

گزارش پایانی پروژه یادگیری ماشین

علیرضا حسینی

شماره دانشجویی : ۸۱۰۱۰۱۱۴۲

رضا رضائیان

شماره دانشجویی : ۸۱۰۱۰۱۱۶۸

علی ضیارزار

شماره دانشجویی : ۸۱۰۱۰۰۴۰۲

یادگیری ماشین

دکتر ابوالقاسمی و دکتر توسلی پور

تابستان ۱۴۰۲

فهرست مطالب

5	1-1- مقدمه
7	1-2- مجموعه داده
7	1-2-1- مقدمه
8	1-2-2- لود کردن داده های دیتاست
10	1-3- استخراج ویژگی های تصویر
10	1-3-1- مقدمه
12	1-3-2- استخراج ویژگی مبتنی بر روش های کلاسیک
15	1-3-3- ویژگی های کلاسیک استفاده شده و پیاده سازی آنها
24	1-3-4- استخراج ویژگی مبتنی بر روش های یادگیری عمیق
27	1-3-5- کد نهایی استخراج ویژگی های کلاسیک و ساخت CSV فیچر ها
29	1-3-6- انتخاب بهترین ستون های فیچر
32	1-4- طبقه بندی
32	1-4-1- مقدمه ای بر طبقه بند های مختلف
34	1-4-2- طبقه بندی بر روی فیچر های استخراج شده با روش های کلاسیک
42	1-4-3- طبقه بندی بر روی فیچر های استخراج شده با روش های یادگیری عمیق
44	1-4-4- مقایسه و تحلیل نهایی
47	1-5- خوش بندی
47	1-5-1- مقدمه
50	1-5-2- معیار های مورد نیاز برای مقایسه خوش بندی
52	1-5-3- خوش بندی بر بروی ویژگی های داده شده در features.csv
61	1-5-4- خوش بندی بر روی ویژگی های کلاسیک استخراج شده
67	1-5-5- خوش بندی Kmeans و GMM بر روی تک تک فیچر های استخراج شده
72	1-5-6- خوش بندی ویژگی های کلاسیک با روش BIRCH و HDBSCAN
79	1-5-7- نتیجه گیری

فهرست اشکال

8 شکل (۱-۱) کد پایتون دانلود و unzip کردن دیتاست
9 شکل (۲-۱) کد پایتون خواندن داده های دیتاست
9 شکل (۳-۱) کد پایتون نمایش 10 نمونه تصادفی از داده های دیتاست
10 شکل (۴-۱) نمونه ای از داده های مجموعه دادگان جمع آوری شده
16 شکل (۵-۱) استخراج هیستوگرام رنگ ها
18 شکل (۶-۱) کد پایتون پیاده سازی LBP
20 شکل (۷-۱) پیاده سازی HOG
21 شکل (۸-۱) پیاده سازی فیلتر گابور
23 شکل (۹-۱) کد پایتون پیاده سازی GLCM
28 شکل (۱۰-۱) کد استخراج فیچر بر روی تمامی داده های دیتاست
29 شکل (۱۱-۱) تقسیم داده های فیچر به 2 دسته آموزش و تست
30 شکل (۱۲-۱) کد پایتون تابع انتخاب بهترین فیچر ها
30 شکل (۱۳-۱) کد پایتون حذف داده های حشو
31 شکل (۱۴-۱) کد پایتون انتخاب بهترین فیچر ها و حذف redundancy و ذخیره در فایل های CSV
31 شکل (۱۵-۱) استخراج هر کدام از فیچر ها جداگانه
35 شکل (۱۶-۱) کد پایتون تفکیک داده ها به X و Y های آموزش و ارزیابی و نرم افزاری آنها
35 شکل (۱۷-۱) کد آموزش SVM و ارزیابی آن
35 شکل (۱۸-۱) دقت روی آموزش و تست و ماتریس آشفتگی هنگام آموزش طبقه بند با SVM با کرnel linear
36 شکل (۱۹-۱) کد آموزش SVM با کرnel RBF و ارزیابی آن
36 شکل (۲۰-۱) دقت روی آموزش و تست و ماتریس آشفتگی هنگام آموزش طبقه بند با SVM با کرnel RBF
37 شکل (۲۱-۱) پیاده سازی و ارزیابی Random Forrest
37 شکل (۲۲-۱) نتایج آموزش با randomForrest
38 شکل (۲۳-۱) کد بررسی 29 مدل طبقه بند کلاسیک
38 شکل (۲۴-۱) نتیجه بررسی 29 مدل طبقه بند
39 شکل (۲۵-۱) کد انتخاب بهترین طبقه بند با h2o
40 شکل (۲۶-۱) مقادیر loss پس از آموزش با الگوریتم طبقه بند
40 شکل (۲۷-۱) Variable importance در طبقه بندی به کمک h2o
43 شکل (۲۸-۱) کد پایتون خواندن داده های فایل features.csv
43 شکل (۲۹-۱) اعمال h2o بر روی فیچر های داده شده
44 شکل (۳۰-۱) انتخاب بهترین مدل طبقه بند برای features.csv
44 شکل (۳۱-۱) مقادیر loss و خطای در طبقه بندی با مدل Gradient boosting
53 شکل (۳۲-۱) کد پایتون خوش بندی با الگوریتم k-means

54 شکل (۱-۳۳) اعمال kmeans با ۲ تا کلاستر
54 شکل (۱-۳۴) خروجی ۲ کلاستر با kmeans بر روی features.csv
54 شکل (۱-۳۵) اعمال kmeans با ۳ تا کلاستر
55 شکل (۱-۳۶) خروجی ۳ کلاستر با kmeans بر روی features.csv
56 شکل (۱-۳۷) اعمال kmeans با ۶ تا کلاستر
56 شکل (۱-۳۸) خروجی ۶ کلاستر با kmeans بر روی features.csv
57 شکل (۱-۳۹) کد پایتون خوش بندی با الگوریتم GMM
58 شکل (۱-۴۰) اعمال GMM با ۲ تا کلاستر
58 شکل (۱-۴۱) خروجی ۲ کلاستر با GMM بر روی features.csv
59 شکل (۱-۴۲) اعمال GMM با ۳ تا کلاستر
60 شکل (۱-۴۳) خروجی ۳ کلاستر با GMM بر روی features.csv
61 شکل (۱-۴۴) خروجی ۶ کلاستر با kmeans بر روی features.csv
62 شکل (۱-۴۵) خروجی ۲ کلاستر با kmeans بر روی فیچر های کلاسیک
63 شکل (۱-۴۶) خروجی ۳ کلاستر با kmeans بر روی فیچر های کلاسیک
64 شکل (۱-۴۷) خروجی ۶ کلاستر با kmeans بر روی فیچر های کلاسیک
65 شکل (۱-۴۸) خروجی ۲ کلاستر با GMM بر روی فیچر های کلاسیک
66 شکل (۱-۴۹) خروجی ۳ کلاستر با GMM بر روی فیچر های کلاسیک
67 شکل (۱-۵۰) تفکیک ۵ کلاستر در فایل های جداگانه
68 شکل (۱-۵۱) خروجی های ۲ کلاستر برای فیچر color hist
68 شکل (۱-۵۲) خروجی های ۳ کلاستر برای فیچر color hist
69 شکل (۱-۵۳) خروجی های ۲ کلاستر برای فیچر lbpt
70 شکل (۱-۵۴) خروجی های ۳ کلاستر برای فیچر lbp
71 شکل (۱-۵۵) خروجی های ۲ کلاستر برای فیچر hog
73 شکل (۱-۵۶) کد پایتون انتخاب بهترین تعداد component در PCA
74 شکل (۱-۵۷) Variance ratio بر حسب تعداد کامپوننت های PCA
74 شکل (۱-۵۸) کاهش بعد فضای اعمال HDBSCAN
75 شکل (۱-۵۹) امتیاز Sillhoette برای تمامی فیچر ها بر حسب کمترین تعداد کلاستر
75 شکل (۱-۶۰) امتیاز Sillhoette برای فیچر LBP بر حسب کمترین تعداد کلاستر
76 شکل (۱-۶۱) امتیاز Sillhoette برای فیچر glcm بر حسب کمترین تعداد کلاستر
76 شکل (۱-۶۲) امتیاز Sillhoette برای فیچر gabor بر حسب کمترین تعداد کلاستر
78 شکل (۱-۶۳) کد پایتون انتخاب بهترین birch clustering
79 شکل (۱-۶۴) کد پایتونتابع BIRCH

۱-۱- مقدمه

در عصر دیجیتال امروز، تکثیر تصاویر جعلی چالش‌های مهمی را در حوزه‌های مختلف، از رسانه‌های اجتماعی گرفته تا پزشکی قانونی دیجیتال، ایجاد می‌کند. در نتیجه، توسعه روش‌های قابل اعتماد برای شناسایی و طبقه‌بندی تصاویر جعلی و واقعی به یک کار بسیار مهم تبدیل شده است. در این گزارش، کاربرد روش‌های طبقه‌بندی کلاسیک را برای یک مجموعه داده بزرگ شامل تصاویر جعلی و واقعی بررسی می‌شود و هدف شناسایی و استخراج ویژگی‌های کلاسیک از تصاویر، طبقه‌بندی تصاویر با استفاده از این ویژگی‌ها و متعاقباً خوش‌بندی آن‌ها در گروه‌های مجزا و آنالیز آن‌ها است.

مجموعه داده ارائه شده طیف گسترده‌ای از تصاویر، از جمله عکس‌ها، تصاویر تولید شده توسط شبکه‌های مولد و .. می‌باشد. این تصاویر به عنوان "جعلی" یا "واقعی" برچسب گذاری شده اند که نشان دهنده صحت آنهاست. با استفاده از این مجموعه داده، می‌توان بینش‌هایی در مورد اثربخشی روش‌های طبقه‌بندی کلاسیک برای تمایز بین تصاویر واقعی و دستکاری شده به دست آورد.

برای کار طبقه‌بندی، از الگوریتم‌های یادگیری ماشین کلاسیک مانند ماشین‌های بردار پشتیبانی (SVM)، رندوم فارست، و نزدیک‌ترین همسایه‌های (KNN) و adaboost و ... استفاده می‌شود. این روش‌ها سابقه اثبات شده‌ای در حل مسائل طبقه‌بندی دارند و می‌توانند بینش‌های ارزشمندی در مورد صحت تصاویر ارائه دهنند.

برای طبقه‌بندی مؤثر تصاویر، باید ویژگی‌های مرتبط را استخراج کرد که بین تصاویر جعلی و واقعی تفاوت قائل شود. ویژگی‌هایی که با روش‌های کلاسیک بدست می‌ایند و معمولاً در کارهای طبقه‌بندی تصویر به کار می‌روند شامل هیستوگرام‌های رنگی، تجزیه و تحلیل بافت و توصیف کننده‌های شکل و .. است. این ویژگی‌ها ویژگی‌های حیاتی تصاویر را ثبت می‌کنند و الگوریتم‌های طبقه‌بندی را قادر می‌سازند تا بین انواع مختلف تصاویر تمایز قائل شوند.

طبقه بندی و خوشه بندی تصاویر جعلی و واقعی چندین چالش را به همراه دارد. اولاً، مجموعه داده ممکن

است حاوی تصاویری با کیفیت، وضوح، شرایط نوری و سطوح نویز متفاوت باشد. این تغییرات استخراج ویژگی و طبقه بندی را چالش برانگیزتر می کند. علاوه بر این، وجود شبکه های مولد با دقت های بسیار بالا، که در آن تصاویر جعلی عمدتاً دستکاری می شوند تا مدل های طبقه بندی را فریب دهند، می توانند مانع از دقت روش های طبقه بندی سنتی شود. علاوه بر این، ویژگی ها و روش های سنتی ممکن است در تعمیم تکنیک های متنوع و در حال تکامل تصویر جعلی تلاش کنند.

این پروژه در چشم انداز دیجیتال امروزی از اهمیت قابل توجهی برخوردار است. اولاً، با شناسایی و طبقه بندی دقیق تصاویر جعلی، به موضوع حیاتی مبارزه با اطلاعات نادرست می پردازد. با توانایی تشخیص تصاویر واقعی و دستکاری شده، می توانیم به حداقل رساندن انتشار اطلاعات نادرست و اطلاعات نادرست کمک کنیم. ثانیاً، این پروژه می تواند با کمک به کارشناسان در شناسایی تصاویر دستکاری شده یا جعلی در طول تحقیقات، پژوهشی قانونی دیجیتال را بهبود بخشد. در نهایت، این تلاش تحقیقاتی پتانسیل پیشرفت در زمینه طبقه بندی تصاویر و کمک به توسعه تکنیک های قوی تر و قابل اعتمادتر را دارد.

در این گزارش ابتدا دیتا است آماده شده بررسی و سپس به بررسی روش های استخراج ویژگی و پیاده سازی ویژگی های مفید پرداخته می شود. در ادامه با استفاده از این مجموعه داده بزرگ، هدف ما شناسایی و استخراج ویژگی های سنتی، طبقه بندی تصاویر با استفاده از الگوریتم های مختلف یادگیری ماشین، و خوشه بندی آنها در گروه های مجزا است. در مورد چالش های مرتبط با این کار، در گزارش اولیه به طور کامل صحبت شده است فلذا در اینجا دیگر مطرح نمی شود.

۱-۲- مجموعه داده

۱-۲-۱- مقدمه

مجموعه داده شامل توزیع متعادلی از تصاویر واقعی و جعلی است که ۱۷۱۰ تصویر به عنوان واقعی و ۱۷۱۰ تصویر به عنوان جعلی برحسب گذاری شده اند. تصاویر صحنه هایی از دریا، مناظر کوهستانی و جنگل های ابیه را به تصویر می کشند. تصاویر جعلی با استفاده از تکنیک های رایج سنتر تصویر مانند DreamStudio، Stable Diffusion و OpenAI توسط DALL-E2 و ... تولید می شوند.

تصاویر واقعی: مجموعه داده شامل ۱۷۱۰ تصویر واقعی است که با دقت از منابع مختلف تهیه شده است. این تصاویر در محیط های واقعی گرفته شده اند و زیبایی های طبیعی دریا، کوه ها و جنگل ها را به نمایش می گذارند. تصاویر واقعی نمایانگر صحنه های واقعی هستند که توسط عکاسان، ماجراجویان و علاقه مندان به طبیعت گرفته شده است.

تصاویر جعلی: مجموعه داده همچنین شامل ۱۷۱۰ تصویر جعلی است که با استفاده از تکنیک های مدرن سنتر تصویر تولید شده اند. هدف این تصاویر جعلی تقلید از ویژگی های تصاویر واقعی و در عین حال ارائه تغییرات و تغییرات است. روش های تولید به کار گرفته شده عبارتند از DreamStudio، یک نرم افزار سنتر تصویر محبوب که امکان ایجاد صحنه های سورئال و روایی را فراهم می کند. Stable Diffusion، تکنیک دیگری که استفاده می شود، بر تولید تصاویر بصری منسجم و با کیفیت تمرکز دارد. علاوه بر این، DALL-E2 توسط OpenAI، یک مدل قدرتمند زبان به تصویر، تصاویر را بر اساس توضیحات متنی تولید می کند و به خروجی های خلاقانه و متنوع اجازه می دهد.

این مجموعه داده به چند دلیل دارای اهمیت است:

- آموزش و ارزیابی: مجموعه داده منبع ارزشمندی برای آموزش و ارزیابی الگوریتم های طبقه بندی

تصاویر به طور خاص برای تمایز بین تصاویر واقعی و جعلی طراحی شده است. به کمک این دیتا میشود مدل هایی را توسعه داد و اصلاح کرد که قادر به شناسایی صحت تصاویری که صحنه های دریا، کوه و جنگل را به تصویر می کشند.

- پیشرفت تشخیص تصویر جعلی: با گسترش تصاویر دستکاری شده و سنتز شده در چشم انداز دیجیتال امروزی، مجموعه داده در زمینه تشخیص تصاویر جعلی را تسهیل می کند. با کاوش در ویژگی ها و الگوهای نمایش داده شده توسط تصاویر جعلی، می توان به توسعه تکنیک های قوی تر و قابل اعتمادتر برای شناسایی محتوای بصری دستکاری شده کمک کنند.

2-1-2- لود کردن داده های دیتاست

برای راحتی ابتدا تمامی عکس ها و داده ها زیپ شده و در درایو آپلود میشود تا در نوتبوک های گوگل کولب به راحتی با دستور gdown بتوان آن ها را دانلود کرد.

Download and unzip Dataset

```
29s [1] !gdown 1tKCABs1GetwXq9pp9pKUCHIjBYNlpeaw
      Downloading...
      From: https://drive.google.com/uc?id=1tKCABs1GetwXq9pp9pKUCHIjBYNlpeaw
      To: /content/dataset_ml.zip
      100% 2.23G/2.23G [00:28<00:00, 78.7MB/s]

24s [2] !unzip -qq "./dataset_ml.zip"
```

شکل (1-1) کد پایتون دانلود و unzip کردن دیتاست

در ادامه آدرس عکس های داده های real و fake را خوانده و در یک متغیر dataset ریخته میشود.

```

real_folder = './real'
fake_folder = './fake'

# Read all files in the real and fake folders
real_files = [os.path.join(real_folder, file) for file in os.listdir(real_folder)]
fake_files = [os.path.join(fake_folder, file) for file in os.listdir(fake_folder)]

# Create the dataset
dataset = [(file, 'Real') for file in real_files] + [(file, 'Fake') for file in fake_files]

# Shuffle the dataset
random.shuffle(dataset)

```

شکل (۲-۱) کد پایتون خواندن داده های دیتابست

در نهایت نیز چند نمونه به صورت تصادفی از مجموعه عکس های واقعی و جعلی نمایش داده میشود.

```

# Display 10 real and fake images randomly
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
axes = axes.flatten()

for i in range(10):
    file_path, label = dataset[i]

    # Read the image using cv2
    image = cv2.imread(file_path)

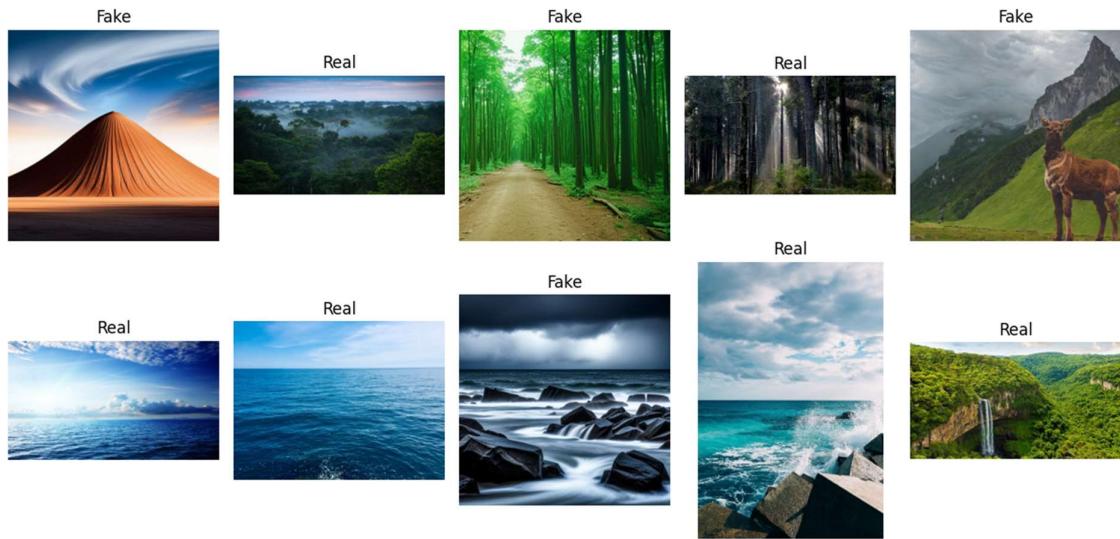
    # Convert BGR to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    axes[i].imshow(image)
    axes[i].axis('off')
    axes[i].set_title(label)

plt.tight_layout()
plt.show()

```

شکل (۳-۱) کد پایتون نمایش 10 نمونه تصادفی از داده های دیتابست



شکل (۱) نمونه ای از داده های مجموعه دادگان جمع آوری شده

۱-۳-۱-۳-۱- استخراج ویژگی های تصویر

۱-۳-۱-۴- مقدمه

استخراج ویژگی نقش مهمی در تجزیه و تحلیل تصویر، طبقه بندی و وظایف خوش بندی ایفا می کند. این شامل شناسایی و ثبت ویژگی ها یا الگوهای مرتبط از تصاویر برای نمایش آنها به شکلی معنادارتر و فشرده تر است. استخراج ویژگی پردازش کارآمد را تسهیل می کند، ابعاد را کاهش می دهد و تجزیه و تحلیل مؤثر را امکان پذیر می سازد.

تکنیک های استخراج ویژگی کلاسیک بر استخراج ویژگی های دست ساز بر اساس دانش قبلی تمرکز دارند، این روش ها به الگوریتم های از پیش تعریف شده و اکتشافی برای ثبت ویژگی های تصویر متمایز متکی هستند. ویژگی های رایج کلاسیک شامل هیستوگرام های رنگی، توصیفگرهای بافت و ویژگی های مبتنی بر شکل و .. است.

به عنوان مثال هیستوگرام های رنگی توزیع اطلاعات رنگ را در یک تصویر نشان می دهند و بینش هایی را در مورد ترکیب رنگ ارائه می دهند. توصیفگرهای بافت، الگوها و تغییرات در شدت پیکسلها را ثبت می کنند، و خصوصیات سطح را در یک تصویر ممکن می سازند. ویژگی های مبتنی بر شکل بر استخراج ویژگی های هندسی مانند خطوط، گوشها یا توصیف کننده های شکل خاص برای نمایش ساختارهای شی تمرکز می کنند. اهمیت استخراج ویژگی کلاسیک در تفسیرپذیری و اثربخشی آن در سناریوهایی است که داده های بر چسب گذاری شده محدودی در دسترس هستند. این ویژگی ها را می توان به راحتی در ک و دستکاری کرد، و به افراد اجازه می دهد تا بینشی در مورد عوامل متمایز کننده های که در طبقه بندی و خوش بندی تصاویر نقش دارند، به دست آورند.

برخلاف رویکردهای کلاسیک، استخراج ویژگی مبتنی بر یادگیری از الگوریتم های یادگیری ماشینی برای یادگیری خودکار ویژگی های متمایز از داده ها استفاده می کند. این روش ها به جای تکیه بر ویژگی های دست ساز، از شبکه های عصبی عمیق مانند شبکه های عصبی کانولوشنال (CNN) برای استخراج ویژگی های سلسله مراتبی مستقیماً از پیکسل های تصویر خام استفاده می کنند.

CNN ها بر روی مجموعه داده های بر چسب گذاری شده در مقیاس بزرگ آموزش می بینند و یاد می گیرند که ویژگی ها در سطوح مختلف انتزاع استخراج کنند. لایه های پایین تر ویژگی های سطح پایین مانند لبه ها و گوشها را ثبت می کنند، در حالی که لایه های بالاتر نمایش های پیچیده تر و انتزاعی تری را به تصویر می کشند. ویژگی های آموخته شده برای طبقه بندی یا کارهای خوش بندی خاص بهینه سازی شده اند، که منجر به نمایش های قوی تر و متمایز تر می شود.

اهمیت استخراج ویژگی مبتنی بر یادگیری در توانایی آن در گرفتن روابط پیچیده و غیرخطی موجود در تصاویر نهفته است. این روش ها در سناریوهایی با مقادیر زیادی از داده های بر چسب گذاری شده برتر هستند و

امکان کشف خودکار ویژگی‌های سطح بالا را می‌دهند که استخراج دستی آنها ممکن است چالش برانگیز باشد.

ویژگی‌های مبتنی بر یادگیری اغلب از نظر دقت و تعمیم از ویژگی‌های کلاسیک بهتر عمل می‌کنند، به ویژه در کارهایی مانند تشخیص اشیا، طبقه‌بندی تصویر و خوشبندی.

به طور کلی استخراج ویژگی هم برای طبقه‌بندی و هم برای کارهای خوشبندی شامل تصاویر ضروری است. در طبقه‌بندی، ویژگی‌های استخراج شده به عنوان ورودی برای الگوریتم‌های طبقه‌بندی عمل می‌کنند و تمايز بین کلاس‌های مختلف را امکان پذیر می‌سازند. ویژگی‌هایی که به خوبی انتخاب شده‌اند می‌توانند ویژگی‌های ذاتی اشیاء یا صحنه‌ها را ثبت کنند و دقت و قابلیت اطمینان مدل طبقه‌بندی را بهبود بخشد.

در کارهای خوشبندی، استخراج ویژگی گروه‌بندی تصاویر مشابه را بر اساس ویژگی‌های مشترک آنها تسهیل می‌کند. با استخراج ویژگی‌های مرتبط، می‌توان تصاویر را در یک فضای کم‌بعدی نشان داد و الگوریتم‌های خوشبندی کارآمد را قادر می‌سازد تا شباهت‌ها را شناسایی کرده و خوشبندی‌های متمایز را تشکیل دهند.

هر دو روش استخراج ویژگی کلاسیک و مبتنی بر یادگیری در کارهای طبقه‌بندی و خوشبندی ضروری هستند. ویژگی‌های کلاسیک بینش‌های قابل تفسیری را ارائه می‌کنند و در سناریوهایی با داده‌های محدود به خوبی کار می‌کنند. از سوی دیگر، ویژگی‌های مبتنی بر یادگیری، نمایش‌های قدرتمندی را ارائه می‌دهند که روابط پیچیده را به تصویر می‌کشد، به ویژه زمانی که مجموعه داده‌های برچسب‌دار بزرگی در دسترس هستند.

3-1- استخراج ویژگی مبتنی بر روش‌های کلاسیک

تکنیک‌های استخراج ویژگی‌های تصویر سنتی به طور گسترده‌ای در بینایی کامپیوتری و پردازش تصویر

برای تجزیه و تحلیل و درک محتوای تصاویر استفاده می شود. هدف این روش ها گرفتن اطلاعات مرتبط و متمایز از تصاویر است که می تواند بیشتر برای کارهای مختلفی مانند تشخیص اشیا، بازیابی تصویر و طبقه بندی تصویر استفاده شود. در اینجا برخی از روش های سنتی استخراج ویژگی ها از تصاویر به همراه توضیح مختصری درباره هر کدام آورده شده است:

• **هیستوگرام رنگ:** هیستوگرام های رنگی نشان دهنده توزیع مقادیر رنگ در یک تصویر هستند. آنها وقوع شدت رنگ های مختلف را کمیت می کنند و می توانند توزیع کلی رنگ را به تصویر بکشند. به طور معمول، هیستوگرام برای هر کanal رنگی (به عنوان مثال، قرمز، سبز و آبی) یا در فضاهای رنگی جایگزین (مانند HSV یا LAB) محاسبه می شود. هیستوگرام های رنگی اطلاعات ارزشمندی در مورد رنگ های غالب در یک تصویر ارائه می دهند.

• **توصیف کننده های بافت (Texture):** هدف توصیف کننده های بافت، به تصویر کشیدن آرایش فضایی الگوها یا بافت های محلی در یک تصویر است. این توصیفگرها می توانند بر اساس معیارهای آماری، مانند ماتریس های همزمان یا الگوهای باینری محلی (LBP) باشند. ماتریس های همزمانی احتمال مشترک مقادیر شدت را برای جفت پیکسل در جابجایی های فضایی مختلف کمیت می کنند. LBP ارتباط بین پیکسل ها در یک محله را با مقایسه شدت آنها رمزگذاری می کند. توصیفگرها باید برای تمایز تصاویر بر اساس ویژگی های بافتی آنها مفید هستند.

• **توصیف کننده های شکل:** توصیف کننده های شکل بر روی ثبت ویژگی های هندسی و خطوط کلی اشیاء موجود در یک تصویر تمرکز می کنند. این توصیفگرها می توانند بر اساس تحلیل کانتور مانند توصیفگرها فوریه یا کدهای زنجیره ای باشند. توصیفگرها فوریه اطلاعات شکل را با محاسبه تبدیل فوریه مرز یک شی استخراج می کنند. کدهای زنجیره ای خطوط شی را با رمزگذاری اتصال بین پیکسل های همسایه نشان می دهند. توصیف کننده های شکل برای کارهایی

که شامل تشخیص شی و بازیابی تصویر مبتنی بر شکل است، سودمند هستند.

- **تشخیص لبه:** هدف تکنیک‌های تشخیص لبه شناسایی و استخراج مرزها یا انتقال واضح بین مناطق

مختلف در یک تصویر است. الگوریتم‌های محبوب تشخیص لبه شامل آشکارساز لبه Canny،

عملگر Sobel و Laplacian of Gaussian (LoG) است. این الگوریتم‌ها نواحی با تغییرات شدت

قابل توجه را شناسایی می‌کنند و یک تصویر باینری یا گرادیان ارائه می‌دهند که لبه‌های

شناسایی شده را برجسته می‌کند. تشخیص لبه یک گام اساسی برای بسیاری از برنامه‌های پردازش

تصویر است.

- **تبدیل ویژگی ثابت مقیاس (SIFT):** SIFT یک الگوریتم استخراج ویژگی قوی است که

ویژگی‌های محلی را در یک تصویر شناسایی و توصیف می‌کند که نسبت به تغییرات مقیاس،

چرخش و روشنایی تغییر نمی‌کند. نقاط کلیدی SIFT به عنوان مناطق تصویر ثابت و متمایز

شناسایی می‌شوند و ظاهر محلی آنها با استفاده از هیستوگرام جهت‌گیری گرادیان و بزرگی توصیف

می‌شود. ویژگی‌های SIFT به طور گسترده برای کارهای مختلف از جمله تطبیق تصویر، تشخیص

اشیا و دوخت تصویر استفاده شده است.

- **هیستوگرام گرادیان‌های جهت دار (HOG):** HOG یک توصیفگر ویژگی است که توزیع

جهت‌های گرادیان محلی را در یک تصویر ثبت می‌کند. اندازه‌ها و جهت‌گیری‌های گرادیان را

در مناطق کوچک تصویر محاسبه می‌کند و هیستوگرام جهت‌گیری گرادیان را تشکیل می‌دهد.

ویژگی‌های HOG معمولاً در تشخیص عابر پیاده و وظایف تشخیص فعالیت‌های انسانی استفاده

می‌شود.

این روش‌های کلاسیک کلی استخراج ویژگی اطلاعات ارزشمندی را در مورد جنبه‌های مختلف تصویر،

مانند رنگ، بافت، شکل و لبه‌ها ارائه می‌کنند. با این حال، با پیشرفت‌های یادگیری عمیق، بسیاری از این تکنیک‌ها

با رویکردهای مبتنی بر شبکه‌های عصبی عمیق تکمیل شده یا حتی جایگزین شده‌اند، که می‌توانند به طور خودکار ویژگی‌های متمایزتر و سطح بالا را مستقیماً از داده‌های تصویر خام بیاموزند.

در این پروژه با توجه به این که ماهیت پروژه تشخیص واقعی یا جعلی بودن عکس‌ها می‌باشد می‌توان گفت استفاده از تکنیک‌هایی مانند SIFT اصلاً به کار نمی‌اید.

در این پروژه از روش‌هایی که در ادامه گفته شده است برای استخراج ویژگی استفاده شده است که انتخاب آن‌ها تماماً با روش‌های forward و backward selection صورت گرفته است.

3-3-1- ویژگی‌های کلاسیک استفاده شده و پیاده‌سازی آنها

3-3-1-1- هیستوگرام رنگ‌ها (Color Hist)

هیستوگرام‌های رنگی معمولاً برای طبقه‌بندی تصاویر و کارهای خوش‌بندی به دلیل اثربخشی آنها در ثبت و نمایش اطلاعات رنگ استفاده می‌شوند.

هیستوگرام‌های رنگی می‌توانند به تمايز بین تصاویر واقعی و مصنوعی کمک کنند. تصاویر واقعی گرفته شده از دنیای فیزیکی دارای توزیع رنگ‌های طبیعی و متنوع تری هستند، در حالی که تصاویر مصنوعی تولید شده توسط الگوریتم‌های کامپیوتری ممکن است الگوهای رنگی یکنواخت یا خاصی را نشان دهند. با مقایسه هیستوگرام رنگی تصاویر واقعی و مصنوعی، طبقه‌بندی کننده‌ها می‌توانند تمايز بین این دو نوع را بیاموزند.

هیستوگرام‌های رنگی برای خوش‌بندی تصاویر بر اساس توزیع رنگ آنها نیز مفید هستند. با گروه بندی تصاویر با هیستوگرام‌های رنگی مشابه در کنار هم، خوش‌های را می‌توان تشکیل داد و الگوهای شباختهای را در محتوای رنگی تصاویر آشکار کرد. این می‌تواند برای کارهایی مانند بازیابی تصویر مبتنی بر محتوا، سازماندهی

تصویر، یا کاوش داده های بصری مفید باشد.

هیستوگرام های رنگی از نظر محاسباتی نیز کارآمد هستند و برای مجموعه داده های تصویری در مقیاس بزرگ مناسب هستند. این محاسبه شامل شمارش تعداد پیکسل هایی است که در هر سطح رنگ قرار می گیرند، که می تواند به طور موثر پیاده سازی شود. علاوه بر این، هیستوگرام های رنگی نمایش ساده و فشرده ای از اطلاعات رنگی را ارائه می دهند و نیازهای ذخیره سازی و محاسباتی را در مقایسه با سایر نمایش های ویژگی کاهش می دهند. این سادگی، هیستوگرام های رنگی را برای تفسیر و استفاده در کاربردهای مختلف آسان می کند.

1-3-3-2- پیاده سازی هیستوگرام رنگ ها

به کمک کتاب خانه OpenCV به راحتی میتوان این ویژگی را از تصاویر استخراج کرده و در نهایت نیز داده ها نرمال میشود.

```
def extract_colorhist_features(image):
    color_hist = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
    color_hist = cv2.normalize(color_hist, color_hist).flatten()
    return color_hist
```

شکل (۱-۵) استخراج هیستوگرام رنگ ها

1-3-3-3- الگوهای باینری محلی (Local Binary Pattern)

الگوهای باینری محلی (LBP) یک توصیفگر بافت محبوب است که در بینایی کامپیوتر و پردازش تصویر استفاده می شود. با مقایسه مقادیر شدت پیکسل ها در یک محله، الگوهای محلی و تغییرات بافت را در یک تصویر ثبت می کند.

الگوریتم LBP با در نظر گرفتن هر پیکسل در یک تصویر و مقایسه مقدار شدت آن با پیکسل های همسایه کار می کند. همسایگی با انتخاب یک ناحیه دایره ای یا مستطیلی در مرکز اطراف پیکسل مورد نظر تعریف می شود. معمولاً از همسایگی 8 پیکسلی استفاده می شود که در آن مقادیر شدت هشت پیکسل مجاور با مقدار شدت پیکسل مرکزی مقایسه می شود.

مقایسه با آستانه گذاری مقادیر شدت انجام می شود که منجر به یک الگوی دودویی می شود. اگر مقدار شدت پیکسل مجاور بزرگتر یا مساوی با مقدار پیکسل مرکزی باشد، مقدار دودویی 1 تخصیص داده می شود. در غیر این صورت، مقدار 0 تخصیص داده می شود. این فرآیند برای تمام پیکسل های مجاور تکرار می شود و در نتیجه یک الگوی باینری 8 بیتی ایجاد می شود. سپس الگوی باینری حاصل به یک مقدار اعشاری تبدیل می شود که نشان دهنده اطلاعات بافت پیکسل مرکزی است.

LBP تغییرات بافت را با رمزگذاری روابط بین پیکسل ها در یک محله محلی ثبت می کند. با محاسبه LBP برای هر پیکسل در یک تصویر، می توان یک هیستوگرام از کدهای LBP ساخت که نشان دهنده توزیع الگوهای بافت محلی در تصویر است. سطلهای هیستوگرام با مقادیر اعشاری به دست آمده از محاسبه LBP مطابقت دارند و مقادیر bin تعداد دفعات وقوع هر الگو را نشان می دهند.

از مزایای LBP می توان به سادگی محاسباتی، کارایی و استحکام نسبت به تغییرات نور اشاره کرد. LBP نسبت به دگرگونی های یکنواخت تصویر ثابت است و آن را برای کاربردهای مختلفی مانند طبقه بندی بافت، تشخیص چهره و تشخیص اشیا مناسب می کند.

برنامه های افروزنی LBP، مانند LBP ثابت چرخشی (RI-LBP) و LBP یکنواخت، برای افزایش قدرت تمایز و استحکام آن توسعه یافته اند. RI-LBP با جابجایی دایره ای الگوی بیت برای انتخاب حداقل مقدار، تغییر ناپذیری چرخشی را در نظر می گیرد. Uniform LBP با طبقه بندی کدهای LBP به عنوان یکنواخت یا غیر یکنواخت بر اساس تعداد انتقال بیت ها، بر گرفتن الگوهای بافت یکنواخت و متمایز تر تمرکز می کند. این

پسوندها عملکرد LBP را در کارهای تجزیه و تحلیل بافت بهبود بخشدیه است.

1-3-3-4- پیاده سازی الگوی باینری محلی (Local Binary Pattern)

برای پیاده سازی LBP پکیج skimage در کتابخانه feature استفاده میشود لازم به ذکر است تصویر ورودی باید ابتدا gray Scale شود.

```
def extract_lbp_features(image):
    lbp = feature.local_binary_pattern(image, 24, 8, method="uniform")
    lbp_hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, 27), range=(0, 26))
    lbp_hist = lbp_hist.astype("float")
    lbp_hist /= (lbp_hist.sum() + 1e-7)
    return lbp_hist
```

شکل (1-6) کد پایتون پیاده سازی LBP

1-3-3-5- هیستوگرام گرادیان ها (HOG)

هیستوگرام گرادیان های جهت دار (HOG) یک توصیفگر ویژگی است که به طور گسترده در بینایی کامپیوتر برای تشخیص اشیا و وظایف طبقه بندی تصویر استفاده می شود. این اطلاعات شکل محلی و لبه را با تجزیه و تحلیل توزیع جهت گیری های گرادیان در یک تصویر می گیرد.

الگوریتم HOG با تقسیم یک تصویر به سلول های کوچک همپوشانی و محاسبه گرادیان ها (یعنی تغییرات شدت) در هر سلول کار می کند. گرادیان نشان دهنده بزرگی و جهت تغییرات در شدت پیکسل است. جهت گیری گرادیان ها سپس به مجموعه ای از سطلهای جهت گیری از پیش تعریف شده کوانتیزه می شوند و هیستوگرام جهت گیری های گرادیان را برای هر سلول ایجاد می کنند.

برای ثبت روابط فضایی و ارائه استحکام به تغییرات محلی، HOG از تکنیکی به نام عادی سازی بلوک استفاده می کند. سلول ها به بلوک های بزرگتر گروه بندی می شوند و هیستوگرام هر بلوک برای محاسبه تغییرات

در روشنایی و کنتراست نرمال می‌شود. این فرآیند عادی سازی قدرت تمایز توصیفگر HOG را افزایش می‌دهد.

HOG به دلیل توانایی آن در ثبت تفاوت‌های ظریف در شکل و بافت، به ویژه برای طبقه‌بندی تصاویر

واقعی و جعلی مفید است.

تصاویر واقعی تمایل به داشتن اشکال متنوع و طبیعی دارند و در نتیجه توزیع غنی‌تری از جهت‌های گرادیان

را به همراه دارند. از سوی دیگر، تصاویر جعلی یا دستکاری شده، مانند گرافیک‌های کامپیوترا یا تصاویر

فتوشاپ شده، اغلب ناهنجاری‌های خاصی را در الگوهای لبه خود نشان می‌دهند. این ناهنجاری‌ها را می‌توان

با تجزیه و تحلیل ویژگی‌های HOG تصاویر و مقایسه آنها با مجموعه داده‌های آموزشی شناسایی کرد.

با آموزش یک طبقه‌بند، مانند ماشین‌های بردار پشتیبانی (SVM) یا شبکه‌های عصبی، با نمونه‌های

برچسب‌گذاری شده از تصاویر واقعی و جعلی، می‌توان از ویژگی‌های HOG برای تمایز بین این دو کلاس

استفاده کرد. طبقه‌بند یاد می‌گیرد که الگوها و ویژگی‌های متمایز مرتبط با تصاویر واقعی و جعلی را تشخیص

دهد و امکان طبقه‌بندی دقیق را فراهم می‌کند.

مزایای استفاده از HOG برای طبقه‌بندی تصاویر واقعی و جعلی شامل استحکام آن در برابر تغییرات در

شرايط نوری، توانایی آن در گرفتن اطلاعات لبه و شکل، و اثربخشی آن در تمایز بین تفاوت‌های ظریف است.

علاوه بر این، HOG را می‌توان با سایر ویژگی‌های تصویر، مانند هیستوگرام‌های رنگی یا توصیفگرهای بافت،

ترکیب کرد تا نمایش جامع‌تری برای کارهای طبقه‌بندی ایجاد کند.

1-3-3-1- پیاده سازی هیستوگرام گرادیان‌ها (HOG)

پیاده‌سازی هیستوگرام گرادیان‌های جهت‌یافته (HOG) را می‌توان با استفاده از کتابخانه تصویری scikit که

طیف وسیعی از پردازش تصویر و عملکردهای بینایی کامپیوترا را در پایتون فراهم می‌کند، انجام داد. Scikit-

image مجموعه‌ای مناسب از ابزارها را برای استخراج و تجسم ویژگی‌های HOG ارائه می‌دهد.

```

def extract_hog_features(image):
    hog_features = feature.hog(image, orientations=9, pixels_per_cell=(8, 8),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hys')
    return hog_features

```

شکل (۱-۷) پیاده سازی HOG

۱-۳-۳-۷- فیلتر گابور

ویژگی های گابور به طور گسترده در پردازش تصویر و بینایی کامپیوتری به عنوان توصیف کننده بافت

استفاده می شود. آنها اطلاعات بافت در تصاویر را با تجزیه و تحلیل تغییرات محلی در شدت و فرکانس با استفاده

از فیلترهای گابور می گیرند.

فیلترهای گابور خانواده ای از فیلترهای خطی هستند که شبیه واکنش سلول های قشر بینایی انسان به محرک

های بینایی هستند. آنها توسط یک پوشش گاوی تعدل شده توسط یک موج صفحه سینوسی تعریف می شوند.

فیلترها با فرکانس مرکزی، جهت گیری و پهنه ای باند مشخص می شوند.

برای استخراج ویژگی های گابور، تصویر با مجموعه ای از فیلترهای گابور در مقیاس ها و جهت های مختلف

ترکیب می شود. فرآیند پیچیدگی یک نقشه پاسخ تولید می کند که نشان دهنده فعال شدن هر فیلتر در مکان های

فضایی مختلف است. این نقشه های پاسخ، اطلاعات بافت محلی موجود در تصویر را ضبط می کنند.

ویژگی های گابور را می توان با استخراج معیارهای آماری از نقشه های پاسخ به دست آورد. آمارهای رایج

شامل میانگین، واریانس، انرژی و آنتروپی است. این معیارها جنبه های مختلف اطلاعات بافت گرفته شده توسط

فیلترهای گابور را نشان می دهند.

ویژگی های گابور چندین مزیت را برای تجزیه و تحلیل بافت ارائه می دهد. آنها قادر به گرفتن اطلاعات

مکانی و فرکانسی هستند که آنها را برای تمایز بین بافت ها با مقیاس ها و جهت گیری های مختلف مناسب می

کنند. آنها همچنین در برابر تغییرات نویز و روشنایی مقاوم هستند که آنها را برای کاربردهای مختلف در دنیای

واقعی مناسب می کند.

ویژگی های گابور با موقعيت در کارهایی مانند طبقه بندی بافت، تشخیص چهره، تشخیص اثر انگشت و تجزیه و تحلیل تصویر پزشکی به کار گرفته شده است. در طبقه بندی بافت، ویژگی های گابور اطلاعات متمایز کننده ای در مورد الگوهای بافت موجود در تصویر ارائه می کنند و امکان طبقه بندی دقیق بین کلاس های بافت مختلف را فراهم می کنند.

اثربخشی ویژگی های گابور در توانایی آنها برای گرفتن ویژگی های بافت در مقیاس ها و جهت گیری های متعدد نهفته است. با در نظر گرفتن طیف وسیعی از فرکانس ها و جهت گیری ها، ویژگی های گابور می توانند الگوهای بافت مختلفی از جمله لبه ها، گوش ها و نظم های بافت را نشان دهند.

1-3-3-8- پیاده سازی فیلتر گابور

برای پیاده سازی میتوان گابور کرنل های open CV استفاده کرد.

```
def extract_gabor_features(image):
    # Convert the image to grayscale if necessary
    if len(image.shape) > 2:
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        gray_image = image

    # Define parameters for Gabor filters
    ksize = (7, 7) # Size of the filter kernel
    sigma = 2.0 # Standard deviation of the Gaussian envelope
    theta = np.pi / 4 # Orientation of the normal to the parallel stripes
    lambd = 10.0 # Wavelength of the sinusoidal factor
    gamma = 0.5 # Spatial aspect ratio
    psi = 0 # Phase offset

    # Create Gabor filter kernel
    gabor_kernel = cv2.getGaborKernel(ksize, sigma, theta, lambd, gamma, ktype=cv2.CV_32F)

    # Apply Gabor filter to the image
    filtered_image = cv2.filter2D(gray_image, cv2.CV_32F, gabor_kernel)

    # Flatten the filtered image into a feature vector
    gabor_features = filtered_image.flatten()

    return gabor_features
```

شکل (1-۸) پیاده سازی فیلتر گابور

9-3-3-1- ویژگی با glcm

ماتریس همزمانی سطح خاکستری (GLCM) یک روش استخراج ویژگی است که معمولاً در تجزیه و تحلیل تصویر و بینایی ماشین استفاده می شود. روابط فضایی بین جفت پیکسل ها در یک تصویر را بر اساس شدت سطح خاکستری آنها کمیت می کند. GLCM اطلاعات بافت با ارزشی را ارائه می دهد که می تواند برای کارهای خوش بندی استفاده شود.

برای محاسبه GLCM، تصویر به مناطق کوچک تقسیم می شود، معمولاً از رویکرد پنجره کشویی استفاده می شود. برای هر پیکسل در تصویر، رابطه بین شدت سطح خاکستری و شدت سطح خاکستری پیکسل های همسایه آن تحلیل می شود. سپس GLCM با شمارش تعداد دفعاتی که ترکیبات خاصی از مقادیر سطح خاکستری در همسایگی تعریف شده رخ می دهند ساخته می شود.

GLCM ویژگی های بافت مختلفی مانند کتراست، همگنی، آنتروپی، انرژی و همبستگی را ثبت می کند. این ویژگی ها توزیع شدت پیکسل ها و رابطه بین پیکسل های همسایه را توصیف می کنند. برای مثال، کتراست تغییرات محلی در شدت پیکسل ها را اندازه گیری می کند، در حالی که همگنی شباهت پیکسل های همسایه را کمی می کند.

GLCM برای خوش بندی مفید است زیرا نمایش فشرده ای از ویژگی های بافت در یک تصویر را ارائه می دهد. با استخراج ویژگی های GLCM از چندین تصویر، الگوریتم های خوش بندی می توانند بافت های مشابه را با هم گروه بندی کنند و امکان کشف الگوهای مقوله های معنادار را در یک مجموعه داده فراهم کنند.

الگوریتم های خوش بندی، مانند K-means، خوش بندی سلسله مراتبی، یا خوش بندی مبتنی بر چگالی، می توانند از ویژگی های GLCM به عنوان ورودی برای تقسیم کردن تصاویر به خوش های مجزا استفاده کنند. ویژگی های GLCM استخراج شده، اطلاعات بافت زیرین را می گیرد و به الگوریتم خوش بندی اجازه می دهد تا شباهت ها و تفاوت های بین تصاویر را بر اساس ویژگی های بافت آنها شناسایی کند.

یکی از مزیت‌های استفاده از ویژگی‌های GLCM برای خوشه‌بندی، توانایی آن‌ها در ثبت تفاوت‌های ظریف در بافت است، که ممکن است به راحتی از طریق روش‌های دیگر استخراج ویژگی قابل تشخیص نباشد. ویژگی‌های GLCM اندازه‌گیری کمی از روابط فضایی بین پیکسل‌ها را فراهم می‌کند و امکان شناسایی ویژگی‌های بافتی منحصر به فرد را فراهم می‌کند.

1-3-3-10 glcm - پیاده سازی

کد زیر کد پیاده سازی glcm می‌باشد.

```
def extract_glcm_features(image):
    # Convert the image to grayscale if necessary
    if len(image.shape) > 2:
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        gray_image = image

    # Compute the GLCM
    distances = [1] # Distance between pixels
    angles = [0, np.pi/4, np.pi/2, 3*np.pi/4] # Angles for co-occurrence
    glcm = greycomatrix(gray_image, distances=distances, angles=angles, symmetric=True, normed=True)

    # Extract desired GLCM properties
    contrast = greycoprops(glcm, 'contrast')[0, 0]
    correlation = greycoprops(glcm, 'correlation')[0, 0]
    energy = greycoprops(glcm, 'energy')[0, 0]
    homogeneity = greycoprops(glcm, 'homogeneity')[0, 0]

    return contrast, correlation, energy, homogeneity
```

شکل (۱-۹) کد پایتون پیاده سازی GLCM

در اینجا خلاصه‌ای از آنچه که کد انجام می‌دهد آورده شده است:

این عملکرد با بررسی اینکه آیا تصویر در مقیاس خاکستری یا فرمت رنگی است شروع می‌شود. اگر تصویر

بیش از دو بعد داشته باشد (که نشان دهنده کانال‌های رنگی است)، با استفاده از تابع cv2.cvtColor() OpenCV

با پرچم COLOR_BGR2GRAY، تصویر را به مقیاس خاکستری تبدیل می‌کند. در غیر این صورت، تصویر

ورودی را مستقیماً به متغیر gray_image اختصاص می‌دهد.

مرحله بعدی محاسبه GLCM با استفاده از تابع `greycomatrix()` از مژول `skimage.feature` است.

GLCM بر اساس `gray_image` و پارامترهای خاصی مانند فاصله بین پیکسل‌ها و زوایای همزمانی محاسبه می‌شود. در این حالت، فاصله بر روی $[0, 90]$ تنظیم می‌شود که نشان‌دهنده پیکسل‌های مجاور است، و زاویه‌ها روی $[0, 45, 90, 135]$ درجه تنظیم می‌شوند که نشان‌دهنده چهار متفاوت است. جهت $(0, 45, 90, 135)$ درجه،

پس از محاسبه GLCM، کد ویژگی‌های GLCM مورد نظر را با استفاده از تابع `greycoprops()` از مژول `skimage.feature`

استخراج می‌کند. مقادیر کنترast، همبستگی، انرژی و همگنی را از GLCM محاسبه می‌کند. این ویژگی‌ها اطلاعاتی در مورد توزیع و روابط شدت پیکسل در تصویر ارائه می‌دهند.

در نهایت، تابع ویژگی‌های GLCM استخراج شده را بر می‌گرداند: کنترast، همبستگی، انرژی و همگنی.

4-3-1- استخراج ویژگی مبتنی بر روش‌های یادگیری عمیق

4-3-1- مقدمه

در سال‌های اخیر، یادگیری عمیق توجه و محبوبیت قابل توجهی را در زمینه بینایی کامپیوتر، به ویژه برای کارهای استخراج ویژگی، به دست آورده است. این تغییر به سمت رویکردهای یادگیری عمیق عمدتاً ناشی از چندین مزیت است که آنها نسبت به روش‌های سنتی استخراج ویژگی ارائه می‌دهند.

- یادگیری سرتاسری: مدل‌های یادگیری عمیق، مانند شبکه‌های عصبی کانولوشنال (CNN)،

می‌توانند نمایش ویژگی‌ها را مستقیماً از داده‌های خام بیاموزند. بر خلاف تکنیک‌های سنتی

استخراج ویژگی که بر ویژگی‌های دست ساز تکیه می‌کنند، مدل‌های یادگیری عمیق می‌توانند به

طور خودکار نمایش‌های سلسله مراتبی و انتزاعی داده‌ها را بیاموزند. این یادگیری انتها به انتها (

END to END) نیاز به مهندسی ویژگی های دستی را از بین می برد و باعث می شود فرآیند

کارآمدتر و کمتر به تخصص دامنه وابسته باشد.

• سلسله مراتب ویژگی ها: مدل های یادگیری عمیق ویژگی ها را به شیوه ای سلسله مراتبی یاد می

گیرند و نمایش های سطح پایین و سطح بالا را به تصویر می کشند. روش های سنتی اغلب بر ثبت

ویژگی های خاص یا الگوهای محلی تمرکز می کنند، در حالی که مدل های یادگیری عمیق

می توانند ویژگی های پیچیده و انتزاعی را با ترکیب ویژگی های سطح پایین تر یاموزند. این نمایش

سلسله مراتبی به مدل های یادگیری عمیق اجازه می دهد تا ویژگی های پیچیده تر و متمایز تری را به

تصویر بکشند، که منجر به بهبود عملکرد در وظایف مختلف می شود.

• سازگاری با داده ها: مدل های یادگیری عمیق بسیار انعطاف پذیر و سازگار با انواع مختلف داده ها

هستند. آن ها می توانند طیف وسیعی از داده های ورودی، از جمله تصاویر، متن و صدا را مدیریت

کنند، که امکان انتقال یادگیری و استفاده از مدل های از پیش آموزش دیده شده را در مجموعه های

داده در مقیاس بزرگ فراهم می کند. روش های استخراج ویژگی سنتی اغلب به انتخاب دقیق و

سفرارشی سازی ویژگی ها بر اساس دانش خاص نیاز دارند و کاربرد آن ها را برای انواع مختلف

داده ها محدود می کند.

• استحکام نسبت به تغییرات: مدل های یادگیری عمیق نسبت به تغییرات در داده های ورودی

استحکام نشان می دهند. آنها می توانند تغییرات در مقیاس، چرخش، شرایط نوری، و درهم

ریختگی پس زمینه را مدیریت کنند، که آنها را برای سناریوهای دنیای واقعی مناسب می کند.

روش های استخراج ویژگی سنتی ممکن است با چنین تغییراتی دست و پنجه نرم کند، زیرا به شدت

به ویژگی های دست ساز متکی هستند که ممکن است در شرایط مختلف به خوبی تعمیم نشوند.

• مقیاس پذیری: مدل های یادگیری عمیق می توانند به خوبی به مجموعه داده های بزرگ و فضاهای

ویژگی های با ابعاد بالا مقیاس شوند. آن ها می توانند از حجم عظیمی از داده ها بیاموزند و امکان

استخراج ویژگی های ظریفتر و متمایز تر را فراهم کنند. روش های استخراج ویژگی های سنتی

معمولاً هنگام برخورد با مجموعه داده های بزرگ یا فضاهای ویژگی با ابعاد بالا با چالش هایی مواجه

می شوند که منجر به مسائلی مانند ناکارآمدی محاسباتی یا برآزش بیش از حد می شود.

• بهبود عملکرد: استخراج ویژگی مبتنی بر یادگیری عمیق عملکرد برتر را در وظایف مختلف بینایی

کامپیوتر از جمله طبقه بندی تصویر، تشخیص اشیا و تقسیم بندی نشان داده است. توانایی مدل های

یادگیری عمیق برای یادگیری ویژگی های خاص کار و قابلیت های یادگیری نمایش سلسله مراتبی

آن ها به عملکرد پیشرفته در بسیاری از معیارها و مسابقات کمک کرده است.

در نتیجه، تغییر به سمت یادگیری عمیق برای وظایف استخراج ویژگی به دلیل مزایایی است که نسبت به

روش های سنتی ارائه می دهد. یادگیری انتها به انتهای، نمایش ویژگی های سلسله مراتبی، سازگاری با داده ها،

استحکام در برابر تغییرات، مقیاس پذیری و بهبود عملکرد برخی از مزایای کلیدی هستند که یادگیری عمیق را

به یک انتخاب ترجیحی برای استخراج ویژگی در بینایی کامپیوتر تبدیل می کند. این مزایا انقلابی در این زمینه

ایجاد کرده است و همچنان باعث پیشرفت در کاربردهای مختلف می شود، از تجزیه و تحلیل و تشخیص تصویر

گرفته تا پردازش زبان طبیعی و تشخیص گفتار.

2-4-3-1- پیاده سازی استخراج ویژگی به کمک شبکه های عصبی عمیق

برای تمایز بین تصاویر واقعی و جعلی، می توان از مدل های آموزش عمیق از قبل آموزش دیده مانند

ResNet یا EfficientNet برای استخراج ویژگی هایی استفاده کنید که ویژگی های متمایز تصاویر را ثبت می

کند. این مدل ها به طور گسترده ای در وظایف بینایی کامپیوتری استفاده می شوند و در مسائل مختلف طبقه

بندی تصاویر موثر هستند.

ResNet و EfficientNet معماری های شبکه عصبی کانولوشنی عمیق هستند که عملکرد قابل توجهی در چالش های طبقه بندی تصاویر در مقیاس بزرگ به دست آورده اند. آنها از لایه های متعددی از لایه های کانولوشن، ادغام و کاملاً متصل تشکیل شده اند که قادر به یادگیری نمایش های سلسله مراتبی پیچیده از تصاویر هستند.

برای به دست آوردن ویژگی ها با استفاده از ResNet یا EfficientNet، باید این مراحل را انجام داد: ابتدا، بارگیری مدل از پیش آموزش دیده: وزنه های از پیش آموزش دیده را برای ResNet یا EfficientNet یا API کتابخانه بارگیری کرده و تصاویر هم پیش پردازش کرده و به مدل داده میشود. در نهایت نیز ویژگی های استخراج شده در یک فایل CSV ذخیره میشود. در این پروژه فایل CSV داده شده همان ویژگی های استخراج شده بر روی تصاویر به کمک EfficientNET میباشد. و نیازی به پیاده سازی این مراحل نبوده و خروجی های ویژگی های استخراج شده به کمک یادگیری عمیق داده شده است.

1-3-5- کد نهایی استخراج ویژگی های کلاسیک و ساخت csv فیچر ها

با توجه به محدودیت های RAM سیستم از batch استفاده کرده و 100 تایی فیچر را گرفته و رم را خالی کرده و مجددا اقدام به استخراج فیچر شده است، برای مشاهده روند استخراج نیز از tqdm استفاده میشود. به کمک کد زیر بر روی تمامی داده ها فیچر ها را استخراج کرده.

لازم به ذکر است به جهت نرمال سازی نهایی تمامی عکس ها به 64 در 64 ریسایز میشود.

* با توجه به اینکه تعداد محدودی از عکسها درست رایت نشده بود از try و except استفاده شده است.

```

# Set the batch size
batch_size = 100

# Initialize empty lists to store features and labels
features = []
labels = []

# Process images in batches to avoid memory issues
with tqdm(total=len(dataset), desc='Extracting features') as pbar:
    for i in range(0, len(dataset), batch_size):
        batch = dataset[i:i+batch_size]
        batch_features = []
        batch_labels = []

        for file_path, label in batch:
            try:
                image = cv2.imread(file_path)

                # Check if image is successfully read
                if image is None:
                    # Skip the image and its label
                    continue

                # Resize
                resized_image = cv2.resize(image, (64, 64))

                # Convert the resized image to grayscale
                gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)

```

```

# Extract features
hog_features = extract_hog_features(gray_image)
lbp_features = extract_lbp_features(gray_image)
colorhist_features = extract_colorhist_features(resized_image)
glcm_features = extract_glcm_features(resized_image)

# Combine features
combined_features = np.concatenate((hog_features, lbp_features, colorhist_features, resized_image))

batch_features.append(combined_features)
batch_labels.append(label)

except Exception as e:
    print(f"Error processing image: {file_path}")
    print(f"Error message: {str(e)}")

pbar.update(1)

features.extend(batch_features)
labels.extend(batch_labels)

# Convert labels to numeric format
label_encoder = LabelEncoder()
numeric_labels = label_encoder.fit_transform(labels)

# Create a DataFrame with features and labels
df = pd.DataFrame(features)
df['Label'] = numeric_labels

# Save the DataFrame as a CSV file
df.to_csv('image_features4_64_64.csv', index=False)

```

Extracting features: 100%|██████████| 3417/3421 [04:12<00:00, 13.56it/s]

شکل (۱-۱۰) کد استخراج فیچر بر روی تمامی داده های دیتابیس

3-1-3-6- انتخاب بهترین ستون های فیچر

با توجه به اینکه به عنوان مثال ، مثلا HOG به ازای هر پیکسل از هر عکس یک مقدار به عنوان ویژگی

برمیگرداند حجم فایل CSV فیچر ها خیلی میشود اما میدانیم بعضی از ستون های فیچر ها خیلی مفید میتوانند

نشاند به همین دلیل آنها حذف میشود.

این کار باعث میشود که سرعت محاسبات هنگام آموزش طبقه بند بالا برود همچنین داده های نامربوط هم

حذف شود.

برای این کار ابتدا داده ها به دو دسته آموزش و تست تقسیم بندی میشود (دلیل این کار این است که انتخاب

فیچر های نامربوط فقط روی داده های آموزش انجام شود و فقط روی داده های تست ستون های مربوطه پاک

شود.

```
df = pd.read_csv('image_features.csv')
X = df.drop('Label', axis=1)
y = df['Label']

# Splitting X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Creating new dataframes for training and testing data
train_df = pd.concat([X_train, y_train], axis=1)
test_df = pd.concat([X_test, y_test], axis=1)

train_df.to_csv('image_features_train.csv')
test_df.to_csv('image_features_test.csv')
```

شکل (1-11) تقسیم داده های فیچر به 2 دسته آموزش و تست

```

# Remove irrelevant features and select important features
def Feature_Importance_IG(data):
    features = data.drop(['Label'],axis=1).values # "Label" should be changed to the target class variable name if different
    labels = data['Label'].values

    # Extract feature names
    feature_names = list(data.drop(['Label'],axis=1).columns)

    # Empty array for feature importances
    feature_importance_values = np.zeros(len(feature_names))
    model = lgb.LGBMRegressor(verbose = -1)
    model.fit(features, labels)
    feature_importances = pd.DataFrame({'feature': feature_names, 'importance': model.feature_importances_})

    # Sort features according to importance
    feature_importances = feature_importances.sort_values('importance', ascending = False).reset_index(drop = True)

    # Normalize the feature importances to add up to one
    feature_importances['normalized_importance'] = feature_importances['importance'] / feature_importances['importance'].sum()
    feature_importances['cumulative_importance'] = np.cumsum(feature_importances['normalized_importance'])

    cumulative_importance=0.90 #

    # Make sure most important features are on top
    feature_importances = feature_importances.sort_values('cumulative_importance')

    # Identify the features not needed to reach the cumulative_importance
    record_low_importance = feature_importances[feature_importances['cumulative_importance'] > cumulative_importance]

    to_drop = list(record_low_importance['feature'])
    # print(feature_importances.drop(['importance'],axis=1))
    return to_drop

```

شکل (۱۲-۱) کد پایتون تابع انتخاب بهترین فیچر ها

```

# Remove redundant features
def Feature_Redundancy_Pearson(data):
    correlation_threshold=0.90 # Only remove features with the redundancy>90%
    features = data.drop(['Label'],axis=1)
    corr_matrix = features.corr()

    # Extract the upper triangle of the correlation matrix
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k = 1).astype(np.bool))

    # Select the features with correlations above the threshold
    # Need to use the absolute value
    to_drop = [column for column in upper.columns if any(upper[column].abs() > correlation_threshold)]

    # Dataframe to hold correlated pairs
    record_collinear = pd.DataFrame(columns = ['drop_feature', 'corr_feature', 'corr_value'])

    # Iterate through the columns to drop
    for column in to_drop:

        # Find the correlated features
        corr_features = list(upper.index[upper[column].abs() > correlation_threshold])

        # Find the correlated values
        corr_values = list(upper[column][upper[column].abs() > correlation_threshold])
        drop_features = [column for _ in range(len(corr_features))]

        # Record the information (need a temp df for now)
        temp_df = pd.DataFrame.from_dict({'drop_feature': drop_features,
                                           'corr_feature': corr_features,
                                           'corr_value': corr_values})
        record_collinear = record_collinear.append(temp_df, ignore_index = True)
    # print(record_collinear)
    return to_drop

```

شکل (۱۳-۱) کد پایتون حذف داده های حشو

```

def Auto_Feature_Engineering(df):
    drop1 = Feature_Importance_IG(df)
    dfh1 = df.drop(columns = drop1)

    drop2 = Feature_Redundancy_Pearson(dfh1)
    dfh2 = dfh1.drop(columns = drop2)

    return dfh2

df = pd.read_csv('image_features_train.csv')
df_selected = Auto_Feature_Engineering(df)
df_selected.to_csv('image_features_selected_train.csv')

df = pd.read_csv('image_features_test.csv')
df_selected = Auto_Feature_Engineering(df)
df_selected.to_csv('image_features_selected_test.csv')

```

شکل (۱۴) کد پایتون انتخاب بهترین فیچر ها و حذف redundancy و ذخیره در فایل های CSV

برای آنکه بتوان از همه فیچر ها جداگانه برای ایجاد clustering استفاده کرد نیز جداگانه نیز از همه فیچر ها

را در CSV های مختلف ریخته میشود.

```

extract_features(feature_extractor=extract_hog_features, dataset=dataset, name='hog')

Extracting features: 100%|██████████| 3417/3421 [05:03<00:00, 11.27it/s]

extract_features(feature_extractor=extract_lbp_features, dataset=dataset, name='lbp')

Extracting features: 100%|██████████| 3417/3421 [04:23<00:00, 12.99it/s]

extract_features(feature_extractor=extract_colorhist_features, dataset=dataset, name='hist')

Extracting features: 100%|██████████| 3417/3421 [04:11<00:00, 13.61it/s]

extract_features(feature_extractor=extract_glcm_features, dataset=dataset, name='glcm')

Extracting features: 100%|██████████| 3417/3421 [06:31<00:00,  8.73it/s]

extract_features(feature_extractor=extract_gabor_features, dataset=dataset, name='gabor')

Extracting features: 100%|██████████| 3417/3421 [04:11<00:00, 13.60it/s]

```

شکل (۱۵) استخراج هر کدام از فیچر ها جداگانه

در اینجا کار تمیز سازی داده‌ها و استخراج ویژگی‌های کلاسیک از تصاویر مطابق با نیاز مساله و انتخاب بهترین ویژگی‌ها به کمک سورت کردن بر اساس Value انجام شده است.

در ادامه به بررسی روش‌های مختلف طبقه‌بندی و خوشبندی پرداخته می‌شود.

۱-۴-۱- طبقه‌بندی

۱-۴-۱-۱- مقدمه‌ای بر طبقه‌بندی‌های مختلف

طبقه‌بندی یک کار اساسی در یادگیری ماشین و داده کاوی است که شامل دسته بندی نقاط داده به کلاس‌ها یا دسته‌های از پیش تعریف شده است. به طور گسترده در زمینه‌های مختلف از جمله تشخیص تصویر، تجزیه و تحلیل متن و ... استفاده می‌شود. هدف طبقه‌بندی ساخت مدلی است که بتواند برچسب‌های کلاس داده‌های دیده نشده را بر اساس الگوهای و روابطی که از داده‌های آموزشی برچسب‌گذاری شده به دست می‌آید، پیش‌بینی کند. چندین روش طبقه‌بندی کلاسیک وجود دارد که به طور گسترده مورد استفاده قرار گرفته و در کاربردهای مختلف مؤثر واقع شده‌اند. در اینجا به برخی از این روش‌ها پرداخته می‌شود.

ماشین‌های بردار پشتیبانی (SVM): یک الگوریتم قدرتمند برای طبقه‌بندی بازیزی است. یک ابر صفحه یا مجموعه‌ای از ابرصفحه‌ها ایجاد می‌کند که نقاط داده را به کلاس‌های مختلف جدا می‌کند و در عین حال حاشیه بین کلاس‌ها را به حداقل می‌رساند. SVM می‌تواند مشکلات طبقه‌بندی خطی و غیرخطی را از طریق استفاده از توابع هسته حل کند.

Random Forest: یک روش یادگیری گروهی است که چندین درخت تصمیم را برای پیش‌بینی ترکیب می‌کند. با انتخاب تصادفی زیرمجموعه‌هایی از ویژگی‌ها و نقاط داده، مجموعه‌ای از درخت های تصمیم را ایجاد می‌کند. پیش‌بینی نهایی با تجمعیت پیش‌بینی‌های تک درختان انجام می‌شود.

Forest در برابر برآوردهای بیش از حد مقاوم است و می‌تواند داده‌های با ابعاد بالا را مدیریت کند.

طبقه‌بندی کننده‌های بیزی: طبقه‌بندی کننده‌های بیزی بر اساس قضیه بیز و مدل‌های احتمالی ساخته شده‌اند. آنها احتمال هر کلاس را با توجه به ویژگی‌های ورودی محاسبه می‌کنند و نقطه داده را به کلاس با بیشترین احتمال اختصاص می‌دهند. طبقه‌بندی کننده ساده بیز فرض می‌کند که ویژگی‌ها با توجه به برچسب کلاس مستقل هستند، که محاسبات را ساده می‌کند.

XGBClassifier و XGBClassifier: LGBMClassifier و LGBMClassifier های تقویت گرادیان هستند که به ترتیب با نام XGBoost و LightGBM شناخته می‌شوند. این الگوریتم‌ها مجموعه‌ای از یادگیرندگان ضعیف (درخت تصمیم) را به صورت متوالی ایجاد می‌کنند که در آن هر درخت بعدی اشتباهات درختان قبلی را تصحیح می‌کند. آنها به دلیل کارایی و دقت بالا معروف هستند.

AdaBoost: AdaBoost (تقویت تطبیقی) یکی دیگر از روش‌های یادگیری گروهی (Ensemble) است که چندین طبقه‌بندی ضعیف را برای ایجاد یک طبقه‌بندی قوی ترکیب می‌کند. وزن‌های بالاتری را به نقاط داده طبقه‌بندی اشتباه اخلاقی کننده‌های بعدی را آموزش می‌دهد تا روی این نمونه‌های طبقه‌بندی اشتباه تمرکز کنند. پیش‌بینی نهایی با رای گیری وزنی بر اساس عملکرد طبقه‌بندی کننده فردی انجام می‌شود.

درخت تصمیم: درخت‌های تصمیم، طبقه‌بندی کننده‌های ساده و در عین حال قدرتمندی هستند که فضای ویژگی را بر اساس مقادیر ویژگی به مناطق تقسیم می‌کنند. هر گره داخلی نشان دهنده یک آزمایش روی یک ویژگی است و هر گره برگ مربوط به یک برچسب کلاس است. درخت‌های تصمیم قابل تفسیر هستند و می‌توانند هر دو ویژگی‌های دسته‌بندی و عددی را مدیریت کنند.

LDA (تحلیل تشخیص خطی): یک تکنیک کاهش ابعاد و طبقه‌بندی است. این داده‌ها در فضایی با ابعاد پایین تر پخش می‌کند در حالی که قابلیت تفکیک کلاس را به حداقل می‌رساند. LDA فرض می‌کند

که داده‌ها از یک توزیع گاوی چند متغیره با میانگین‌های کلاس خاص و یک ماتریس کوواریانس مشترک پیروی می‌کنند.

اینها تنها چند نمونه از روش‌های طبقه‌بندی کلاسیک هستند. هر روش نقاط قوت و ضعف خاص خود را دارد و انتخاب الگوریتم مناسب به ویژگی‌های داده‌ها و مسئله مورد نظر بستگی دارد. علاوه بر این، تکنیک‌های یادگیری گروهی، مانند bagging و boosting، می‌توانند برای ترکیب چند طبقه‌بندی کننده برای بهبود عملکرد و استحکام کلی استفاده شود.

1-4-2- طبقه‌بندی بر روی فیچر‌های استخراج شده با روش‌های کلاسیک

در اینجا ابتدا بررسی می‌شود چگونه می‌توان الگوریتم‌های کلاسیک را اعمال کرد سپس از مدل‌های آماده به بررسی و انتخاب بهترین مدل‌ها پرداخته می‌شود.

1-4-2-1- نرمال‌سازی داده‌ها و تبدیل به X_{train} و Y_{train} و ..

باید ابتدا داده‌های فیچر‌ها نرمال شده و همچنین ستون target را از بقیه موارد جدا کرد.

```
def preprocess_dataframe(df):
    # Copy the original dataframe to avoid modifying the original data
    preprocessed_df = df.copy()

    # Normalize the last column using Min-Max scaling
    scaler = MinMaxScaler()
    preprocessed_df.iloc[:, -1:] = scaler.fit_transform(preprocessed_df.iloc[:, -1:])

    return preprocessed_df

train_df = pd.read_csv('image_features_train5_64_64_normalized_.csv')
test_df = pd.read_csv('image_features_test5_64_64_normalized_.csv')

# Separate the features (X) and the labels (y) in the train dataframe
X_train = train_df.iloc[:, :-1] # All columns except the last one
y_train = train_df.iloc[:, -1] # Last column

# Separate the features (X) and the labels (y) in the test dataframe
X_test = test_df.iloc[:, :-1] # All columns except the last one
y_test = test_df.iloc[:, -1] # Last column
```

شکل (۱-۱۶) کد پایتون تفکیک داده ها به X و Y های آموزش و ارزیابی و نرمال سازی آنها

1-4-2-2- طبقه بند SVM و Random Forrest

برای پیاده سازی از کتاب خانه Scikitlearn استفاده میشود.

ابتدا برای SVM از کرnel linear استفاده میشود.

```
# Train the SVM model
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)

# Predict the labels for training and testing sets
y_train_pred = svm_model.predict(X_train)
y_test_pred = svm_model.predict(X_test)

# Calculate the accuracy of the model
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

# Print the accuracy of the model
print(f"Training Accuracy: {train_accuracy}")
print(f"Testing Accuracy: {test_accuracy}")

# Calculate the confusion matrix for the testing set
confusion_matrix_test = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix (Testing Set):")
print(confusion_matrix_test)
```

شکل (۱-۱۷) کد آموزش SVM و ارزیابی آن

نتیجه آموزش به صورت زیر میباشد.

```
Training Accuracy: 0.9388949871935602
Testing Accuracy: 0.6403508771929824
Confusion Matrix (Testing Set):
[[216 136]
 [110 222]]
```

شکل (۱-۱۸) دقت روی آموزش و تست و ماتریس آشنتگی هنگام آموزش طبقه بند با SVM با کرnel linear

مشاهده میشود که اورفیت شده است و دقت خوبی روی تست ندارد به همین دلیل از کرnel RBF استفاده

میشود.

```
# Train the SVM model
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)

# Predict the labels for training and testing sets
y_train_pred = svm_model.predict(X_train)
y_test_pred = svm_model.predict(X_test)

# Calculate the accuracy of the model
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

# Print the accuracy of the model
print(f"Training Accuracy: {train_accuracy}")
print(f"Testing Accuracy: {test_accuracy}")

# Calculate the confusion matrix for the testing set
confusion_matrix_test = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix (Testing Set):")
print(confusion_matrix_test)
```

شکل (۱-۱۹) کد آموزش SVM با کرنل RBF و ارزیابی آن

نتیجه آموزش به صورت زیر میباشد.

```
Training Accuracy: 0.880717160629345
Testing Accuracy: 0.6900584795321637
Confusion Matrix (Testing Set):
[[226 126]
 [ 86 246]]
```

شکل (۱-۲۰) دقت روی آموزش و تست و ماتریس آشونگی هنگام آموزش طبقه بند با SVM با کرنل RBF

مشاهده میشود که دقت روی تست بهتر میشود ولی همچنان خوب نیست.

لازم به ذکر است در مراحل انجام پروژه یک بار عکس ها 256 در 256 ریسایز شد که مشاهده شد در این

حالت دقت روی ترین به 100 درصد و روی تست به 74 درصد میرسد اما دلیل آنکه این کار را نکردیم این بود

که در ادامه مشاهده میشود با همین ابعاد 64 در 64 با طبقه بند های دیگر به دقت بالاتری هم میشود رسید و نیازی

نیست فیچر های بیشتری هم گرفته شود.

با توجه به بررسی روش های مناسب برای تفکیک عکس های واقعی و جعلی مشاهده شده است که random Forrest عملکرد به مراتب بهتری میتواند داشته باشد و به همین دلیل آن هم در اینجا پیاده سازی میشود.

```
# Train the Random Forest model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Predict the labels for training and testing sets
y_train_pred = rf_model.predict(X_train)
y_test_pred = rf_model.predict(X_test)

# Calculate the accuracy of the model on the training set
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {train_accuracy}")

# Calculate the accuracy of the model on the testing set
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Testing Accuracy: {test_accuracy}")

# Calculate the confusion matrix for the testing set
confusion_matrix_test = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix (Testing Set):")
print(confusion_matrix_test)
```

شکل (۱-۲۱) پیاده سازی و ارزیابی Random Forrest

نتیجه به صورت زیر میباشد.

```
Training Accuracy: 1.0
Testing Accuracy: 0.7368421052631579
Confusion Matrix (Testing Set):
[[266  86]
 [ 94 238]]
```

شکل (۱-۲۲) نتایج آموزش با randomForrest

1-4-2-3 طبقه بندی روی 29 مدل با LazyPredict

یکی از کتاب خانه های معروف جهت مقایسه و انتخاب بهترین طبقه بند lazy predict میباشد که داده ها را

روی 29 مدل آموزش میدهد و بهترین آن ها را انتخاب میکند.

```
df = pd.read_csv('image_features.csv')
features = df.drop('Label', axis=1).values
labels = df['Label'].values

# Identify missing values
missing_mask = np.isnan(features)

# Perform imputation or removal of rows/columns with missing values
imputer = SimpleImputer(strategy='mean') # Example: Impute missing values with the mean
features_imputed = imputer.fit_transform(features)

# Apply feature normalization
scaler = MinMaxScaler()
features_normalized = scaler.fit_transform(features_imputed)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_normalized, labels, test_size=0.2, random_state=42)

clf = LazyClassifier(verbosity=0, ignore_warnings=True, custom_metric=None)
models, predictions = clf.fit(X_train, X_test, y_train, y_test)

print(models)
```

شکل (۱-۲۳) کد بررسی 29 مدل طبقه بند کلاسیک

در کد فوق ابتدا داده های miss شده را هندل کرده و سپس به lazy Classifier داده میشود.

```
100%|██████████| 29/29 [07:19<00:00, 15.15s/it]
          Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
XGBClassifier          0.76          0.76          0.76          0.76
RandomForestClassifier 0.74          0.74          0.74          0.74
LGBMClassifier          0.73          0.73          0.73          0.73
AdaBoostClassifier      0.72          0.72          0.72          0.72
SVC                    0.72          0.72          0.72          0.72
NuSVC                  0.70          0.70          0.70          0.70
BaggingClassifier       0.69          0.69          0.69          0.69
ExtraTreesClassifier    0.68          0.68          0.68          0.68
LogisticRegression      0.64          0.64          0.64          0.64
PassiveAggressiveClassifier 0.64          0.64          0.64          0.64
CalibratedClassifierCV 0.63          0.64          0.64          0.63
DecisionTreeClassifier 0.63          0.63          0.63          0.63
LinearSVC               0.63          0.63          0.63          0.63
Perceptron               0.63          0.63          0.63          0.63
KNeighborsClassifier    0.62          0.62          0.62          0.60
SGDClassifier            0.62          0.61          0.61          0.62
BernoulliNB               0.61          0.61          0.61          0.61
RidgeClassifierCV         0.60          0.60          0.60          0.60
NearestCentroid            0.60          0.60          0.60          0.60
ExtraTreeClassifier        0.59          0.59          0.59          0.59
LinearDiscriminantAnalysis 0.57          0.57          0.57          0.57
RidgeClassifier            0.56          0.56          0.56          0.56
GaussianNB                 0.53          0.54          0.54          0.43
LabelSpreading               0.55          0.54          0.54          0.42
LabelPropagation              0.55          0.54          0.54          0.42
QuadraticDiscriminantAnalysis 0.55          0.53          0.53          0.42
DummyClassifier              0.49          0.50          0.50          0.32
```

شکل (۱-۲۴) نتیجه بررسی 29 مدل طبقه بند

همانطور که مشاهده میشود با توجه به ویژگی های استخراج شده بهترین دقت برای XGB Classifier میباشد.

که توضیحات آن در بالا آورده شده است که در این حالت دقت روی داده های تست 76 درصد میباشد.

لازم به ذکر است بالا بردن ابعاد تصویر و زیاد کردن فیچر ها باعث میشود که روی این مدل XGB دقت به

83 درصد هم برسرد که طبیعی هم میباشد اما هم تعداد ویژگی ها به مراتب خیلی زیاد میشود که این باعث

میشود حجم زمان پردازشی بسیار زیاد شود (حدود 8 برابر) و در نهایت نیز دقت فقط حدود 7 درصد بیشتر

میشود.

4-2-4-1- انتخاب بهترین طبقه بند به کمک H2O

یکی دیگر از پکیج های معروف برای انتخاب بهترین طبقه بند h2o میباشد که در آن میتوان گفت مشابه

روی مدل های مختلف تست کند و نتیجه را گزارش کند که کد آن به صورت زیر میباشد.

```
def select_best_classification_model(train, validation, test, max_runtime_sec=300, max_models=None, nfolds=0):

    # Start H2O cluster
    h2o.init()

    # Convert dataframes to H2O frames
    train_h2o = h2o.H2OFrame(train)
    validation_h2o = h2o.H2OFrame(validation)
    test_h2o = h2o.H2OFrame(test)

    # Set the last column as the label
    label_index = train_h2o.ncols - 1
    train_h2o[label_index] = train_h2o[label_index].asfactor()
    validation_h2o[label_index] = validation_h2o[label_index].asfactor()
    test_h2o[label_index] = test_h2o[label_index].asfactor()

    # Train H2O AutoML
    if max_models:
        aml = H2OAutoML(max_models=max_models, seed=1, nfolds=nfolds, exclude_algos=["DeepLearning"])
    else:
        aml = H2OAutoML(max_runtime_secs=max_runtime_sec, seed=1, nfolds=nfolds, exclude_algos=["DeepLearning"])
        aml.train(x=train_h2o.columns[:-1], y=train_h2o.columns[label_index], training_frame=train_h2o, validation_frame=validation_h2o)

    # Get the best model
    best_model = aml.leader

    # Print the accuracy on train, validation, and test data
    train_accuracy = best_model.model_performance(train_h2o).accuracy()
    validation_accuracy = best_model.model_performance(validation_h2o).accuracy()
    test_accuracy = best_model.model_performance(test_h2o).accuracy()
    print(f"Train Accuracy: {train_accuracy}")
    print(f"Validation Accuracy: {validation_accuracy}")
    print(f"Test Accuracy: {test_accuracy}")

    return best_model
```

شکل (۱-۲۵) کد انتخاب بهترین طبقه بند با h2o

این پکیج هم بهترین را همان XGBoost میداند با همان دقت ها ولی جز آن اطلاعات دیگری نیز میدهد که در ادامه گزارش شده است.

```
MSE: 0.029040139078203256
RMSE: 0.1704116752989749
LogLoss: 0.1485164790433812
Mean Per-Class Error: 0.010252758711926662
AUC: 0.999236933623169
AUCPR: 0.9992514550815524
Gini: 0.998473867246338
```

شکل (۱-۲۶) مقادیر loss پس از آموزش با الگوریتم طبقه بند

همچنین h2o در فضای ویژگی ها متغیر های مهم هم در طبقه بندی گزارش میکند که میتواند بررسی آن ها مفید و سودمند باشد. (اینکه کدام ستون ها مهم ترند)

Variable Importances:			
variable	relative_importance	scaled_importance	percentage
1968	195.6201171875	1.0	0.045245795399663814
2255	86.996826171875	0.4447233107855128	0.020121859929262416
1390	65.54590606689453	0.3350673080523176	0.01516038686525129
455	51.98220443725586	0.2657303613995457	0.01202318156336257
1579	46.01121139526367	0.2352069513953024	0.010642125599410523
506	45.04158020019531	0.23025024648678333	0.01041785554326316
2737	40.458457946777344	0.20682156072935126	0.009357806020999372
1471	35.54043960571289	0.18168090335846043	0.008220296981382994
1521	35.26609420776367	0.18027846376332277	0.008156842486401009
176	31.788936614990234	0.1625034125939146	0.007352596157971411
---	---	---	---
1387	1.8206892013549805	0.009307269761063825	0.00042111482333857174
1374	1.604644775390625	0.00820286174275516	0.00037114500410442977
976	1.349569320678711	0.006898929108529067	0.0003121475349212912
1532	1.3007698059082031	0.0066494684933729375	0.00030086049096766275
5983	0.8919868469238281	0.00455979098544792	0.00020631136999280802
2728	0.6874923706054688	0.0035144257169957317	0.00015901298693850568
798	0.6767730712890625	0.0034596292089958316	0.00015653367534892616
5426	0.518524169921875	0.0026506689464094054	0.0001199316248214824
194	0.1791229248046875	0.0009156671991613209	4.143009074743635e-05
1351	0.023181915283203125	0.00011850476125102962	5.361842181450095e-06
[443 rows x 4 columns]			

شکل (۱-۲۷) در طبقه بندی به کمک h2o Variable importance

1-4-2-5- مقایسه و تحلیل نهایی

جهت ارزیابی عملکرد مدل های طبقه بندی مختلف در تشخیص تصاویر واقعی و جعلی با استفاده از

ویژگی‌های سنتی استخراج شده از تصاویر، ویژگی‌های کلاسیک شامل Hog، هیستوگرام رنگی، LBF، فیلتر گابور و غیره می‌باشد. این بخش 29 مدل از جمله AdaBoost، XGBoost، Random Forest، SVM و موارد دیگر را تجزیه و تحلیل می‌کند تا بهترین مدل را بر اساس دقت تعیین کند.

راهاندازی آزمایشی شامل آموزش و آزمایش مدل‌های طبقه‌بندی با استفاده از ویژگی‌های کلاسیک استخراج شده از مجموعه داده‌ای حاوی تصاویر واقعی و جعلی بود. مجموعه داده به یک مجموعه آموزشی و یک مجموعه آزمایشی تقسیم شد. مدل‌ها بر روی مجموعه آموزشی آموزش داده شدند و در مجموعه تست مورد ارزیابی قرار گرفتند. عملکرد هر مدل از نظر دقت اندازه گیری شد که نشان دهنده درصد موارد طبقه‌بندی صحیح است.

پس از انجام آزمایشات، نتایج زیر به دست آمد: مدل XGBoost بالاترین دقت را در بین تمام مدل‌های آزمایش شده به دست آورد. دقت 76٪ در مجموعه تست و 100٪ در مجموعه آموزشی به دست آورد. این نتایج نشان می‌دهد که مدل به خوبی تعمیم می‌یابد و از برازش بیش از حد ندارد. لازم به ذکر است که افزایش ابعاد تصویر و تعداد ویژگی‌ها منجر به دقت بالاتر 83 درصدی در مجموعه تست شد. با این حال، این بهبود به قیمت افزایش قابل توجه زمان پردازش، تقریباً هشت برابر بیشتر، انجام شد. علاوه بر این، افزایش دقت نسبتاً کم بود، حدود 7٪.

مدل جنگل تصادفی (random Forrest) با دقت 74 درصد در مجموعه تست و 100 درصد در مجموعه آموزشی، دومین عملکرد برتر را داشت. Random Forest به دلیل توانایی خود در مدیریت داده‌های با ابعاد بالا و مقاومت در برابر بیش از حد شناخته شده است. اگرچه در مقایسه با XGBoost دقت کمی کمتری داشت، اما باید توجه داشت که زمان پردازش به طور قابل توجهی کوتاه‌تر بود.

شایان ذکر است که عملکرد مدل‌های طبقه‌بندی می‌تواند تحت تأثیر عوامل مختلفی از جمله کیفیت و تنوع مجموعه داده‌ها، انتخاب ویژگی‌ها و تنظیم ها پرپارامتر مدل‌ها باشد. ممکن است تحقیقات و آزمایش‌های بیشتری

برای کشف این عوامل و بهبود بالقوه دقت مدل‌ها انجام شود.

در نتیجه، تحلیل مقایسه‌ای مدل‌های طبقه‌بندی برای تشخیص تصویر واقعی و جعلی با استفاده از ویژگی‌های کلاسیک، بینش‌های ارزشمندی را در مورد عملکرد آنها ارائه می‌کند. این یافته‌ها می‌توانند به عنوان پایه‌ای برای تحقیقات بیشتر در توسعه مدل‌های قوی و کارآمد برای وظایف طبقه‌بندی تصویر باشد.

4-1-4-3- طبقه‌بندی بر روی فیچر‌های استخراج شده با روش‌های یادگیری عمیق

4-1-4-3-1- انتخاب بهترین طبقه به کمک $h2o$

همانند قبل تمامی کد‌ها این بار بروی فیچر‌های داده شده که خروجی‌های شبکه‌های عصبی عمیق میباشند اعمال می‌شود.

به کمک کد زیر ابتدا داده‌ها را بر اساس هر عکس جدا کرده و تقسیم‌بندی‌های لازم انجام می‌شود. کد زیر ابتدا داده‌های فیچر را خوانده سپس به داده‌های آموزش و ارزیابی تقسیم‌بندی شده و لیل آن‌ها نیز assign می‌شود.

همچنین تمامی داده‌های ویژگی داده شده ابتدا نرمالایز هم می‌شوند که مقادیر زیاد در بعضی فیچر‌ها مشکلی بابت بایاس به سمت آن‌ها ایجاد نکند.

```

# Separate the features from the labels
features = pd.read_csv('Features/features.csv', header=None)
labels = pd.read_csv('new_labels.csv')

# Apply Min-Max scaling to the features
scaler = MinMaxScaler()
normalized_features = scaler.fit_transform(features)

# Create a new DataFrame with the normalized features and labels
normalized_df = pd.DataFrame(normalized_features, columns=features.columns)
normalized_df['Label'] = labels['real_fake']

# Split the normalized DataFrame into train, validation, and test sets
train_val_df, test_df = train_test_split(normalized_df, test_size=0.3, random_state=42)
train_df, val_df = train_test_split(train_val_df, test_size=0.2, random_state=42)

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Fit and transform the "Label" column in train_df
train_df['Label'] = label_encoder.fit_transform(train_df['Label'])

# Transform the "Label" column in val_df and test_df using the fitted encoder
val_df['Label'] = label_encoder.transform(val_df['Label'])
test_df['Label'] = label_encoder.transform(test_df['Label'])

```

شکل (۱-۲۸) کد پایتون خواندن داده های فایل features.csv

در ادامه بهترین مدل برای طبقه بندی پیدا میشود که این کارها به کمک کد زیر انجام میشود.

```

best_model = select_best_classification_model(train_df, val_df, test_df, max_runtime_sec=300)
Checking whether there is an H2O instance running at http://localhost:54321. connected.

H2O_cluster_uptime: 1 hour 47 mins
H2O_cluster_timezone: Etc/UTC
H2O_data_parsing_timezone: UTC
H2O_cluster_version: 3.40.0.4
H2O_cluster_version_age: 2 months and 6 days
H2O_cluster_name: H2O_from_python_unknownUser_4vayef
H2O_cluster_total_nodes: 1
H2O_cluster_free_memory: 2.353 Gb
H2O_cluster_total_cores: 2
H2O_cluster_allowed_cores: 2
H2O_cluster_status: locked, healthy
H2O_connection_url: http://localhost:54321
H2O_connection_proxy: {"http": null, "https": null, "colab_language_server": "/usr/colab/bin/language_service"}
H2O_internal_security: False
Python_version: 3.10.12 final

Parse progress: |██████████| (done) 100%
Parse progress: |██████████| (done) 100%
Parse progress: |██████████| (done) 100%
AutoML progress: |██████████| (done) 100%
Train Accuracy: [[0.7961747948643494, 0.9989545216936748]]
Validation Accuracy: [[0.5378054739570671, 0.9874739039665971]]
Test Accuracy: [[0.4194223474405893, 0.9892787524366472]]

```

شکل (۱-۲۹) اعمال h2o بر روی فیچر های داده شده

مشاهده میشود که در این حالت دقت روی اکثر مدل ها بالای 95 میباشد (روی داده های تست) که نشان

از این دارد استخراج ویژگی در این مساله به کمک شبکه های یادگیری عمیق نتایج به مراتب بهتری حاصل میکند.

```
Model Details
=====
H2OGradientBoostingEstimator : Gradient Boosting Machine
Model Key: GBM_1_AutoML_8_20230704_145749
```

شکل (۱-۳۰) انتخاب بهترین مدل طبقه بند برای features.csv

```
ModelMetricsBinomial: gbm
** Reported on train data. **

MSE: 0.0018978032399366424
RMSE: 0.04356378358151002
LogLoss: 0.009632680261227435
Mean Per-Class Error: 0.0010504201680672268
AUC: 0.9999945347545885
AUCPR: 0.9999944956018948
Gini: 0.9999890695091771
```

شکل (۱-۳۱) مقادیر loss و خطا در طبقه بندی با مدل Gradient boosting

مشاهده میشود که بهترین مدل gradient boosting میباشد البته با روش های random forest هم به همین

دقت 98 مشاهده شد که میتوان رسید.

۱-۴-۴- مقایسه و تحلیل نهایی

در این بخش هدف مقایسه عملکرد روش های یادگیری ماسین کلاسیک را با رویکردهای یادگیری عمیق (EfficientNet و ResNet) برای طبقه بندی تصاویر واقعی و جعلی از مناظر دریا، کوه و جنگل مقایسه میباشد. مقایسه بر اساس دقت به دست آمده در مجموعه تست است.

دقت:

- یادگیری ماشین کلاسیک: مدل Random Forest در مجموعه آزمایشی به دقت 74٪ دست یافت، در حالی که مدل XGBoost دقت کمی بالاتر از 76٪ را به دست آورد.
- یادگیری عمیق: مدل های یادگیری عمیق به طور قابل توجهی از روش های یادگیری ماشین کلاسیک بهتر عمل کردند. هر دو EfficientNet و ResNet به دقت 98٪ در مجموعه آزمایشی هنگام استفاده در ترکیب با ماشین های تقویت گرادیان و طبقه بندی کننده های جنگل تصادفی دست یافند.

استخراج ویژگی:

- یادگیری ماشین کلاسیک: روش های یادگیری ماشین کلاسیک بر ویژگی های دست ساز مانند هیستوگرام های رنگی و هیستوگرام گرادیان های جهت دار (HOG) متکی بودند.
- یادگیری عمیق: مدل های یادگیری عمیق (EfficientNet و ResNet) به طور خود کار ویژگی هایی (CNN) را از تصاویر در طول آموزش یاد می گیرند و از قدرت شبکه های عصبی کانولوشنال (CNN) استفاده می کنند.

پیچیدگی مدل:

- یادگیری عمیق: EfficientNet و ResNet معماری های شبکه عصبی کانولوشنال عمیق هستند. آنها از چندین لایه تشکیل شده اند و به منابع محاسباتی بیشتری نیاز دارند. مدل های یادگیری عمیق معمولاً در مقایسه با مدل های یادگیری ماشین کلاسیک، سطح پیچیدگی بیشتری دارند.

ارتقای کارایی:

مدل‌های یادگیری عمیق نسبت به روش‌های یادگیری ماشین کلاسیک از نظر دقت، بهبود عملکرد قابل توجهی را نشان دادند. این پیشرفت را می‌توان به توانایی مدل‌های عمیق در گرفتن نمایش‌های پیچیده و سلسله مراتبی از تصاویر، یادگیری خودکار ویژگی‌های مرتبط نسبت داد.

آموزش انتقالی (transfer learning):

مدل‌های یادگیری عمیق، مانند ResNet و EfficientNet، در استفاده از یادگیری انتقالی مزیت دارند. با استفاده از مدل‌های از پیش آموزش دیده آموزش دیده بر روی مجموعه داده‌های در مقیاس بزرگ، آنها می‌توانند به طور موثر ویژگی‌های معناداری را از تصاویر استخراج کنند، حتی با داده‌های آموزشی محدود. این به عملکرد برتر آنها در کارهای طبقه‌بندی تصویر کمک می‌کند.

تفسیر پذیری:

یادگیری ماشین کلاسیک: رتبه‌بندی اهمیت ویژگی‌ها را ارائه می‌کنند و امکان تفسیر و درک ویژگی‌های مهمی را که در تصمیم‌گیری طبقه‌بندی نقش دارند، فراهم می‌کنند.

یادگیری عمیق: مدل‌های یادگیری عمیق، مانند ResNet و EfficientNet، معمولاً کمتر قابل تفسیر هستند. نمایش‌های سلسله مراتبی پیچیده ای که توسط این مدل‌ها آموخته می‌شوند ممکن است به راحتی قابل درک یا توضیح نباشد.

در نتیجه، رویکردهای یادگیری عمیق، به طور خاص با استفاده از ResNet و EfficientNet برای استخراج ویژگی، از روش‌های استخراج کلاسیک در طبقه‌بندی تصاویر واقعی و جعلی از مناظر دریا، کوه و جنگل بهتر عمل کردند. مدل‌های یادگیری عمیق در مجموعه آزمایشی به دقت 98٪ دست یافتند، در حالی که روش‌های یادگیری ماشین کلاسیک به دقت 74٪ و 76٪ رسیدند. این بهبود عملکرد را می‌توان به توانایی مدل‌های

یادگیری عمیق در یادگیری خودکار ویژگی‌های مرتبط و متمایز از تصاویر، با استفاده از نمایش سلسله مراتبی آنها نسبت داد. با این حال، توجه به این نکته مهم است که مدل‌های یادگیری عمیق با پیچیدگی بالاتری همراه هستند و ممکن است در مقایسه با روش‌های یادگیری ماشین کلاسیک، قابلیت تفسیر محدودی داشته باشند.

1-5- خوشبندی

1-5-1 مقدمه

خوشبندی یک تکنیک یادگیری بدون نظارت رایج است که هدف آن گروه‌بندی نقاط داده مشابه بر اساس ویژگی‌های ذاتی آنهاست. در زمینه تشخیص تصاویر واقعی و جعلی، تجزیه و تحلیل خوشبندی نقش مهمی در شناسایی الگوهای تشخیص تصاویر واقعی از تصاویر دستکاری شده یا جعلی دارد. این گزارش روش‌های مختلف خوشبندی را بررسی می‌کند و اهمیت تجزیه و تحلیل خوشبندی را در تمايز بین تصاویر واقعی و جعلی بر جسته می‌کند.

روش‌های خوشبندی:

- خوشبندی K-Means یکی از پرکاربردترین الگوریتم‌های خوشبندی است. این داده‌ها را به خوشه‌های K تقسیم می‌کند، جایی که K یک پارامتر تعريف شده توسط کاربر است. الگوریتم به طور مکرر هر نقطه داده را با نزدیکترین میانگین به خوشه اختصاص می‌دهد و مرکزهای خوش را بر این اساس به روز می‌کند. خوشبندی K-means از نظر محاسباتی کارآمد است اما نیاز به تعیین تعداد خوشه‌ها از قبل دارد.

- DBSCAN (خوشبندی فضایی برنامه‌های کاربردی با نویز مبتنی بر چگالی): DBSCAN یک الگوریتم خوشبندی مبتنی بر چگالی است که نقاط داده‌ای را که کاملاً بسته‌بندی شده‌اند را در

کنار هم قرار می‌دهد و نقاط پرت را به عنوان نویز جدا می‌کند. این خوش‌های را به عنوان مناطقی با

چگالی بالا تعریف می‌کند که توسط مناطق با چگالی کم از هم جدا شده‌اند. DBSCAN نیازی

به تعیین تعداد خوش‌های از قبل ندارد و می‌تواند خوش‌هایی از اشکال دلخواه را مدیریت کند.

• خوش‌بندی سلسله مراتبی: خوش‌بندی سلسله مراتبی سلسله مراتبی از خوش‌های را با ادغام یا تقسیم

بازگشتهای خوش‌های بر اساس معیار تشابه ایجاد می‌کند. خوش‌بندی سلسله مراتبی با هر نقطه داده به

عنوان یک خوش‌گاه شروع می‌شود و متوالی شبیه‌ترین خوش‌های را تا زمانی که یک خوش-

باقی می‌ماند ادغام می‌کند. خوش‌بندی سلسله مراتبی تقسیمی با تمام نقاط داده در یک خوش-

شروع می‌شود و به صورت بازگشتهای خوش‌های را تا زمانی که هر نقطه داده در خوش‌های خود قرار می-

گیرد تقسیم می‌کند.

• همچنین روش‌های دیگری مانند GMM و ... نیز وجود دارد.

اهمیت تجزیه و تحلیل خوش‌بندی برای تشخیص تصاویر واقعی و جعلی:

• شناسایی الگو: تجزیه و تحلیل خوش‌بندی به شناسایی الگوهای درون داده‌ها کمک می‌کند و امکان

تشخیص ویژگی‌ها یا ویژگی‌هایی را می‌دهد که تصاویر واقعی و جعلی را تشخیص می‌دهند. با

گروه بندی تصاویر مشابه با هم، خوش‌بندی می‌تواند ویژگی‌ها یا ناهنجاری‌های مشترکی را که

ممکن است نشان دهنده دستکاری یا جعل تصویر باشد، آشکار کند. این تجزیه و تحلیل می‌تواند

به درک ساختار زیربنایی مجموعه داده و استخراج بینش معنی دار کمک کند.

• تشخیص ناهنجاری: روش‌های خوش‌بندی همچنین می‌توانند برای تشخیص ناهنجاری‌ها یا نقاط

پرت در یک مجموعه داده استفاده شوند. در مورد تشخیص تصویر جعلی، تجزیه و تحلیل

خوش‌بندی می‌تواند تصاویری را شناسایی کند که به طور قابل توجهی از هنجار انحراف دارند و به

طور بالقوه وجود تصاویر دستکاری شده یا تقلبی را نشان می‌دهند. ناهنجاری‌ها ممکن است

الگوهای متمایزی را نشان دهنده، مانند ناهمانگی در شدت پیکسل، توزیع بافت غیرعادی، یا اشکال

نامنظم، که می‌تواند از طریق خوشبندی آشکار شود.

• استخراج و نمایش ویژگی: خوشبندی می‌تواند استخراج و نمایش ویژگی‌های مرتبط از تصاویر

را تسهیل کند. با خوشبندی تصاویر با ویژگی‌های مشابه، الگوهای ویژگی مشترک را می‌توان

شناسایی کرد، که به توسعه روش‌های استخراج ویژگی قوی برای تمایز بین تصاویر واقعی و جعلی

کمک می‌کند. این ویژگی‌ها می‌توانند متعاقباً توسط مدل‌های طبقه‌بندی برای طبقه‌بندی دقیق

تصاویر به عنوان واقعی یا دستکاری شده استفاده شوند.

• خوشبندی یک تکنیک یادگیری بدون نظارت است، به این معنی که برای آموزش به داده‌های

برچسب دار نیاز ندارد. این امر به ویژه در زمینه تشخیص تصاویر واقعی و جعلی سودمند است، زیرا

به دست آوردن داده‌های آموزشی برچسب گذاری شده می‌تواند چالش برانگیز و وقت‌گیر باشد.

خوشبندی امکان تجزیه و تحلیل اکتشافی داده‌ها را فراهم می‌کند و می‌تواند ساختارها و روابط

پنهان را حتی در غیاب دانش قبلی در مورد تصاویر آشکار کند.

بنابراین به طور خلاصه، تجزیه و تحلیل خوشبندی با کشف الگوهای شناسایی ناهمجارتی، استخراج

ویژگی‌های مرتبط و فعال کردن یادگیری بدون نظارت، نقش حیاتی در تشخیص تصاویر واقعی و جعلی دارد. با

استفاده از الگوریتم‌های مختلف خوشبندی، مانند K-means، DBSCAN، و خوشبندی سلسله مراتبی، می‌توان

بینش‌هایی در مورد ساختار زیربنایی مجموعه داده تصویر به دست آورد و بین تصاویر واقعی و دستکاری شده

تمایز قائل شد. استفاده از خوشبندی در ترکیب با تکنیک‌های طبقه‌بندی می‌تواند دقت و اثربخشی سیستم‌های

تمایز را افزایش داده و به پیشرفت پژوهشی قانونی تصویر و تضمین یکپارچگی محتوای دیجیتال

کمک کند.

1-5-2- معیار های مورد نیاز برای مقایسه خوشه بندی

Silhouette Score -1-5-2-1

امتیاز Silhouette معیاری است که معمولاً برای ارزیابی کیفیت الگوریتم های خوشه بندی استفاده می شود. میزان تناسب هر نقطه داده با خوشه اختصاص داده شده را اندازه گیری می کند و نشانه ای از اثربخشی کلی مدل خوشه بندی را ارائه می دهد. امتیاز از 1- تا 1 متغیر است که مقادیر بالاتر نشان دهنده نتایج خوشه بندی بهتر است.

محاسبه امتیاز Silhouette :

امتیاز Silhouette برای هر نقطه داده با در نظر گرفتن دو عامل کلیدی محاسبه می شود: انسجام و جدایی.

- انسجام: انسجام میزان نزدیکی یک نقطه داده به سایر نقاط درون خود را اندازه گیرد. معمولاً به عنوان فاصله متوسط بین یک نقطه داده و سایر نقاط در همان خوشه محاسبه می شود.
- جداسازی: جداسازی میزان فاصله یک نقطه داده از نقاط در خوشه های همسایه را کمیت می کند. معمولاً به عنوان فاصله متوسط بین یک نقطه داده و همه نقاط در نزدیکترین خوشه همسایه محاسبه می شود.

امتیاز Silhouette برای یک نقطه داده خاص با استفاده از فرمول زیر محاسبه می شود:

$$\text{Silhouette Score} = (\text{Separation} - \text{Cohesion}) / \max(\text{Separation}, \text{Cohesion})$$

امتیاز کلی Silhouette برای کل مجموعه داده با میانگین نمرات تمام نقاط داده فردی محاسبه می شود.

تفسیر امتیاز سیلوئت:

امتیاز Silhouette بیشتر در مورد کیفیت و ثبات نتایج خوشه بندی فراهم می کند. تفسیر امتیاز را می توان به صورت زیر خلاصه کرد:

- نزدیک به 1: امتیاز نزدیک به 1 نشان می دهد که نقطه داده به خوبی با خوش اختصاص داده شده مطابقت دارد و به طور قابل توجهی به نقاط در خوش خود نزدیکتر از نقاط در خوش های همسایه است. این نشان دهنده جدایی خوب بین خوش ها است.

- نزدیک به 0: نمره حدود 0 نشان می دهد که نقطه داده نزدیک به مرز تصمیم بین دو خوش های همسایه است یا ممکن است به یک خوش نامناسب اختصاص داده شود. این نشان دهنده خوش های همپوشانی یا مبهم است.

- نزدیک به -1: امتیاز نزدیک به -1 نشان می دهد که نقطه داده به طور بالقوه به خوش اشتباهی اختصاص داده شده است، زیرا بیشتر شبیه به نقاط یک خوش های همسایه است تا نقاطی در خوش اختصاص داده شده آن. این نشان دهنده نتایج ضعیف خوش بندی است.

مقایسه نمرات Silhouette

هنگام مقایسه امتیازهای Silhouette بین مدل های مختلف خوش بندی یا پیکربندی پارامترها، می توان دستورالعمل های زیر را در نظر گرفت:

- امتیاز بالاتر: امتیاز Silhouette بالاتر نشان دهنده نتایج بهتر خوش بندی است. مدل ها یا

پیکربندی هایی با امتیازهای بالاتر ترجیح داده می شوند زیرا خوش های متمایز تر و به خوبی جدا شده را نشان می دهند.

- سازگاری: سازگاری امتیازهای Silhouette در چندین آزمایش یا تنظیمات پارامترهای مختلف مهم است. اگر یک مدل به طور پیوسته نمرات بالاتری را در اجراهای مختلف کسب کند، عملکرد خوش بندی قابل اعتماد تر و پایدار تری را نشان می دهد.

- تجزیه و تحلیل مقایسه ای: از امتیازهای Silhouette می توان برای مقایسه الگوریتم های مختلف

خوشه بندی یا تغییرات در یک الگوریتم استفاده کرد. مدل‌هایی که امتیازات بالاتری دارند معمولاً

نتایج خوشه بندی بهتری دارند.

• دانش دامنه: در نظر گرفتن دانش و قابلیت تفسیر دامنه در کنار امتیازهای Silhouette ضروری

است. در حالی که امتیاز بالاتر مطلوب است، ارزیابی اینکه آیا خوشه های حاصل با الگوهای مورد

انتظار یا شناخته شده در مجموعه داده همسو هستند یا خیر، بسیار مهم است.

محدودیت ها و ملاحظات:

• امتیاز Silhouette هنگام ارزیابی الگوریتم‌های خوشه بندی که هدف‌شان تولید خوشه‌های فشرده و

کاملاً جدا شده است، مؤثرتر است. ممکن است برای ارزیابی الگوریتم‌هایی که هدف‌شان شناسایی

خوشه های سلسله مراتبی یا همپوشانی است، مناسب نباشد.

• امتیاز Silhouette به متريک فاصله انتخابی بستگی دارد و ممکن است با معيارهای فاصله متفاوت

متفاوت باشد.

• تفسیر امتیازهای Silhouette ذهنی است و نیاز به تجزیه و تحلیل دقیق در ارتباط با دانش حوزه

دارد.

3-5-1- خوشه بندی بر بروی ویژگی های داده شده در features.csv

3-5-1-1- خوشه بندی با الگوریتم k-means

برای خوشه بندی به کمک الگوریتم k-means از کد زیر استفاده می‌شود.

```

def perform_kmeans_clustering(features_df, labels_df, num_clusters, plot=True):
    # Step 1: Merge the features and labels dataframes
    merged_df = pd.merge(features_df, labels_df, left_index=True, right_index=True)

    # Step 2: Extract relevant columns
    real_fake = merged_df['real_fake']
    category = merged_df['category']

    # Step 3: Preprocess the feature data - Normalize the features
    scaler = MinMaxScaler()
    scaled_features = scaler.fit_transform(features_df)

    # Step 4: Perform K-means clustering
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(scaled_features)

    # Step 5: Analyze the clusters
    cluster_labels = kmeans.labels_

    # Calculate Silhouette score
    silhouette_avg = silhouette_score(scaled_features, cluster_labels)

    # Create a new dataframe with cluster labels and original labels
    clustered_data = pd.DataFrame({'Cluster': cluster_labels, 'Real_Fake': real_fake, 'Category': category})

    # Print Silhouette score
    print(f"Silhouette Score for {num_clusters} Clusters: {silhouette_avg}")

    # Iterate over clusters
    for i in range(num_clusters):
        print(f"\nCluster {i}:")
        cluster_data = clustered_data[clustered_data['Cluster'] == i]

        # Count the occurrences of each unique value in 'Real_Fake' column
        real_fake_counts = cluster_data['Real_Fake'].value_counts()
        print("Real vs. Fake:")
        print(real_fake_counts)

        # Count the occurrences of each unique value in 'Category' column
        category_counts = cluster_data['Category'].value_counts()
        print("\nCategory:")
        print(category_counts)

        # Plot bar charts for label counts within the cluster if plot=True
        if plot:
            fig, axes = plt.subplots(1, 2, figsize=(10, 4))
            axes[0].bar(real_fake_counts.index.astype(str), real_fake_counts.values)
            axes[0].set_title(f'Cluster {i}: Real vs. Fake')

            axes[1].bar(category_counts.index.astype(str), category_counts.values)
            axes[1].set_title(f'Cluster {i}: Category')

            plt.tight_layout()
            plt.show()

```

شکل (۱-۳۲) کد پایتون خوش‌بندی با الگوریتم k-means

در کد فوق ابتدا فیچر ها را خوانده سپس الگوریتم را اعمال کرده و در ادامه به تعدادی cluster هایی که

تعیین شده بر اساس لیل هایی که از قبل موجود است بررسی می‌شود که هر کدام از کلاستر ها شامل چه تعداد

داده real و چه تعداد fake و همچنین چه تعداد از داده های شامل کوه و دریا و جنگل می‌باشد. در ادامه هم برای

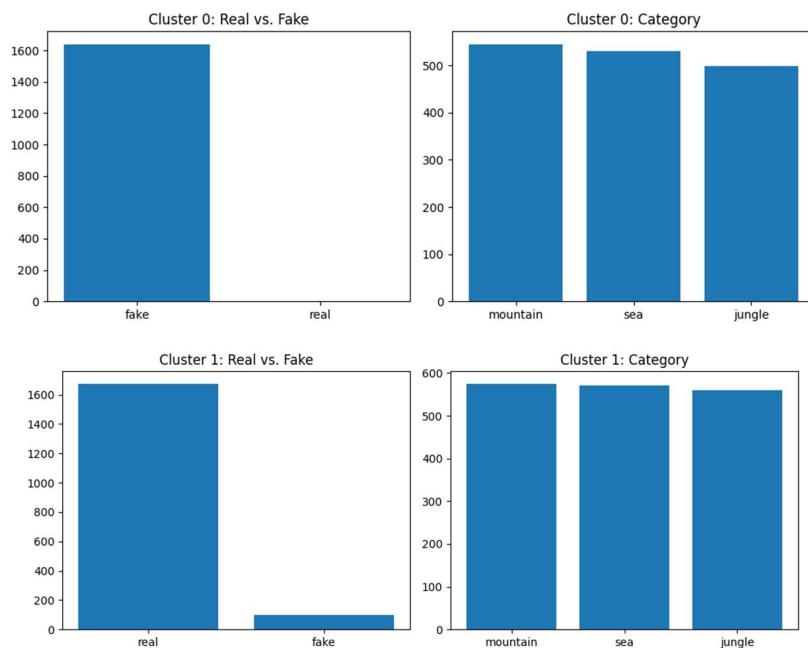
دید بهتر منحنی میله آن ها رسم می‌شود تا بدین ترتیب بتوانیم مقایسه مناسبی از خوش‌بندی شود.

روی Kmeans با 2 کلاستر:

```
perform_kmeans_clustering(features_df, labels_df, 2)
```

شکل (۱-۳۳) اعمال kmeans با 2 تا کلاستر

خروجی کلاستر ها به صورت زیر میباشد و امتیاز Silhouette برابر با 0.340332639856897 است.



شکل (۱-۳۴) خروجی 2 کلاستر با روی kmeans بر روی features.csv

مشاهده میشود که به خوبی توانسته 2 کلاستر جدا کند که یکی برای fake ها و یکی برای real ها میباشد.

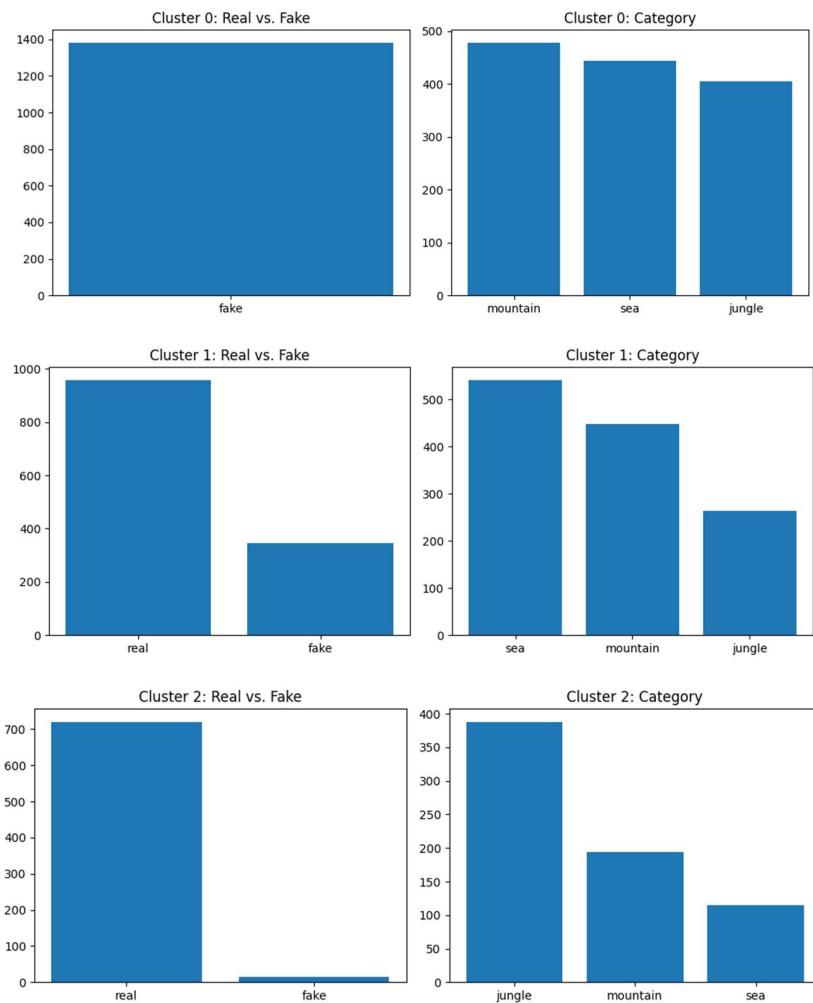
از نظر توزیع category ها هم توزیع یکنواخت میباشد.

روی Kmeans با 3 کلاستر:

```
perform_kmeans_clustering(features_df, labels_df, 3)
```

شکل (۱-۳۵) اعمال kmeans با 3 تا کلاستر

خروجی کلاستر ها به صورت زیر میباشد و امتیاز Silhouette برابر با 0.26732381930414895 است.



شکل (۱-۳۶) خروجی ۳ کلاستر با kmeans بر روی features.csv

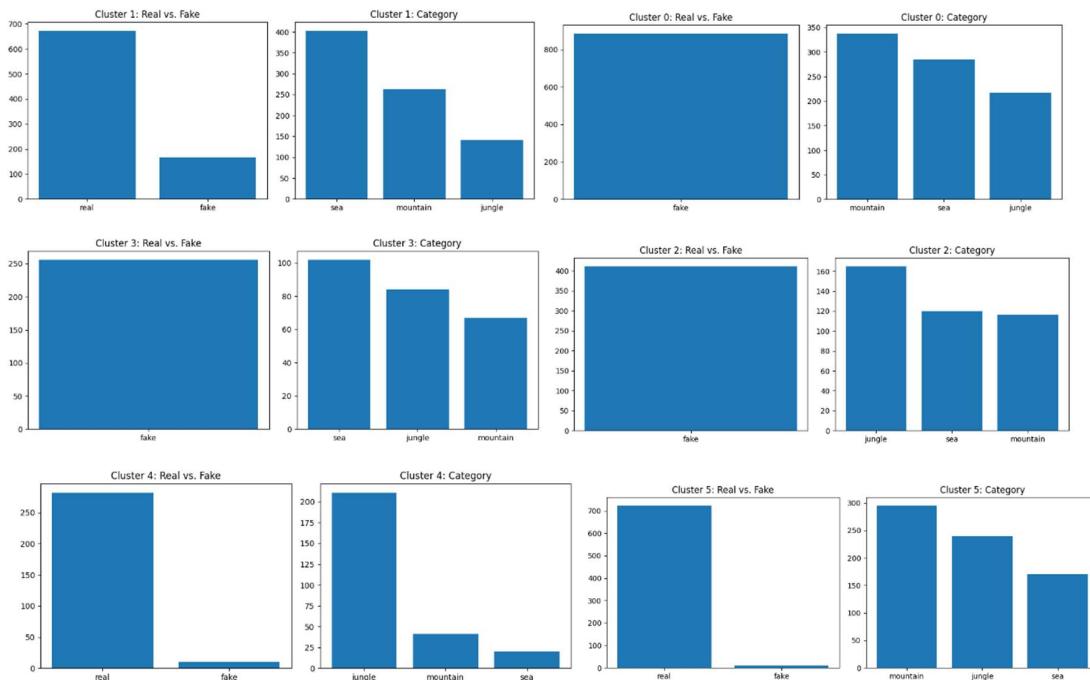
وقتی تعداد کلاستر ها ۳ تا میشود مشاهده میشود یک کلاستر تقریبا شامل real ها یکی کلاستر تماما fake و یک کلاستر هم شامل هم تصاویر real هم fake (بیشتر real) میباشد که از نظر توزیعی این کلاستر بیشتر شامل کلاس دریا میباشد در حالی که آن کلاس دیگر real کمترین تعداد را در دریا دارد بنابراین میتوان گفت که ۳ کلاستر یکی کلاس fake را جدا میکند یکی کلاس های real دریا و یکی هم سایر کلاس های real را جدا میکند.

روی features.csv با ۶ کلاستر: Kmeans

```
perform_kmeans_clustering(features_df, labels_df, 6)
```

شکل (۱-۳۷) اعمال kmeans با ۶ تا کلاستر

خروجی کلاستر ها به صورت زیر بوده و امتیاز silhouette برابر با 0.20700614642138343 است.



شکل (۱-۳۸) خروجی ۶ کلاستر با روی kmeans بر features.csv

مشاهده میشود که ۳ کلاس همه ها و fake تقریبا هم real میباشد همچنین اگر به توزیع داده ها

دقیق شود مشاهده میشود که تقریبا در هر کلاستر یک کلاس از کلاس های دریا و جنگل و کوه غالب میباشد

مثلا همانطور که در شکل فوق میبینید کلاستر ۲ تقریبا میتوان گفت کلاس fake و جنگل و کلاستر ۴ کلاس

real و جنگل میباشد.

روش فوق تا کلاستر های ۹ تایی و ۱۲ هم تکرار شد که برای جلوگیری از زیاد شدن گزارش آن ها آورده

نشده است اما خروجی های آن در کد ارسالی قابل مشاهده میباشد.

۱-۵-۳-۲- خوش بندی با الگوریتم GMM

برای خوش بندی به کمک الگوریتم GMM از کد زیر استفاده میشود.

```
def perform_gmm_clustering(features_df, labels_df, num_clusters, plot=True):
    # Step 1: Merge the features and labels dataframes
    merged_df = pd.merge(features_df, labels_df, left_index=True, right_index=True)

    # Step 2: Extract relevant columns
    real_fake = merged_df['real_fake']
    category = merged_df['category']

    # Step 3: Preprocess the feature data - Normalize the features
    scaler = MinMaxScaler()
    scaled_features = scaler.fit_transform(features_df)

    # Step 4: Perform GMM clustering
    gmm = GaussianMixture(n_components=num_clusters)
    gmm.fit(scaled_features)

    # Step 5: Analyze the clusters
    cluster_labels = gmm.predict(scaled_features)

    # Calculate Silhouette score
    silhouette_avg = silhouette_score(scaled_features, cluster_labels)

    # Create a new dataframe with cluster labels and original labels
    clustered_data = pd.DataFrame({'Cluster': cluster_labels, 'Real_Fake': real_fake, 'Category': category})

    # Print Silhouette score
    print(f"Silhouette Score for {num_clusters} Clusters: {silhouette_avg}")

    # Iterate over clusters
    for i in range(num_clusters):
        print(f"\nCluster {i}:")
        cluster_data = clustered_data[clustered_data['Cluster'] == i]

        # Count the occurrences of each unique value in 'Real_Fake' column
        real_fake_counts = cluster_data['Real_Fake'].value_counts()
        print("Real vs. Fake:")
        print(real_fake_counts)

    # Count the occurrences of each unique value in 'Category' column
    category_counts = cluster_data['Category'].value_counts()
    print("\nCategory:")
    print(category_counts)

    # Plot bar charts for label counts within the cluster if plot=True
    if plot:
        fig, axes = plt.subplots(1, 2, figsize=(10, 4))
        axes[0].bar(real_fake_counts.index.astype(str), real_fake_counts.values)
        axes[0].set_title(f'Cluster {i}: Real vs. Fake')

        axes[1].bar(category_counts.index.astype(str), category_counts.values)
        axes[1].set_title(f'Cluster {i}: Category')

        plt.tight_layout()
        plt.show()
```

شکل (۱-۳۹) کد پایتون خوش بندی با الگوریتم GMM

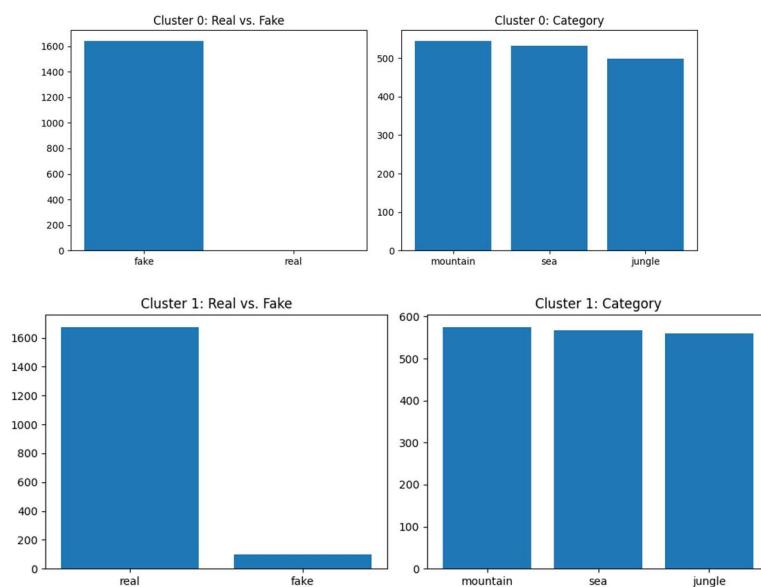
در کد فوق ابتدا فیچر ها را خوانده سپس الگوریتم را اعمال کرده و در ادامه به تعدادی cluster هایی که تعیین شده بر اساس لیل هایی که از قبل موجود است بررسی میشود که هر کدام از کلاستر ها شامل چه تعداد داده real و چه تعداد از داده های شامل کوه و دریا و جنگل میباشد. در ادامه هم برای دید بهتر منحنی میله آن ها رسم میشود تا بدین ترتیب بتوانیم مقایسه مناسبی از خوش بندی شود.

روی GMM با 2 کلاستر: features.csv

```
perform_gmm_clustering(features_df, labels_df, 2)
```

شکل (۱-۴۰) اعمال GMM با 2 تا کلاستر

خروجی کلاستر ها به صورت زیر میباشد و امتیاز Silhouette برابر با 0.3401770154797122 است که از K-means بیشتر است و این به این معناست که این روش از clustering جداسازی اندک بهترین را انجام میدهد.



شکل (۱-۴۱) خروجی ۲ کلاستر با features.csv بر روی GMM

مشاهده میشود که به خوبی توانسته ۲ کلاستر جدا کند که یکی برای fake ها و یکی برای real ها میباشد.

از نظر توزیع category ها هم توزیع یکنواخت میباشد. از نظر دقت هم از kmeans بهتر است

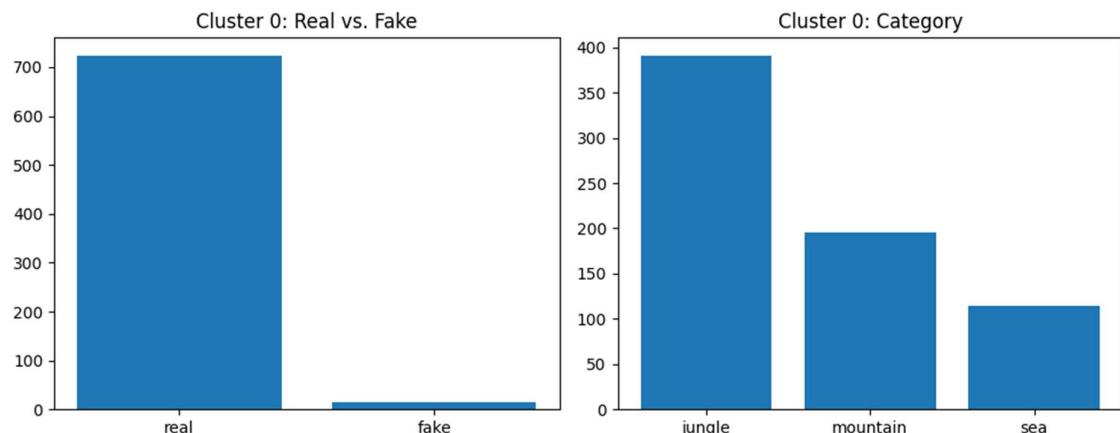
روی GMM با ۳ کلاستر:

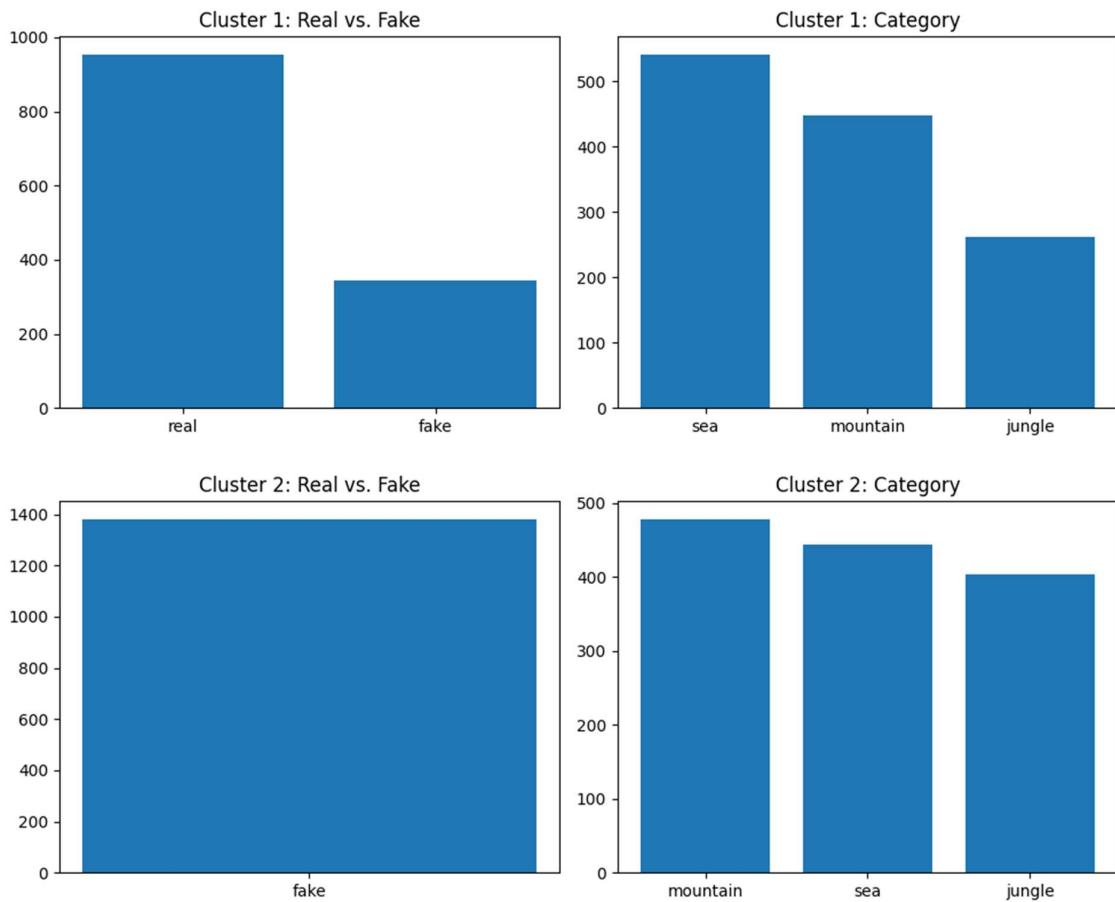
```
perform_gmm_clustering(features_df, labels_df, 3)
```

شکل (۱-۴۲) اعمال GMM با ۳ تا کلاستر

خروجی کلاستر ها به صورت زیر میباشد و امتیاز Silhouette برابر با ۰.۲۶۶۸۷۸۶۸۵۷۲۱۹۷۱۱ است.

که همچنان از kmeans بهتر است.





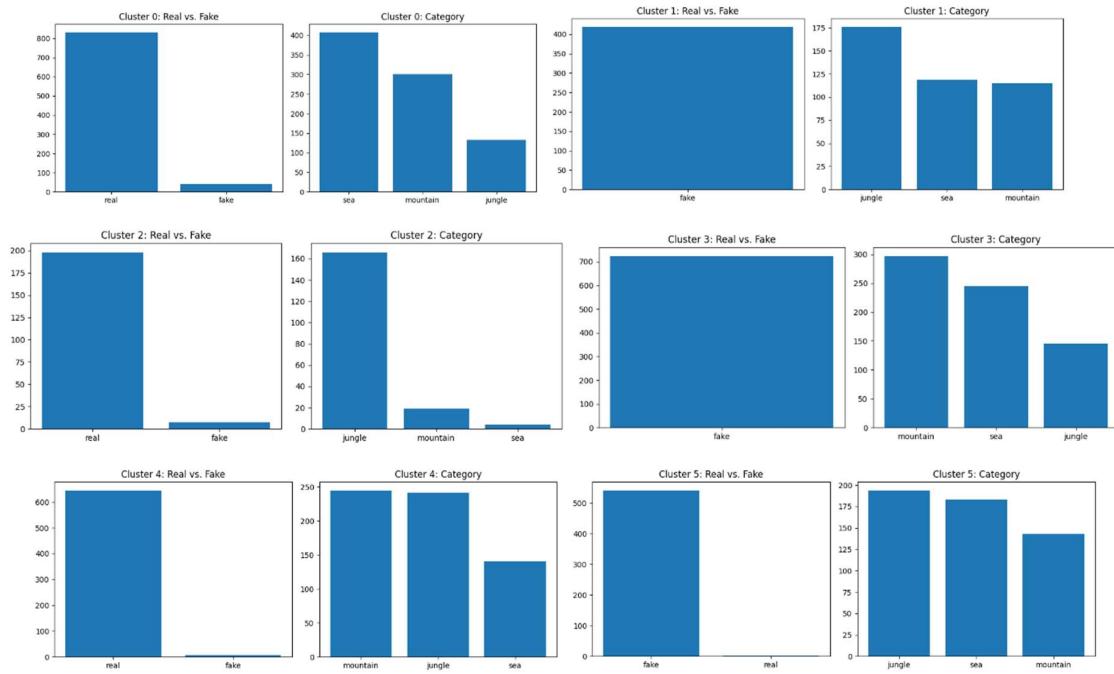
شکل (۱-۴۳) خروجی ۳ کلاستر با GMM بر روی features.csv

وقتی تعداد کلاستر ها ۳ تا میشود مشاهده میشود یک کلاستر تقریبا شامل real ها یکی کلاستر تماما fake و یک کلاستر هم شامل هم تصاویر fake (بیشتر real) هم میباشد که از نظر توزیعی مانند قبل میباشد.

روی GMM با ۶ کلاستر:

خروجی کلاستر ها به صورت زیر بوده و امتیاز silhouette برابر با 0.1868478147777128 است. که

در این حالت k-means عملکرد بهتری داشت.



شکل (۱-۴۴) خروجی ۶ کلاستر با kmeans بر روی features.csv

مشاهده میشود که ۳ کلاس همه fake ها و ۳ کلاس تقریبا هم real میباشد و تا حدی در هر یک کلاستر هر کدام از کلاس های دریا و جنگل و کوه یکی از ان ها کلاس غالب میباشد.

۱-۵-۴- خوشه بندی بر روی ویژگی های کلاسیک استخراج شده

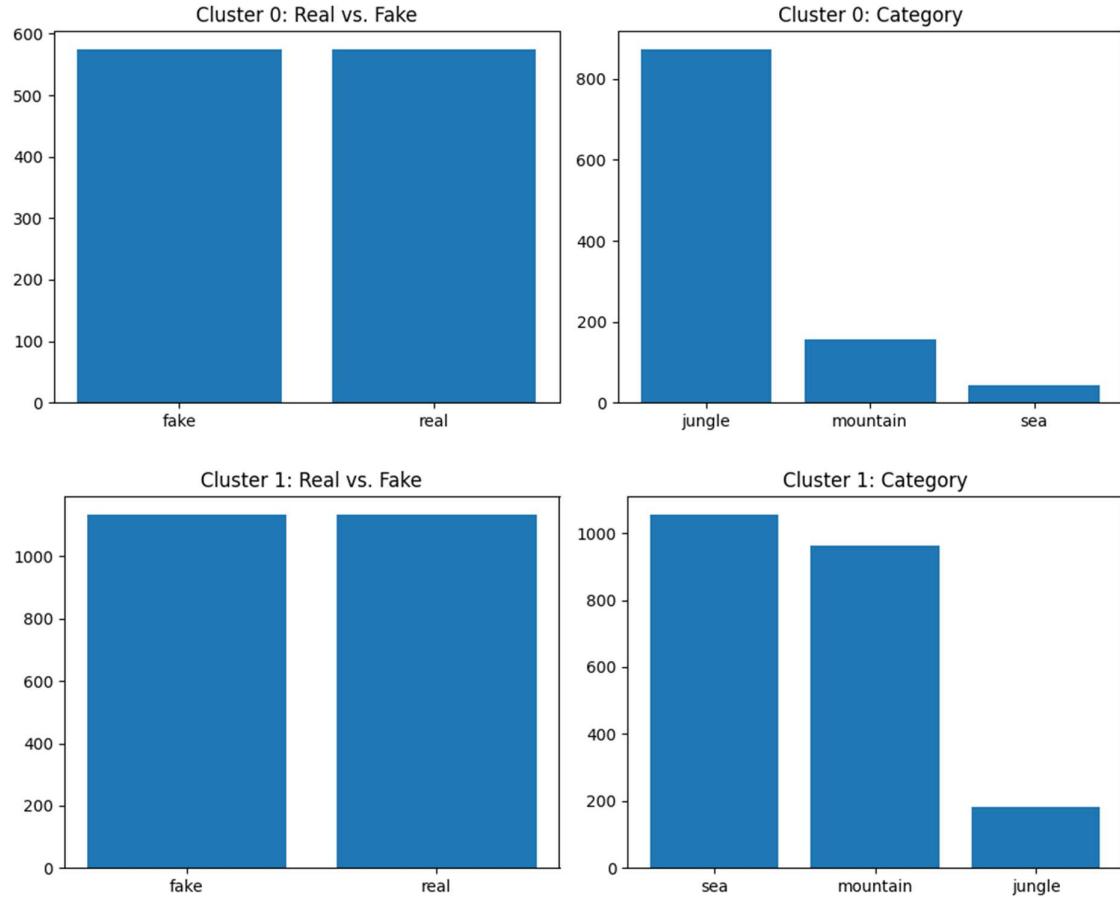
حال همین مراحل خوشه بندی بر روی ویژگی هایی که به صورت کلاسیک استخراج شده است اعمال میشود.

۱-۵-۴-۱- خوشه بندی با الگوریتم k-means

روی فیچر های کلاسیک با 2 کلاستر:

خروجی کلاستر ها به صورت زیر میباشد و امتیاز Silhouette برابر با 0.09645659911886657 است.

که بسیار کمتر از حالت قبل با فیچر های حاصل از شبکه های عصبی میباشد.



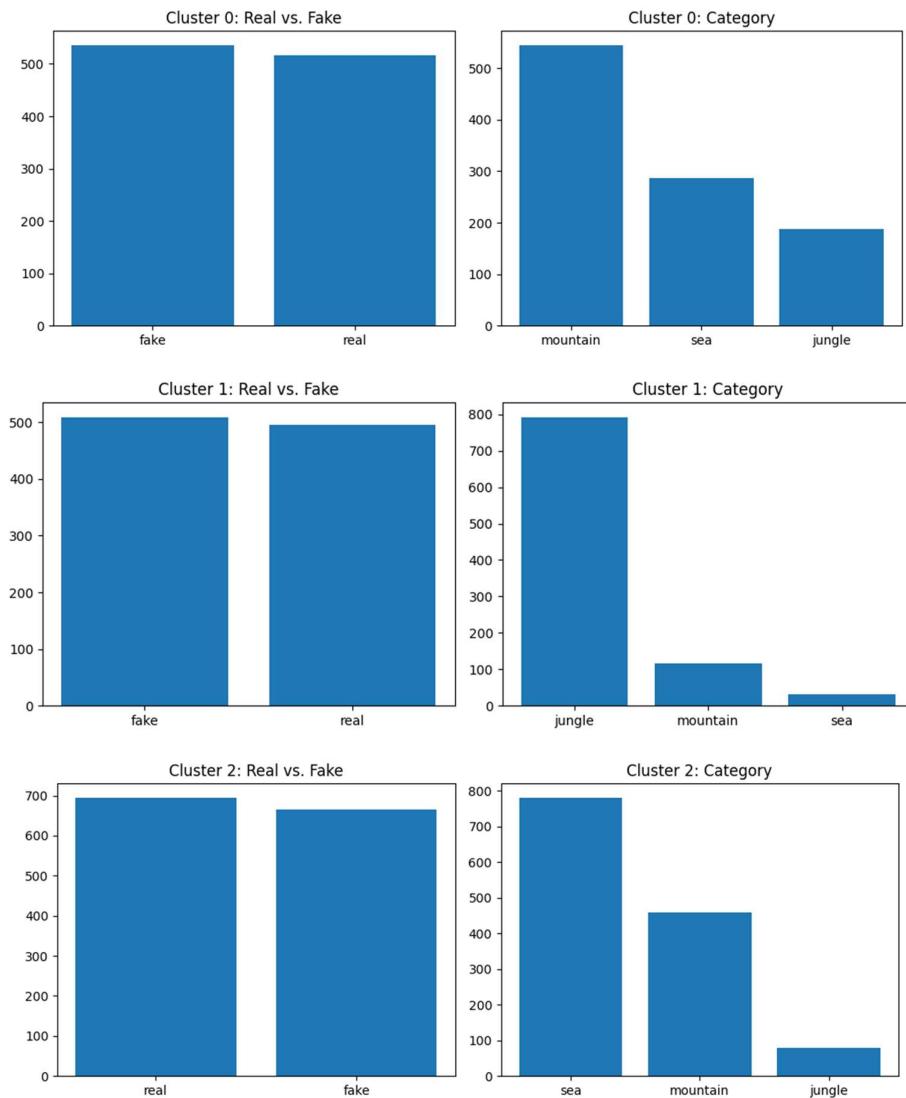
شکل (۱-۴۵) خروجی ۲ کلاستر با kmeans بر روی فیچر های کلاسیک

مشاهده میشود که در این حالت بر روی فیچر های کلاسیک الگوریتم عملابه جای آنکه ۲ کلاس real و fake را از هم جدا کند ۲ تا کلاس جنگل و (دریا و کوه) را از هم جدا کرده است.

روی ویژگی های کلاسیک با ۳ کلاستر:

خروجی کلاستر ها به صورت زیر میباشد و امتیاز Silhouette برابر با 0.06392129357709256 است.

که نسبت به حالت استفاده از فیچر های خروجی شبکه عصبی کمتر است.



شکل (۱-۴۶) خروجی ۳ کلاستر با kmeans بر روی فیچر های کلاسیک

وقتی تعداد کلاستر ها ۳ تا میشود مشاهده میشود که عملا ۳ تا کلاس دریا و جنگل و کوه فارغ از real یا بودن آنها جدا میشود.

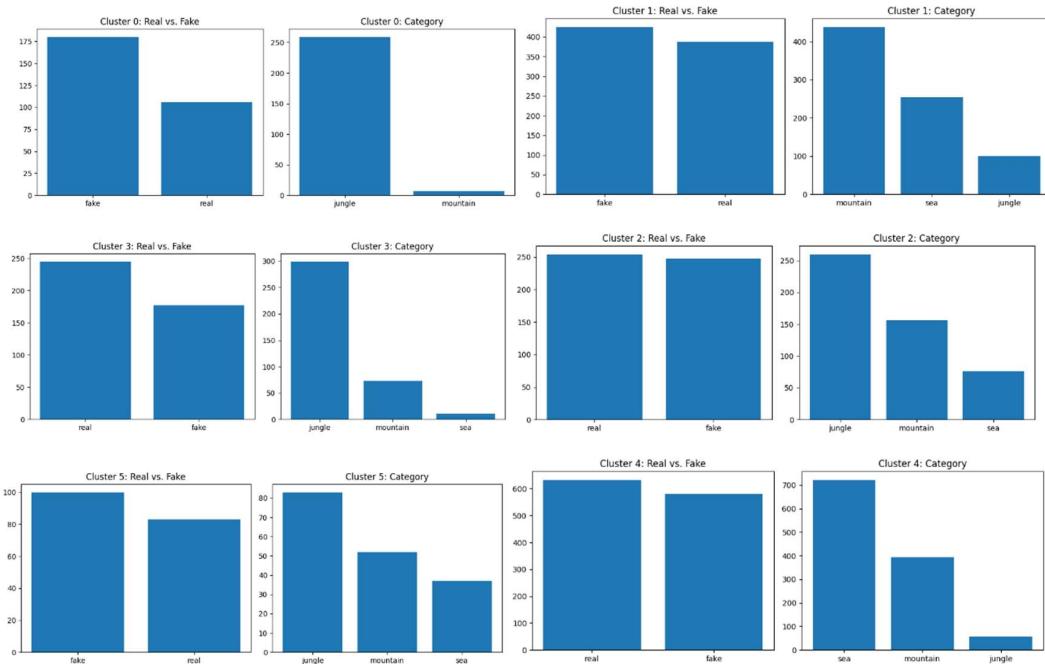
مشاهدات نشان میدهد که فیچر های کلاسیک جداپذیری بیشتری روی ۳ کلاس دریا و جنگل و کوه داشته

است که کاملا قابل پیش بینی نیز میباشد چرا که به عنوان مثال فیچری مثل colorhist عملا رنگ آبی دریا از سبز

جنگل و قهوه ای کوه جدا میکند و

روی ویژگی های کلاسیک با ۶ کلاستر:

خروجی کلاستر ها به صورت زیر بوده و امتیاز silhouette برابر با 0.04059591116169582 است.



شکل (۱-۴۷) خروجی ۶ کلاستر با kmeans بر روی فیچر های کلاسیک

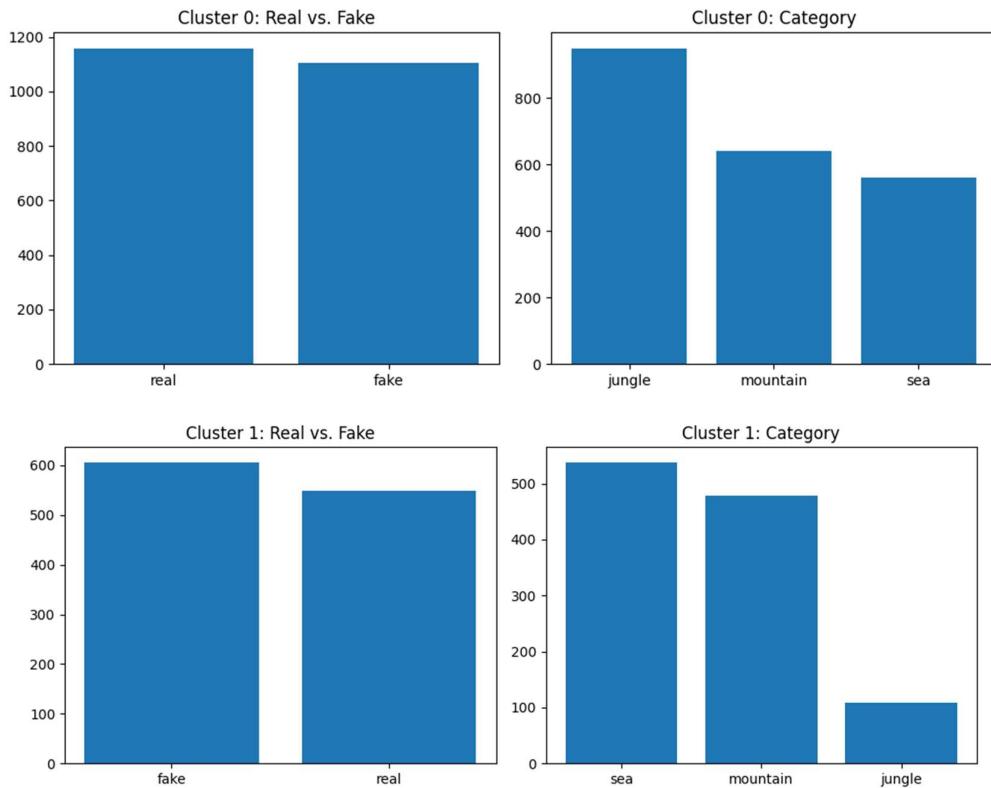
در این حالت خیلی مورد معنا داری پیدا نمیشود.

4-5-1- خوشبندی با الگوریتم GMM

GMM روی فیچر های کلاسیک با ۲ کلاستر:

خروجی کلاستر ها به صورت زیر میباشد و امتیاز Silhouette برابر با 0.3518257575008264 است.

که حالت k-means بسیار بیشتر میباشد.

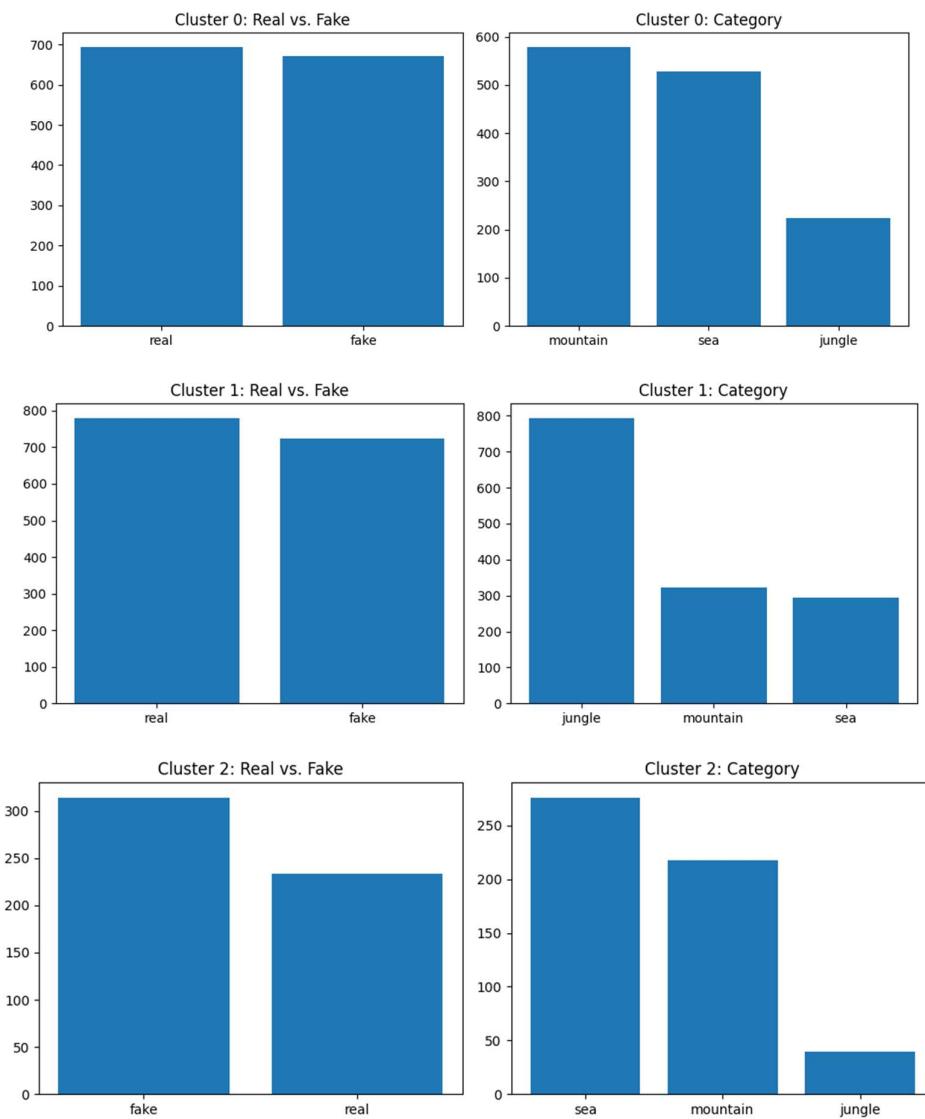


شکل (۱-۴۸) خروجی ۲ کلاستر با GMM بر روی فیچر های کلاسیک

مشاهده میشود که در این حالت بر روی فیچر های کلاسیک الگوریتم عملاً به جای آنکه ۲ کلاس real و fake را از هم جدا کند ۲ تا کلاس جنگل و (دریا و کوه) را البته نه با دقت خوبی از هم جدا کرده است.

روی ویژگی های کلاسیک با ۳ کلاستر:

خروجی کلاستر ها به صورت زیر میباشد و امتیاز Silhouette برابر با 0.24348652850204747 است.



شکل (۱-۴۹) خروجی ۳ کلاستر با GMM بر روی فیچر های کلاسیک

وقتی تعداد کلاستر ها ۳ تا میشود مشاهده میشود که عملاً ۳ تا کلاس دریا و جنگل و کوه فارغ از real یا fake بودن آنها جدا میشود. (البته دقیق در اینجا کمتر از k-means است ولی به هر حال در هر کلاستر یک category غالب تر است)

1-5-5- خوش بندی GMM و Kmeans بر روی تک تک فیچر های استخراج شده

در این بخش به جای استفاده از تمامی ویژگی های کلاسیک استخراج شده از جدا جدا با تک تک ویژگی

ها کلاستر کردن انجام میشود.

```
df = pd.read_csv('hist_features_64_64.csv')
df.rename(columns={'Label':'real_fake', 'Category':'category'}, inplace=True)
df1 = df.drop(['real_fake', 'category'], axis=1)
labels1 = df[['real_fake', 'category']].copy()
labels1['real_fake'] = labels1['real_fake'].str.lower()
df1.name = 'hist_features_64_64'

df = pd.read_csv('lbp_features_64_64.csv')
df.rename(columns={'Label':'real_fake', 'Category':'category'}, inplace=True)
df2 = df.drop(['real_fake', 'category'], axis=1)
labels2 = df[['real_fake', 'category']].copy()
labels2['real_fake'] = labels1['real_fake'].str.lower()
df2.name = 'lbp_features_64_64'

df = pd.read_csv('hog_features_64_64.csv')
df.rename(columns={'Label':'real_fake', 'Category':'category'}, inplace=True)
df3 = df.drop(['real_fake', 'category'], axis=1)
labels3 = df[['real_fake', 'category']].copy()
labels3['real_fake'] = labels1['real_fake'].str.lower()
df3.name = 'hog_features_64_64'

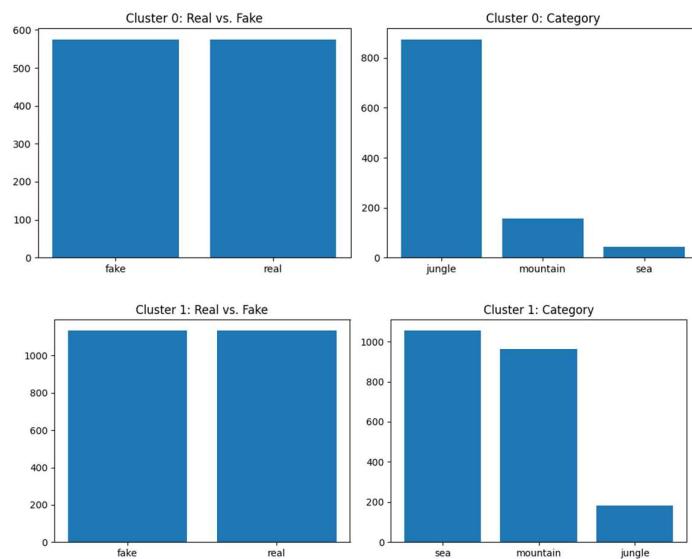
df = pd.read_csv('glcm_features_64_64.csv')
df.rename(columns={'Label':'real_fake', 'Category':'category'}, inplace=True)
df4 = df.drop(['real_fake', 'category'], axis=1)
labels4 = df[['real_fake', 'category']].copy()
labels4['real_fake'] = labels1['real_fake'].str.lower()
df4.name = 'glcm_features_64_64'

df = pd.read_csv('gabor_features_64_64.csv')
df.rename(columns={'Label':'real_fake', 'Category':'category'}, inplace=True)
df5 = df.drop(['real_fake', 'category'], axis=1)
labels5 = df[['real_fake', 'category']].copy()
labels5['real_fake'] = labels1['real_fake'].str.lower()
df5.name = 'gabor_features_64_64'
```

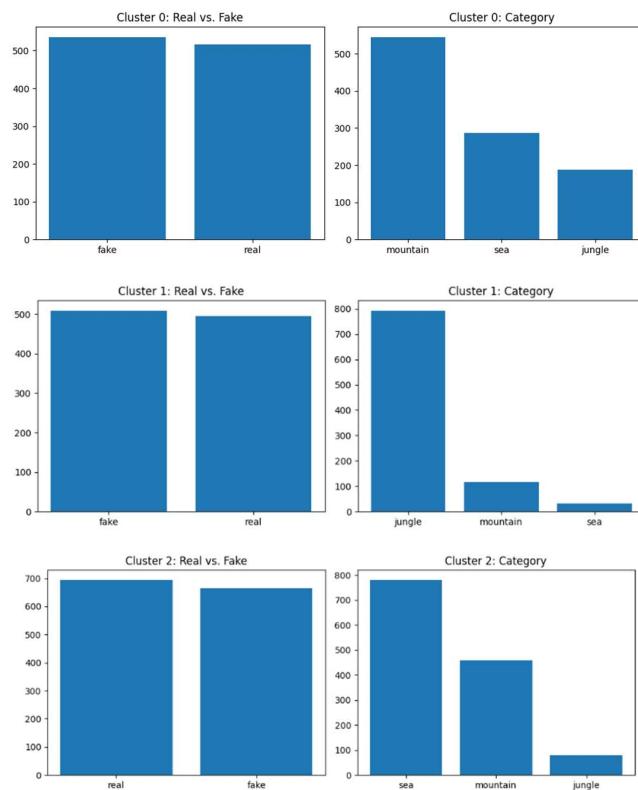
شکل (1-5۰) تفکیک ۵ کلاستر در فایل های جداگانه

1-5-5-2- کلاستر با فیچر color hist

خروجی های kmeans برای حالت های 2 و 3 کلاستر به صورت زیر میباشد.



شکل (۱-۵۱) خروجی های ۲ کلاستر kmeans برای فیچر color hist



شکل (۱-۵۲) خروجی های ۳ کلاستر kmeans برای فیچر color hist

ویژگی Color Hist در 2 کلاستر ، کلاس جنگل را از بقیه جدا میکند و در حالت 3 کلاستر بر اساس

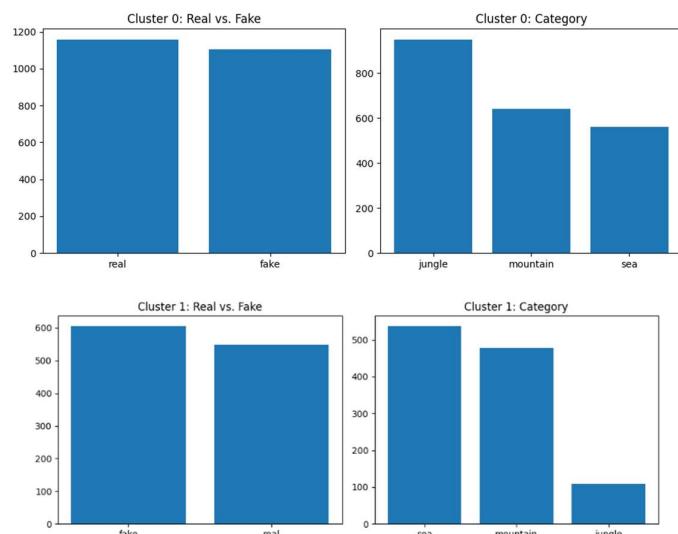
رنگ 3 کلاستر دریا و جنگل و کوه را جدا میکند.

دلیل آن که جنگل را هم از کوه و دریا جدا میکند منطقی به نظر میرسد (تصاویر جنگل همگی سبز هستند ولی تصاویر دریا خودش آبی و ساحل قهوه ای و همچنین کوه ها خودش قهوه ای و آسمانش آبی میباشد و به همین دلیل کوه و دریا در یک دسته و جنگل در یک دسته قرار میگیرد)

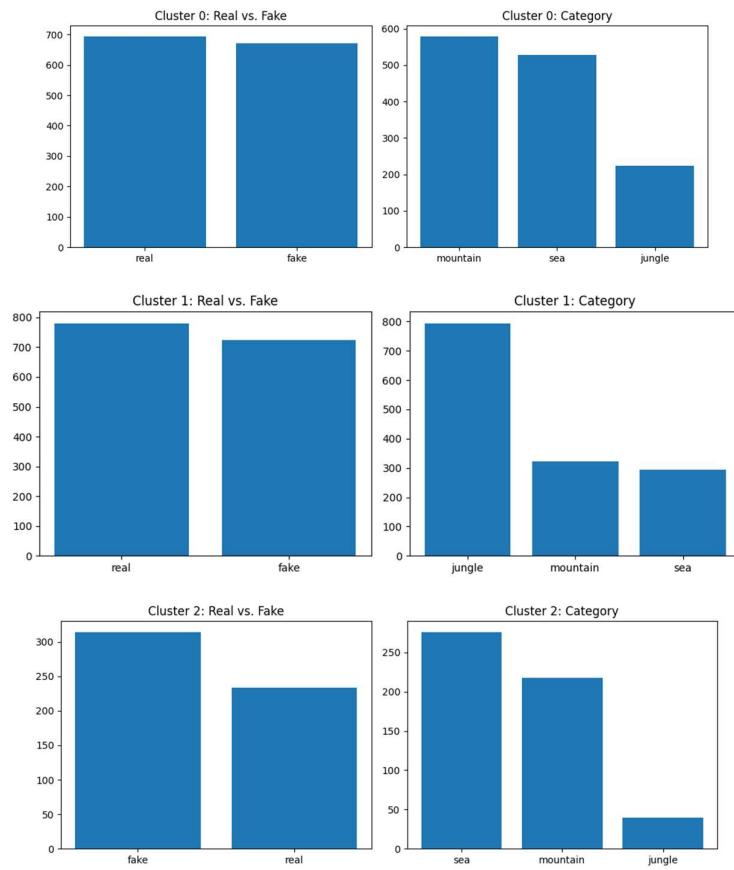
** خروجی ها با GMM هم تقریبا به همین صورت است که خروجی ها در کد ارسالی آمده است.

1-5-5-3 - کلاستر با فیچر lbp

خروجی های kmeans برای حالت های 2 و 3 کلاستر به صورت زیر میباشد.



شکل (1-5-3) خروجی های 2 کلاستر kmeans برای فیچر lbp



شکل (۱-۵۴) خروجی های ۳ کلاستر kmeans برای فیچر lbp

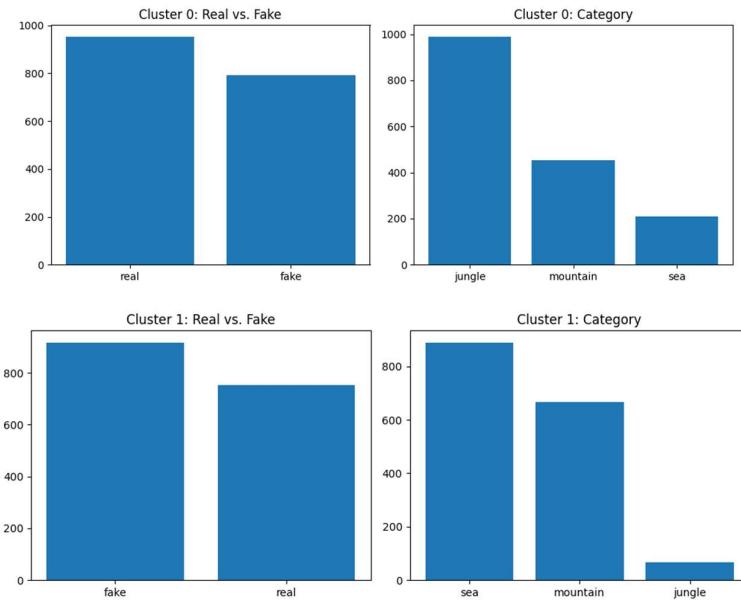
ویژگی LBP در ۲ کلاستر، کلاس جنگل را از بقیه جدا میکند و در حالت ۳ کلاستر عملاً انگار در یک کلاستر کلاس جنگل جدا میشود.

** خروجی های با GMM هم تقریباً به همین صورت است که خروجی ها در کد ارسالی آمده است. همچنین

کلاستر های ۶ و ۹ و ۵۰ و ... از این ویژگی هم در کد ارسالی آمده است.

1-5-5-4 - کلاستر با فیچر hog

خروجی های kmeans برای حالت ۲ کلاستر به صورت زیر میباشد.



شکل (۱-۵۵) خروجی های 2 کلاستر kmeans برای فیچر hog

ویژگی HOG نیز در 2 کلاستر ، کلاس جنگل را از بقیه جدا میکند.

** خروجی ها با GMM هم تقریبا به همین صورت است که خروجی ها در کد ارسالی آمده است. همچنین

کلاستر های 6 و 9 و 50 و ... از این ویژگی هم در کد ارسالی آمده است.

1-5-5-5 - کلاستر با فیچر glcm

شرایط کاملا مشابه 3 ویژگی دیگر میشود ، خروجی ها در کد های ارسالی آمده است.

1-5-5-6 - کلاستر با فیچر gabor

شرایط کاملا مشابه 3 ویژگی دیگر میشود ، خروجی ها در کد های ارسالی آمده است.

1-5-6- خوشه بندی ویژگی های کلاسیک با روش BIRCH و HDBSCAN

HDBSCAN -1-5-6-1

HDBSCAN (خوشه بندی فضایی مبتنی بر تراکم سلسله مراتبی برنامه ها با نویز) یک الگوریتم خوشه بندی مبتنی بر چگالی است که می تواند برای کشف خوشه ها در یک مجموعه داده استفاده شود. این یک توسعه از الگوریتم DBSCAN (خوشه بندی فضایی مبتنی بر چگالی برنامه ها با نویز) است که یک تکنیک محبوب برای خوشه بندی داده های مکانی است.

HDBSCAN برای غلبه بر برخی محدودیت های الگوریتم های خوشه بندی سنتی، مانند نیاز به تعیین تعداد خوشه ها از قبل و حساسیت به تنظیمات پارامتر، طراحی شده است. HDBSCAN از یک رویکرد سلسله مراتبی برای ساخت یک درخت خوشه ای مبتنی بر چگالی استفاده می کند که امکان تعیین خودکار اندازه های خوشه و شناسایی خوشه ها در سطوح مختلف دانه بندی را فراهم می کند.

این الگوریتم ابتدا با ساخت یک نمودار دسترسی متقابل کار می کند، که روابط زوجی بین نقاط مجموعه داده را بر اساس چگالی آنها نشان می دهد. سپس با ادغام تدریجی خوشه ها، از متراکم ترین مناطق شروع به یک درخت خوشه ای متراکم می کند. درخت خوشه یک نمایش سلسله مراتبی از مجموعه داده را ارائه می دهد، که در آن هر سطح یک راه حل خوشه بندی متفاوت را نشان می دهد. با بررسی پایداری خوشه ها در سطوح مختلف درخت، HDBSCAN می تواند قوی ترین نتیجه خوشه بندی را تعیین کند.

یکی از مزایای کلیدی HDBSCAN توانایی آن در کنترل خوشه هایی با چگالی های مختلف و اشکال نامنظم است. می تواند خوشه هایی با اندازه ها و شکل های مختلف را بدون نیاز به دانش قبلی از ویژگی های آنها

شناسایی کند. علاوه بر این، HDBSCAN همچنین می‌تواند نقاط پرت یا نویز را در مجموعه داده شناسایی کند،

که می‌تواند برای تمیز کردن داده‌ها و وظایف تشخیص ناهنجاری مفید باشد.

HDBSCAN به طور گسترده در حوزه‌های مختلف، از جمله پردازش تصویر، بیوانفورماتیک، تجزیه و

تحلیل شبکه‌های اجتماعی و تقسیم‌بندی مشتریان استفاده شده است. این یک رویکرد انعطاف‌پذیر و قوی برای

خوشه‌بندی ارائه می‌کند که می‌تواند با مجموعه داده‌های مختلف سازگار شود و ساختارهای پیچیده را ثبت کند.

با این حال، توجه به این نکته مهم است که HDBSCAN، مانند هر الگوریتم خوشه‌بندی، به تنظیم دقیق پارامتر و

تفسیر نتایج برای به دست آوردن بینش معنی‌دار از داده‌ها نیاز دارد.

برای پیاده‌سازی از کد‌های زیر استفاده شده است ابتدا ابعاد فضای کم کرده سپس HDBSCAN اعمال

می‌شود.

```
def best_pca(df, th=0.8):
    # Generate some random high-dimensional data
    np.random.seed(42)
    X = df.iloc[:, :-1].values # 100 samples, 50 features

    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
    normalized_X = scaler.fit_transform(X)

    # Perform PCA
    pca = PCA()
    pca.fit(normalized_X)

    # Scree plot
    explained_variance_ratio = pca.explained_variance_ratio_
    cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

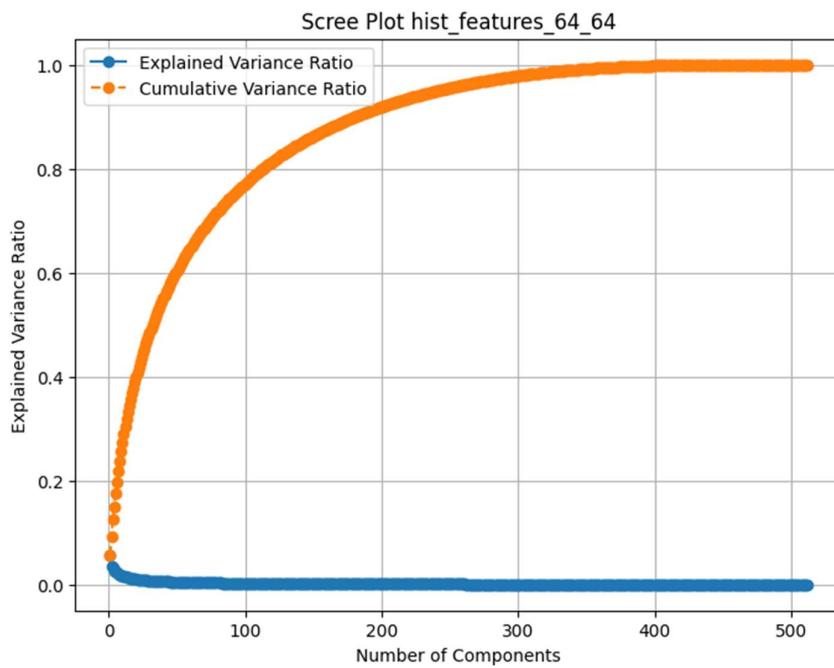
    # Plot explained variance ratio and cumulative variance ratio
    plt.figure(figsize=(8, 6))
    plt.plot(np.arange(1, len(explained_variance_ratio) + 1), explained_variance_ratio, marker='o')
    plt.plot(np.arange(1, len(cumulative_variance_ratio) + 1), cumulative_variance_ratio, marker='o', linestyle='--')
    plt.xlabel('Number of Components')
    plt.ylabel('Explained Variance Ratio')
    plt.title(f'Scree Plot ({df.name})')
    plt.legend(['Explained Variance Ratio', 'Cumulative Variance Ratio'])
    plt.grid(True)
    plt.show()

    print(f'\nName of Features: {df.name}')

    # Determine the optimal number of components
    optimal_num_components = np.argmax(cumulative_variance_ratio >= th) + 1
    print("Optimal number of components:", optimal_num_components)

    return optimal_num_components
```

شکل (1-۵۶) کد پایتون انتخاب بهترین تعداد component در PCA



شکل (۱-۵۷) Variance ratio بر حسب تعداد کامپونت های PCA

```
def reduce_dimensions_and_cluster(data, n_components, min_cluster_size=5):
    # Extract features and labels
    features = data.iloc[:, :-1]
    labels = data.iloc[:, -1]

    # Scale the features
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(features)

    # Perform PCA dimensionality reduction
    pca = PCA(n_components=n_components, random_state=42)
    reduced_features = pca.fit_transform(scaled_features)

    # Perform clustering using HDBSCAN
    clusterer = HDBSCAN(min_cluster_size=min_cluster_size)
    clusters = clusterer.fit_predict(reduced_features)

    # Compute evaluation metrics
    silhouette = silhouette_score(reduced_features, clusters)

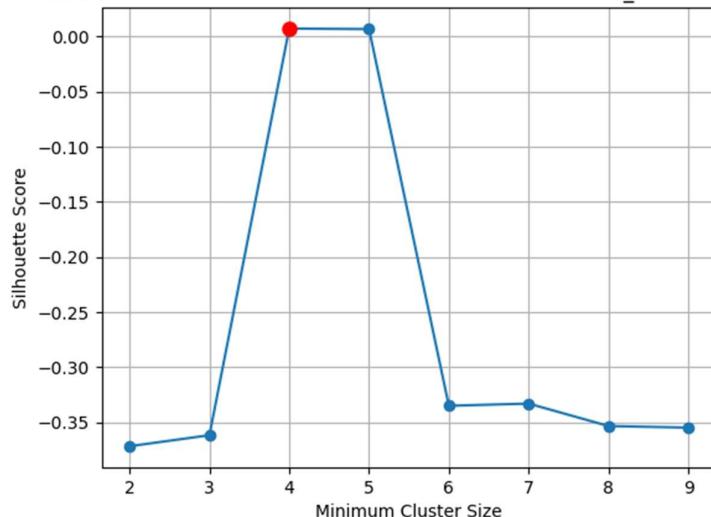
    # Create a new dataframe with reduced features and assigned clusters
    result = pd.DataFrame(reduced_features, columns=[f"Dimension {i+1}" for i in range(n_components)])
    result["Cluster"] = clusters

    return result, silhouette
```

شکل (۱-۵۸) کاهش بعد فضای اعمال HDBSCAN

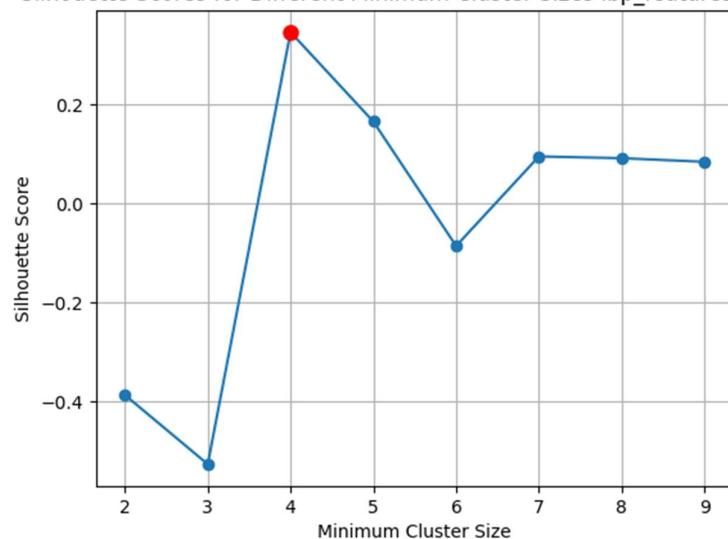
در ادامه نیز مقادیر Silhouette در حالت کلی و به ازای هر فیچر به صورت زیر میباشد. که میزان تفکیک پذیری را نشان میدهد و میتوان در این حالت فیچری که بیشترین تفکیک پذیری و اینکه در چند کلاستر را دارا میباشد پیدا کرد.

Silhouette Scores for Different Minimum Cluster Sizes hist_features_64_64

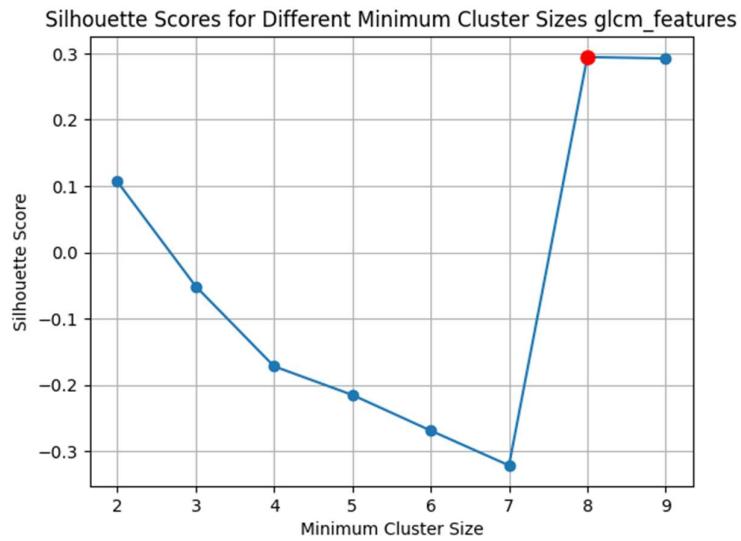


شکل (۱-۵۹) امتیاز Sillhoette برای تمامی فیچر ها بر حسب کمترین تعداد کلاستر

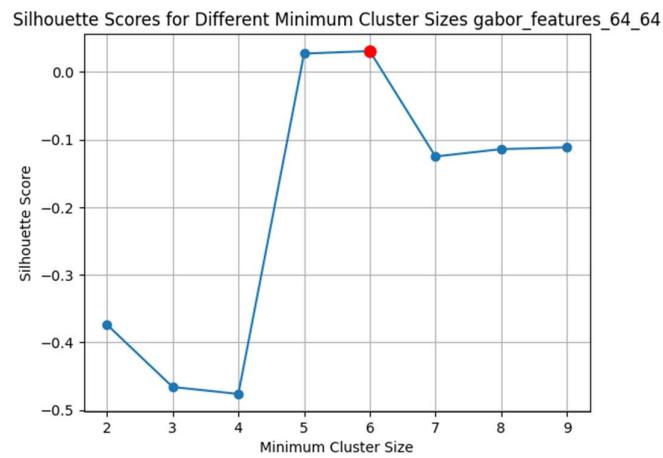
Silhouette Scores for Different Minimum Cluster Sizes lbp_features_64_64



شکل (۱-۶۰) امتیاز Sillhoette برای فیچر LBP بر حسب کمترین تعداد کلاستر



شکل (۱-۶۱) امتیاز Sillhoette برای فیچر glcm بحسب کمترین تعداد کلاستر



شکل (۱-۶۲) امتیاز Sillhoette برای فیچر gabor بحسب کمترین تعداد کلاستر

با توجه به مقادیر بهترین فیچر برای کلاستر فیچر lbp میباشد.

BIRCH -1-5-6-2

(کاهش و خوشه بندی متوازن تکرار شونده با استفاده از سلسله مراتب) یک الگوریتم خوشه بندی

سلسله مراتبی است که به ویژه برای مجموعه داده های بزرگ مناسب است. که توسط Raghu Tian Zhang

در سال 1996 معرفی شد. BIRCH برای خوشبندی کارآمد داده ها با ایجاد Miron Livny و Ramakrishnan

ساختار سلسله مراتبی از زیرخوشه ها طراحی شده است که امکان عملیات خوشبندی سریع و مقیاس پذیر را فراهم می کند.

ایده اصلی پشت BIRCH این است که مجموعه داده را با استفاده از یک ساختار داده فشرده و کارآمد به نام درخت ویژگی خوش ای (CFT) نشان دهد. CFT یک ساختار درختی است که اطلاعات ضروری در مورد توزیع داده ها و روابط خوشبندی را جمع آوری می کند. از مجموعه ای از زیرخوشه های غیر همپوشانی تشکیل شده است که هر زیرخوش با یک مرکز، یک شعاع و تعدادی نقطه نشان داده می شود. CFT امکان به روز رسانی کارآمد و افزایشی را فراهم می کند زیرا نقاط داده جدید به فرآیند خوشبندی اضافه می شوند.

الگوریتم BIRCH در دو مرحله اصلی عمل می کند: خوشبندی و پالایش. در مرحله خوشبندی، BIRCH نقاط داده را به صورت جریانی پردازش می کند. از مجموعه ای از پارامترها مانند ضریب انشعاب و مقادیر آستانه برای شعاع و تعداد نقاط استفاده می کند تا نحوه تشکیل زیرخوشه ها را تعیین کند. در ابتدا، هر نقطه داده به عنوان یک زیرخوشه جدید در CFT درج می شود. با رسیدن نقاط داده جدید، BIRCH به صورت بازگشتی زیرخوشه ها را ادغام و تقسیم می کند تا ویژگی های مورد نظر مانند فشردگی و جدایی را حفظ کند.

در مرحله Refinement، BIRCH یک الگوریتم خوشبندی، معمولاً یک روش مبتنی بر چگالی مانند DBSCAN را برای زیرخوشه های به دست آمده از مرحله خوشبندی اعمال می کند. این امر کیفیت نتایج خوشبندی را با ثبت ساختارهای ظریفتر در هر زیرخوشه بهبود می بخشد.

یکی از مزایای BIRCH توانایی آن در مدیریت کارآمد مجموعه داده های بزرگ است. با استفاده از ساختار داده CFT، BIRCH نیاز به حافظه و پیچیدگی محاسباتی را در مقایسه با الگوریتم های خوشبندی سنتی کاهش می دهد. این می تواند نقاط داده را در یک پاس پردازش کند و ساختار خوشبندی را به صورت تدریجی به روز

کند و آن را برای سناریوهای خوش بندی بلاذرنگ یا آنلاین مناسب کند.

با این حال، توجه به این نکته مهم است که BIRCH محدودیت هایی دارد. ممکن است با مجموعه های داده ای که دارای خوش هایی با اندازه ناهموار یا خوش هایی با تراکم های متفاوت هستند، مشکل داشته باشد. علاوه بر این، BIRCH به تنظیم دقیق پارامترها، مانند تعیین آستانه های مناسب و فاکتورهای انشعاب، برای دستیابی به نتایج خوش بندی رضایت بخشی نیاز دارد.

از کد های زیر برای پیاده سازی استفاده می شود.

```
def find_best_birch_clustering(dataframe, threshold_list, branching_factor_list):
    best_silhouette_score = -1
    best_threshold = None
    best_branching_factor = None
    best_clustered_df = None

    for threshold in threshold_list:
        for branching_factor in branching_factor_list:
            try:
                # Initialize BIRCH with the specified parameters
                birch = Birch(threshold=threshold, branching_factor=branching_factor)

                # Fit the BIRCH model on the data
                birch.fit(dataframe)

                # Predict the cluster labels for the data
                cluster_labels = birch.predict(dataframe)

                # Compute the Silhouette score
                silhouette_avg = silhouette_score(dataframe, cluster_labels)

                # Update the best scores and parameters if necessary
                if silhouette_avg > best_silhouette_score:
                    best_silhouette_score = silhouette_avg
                    best_threshold = threshold
                    best_branching_factor = branching_factor
                    best_clustered_df = dataframe.copy()
                    best_clustered_df['Cluster Label'] = cluster_labels

            except:
                pass

    # Return the best results
    return best_clustered_df, best_silhouette_score, best_threshold, best_branching_factor
```

شکل (۱-۶۳) کد پایتون انتخاب بهترین birch clustering

```

def birch(dataframes):
    for df in dataframes:
        # Create a MinMaxScaler object
        scaler = MinMaxScaler()

        # Normalize the DataFrame
        normalized_df = scaler.fit_transform(df)

        # Convert the normalized array back to a DataFrame
        normalized_df = pd.DataFrame(normalized_df, columns=df.columns)

        # Print the normalized DataFrame

        # Specify the lists of threshold and branching_factor values to try
        threshold_list = [0.1, 0.3, 0.5, 0.7]
        branching_factor_list = [10, 20, 30, 40, 50]

        # Find the best BIRCH clustering parameters and compute Silhouette score
        best_clustered_df, best_silhouette_score, best_threshold, best_branching_factor = find_best_birch_clustering(normalized_df, threshold_list)

        # Display the modified dataframe with cluster labels
        #print(best_clustered_df)
        print('\nName of Features: ', df.name)
        # Display the best Silhouette score and corresponding parameters
        print("Best Silhouette score:", best_silhouette_score)
        print("Best threshold:", best_threshold)
        print("Best branching factor:", best_branching_factor)

```

شکل (۱-۶۴) کد پایتون تابع BIRCH

خروجی ها در نهایت به شرح ذیل میباشد.

```

Name of Features: hist_features_64_64
Best Silhouette score: 0.1377217652732881
Best threshold: 0.7
Best branching factor: 50

Name of Features: lbp_features_64_64
Best Silhouette score: 0.4191079379702569
Best threshold: 0.7
Best branching factor: 10

Name of Features: glcm_features
Best Silhouette score: 0.7139869655768377
Best threshold: 0.3
Best branching factor: 10

Name of Features: gabor_features_64_64
Best Silhouette score: 0.14952556738228934
Best threshold: 0.3
Best branching factor: 50

```

۱-۵-۷- نتیجه گیری

در نتیجه، با استفاده از روش های استخراج ویژگی مبتنی بر یادگیری عمیق، می توانیم به طور موثر تصاویر

واقعی و جعلی را در محیط های مختلف مانند دریا، جنگل و کوه ها خوشه بندی کرد. با استفاده از تکنیک هایی

مانند K-means یا مدل‌های Gaussian Mixture (GMM)، می‌توانیم با موفقیت بین تصاویر واقعی و جعلی با

دو خوشه تمايز قائل شد. علاوه بر این، با استفاده از شش خوشه، می‌توان بین تصاویر واقعی و جعلی تمايز قائل شویم و در عین حال آنها را در محیط‌های مربوطه خود طبقه‌بندی کنیم: دریا، کوه یا جنگل.

با این حال، هنگام استفاده از روش‌های خوشه‌بندی کلاسیک، با محدودیت‌هایی در خوشه‌بندی تصاویر

واقعی و جعلی به عنوان کلاس‌های مجزا مواجه شده. با این وجود، این روش‌های کلاسیک هنوز برخی از قابلیت‌ها را در خوشه‌بندی تصاویر بر اساس محیط خود نشان می‌دهند. به عنوان مثال، با استفاده از دو خوشه، می‌توانی

تصاویر جنگل را از انواع دیگر تصاویر جدا کرد، در حالی که استفاده از سه خوشه به ما امکان می‌دهد تصاویر کوه و جنگل نیز به گروه‌های مجزا خوشه‌بندی شود.

به طور خلاصه، روش‌های استخراج ویژگی مبتنی بر یادگیری عمیق، در ترکیب با الگوریتم‌های خوشه‌بندی، رویکرد جامع‌تری برای خوشه‌بندی تصاویر واقعی و جعلی در محیط‌های مختلف ارائه می‌دهند. این تکنیک‌ها

دقت و انعطاف‌پذیری بیشتری را در مقایسه با روش‌های خوشه‌بندی سنتی هنگام برخورد با مجموعه داده‌های تصویری پیچیده فراهم می‌کنند.