است.

در مرحله ارزیابی هر دو تصویر را به شبکه عصبی می‌دهیم و مدل برای هر کدام از تصاویر یک بردار بازنمایی تولید می‌کند. اگر مدل به درستی آموزش دیده باشد، آنگاه فاصله بردار بازنمایی تصاویری که مربوط به یک شخص هستند کم و فاصله بردار تصاویری که مربوط به افراد مختلفی هستند زیاد است. به این ترتیب می‌توانیم یک حد آستانه[1] به نام T مشخص کنیم. اگر فاصله بردار تصاویر از این حد بیشتر باشد، یعنی مدل ما پیش‌بینی کرده است که دو تصویر مربوط به افراد مختلفند و بر عکس. با توجه به پیش‌بینی های مدل و برچسب‌های موجود هر پیش‌بینی در یکی از ۴ حالت زیر قرار می‌گیرد.

۱- TP (True Positive): زمانی که جفت تصویر مربوط به یک شخص هستند و مدل به درستی این موضوع را تشخیص می‌دهد.
۲- TN (True Negative): زمانی که جفت تصویر مربوط به افراد مختلف هستند و مدل به درستی این موضوع را تشخیص می‌دهد.
۳- FP (False Positive): زمانی که جفت تصویر مربوط به افراد مختلفند، اما مدل به اشتباه پیشبینی می‌کند که مربوط به یک فرد هستند.
۴- FN (False Negative): زمانی که جفت تصویر مربوط به یک شخص هستند، اما مدل به اشتباه پیشبینی می‌کند که مربوط به افراد مختلف هستند.

با این تعاریف ساده‌ترین معیار برای ارزیابی کیفیت مدل به نام دقت[2] به صورت زیر تعریف می‌شود.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \qquad (۲ - ۴)$$

[1] threshold

[2] Accuracy

معیار دقت به ما نشان می‌دهد که مدل از بین تمام زوج تصاویر چه تعداد را به درستی تشخیص می‌دهد که مربوط به یک شخص هستند یا نیستند.

هرچند معیار دقت، تا حد خوبی کیفیت مدل را ارزیابی می‌کند، اما اشکالاتی نیز دارد. برای مثال معیار دقت بدست آمده، به مقدار انتخاب شده برای حد آستانه (T) وابسته است و واضح است که اگر T عوض شود، دقت مدل ممکن است تغییر کند. در صورتی که در حالت کلی می‌خواهیم ارزیابی ما مستقل از T باشد. علاوه بر این، دقت مدل اطلاعی راجع به عملکرد مدل راجع به زوج‌های مثبت و منفی نمی‌دهد و مدل را به صورت کلی ارزیابی می‌کند.

با توجه به آن‌چه گفته شد می‌توانیم معیارهای TPR[1] و FPR[2] را به شکل زیر تعریف کنیم:

$$TPR = Recall = \frac{TP}{TP + FN} \qquad (۳-۴)$$

$$FPR = \frac{FP}{TN + FP} \qquad (۴-۴)$$

با تغییر حد آستانه T مقدار TPR و FPR تغییر می‌کنند. به این ترتیب می‌توانیم نمودار TPR بر حسب FPR را رسم کنیم. به این نمودار، نمودار ROC[3] گفته می‌شود. شکل ۴-۸ یک نمونه از این نمودار را نشان می‌دهد. اگر مدل ما کاملا به صورت تصادفی عمل کند، آنگاه نمودار ROC خطچین آبی رنگ خواهد بود و اگر از حالت تصادفی بهتر عمل کند، مانند نمودار نارنجی رنگ بالای خطچین قرار میگیرد.

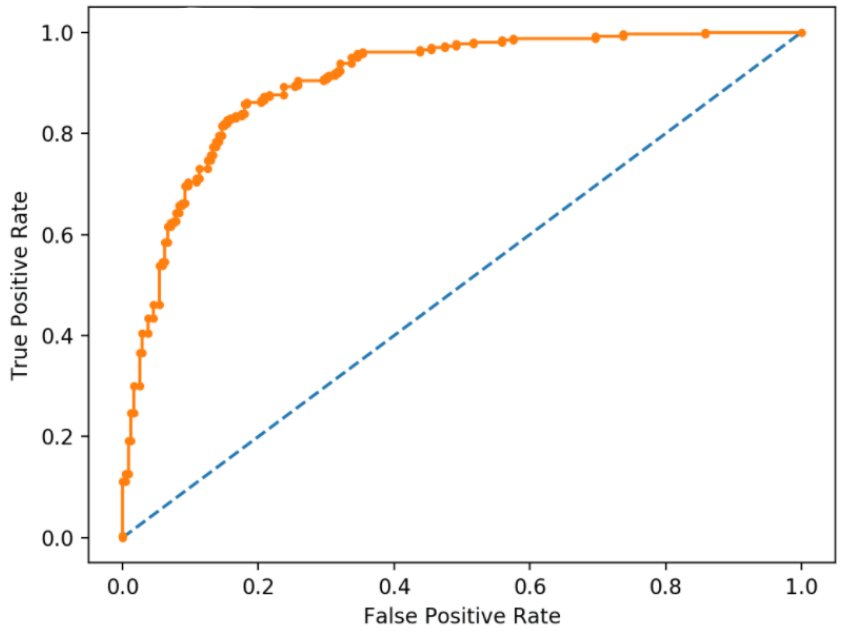
هر چه سطح زیر نمودار ROC بیشتر باشد یعنی کیفیت مدل بهتر بوده است. بنابراین اندازه سطح زیر نمودار ROC یک معیار برای ارزیابی کیفیت مدل است. به این معیار AUC[4] می‌گوییم. بیشترین مقدار برای این معیار ۱ است و اگر مدل کاملا تصادفی باشد AUC آن 0.5 خواهد بود. نکته مهم در مورد معیار AUC اینست که مستقل از حد آستانه انتخابی است و بنابراین کیفیت کلی مدل را می‌سنجد.

[1] True Positive Rate

[2] False Positive Rate

[3] Receiver Operating Characteristic

[4] Area Under Curve



شکل ۴-۸: یک نمونه نمودار ROC

با توجه به معیارهای تعریف شده می‌توانیم عملکرد مدل را روی دیتاست تست LFW ارزیابی کنیم. نتایج ارزیابی این ارزیابی در جدول ۳-۱ دیده می‌شود. لازم به ذکر است که کار detection بر روی این دیتاست با استفاده از الگوریتم MTCNN انجام شده است.

جدول ۴-۱: ارزیابی مدل بر روی دیتاست LFW. برای معیار دقت از حد آستانه ۱,۱ استفاده شده است

	دقت	مساحت زیر سطح نمودار ROC
نتایج روی مجموعه LFW	۹۷ درصد	۹۹/۵۷ درصد

بین تمام حد آستانه‌های ممکن، 1.1 بیشترین دقت را بدست می‌آورد. همانطور که دیده می‌شود نتایج بدست آمده از این آزمایش نتایج خوب و قابل قبولی هستند و استفاده از این مدل برای انجام خوشه‌بندی احتمالا مطلوب خواهد بود، زیرا در این آزمایش عملکرد خوبی داشته است.

۴ – ۳ – ۲ ارزیابی سرعت

علاوه بر کیفیت مدل، سرعت عملکرد آن نیز مهم است. برای مثال سیستمی را در نظر بگیرید که دقت بسیار بالایی دارد اما سرعت پایین. استفاده از این سیستم در یک برنامه کاربردی امکان پذیر نیست، زیرا کند بودن یک برنامه کاربرپسند نیست و کاربران ممکن است استفاده از برنامه را متوقف کنند و به سراغ برنامه ای بروند که سرعت بالاتری داشته باشد. هرچند ممکن است دقت آن کمتر باشد. به همین دلیل ارزیابی عملکرد سیستم از نظر سرعتی نیز مهم است. در این قسمت به ارزیابی سرعت عملکرد شبکه عصبی Inception-ResNet-v1 میپردازیم و متوجه می‌شویم که برای پیدا کردن بازنمایی برای یک تصویر به چه زمانی نیاز داریم.

یک نکته مهم اینست که در این بخش ورودی شبکه عصبی تنها تصویر مربوط به چهره است، بنابراین اندازه آن می‌تواند کوچک و در حد اندازه یک چهره باشد. به همین دلیل اندازه ورودی مدل را 112x112 در نظر می‌گیریم. جدول ۴-۲ سرعت اجرای شبکه عصبی این فصل را نشان می‌دهد.

جدول ۴-۲: ارزیابی سرعت شبکه عصبی Inception-ResNet-v1

مدل	اندازه بسته	اندازه تصویر	انتقال سریع از دیسک به حافظه اصلی	تعداد تصاویر پردازش شده در ثانیه
Inception-ResNet-v1	۶۴	۱۱۲ در ۱۱۲	خیر	۳۰۷
Inception-ResNet-v1	۶۴	۱۱۲ در ۱۱۲	بله	۵۱۲

با دقت در نتایج بدست آمده در این قسمت و مقایسه آن با نتایج سرعت مدل فصل قبل به نکات زیر پی می‌بریم:

۱- در فصل قبل توضیح داده شد که چگونه می‌توان تصاویر را با سرعت بیشتر از دیسک به حافظه اصلی انتقال داد. اما تاثیر آن در فصل قبل قابل توجه نبود، اما همانطور که در جدول ۴-۲ می‌بینیم

تاثیر استفاده از این قابلیت در اینجا بسیار زیاد است. دلیل این موضوع نیز اینست که در اینجا تعداد تصاویر پردازش شده در ثانیه بسیار بیشتر از مدل فصل قبل است و هرچه این تعداد بیشتر باشد مشخص است که تاثیر بهینگی انتقال از دیسک به حافظه اصلی نیز بیشتر خواهد شد.

۲- شبکه عصبی استفاده شده در این بخش بسیار بزرگتر از مدل مورد استفاده در فصل قبل است، اما با این حال می‌بینیم که مدل این بخش تعداد بیشتری تصویر در یک ثانیه را پردازش می‌کند. دلیل این امر نیز اندازه ورودی است زیرا اندازه ورودی شبکه Inception-ResNet-v1 تنها 112x112 است، اما مدل بخش قبل بسیار بزرگتر از این اندازه است. (چون ورودی شبکه MTCNN باید کل یک تصویر باشد نمی‌توانیم آن را خیلی کوچک کنیم اما ازانجایی که ورودی Inception-ResNet-v1 تنها تصویر مربوط به چهره است می‌توانیم آن را تا این اندازه کوچک کنیم.)

۴ – ۴ کارهای آینده

آنچه در این بخش انجام دادیم یک بخش مهم از برنامه مورد نظر است. در این بخش توانستیم یک تصویر از چهره را که یک بردار در فضای برداری با ابعاد بسیار بالاست را با یک تبدیل به برداری با ابعاد بسیار کمتر تبدیل کنیم. این بردار جدید را یک بازنمایی از تصویر اولیه دانستیم و دیدیم که تصاویر مربوط به یک چهره دارای بازنمایی نزدیک به همند و تصاویر مربوط به چهره‌های مختلف دارای بازنمایی با فاصله زیاد هستند. برای انجام این کار از یک شبکه عصبی بزرگ به نام Inception-ResNet-v1 استفاده کردیم. اما این شبکه تنها مدل موجود برای این‌کار نیست. روش های بسیار دیگری نیز وجود دارند که می‌توانیم از آنها استفاده کنیم. شکل ۴-۹ برخی از این روش‌ها را نشان می‌دهد.

همانطور که میبینیم دقت این الگوریتم‌ها روی دیتاست LFW بیشتر از شبکه‌ای است که ما از آن استفاده کردیم. دلیل این موضوع اینست که شبکه‌ی Inception-ResNet-v1 قدیمی‌تر از مدل‌های آورده شده است، اما با این حال عملکرد قابل قبولی دارد.

با این وجود در آینده می‌توانیم برای پیدا کردن بازنمایی از هرکدام از شبکه‌های آورده شده استفاده شود که موجب افزایش دقت این ماژول و کل در نتیجه کل برنامه می‌شود.

Rank	Model	Accuracy	Extra Training Data	Paper	Code	Result	Year	Tags
1	VarGFaceNet	99.85%	×	VarGFaceNet: An Efficient Variable Group Convolutional Neural Network for Lightweight Face Recognition			2019	
2	ArcFace + MS1MV2 + R100,	99.83%	×	ArcFace: Additive Angular Margin Loss for Deep Face Recognition			2018	
3	PFEfuse+match	99.82%	×	Probabilistic Face Embeddings			2019	
4	VarGNet	99.733%	×	VarGNet: Variable Group Convolutional Neural Network for Efficient Embedded Computing			2019	
5	CosFace	99.73%	×	CosFace: Large Margin Cosine Loss for Deep Face Recognition			2018	
6	Dyna. AdaCos	99.73%	×	AdaCos: Adaptively Scaling Cosine Logits for Effectively Learning Deep Face Representations			2019	
7	PAENet	99.67%	×	Increasingly Packing Multiple Facial-Informatics Modules in A Unified Deep-Learning Model via Lifelong Learning			2019	

شکل ۴-۹: الگوریتم‌های جایگزین برای استفاده به جای Inception-ResNet-v1

۴ – ۵ جمع‌بندی

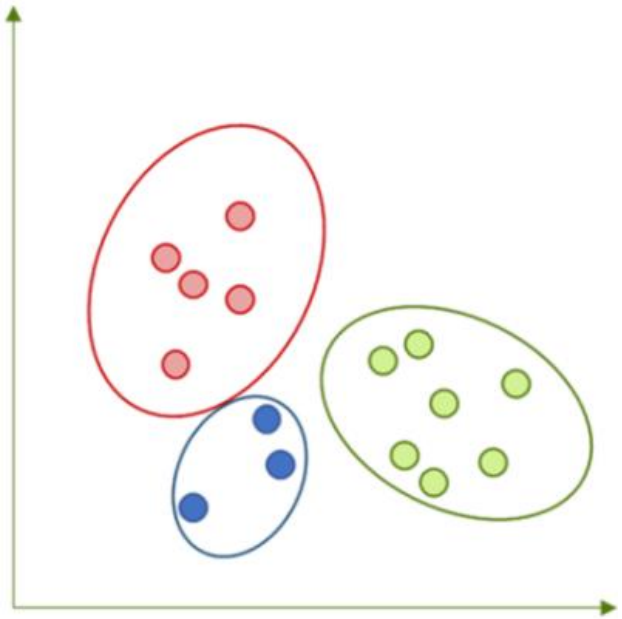
در این فصل با چگونگی استخراج ویژگی از تصاویر چهره آشنا شدیم و آموختیم که چگونه می‌توانیم از یک شبکه عصبی عمیق استفاده کنیم تا برای تصاویر بازنمایی مناسب پیدا کنیم. سپس با معماری شبکه Inception-ResNet-v1 آشنا شدیم و بررسی کردیم که چگونه می‌توانیم این شبکه را آموزش دهیم. در انتها به ارزیابی عملکرد این شبکه از نظر کیفیت و سرعت پرداختیم. در فصل بعد با چگونگی خوشه‌بندی بازنمایی‌های بدست آمده آشنا می‌شویم.

فصل پنجم خوشه‌بندی

فصل ۵ خوشه‌بندی[1]

در فصل ۳ با تشخیص چهره آشنا شدیم و فهمیدیم که چگونه می‌توانیم قسمت‌های مربوط به چهره را در یک تصویر تشخیص دهیم و جدا کنیم. سپس در فصل ۴ برای چهره‌های بدست آمده بازنمایی جدید بدست آوردیم.

هدف اصلی کار در این فصل انجام خوشه بندی است. خوشه‌بندی در کلیت به این معناست که تعدادی اشیا را در گروه‌های مختلف قرار دهیم به صورتی که اشیا با ویژگی‌های یکسان در یک گروه (خوشه) قرار گیرند و اشیا با ویژگی‌های متفاوت در خوشه‌های متفاوت. شکل ۵-۱ یک نمونه خوشه‌بندی اشیا را نشان می‌دهد همانطور که دیده می‌شود در این شکل ۳ خوشه داریم که اشیا نزدیک به هم در خوشه‌های یکسان قرار داده شده اند.



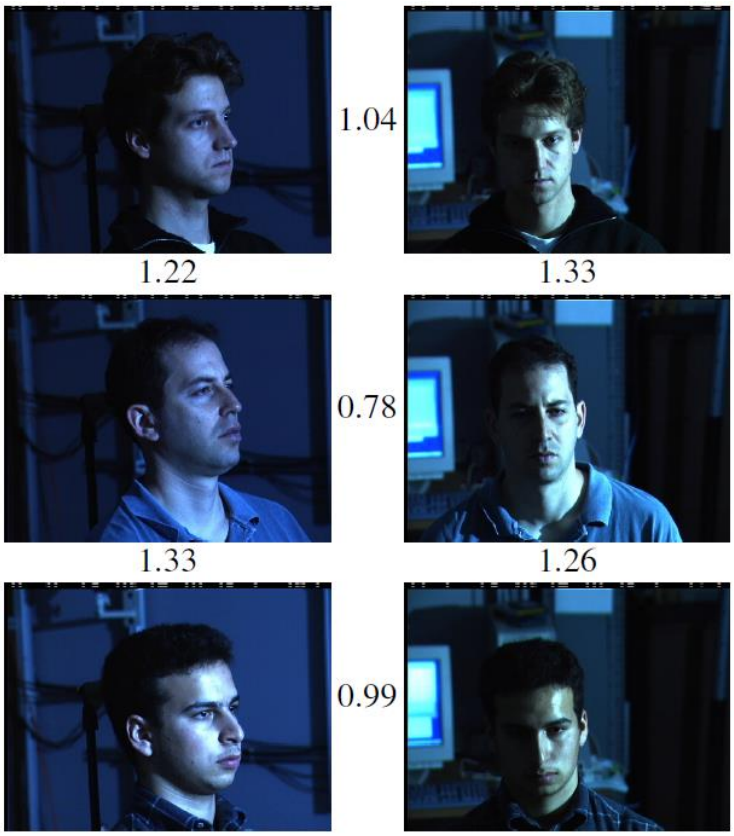
شکل ۵-۱: یک نمونه خوشه‌بندی نقاط در دو بعد

در این فصل می‌خواهیم خوشه بندی را بر روی تصاویر چهره انجام دهیم. به این صورت که تصاویر مربوط به افراد یکسان را در یک خوشه و تصاویر افراد مختلف را در خوشه‌های مختلف قرار دهیم.

[1] Clustering

۵ – ۱ یک روش ساده (Threshold Clustering)

در ابتدا برای انجام خوشه‌بندی از یک روش ساده استفاده می‌کنیم. در فصل قبل یک روش برای یافتن بازنمایی برای تصاویر چهره معرفی کردیم. همچنین گفتیم که بازنمایی بدست آمده برای تصاویر در این فضای برداری جدید دارای ویژگی‌هایی است. یکی از این ویژگی‌ها این بود که بازنمایی تصاویر مربوط به افراد مختلف دارای فاصله زیاد و بازنمایی تصاویر مربوط به افراد یکسان دارای فاصله کم هستند و در نهایت یک حد آستانه برای این فاصله بدست آوردیم و گفتیم اگر فاصله بازنمایی‌ها ازین مقدار کمتر باشد، یعنی تصاویر مربوط به یک شخص هستند و بالعکس. با استفاده از این روش می‌توانیم یکی بودن یا نبودن دو تصویر را مشخص کنیم. تصویر ۵-۲ این موضوع را نشان می‌دهد.



شکل ۵-۲: استفاده از حد آستانه برای تشخیص اینکه آیا دو تصویر مربوط به یک شخص هستند یا نه. در این مثال استفاده از حد آستانه ۱,۱ به درستی همه‌ی حالت ها را تشخیص می‌دهد[۱۴]

از این روش برای انجام خوشه بندی نیز می‌توانیم استفاده کنیم. در زیر مراحل الگوریتم خوشه‌بندی با این روش آورده شده است. این الگوریتم را Threshold Clustering می‌نامیم.

۱- ابتدا هر داده را یک خوشه جدا فرض می‌کنیم.

۲- دو خوشه که فاصله آن‌ها مینیمم است را انتخاب می‌کنیم اگر این فاصله از حد آستانه کمتر بود یعنی دو خوشه مربوط به یک فرد هستند و آن‌دو خوشه را یکی می‌کنیم و دوباره مرحله ۲ را تکرار می‌کنیم. در غیر این صورت الگوریتم پایان می‌یابد.

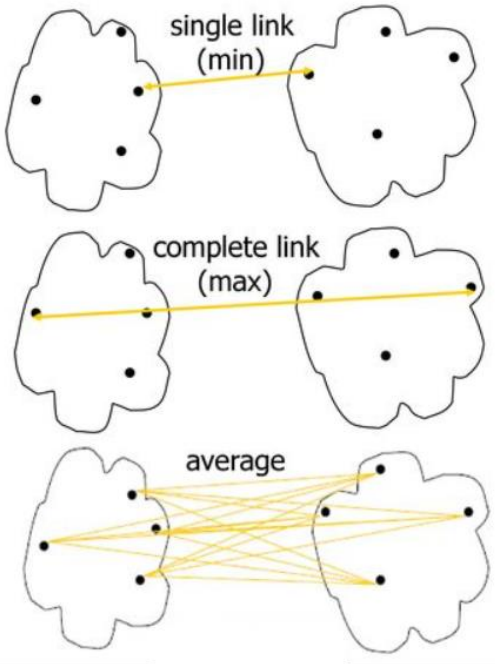
فاصله بین دو خوشه از سه روش می‌تواند محاسبه شود:

۱- Single Link: فاصله بین دو خوشه برابر است با فاصله نزدیک‌ترین داده‌های دو خوشه.

۲- Complete Link: فاصله بین دو خوشه برابر است با فاصله دورترین داده‌های دو خوشه.

۳- Average Link: فاصله بین دو خوشه برابر است با میانگین فاصله‌های بین هر جفت داده از دو خوشه.

شکل ۵-۳ این سه حال را نشان می‌دهد. در پیاده سازی انجام شده در این پروژه از حالت average link استفاده شده است.

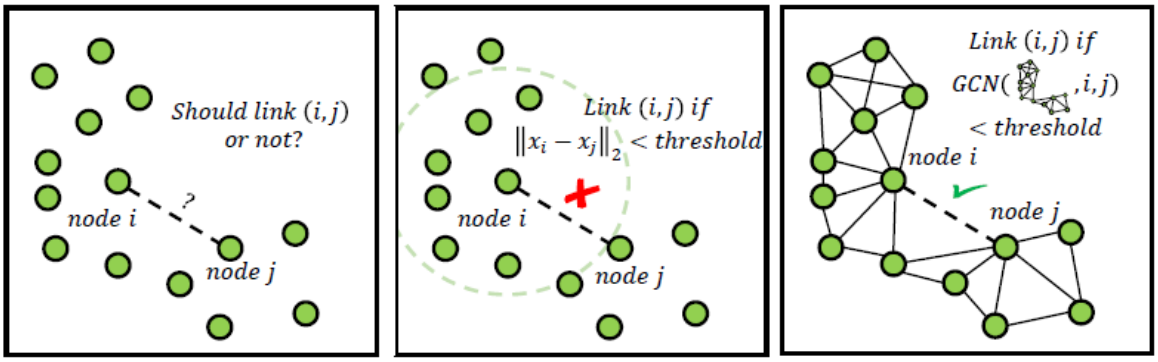


شکل ۵-۳: انواع روش‌های محاسبه فاصله بین دو خوشه

۵ - ۲ روش بهتر ([15]Clustering GCN[1])

در قسمت قبل با یک روش ساده برای خوشه‌بندی آشنا شدیم. هرچند این روش دارای سرعت بالایی است اما دقت آن پایین است. در این قسمت با یک روش جدید برای خوشه‌بندی تصاویر چهره آشنا می‌شویم. در این روش برای خوشه‌بندی از یک نوع جدید از شبکه‌های عصبی به نام شبکه عصبی گرافی استفاده می‌کنیم.

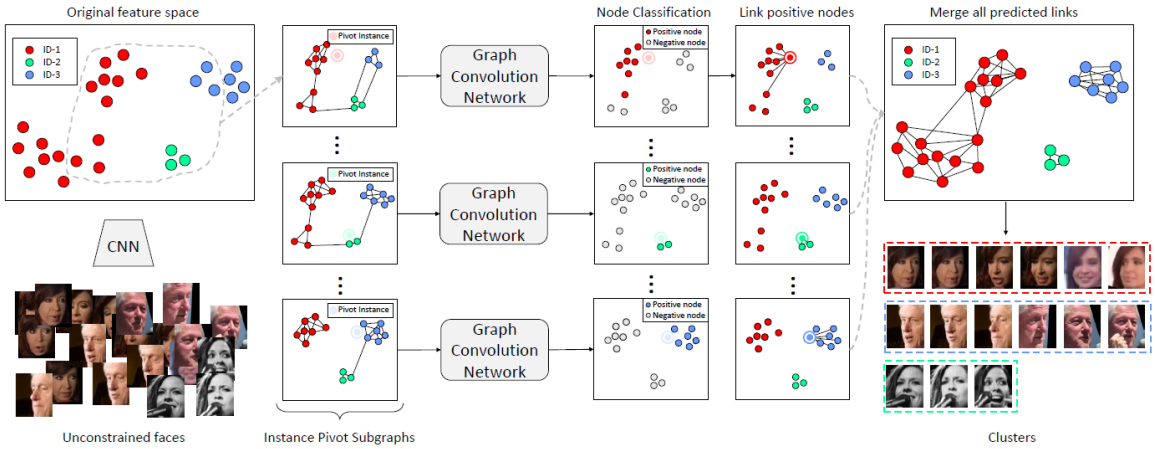
با دقت در مسئله خوشه‌بندی، این‌بار می‌توانیم به این مسئله از دید یک گراف نگاه کنیم. گره‌های گراف همان بازنمایی‌های بدست آمده از تصاویر هستند و اگر بین دو گره از گراف یالی وجود داشته باشد، یعنی دو گره مربوط به تصویر یک فرد هستند. بنابراین اگر بتوانیم یال‌ها را به درستی پیدا کنیم، مسئله خوشه‌بندی را حل کرده‌ایم. به این ترتیب مسئله خوشه‌بندی به مسئله پیشبینی یال در گراف تبدیل می‌شود. شکل ۵-۴ این موضوع را نشان می‌دهد.



شکل ۵-۴: شکل سمت چپ تبدیل مسئله خوشه‌بندی به مسئله پیشبینی یال را نشان می‌دهد. شکل وسط راه‌حل مورد استفاده در قسمت قبل را نشان می‌دهد (Threshold Clustering) و شکل سمت راست راه‌حل پیشنهادی در این قسمت است[۱۵]

شکل ۵-۵ به طور کلی روند این الگوریتم را نشان می‌دهد. در ادامه هر کدام از قسمت ها به طور دقیق‌تر تشریح می‌شوند.

[1] Graph Convolution Network



شکل ۵-۵: ساختار کلی خوشه‌بندی با استفاده از GCN [۱۵]

در ابتدا با استفاده از یک شبکه عصبی (در اینجا Inception-ResNet-v1) هر تصویر را به یک بردار بازنمایی در یک فضای برداری تبدیل می‌کنیم. در مرحله بعد هر داده را یکبار به عنوان pivot انتخاب می‌کنیم و سپس برای هر کدام یک IPS[1] می‌سازیم. پس از آن هر IPS به یک شبکه عصبی گرافی داده می‌شود و این شبکه مشخص می‌کند که pivot با کدام یک از نودها در یک خوشه است و با کدام در یک خوشه نیست. در نهایت یال‌های به دست آمده در هر کدام از IPS ها باهم تلفیق می‌شوند و خوشه‌بندی نهایی به دست می‌آید.

۵ – ۲ – ۱ نحوه ساختن IPS (Instance Pivot Subgraph)

پس از اینکه برای هر تصویر بازنمایی آن را پیدا کردیم وقت آن است که IPS ها را بسازیم. هر بار یکی از گره‌ها را به عنوان pivot انتخاب می‌کنیم و برای آن یک IPS می‌سازیم. برای ساختن IPS سه مرحله را باید طی کنیم.

۱- مرحله یافتن گره‌ها (Node Discovery): فرض کنید یک pivot به نام p داریم برای پیدا کردن IPS آن از همسایگان آن تا h قدم استفاده می‌کنیم. تعداد همسایگان مورد استفاده در هر قدم می‌تواند متفاوت باشد. برای مثال برای h=2 و k1=10 و k2=5 به این صورت عمل می‌کنیم که ۱۰ عدد از

[1] Instance Pivot Subgraph

نزدیک ترین همسایگان p را پیدا می‌کنیم و برای هرکدام از این ۱۰ همسایه ۵ عدد از نزدیک‌ترین همسایگان آن‌ها را نیز پیدا می‌کنیم. دقت کنید که خود نود p در IPS آن وجود ندارد.

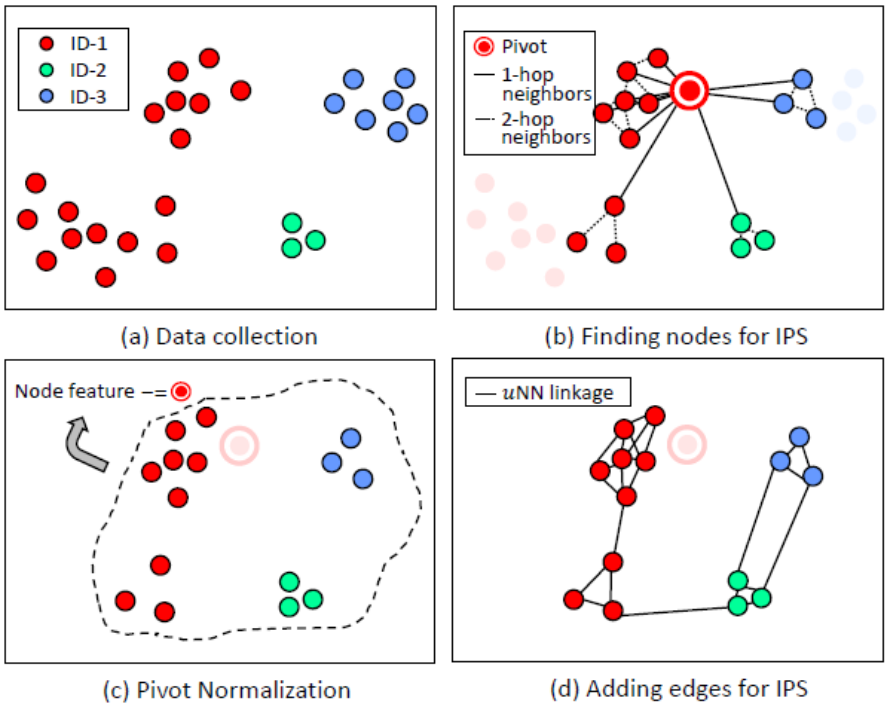
۲- بهنجارسازی ویژگی گره‌ها (Node Feature Normalization): فرض کنید x_p بردار بازنمایی مربوط به نود p باشد و x_q بردار بازنمایی مربوط به نود q که یکی از نودهای IPS است. گفتیم که نود p در IPS وجود ندارد. برای اینکه اطلاعات مربوط به p را در IPS آن بیاوریم بردارهای هرکدام از نودهای IPS را نسبت به p، با کم کردن p از q بهنجار می‌کنیم. بنابراین داریم:

$$F_p = [\dots, x_q - x_p, \dots]^T \; for\ all\ q \in V_p \qquad (۱-۵)$$

در رابطه بالا V_p گره‌های مربوط به IPS نود p را نشان می‌دهد و F_p بردارهای نرمال شده IPS است.

۳- اضافه کردن یال بین گره‌ها: برای هر نود q در نودهای IPS مربوط به p، u عدد از نزدیک‌ترین داده‌ها به آن را پیدا می‌کنیم (در بین کل داده). اگر نود r که در این u نود است در IPS مربوط به p نیز وجود داشت، آنگاه یک یال بین q و r در IPS نود p ایجاد می‌کنیم. این‌کار را برای همه‌ی گره‌ها انجام می‌دهیم تا گراف کامل شود.

شکل ۵-۶ فرایند ساخت IPS را نشان می‌دهد.



شکل ۵-۶: ساختن IPS مربوط به یک گره

۵ – ۲ – ۲ شبکه پیچشی گرافی

همانطور که در شکل ۵-۵ می‌بینیم پس از ساخته شدن، هر IPS به یک شبکه عصبی کانولوشنی داده می‌شود و خروجی این شبکه اینست که آیا یال بین نود pivot و سایر گره‌های IPS وجود دارد یا خیر.

IPS مربوط به p را به صورت زیر نشان می‌دهیم.

$$G_p(V_p, E_p) \qquad (۲ - ۵)$$

V_p مجموعه گره‌های موجود در IPS است و E_p نیز یال ها را نشان می‌دهد.

اگر بردارهای نرمال شده IPS را X بنامیم (F_p در رابطه (۵-۱)) آنگاه یک لایه از شبکه عصبی گرافی به شکل زیر تعریف می‌شود.

$$Y = \sigma([X||GX]W) \qquad (۳ - ۵)$$

علامت || اتصال دو ماتریس X و GX را نشان می‌دهد. W پارامترهای شبکه عصبی است و Y خروجی لایه. همچنین اگر N تعداد گره‌ها باشد داریم: $X \in R^{N \times d_{in}}$ و $G \in R^{N \times N}$ و $W \in R^{d_{in} \times d_{out}}$ و $Y \in R^{N \times d_{out}}$.

ماتریس G ماتریس Aggregation نام دارد. هدف این ماتریس جمع کردن اطلاعات مربوط به گره‌های مختلف است و به چند حالت مختلف می‌تواند تعریف شود. یکی از این حالات Mean Aggregation نام دارد که به صورت زیر است:

$$G = \Lambda^{-\frac{1}{2}} A \Lambda^{-\frac{1}{2}} \qquad (۴ - ۵)$$

که در رابطه بالا A همان ماتریس مجاورت است و Λ یک ماتریس قطری است و $\Lambda_{ii} = \sum_j A_{ij}$

شبکه عصبی نهایی شامل ۴ لایه از این لایه شبکه کانولوشنی گرافی است و از تابع فعالیت ReLU استفاده شده است. همچنین برای آموزش وزن‌ها از تابع هزینه cross-entropy استفاده می‌شود.

۵ – ۲ – ۳ تلفیق یال‌های نهایی

برای انجام خوشه‌بندی روی یک مجموعه داده از نوع چهره، بر روی داده‌ها پیمایش می‌کنیم هر بار یک داده را به عنوان pivot میگیریم و IPS مربوط به آن را می‌سازیم سپس با استفاده از شبکه کانولوشنی

گرافی پیش‌بینی می‌کنیم که pivot با کدام گره‌ها یال دارد - با استفاده از امتیاز softmax پایانی. به این ترتیب در نهایت یک گراف با تعدادی یال خواهیم داشت. وزن هر یال امتیاز softmax شبکه خواهد بود.

در نهایت برای پایان خوشه‌بندی به صورت زیر عمل می‌کنیم:

در هر بار تکرار یال‌هایی که از یک حد آستانه بیشتر هستند را قطع می‌کنیم خوشه‌هایی که اندازه آن‌ها از یک اندازه خاص کمتر باشد به عنوان یک خوشه در نظر گرفته می‌شوند. سایر خوشه‌ها در یک صف قرار می‌گیرند و دوباره در تکرار بعد با افزایش حد آستانه به خوشه‌های ریزتر تقسیم می‌شوند. این فرایند آنقدر ادامه پیدا می‌کند که اندازه همه‌ی خوشه‌ها از آن اندازه خاص کمتر شود.

۵ – ۳ ارزیابی خوشه‌بندی

پس از انجام خوشه‌بندی به یکی از دو روش مطرح شده وقت آن می‌رسد که عملکرد سیستم خوشه‌بند را به صورت کمی بررسی کنیم. برای ارزیابی از معیار F Score استفاده می‌کنیم.

فرض کنید برای داده i ام L(i) خوشه صحیح مربوط به این داده و C(i) خوشه ای است که برای آن داده پیش‌بینی کرده ایم. به این ترتیب تابع زیر را تعریف می‌کنیم:

$$Correct(i,j) = \begin{cases} 1\,; & if\ L(i) = L(j)\ and\ C(i) = C(j) \\ 0\,; & otherwise \end{cases} \qquad (۵-۵)$$

با تعریف بالا Precision و Recall را به شکل زیر تعریف می‌کنیم.

$$P = \mathbb{E}_i[\mathbb{E}_{j:C(j)=C(i)}[Correct(i,j)]] \qquad (۵-۶)$$

$$R = \mathbb{E}_i[\mathbb{E}_{j:L(j)=L(i)}[Correct(i,j)]] \qquad (۵-۷)$$

نحوه محاسبه Precision طبق رابطه (۵-۶): تعدادی زوج داده که الگوریتم ما در یک خوشه قرار داده را انتخاب می‌کنیم و بررسی می‌کنیم که چند درصد آنها واقعا متعلق به خوشه یکسانی هستند.

نحوه محاسبه Recall طبق رابطه (۵-۷): تعدادی زوج داده که متعلق به خوشه یکسانی هستند را انتخاب می‌کنیم و بررسی می‌کنیم که الگوریتم ما چند درصد آنها را در خوشه یکسان قرار داده است.

پس از محاسبه P و R، به راحتی می‌توانیم F Score را از رابطه زیر به دست آوریم. این معیار هم Precision و هم Recall را در نظر می‌گیرد.

$$F = \frac{2.P.R}{P+R} \qquad (۵-۸)$$

حال که معیار ارزیابی را تعریف کردیم، باید یک مجموعه برای ارزیابی انتخاب کنیم و مقدار معیار F را برای الگوریتم خود در این مجموعه بیابیم. مجموعه داده انتخابی برای ارزیابی IJB-B 512[16] نام دارد. این مجموعه شامل 18171 تصویر از 512 شخص مختلف است. متاسفانه امکان دسترسی به این مجموعه داده وجود نداشت، اما توانستیم بازنمایی های استخراج شده برای این مجموعه داده را پیدا کنیم. برای استخراج بازنمایی ها از arcface[17] استفاده شده است – در فصل قبل با استفاده از Inception-ResNet-v1 برای تصاویر بازنمایی ایجاد کردیم. arcface روش دیگری برای تولید این بازنمایی هاست. به این ترتیب ویژگی استخراج شده را به صورت آماده در اختیار داریم. با استفاده از الگوریتم‌های مطرح شده، خوشه‌بندی را انجام می‌دهیم. نتایج را در جدول ۵-۱ می‌بینیم. همانطور که در این جدول دیده می‌شود معیار F که تلفیق Precision و Recall است در الگوریتم GCN Clustering بسیار بهتر است. همچنین بالاتر بودن Recall در Threshold Clustering نشان دهنده بهتر بودن این الگوریتم نیست زیرا افزایش Recall با بالا نگه داشتن Precision مفید است و همانطور که می‌بینیم Precision بسیار پایین است.

جدول ۵-۱: ارزیابی الگوریتم‌های خوشه‌بندی

نام الگوریتم	معیار Precision	معیار Recall	معیار F
Threshold Clustering	۴۳/۳ درصد	۹۲/۸ درصد	۰/۵۹۱
GCN Clustering	۹۰/۲۳ درصد	۷۱/۲۷ درصد	۰/۷۹۶۴

برای اینکه الگوریتم GCN بتواند خوشه‌بندی را انجام دهد، باید برای هر داده k نزدیک ترین همسایه را حساب کند. محاسبه kNN وقتی تعداد داده‌ها و ابعاد هر داده زیاد باشند زمان زیادی می‌برد. این موضوع کاربرپسند نیست. روش های مختلفی برای افزایش سرعت این‌کار وجود دارد که یکی از آنها استفاده از کتابخانه faiss[18] است. اما متاسفانه این کتابخانه در سیستم عامل ویندوز قابل نصب و استفاده نیست بنابراین در این پروژه به خاطر این مشکل از Threshold Clustering استفاده شده است تا سرعت بیشتری

داشته باشد. اما با توجه به اینکه GCN نیز سازی شده است در صورت بهبود سرعت آن می‌توانیم در برنامه آنرا مورد استفاده قرار دهیم.

۵ – ۴ جمع‌بندی

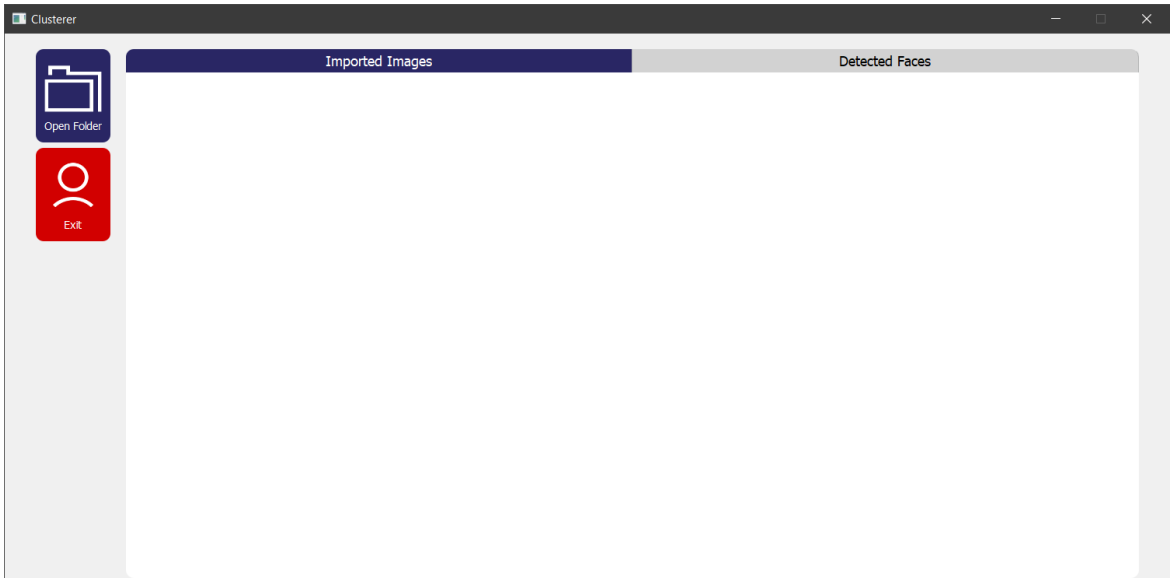
در این فصل با چگونگی خوشه‌بندی بازنمایی‌های تصاویر آشنا شدیم. برای انجام خوشه‌بندی دو الگوریتم Threshold Clustering و GCN Clustering را معرفی کردیم. همچنین گفتیم که الگوریتم Threshold Clustering دقت کمتری نسبت به الگوریتم خوشه‌بندی GCN دارد، اما سرعت اجرای آن بالاتر است. به این ترتیب با استفاده از یکی از این دو الگوریتم برای خوشه‌بندی کار اصلی پروژه پایان می‌پذیرد. در فصل بعد، به بررسی اجمالی رابط کاربری ساخته شده برای برنامه خواهیم پرداخت.

فصل ششم رابط کاربری گرافیکی

فصل ۶ رابط کاربری گرافیکی

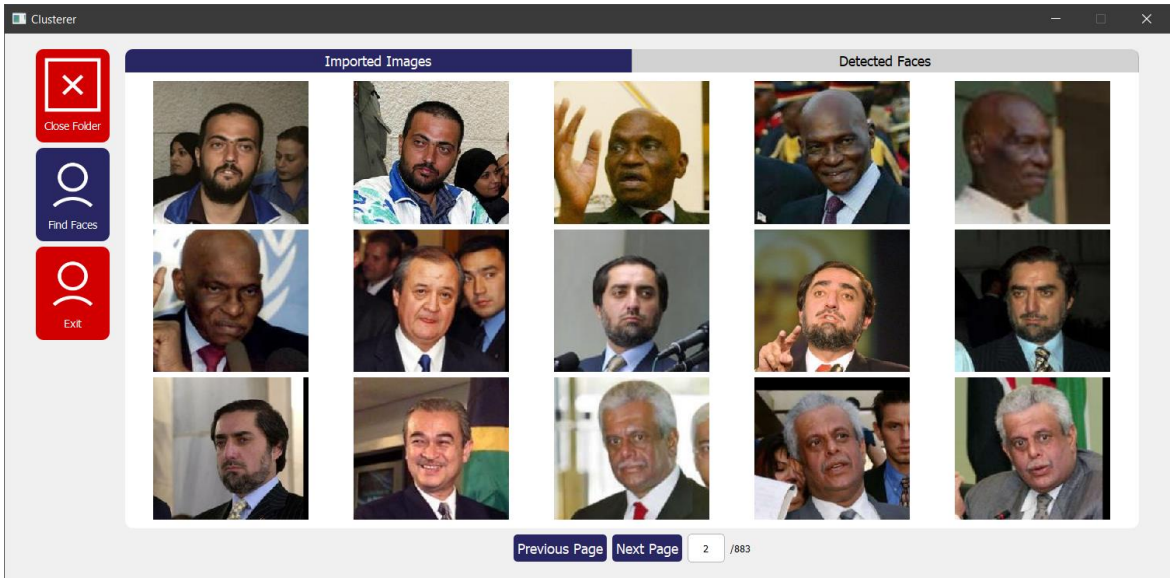
برای برنامه نوشته شده یک رابط کاربری ساده نیز طراحی شده است که در این فصل به طور خلاصه به بخش‌های مختلف آن می‌پردازیم.

با آغاز برنامه یک پنجره مطابق شکل ۶-۱ باز می‌شود. همانطور که دیده می‌شود در سمت چپ دو گزینه Open Folder و Exit وجود دارند. در سمت راست نیز دو تب به نام‌های Imported Images و Detected Faces دیده می‌شود که در حال حاضر هر دو خالی هستند.

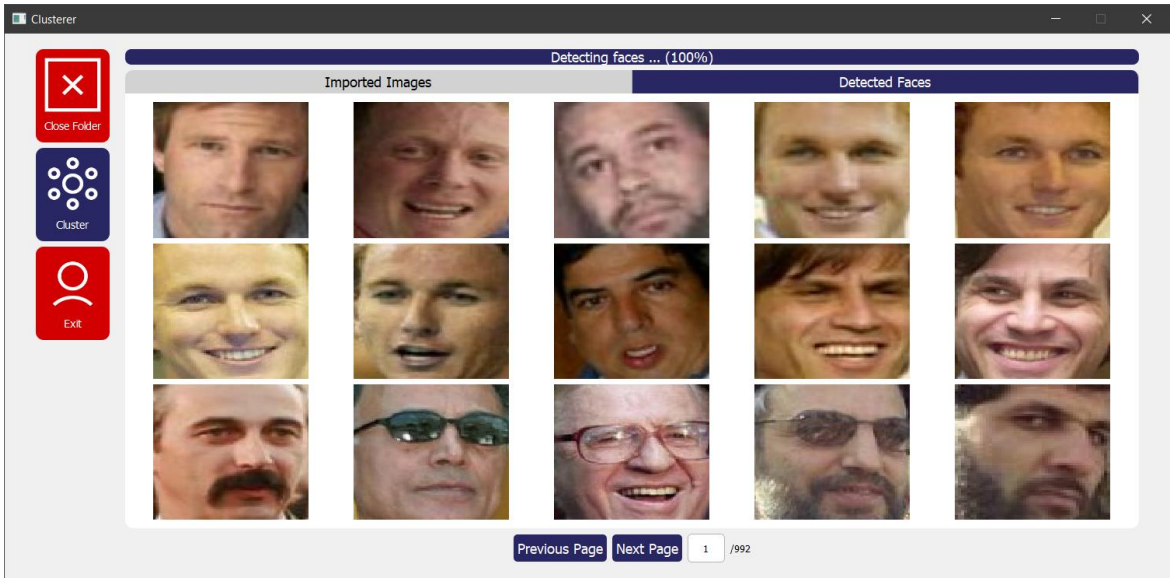


شکل ۶-۱: صفحه اصلی رابط کاربری

با کلیک بر روی گزینه Open Folder یک پنجره باز می‌شود که می‌توان در آن یک پوشه را انتخاب کرد پس از انتخاب پوشه تمام تصاویر موجود در آن پوشه به برنامه وارد می‌شوند و در تب Imported Images قابل مشاهده خواهند بود. شکل ۶-۲ رابط کاربری را پس از باز کردن یک پوشه نشان می‌دهد. همانطور که می‌بینید تصاویر وارد شده در تب Imported Images دیده می‌شوند و اگر تعداد تصاویر زیاد باشد این تصاویر در چند صفحه قرار می‌گیرند که از قسمت پایین می‌توان صفحه‌ها را تغییر داد. در سمت چپ گزینه‌ی Find Faces دیده می‌شود. با کلیک بر روی این گزینه یک progressbar در بالا ظاهر می‌شود که روند پیشرفت عملیات تشخیص چهره را نشان می‌دهد. پس از کامل شدن این فرایند تب Detected Faces چهره‌های تشخیص داده شده را نشان خواهد داد.



شکل ۶-۲: رابط کاربری پس از باز کردن یک پوشه



شکل ۶-۳: رابط کاربری پس از تشخیص چهره

در این مرحله در سمت چپ گزینه ای به نام Cluster ظاهر می‌شود با کلیک بر روی این گزینه فرایند خوشه‌بندی آغاز می‌شود و پس از اتمام تصاویر افراد مختلف در پوشه‌های مختلف ذخیره می‌شوند.

فصل هفتم جمع‌بندی

فصل ۷ جمع‌بندی

خوشه‌بندی چهره کاربردهای زیادی در زمینه‌های مختلف دارد. برای مثال می‌توان از چنین سیستمی در گوشی‌های موبایل استفاده کرد تا تصاویر گالری افراد را با توجه به اشخاص موجود در آن‌ها دسته‌بندی کرد. همچنین از این سیستم میتوان در دوربین‌های مدار بسته استفاده کرد تا افراد مختلفی که در منطقه‌ای خاص تردد کرده‌اند را تشخیص داد و سپس به صورت دستی هویت آن‌ها را شناسایی کرد به این ترتیب دیگر نیازی به مشاهده و بررسی تمام ویدیوهای ضبط شده نیست.

در این پروژه یک برنامه کاربری ساخته شد که توسط آن می‌توانیم تصاویر شخصی را بر اساس افراد موجود در آنها دسته‌بندی کنیم. برای این‌کار سه مرحله اصلی را پیمودیم:

۱- تشخیص چهره‌ی افراد در تصاویر
۲- پیدا کردن یک بازنمایی مناسب برای تصاویر
۳- خوشه‌بندی تصاویر

هر کدام از موارد بالا خود مسئله‌ای جدا بودند که در فصل‌های مختلف به بررسی و حل هر یک پرداختیم. همچنین در انتهای هر بخش راهکارهایی را برای بهبود کیفیت هر کدام از این ماژول‌ها معرفی کردیم.

آنچه در این پروژه انجام شد می‌تواند بخشی از یک برنامه بزرگتر هم باشد. در آینده می‌توانیم برنامه نوشته شده در این پروژه را گسترش دهیم و یک جعبه ابزار جامع برای پردازش چهره بسازیم. برای مثال می‌توانیم در این جعبه ابزار قابلیت‌های زیر را داشته باشیم.

۱- تشخیص حالت چهره‌ی افراد در تصاویر و دسته‌بندی به کمک آن‌ها.
۲- افزایش کیفیت چهره‌ها با استفاده از GAN[19] ها.

در حالت کلی انجام چنین کارهایی با استفاده از شبکه‌های عصبی از نظر پردازشی پرهزینه است. به همین دلیل است که اکثر برنامه‌ها با چنین قابلیت‌هایی پردازش‌ها را در سرورها انجام می‌دهند. به همین منظور می‌توانیم در آینده برنامه نوشته شده را به یک برنامه تحت وب تبدیل کنیم تا امکان استفاده از آن با سیستم‌های قوی تر و برای همگان به شکل آسان‌تری وجود داشته باشد.

References

[1] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters,* vol. 23, no. 10, pp. 1499-1503, 2016.

[2] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[3] Q. M. Rizvi, B. G. Agarwal, and R. Beg, "A review on face detection methods," *Journal of Management Development and Information Technology,* vol. 11, no. 02, 2011.

[4] G. Yang and T. S. Huang, "Human face detection in a complex background," *Pattern recognition,* vol. 27, no. 1, pp. 53-63, 1994.

[5] T. Sakai, M. Nagao, and S. Fujibayashi, "Line extraction and pattern detection in a photograph," *Pattern recognition,* vol. 1, no. 3, pp. 233-248, 1969.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, 1996, vol. 96, no. 34, pp. 226-231.

[7] T. Esler. (August 20, 2021). *Face Recognition Using Pytorch*. Available: https://github.com/timesler/facenet-pytorch

[8] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," in *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[9] S. Yang, P. Luo, C.-C. Loy, and X. Tang, "Wider face: A face detection benchmark," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5525-5533.

[10] J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, and S. Zafeiriou, "Retinaface: Single-stage dense face localisation in the wild," *arXiv preprint arXiv:1905.00641,* 2019.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research,* vol. 15, no. 1, pp. 1929-1958, 2014.

[12] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age," in *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*, 2018, pp. 67-74: IEEE.

[13] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *arXiv preprint arXiv:1411.7923,* 2014.

[14] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815-823.

[15] Z. Wang, L. Zheng, Y. Li, and S. Wang, "Linkage based face clustering via graph convolution network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1117-1125.

[16] C. Whitelam *et al.*, "Iarpa janus benchmark-b face dataset," in *proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 90-98.

[17] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4690-4699.

[18] J. Johnson, M. Douze and H. Jégou, "Billion-Scale Similarity Search with GPUs," *in IEEE Transactions on Big Data,* vol. 7, no. 3, pp. 535-547

[19] I. Goodfellow *et al.*, "Generative adversarial nets," *Advances in neural information processing systems,* vol. 27, 2014.

Abstract

The goal of this project is to build an intelligent system that performs clustering on face images. Face clustering is used in mobile phones to categorize gallery images. Such a system can also be used to review CCTV videos. This system is implemented in the form of a desktop software. The software has a graphical interface suitable for communication with user and consists of several major parts. In the first part the user identifies the folder where there are a number of image. The program finds all pictures in this folder and presents them to the user, then it processes the images and finds every face image in them. There are plenty of algorithms that we can use for face detection e.g. MTCNN which is used in this project. After detecting the face images, another algorithm is used to find a representation for each face. This representation is later used in a clustering algorithm and images are categorized according to the identity of the people in them. Finally images of different people are saved in separate folders and the user can see them.

Key Words: Face Clustering, Face Detection, Neural Networks, Deep Learning, Graph Convolutional Network

Amirkabir University of Technology
(Tehran Polytechnic)

Department of Computer Engineering

BSc Thesis

An intelligent system for face clustering

By
Mohammad Mozafari

Supervisor
Dr. Reza Safabakhsh

August 2021