پروژه پایانی درس مبانی بینایی کامپیوتر

موضوع پروژه: طراحی سیستم‌های Anti-Spoofing

استاد درس: آقای دکتر محمدی

اعضای تیم: کامیار مرادیان زه آب، ارشیا حسین‌زاده

نیم‌سال دوم ۱۴۰۲-۱۴۰۳

پروژه پایانی – Anti-Spoofing

هدف این پروژه، توسعه یک الگوریتم spoofing-Anti برای تشخیص زنده بودن یک چهره در ویدیو می‌باشد. که در آن یک فریم از ویدیو را گرفته و به مدل داده تا تشخیص دهد که تصویر زنده شخص است یا خیر. به‌طور مثال تصویر دارای گریم یا پرینت شده از وی است یا تصویر از یک دستگاه دیگر درحال پخش است یا نه.

برای این منظور نیاز است که به دو روش پیش رویم. روش نخست با استفاده از مهندسی ویژگی و روش دوم مبتنی بر یادگیری عمیق است. شرح هر یک از روش‌ها در زیر آورده شده است.

روش نخست – مهندسی ویژگی

دیتاست: دیتاست مورد نظر برای این بخش، دیتاست CASIA FASD است. این دیتاست در Kaggle موجود است:

https://www.kaggle.com/datasets/minhnh2107/casiafasd

این دیتاست متشکل از تصاویر دوبعدی و همچنین عمق مربوط به هر تصویر است. برای آموزش مدل، تنها بخش رنگی تصاویر در نظر گرفته شده است. کلاس‌های موجود در این دیتاست به دو صورت real و spoof هستند که بخش spoof هم شامل تصاویر بازپخش و هم شامل تصاویر با چهرهٔ پرینت شده می‌باشد.

تعداد داده‌های مربوط به تست برابر با ۲۴۰۸ و همچنین داده‌های آموزشی برابر با ۱۶۵۵ مورد می‌باشند. از طرفی داده‌ها بیشتر شامل داده‌های spoof بوده و به عبارتی مقداری ناهمگونی در داده موجود است. توزیع داده‌ها در این دیتاست به صورت زیر است:

	train	val	test
real	1223	405	314
spoof	7076	2543	7266

در کنار دیتاست بالا از دیتاست LLC and CASIA نیز استفاده شده است. که یک دیتاست با شمار بیشتری داده نسبت به دیتاست قبلی است. این دیتاست از ترکیب دو دیتاست LLC و نیز CASIA ساخته شده است. لینک این دیتاست در زیر است:

https://www.kaggle.com/datasets/ahmedruhshan/lcc-fasd-casia-combined

همچنین برای مرحلهٔ تست، از یک دیتاست متشکل از ویدئوها استفاده شده است. این دیتاست شامل ویدئوهایی است که مربوط به دو دستهٔ زنده و غیر آن تقسیم‌بندی شده‌اند. ویدئوها کوتاه بوده و اغلب در حد ۲ تا ۳ ثانیه و حتی کوتاه‌تر هستند. لینک مربوط به این دیتاست در زیر آورده شده است:

https://www.kaggle.com/datasets/trainingdatapro/ibeta-level-1-liveness-detection-dataset-part-1

توضیحات بیشتر در رابطه با این دیتاست در بخش تست مربوط به یادگیری عمیق آورده شده است.

کتابخانه‌ها و ابزار استفاده شده: به طور ویژه از کتابخانهٔ scipy استفاده شده است. چرا که بخش عظیمی از کد مربوط به feature engineering با استفاده از این کتابخانه نوشته شده است. همچنین در کنار این کتابخانه از کتابخانهٔ tensorflow به منظور بررسی استفاده از معماری شبکهٔ عمیق تماماً متصل به عنوان دسته‌بند برای ویژگی‌های استخراج شده، استفاده شده است. از کتابخانه‌های کمکی دیگر نظیر pandas، numpy و matplotlib و همچنین open-cv نیز استفاده شده است.

ویژگی‌های استخراج شده: برای بخش ویژگی‌ها هفت ویژگی در نظر گرفته شده است.

پروژه پایانی – Anti-Spoofing

- استخراج چهره: برای این ویژگی، از دسته‌بند haarcascade_frontalface_default استفاده شده است. این دسته‌بند توسط کتابخانهٔ Open-CV پیاده‌سازی شده است و برای استفاده از آن کافی‌ست مدل را لود کرده و تصویر ورودی را به آن وارد کنیم. این دسته‌بند از ویژگی‌های Haar استفاده می‌کند و ویژگی‌هایی نظیر لبه‌ها و اشکال دایره‌ای در تصویر را تشخیص می‌دهد و با استفاده از روش‌هایی نظیر AdaBoost آموزش دیده است تا بتواند در مدت زمان کوتاهی از میان ویژگی‌های موجود در یک تصویر مهم‌ترین ویژگی‌ها را انتخاب کند. پس از آنکه بخشی از تصویر که به عنوان چهره تشخیص داده شد، آن بخش از تصویر را می‌بریم و به مدل بعدی تحویل می‌دهیم. همچنین در صورتی که چهره‌ای تشخیص داده نشود، تمام تصویر پس از خاکستری شدن، به عنوان خروجی بیرون ارسال می‌شود. به منظور انسجام میان خروجی‌ها، همگی به اندازهٔ مشخصی کراپ شده‌اند. برای رسیدن به این ویژگی از کلاس FaceDetector استفاده شده است. اطلاعات بیشتر در رابطه با این مدل و روش آموزش آن در لینک زیر آورده شده است:

https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

- ویژگی‌های محلی دودویی: این ویژگی مربوط به بافت تصویر است. به این صورت که در ابتدا تصویر را به یک شبکهٔ چند سلولی تقسیم می‌کنیم (برای این مدل به صورت ۸ × ۸) و در ادامه برای هر سلول هیستوگرام LBP را محاسبه می‌کنیم. شعاع در نظر گرفته شده برای این ویژگی ۱ بوده و تعداد نقاط نیز برابر با ۸ می‌باشد. همچنین برای محاسبهٔ LBP از متد یکنواخت استفاده شده است، تا هیستوگرام بدست آمده را تا حد امکان کوچکتر کند. این ویژگی در کلاس MultiGridLBP پیاده‌سازی شده است. روش کار آن نیز به صورت زیر است:
  - این روش ویژگی‌های LBP را از تصاویر ورودی استخراج می‌کند.
  - ابتدا اندازه تصویر به ۱۲۸×۱۲۸ تغییر می‌کند.
  - اگر تصویر رنگی باشد، به تصویر خاکستری تبدیل می‌شود.
  - تصویر به شبکه‌های کوچکتر تقسیم می‌شود و LBP برای هر شبکه محاسبه می‌شود.
  - هیستوگرام LBP برای هر شبکه محاسبه و به ویژگی‌ها اضافه می‌شود.
  - در نهایت، ویژگی‌ها به صورت آرایه‌ای از ویژگی‌های LBP برگردانده می‌شوند.
- فیلترهای گابور: این فیلترها مجموعه‌ای از فیلترهای خطی‌اند که برای تحلیل فرکانس فضایی تصویر استفاده می‌شود. در ساخت این فیلترها از ضرب موج‌های سینوسی در توابع گاوسی استفاده می‌شود. با توجه به ویژگی منحصر به فردشان در استخراج بافت به هر دو شکل محلی و سراسری، و همچنین معروفیتشان به نزدیک بودن به بنایی انسان، به طور ویژه‌ای از آن در استخراج ویژگی‌های مربوط به بافت از تصاویر استفاده می‌شود. برای این فیلتر، پارامترهایی نظیر لامبدا (برای در نظر گرفتن ضخامت هر ویژگی)، سایز بلاک‌ها، تتا (برای زاویهٔ هر یک از فیلترها) و ... قابل تعریف است. اطلاعات بیشتر در رابطه با فیلتر و نیز پارامترها در لینک زیر موجود است.

https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

برای پیاده‌سازی این ویژگی کلاس GaborFeatureExtractor پیاده‌سازی شده است. روش کار این کلاس به این شرح است:

  - ابتدا اندازه تصویر به ۱۲۸×۱۲۸ تغییر می‌کند.
  - اگر تصویر رنگی باشد، به تصویر خاکستری تبدیل می‌شود.
  - تصویر به داده‌های float32 تبدیل می‌شود.

- اگر تصویر خالی باشد، ویژگی‌های صفر برمی‌گرداند، در غیر این صورت فیلترهای گابور اعمال شده و میانگین و واریانس نتایج به عنوان ویژگی استخراج می‌شود. تعداد ویژگی‌های بدست آمده از این کلاس، برابر با تمام حالات در نظر گرفته شده برای پارامترها ضرب‌در۲ (برای میانگین و واریانس) است.

- ویژگی HOG: ویژگی‌های HOG برای استخراج الگوهای محلی از تصاویر به کار می‌روند. این ویژگی‌ها به‌ویژه در تشخیص اشیاء مانند شناسایی چهره و انسان بسیار مؤثر هستند. HOG با محاسبه هیستوگرام گرادیان‌های جهت‌دار در نواحی کوچک از تصویر، ویژگی‌های قدرتمندی را استخراج می‌کند که برای تشخیص الگوهای محلی مفید هستند. بنابراین در اولین گام نیاز است که تصویر ورودی را به سلول‌های کوچکتری تقسیم کنیم و HOG را در هر ناحیه از تصویر اعمال کنیم. این ویژگی در کلاس HOGFeatureExtractor پیاده‌سازی شده است که در آن به منظور نتیجه‌گیری بهتر از K-means استفاده شده است. روش کار این کلاس به صورت زیر است:
  - این روش ویژگی‌های HOG را از تصاویر ورودی استخراج می‌کند.
  - ابتدا اندازه تصویر به x128۱۲۸ تغییر می‌کند.
  - یک HOGDescriptor برای محاسبه ویژگی‌های HOG از تصویر ایجاد می‌شود.
  - ویژگی‌های HOG محاسبه شده و سپس با استفاده از KMeans خوشه‌بندی می‌شوند.
  - ویژگی‌های خوشه‌بندی شده به عنوان ویژگی نهایی استخراج شده و بازگردانده می‌شوند.
- ویژگی DOG: روش Difference of Gaussians یا DOG یک تکنیک در پردازش تصویر است که برای شناسایی لبه‌ها و ویژگی‌های بافتی در تصاویر به کار می‌رود. این روش با کاهش دو تصویر گوسی که با مقیاس‌های مختلف صاف شده‌اند، لبه‌ها و تغییرات ناگهانی در شدت نور را برجسته می‌کند. بنابراین این ویژگی شامل سه گام اصلی است:
  - اعمال فیلتر گوسی:
    - اعمال فیلتر گوسی با دو انحراف استاندارد مختلف (sigma1 و sigma2) به تصویر.
  - محاسبه تفاوت:
    - تفاوت بین دو تصویر گوسی به دست می‌آید که نتایج آن تغییرات ناگهانی در شدت نور را نشان می‌دهد.
  - چند مقیاسه:
    - این فرآیند در چندین مقیاس مختلف انجام می‌شود تا ویژگی‌ها در سطوح مختلف تصویر استخراج شوند.

برای این ویژگی، کلاس DOGFeatureExtractor پیاده‌سازی شده است. روش کار این کلاس به صورت زیر است:

  - ابتدا اندازه تصویر به x128۱۲۸ تغییر می‌کند.
  - در هر مقیاس، فیلتر گوسی با دو انحراف استاندارد متفاوت اعمال می‌شود.
  - تفاوت بین دو تصویر گوسی محاسبه و به عنوان ویژگی DOG ذخیره می‌شود.
  - ویژگی‌های چند مقیاسه به یکدیگر پیوسته و به عنوان ویژگی نهایی بازگردانده می‌شوند.

- ویژگی نقاط کلیدی با استفاده از SIFT: SIFT یک الگوریتم قدرتمند در پردازش تصویر و بینایی کامپیوتری است که برای استخراج و توصیف ویژگی‌های محلی به ویژه نقاط کلیدی (نظیر گوشه‌ها) از تصاویر به کار می‌رود. ویژگی‌های SIFT مقیاس‌پذیر و نسبت به تغییرات چرخشی و نوری مقاوم هستند. این ویژگی‌ها به‌ویژه در تطابق و شناسایی اشیا بسیار مؤثر هستند. پس از بدست آوردن نقاط کلیدی موجود در یک تصویر، برای هر یک از آن‌ها توصیفگرهای استخراج می‌شوند که هر یک نسبت به تغییراتی نظیر چرخش و انتقال حساس نیستند. برای پیاده‌سازی این ویژگی کلاس SIFTFeatureExtractor پیاده‌سازی شده است:

- ابتدا اندازه تصویر به ۱۲۸×۱۲۸ تغییر می‌کند.
- نقاط کلیدی و توصیف‌گرها (descriptors) از تصویر استخراج می‌شوند.
- اگر تعداد توصیف‌گرها کمتر از n_keypoints (تعداد نقاط کلیدی مطلوب) باشد، با مقادیر صفر پر می‌شوند.
- اگر تعداد توصیف‌گرها بیشتر از n_keypoints باشد، به تعداد n_keypoints محدود می‌شوند. به این صورت که به تعداد مطلوب از یک سر آرایه ویژگی‌ها استخراج می‌شوند.
- ویژگی‌های نهایی به صورت تخت شده (flattened) ذخیره می‌شوند.

- ویژگی تبدیل فوریه و فرکانس تصویر: FFT یک الگوریتم کارآمد برای محاسبه تبدیل فوریه‌ی سریع است که برای تحلیل فرکانسی سیگنال‌ها و تصاویر استفاده می‌شود. در تحلیل تصاویر، FFT به ما امکان می‌دهد که اطلاعات فرکانسی تصویر را استخراج کنیم و الگوها و ساختارهای تکرارشونده را شناسایی کنیم. این ویژگی با هدف به دست آوردن ویژگی‌های مربوط به حالت تست (یعنی تست با فرکانس) تعریف شده است. برای محاسبهٔ این ویژگی، سه مرحلهٔ زیر را طی می‌کنیم:
  - محاسبه تبدیل فوریه‌ی تصویر برای استخراج اطلاعات فرکانسی.
  - جابجایی طیف فرکانسی برای قرار دادن فرکانس‌های پایین در مرکز.
  - محاسبه طیف اندازه از طریق تبدیل مقادیر مختلط به مقادیر حقیقی.

برای پیاده‌سازی این ویژگی کلاس FFTFeatureExtractor استفاده شده است. روش کار این کلاس به صورت زیر است:

- ابتدا اگر تصویر به صورت رنگی باشد، به تصویر خاکستری تبدیل می‌شود.
- سپس اندازه تصویر به ۱۲۸×۱۲۸ تغییر می‌کند.
- تبدیل فوریه دوبعدی (FFT 2D) بر روی تصویر اعمال می‌شود.
- طیف فرکانسی تصویر جابجا می‌شود تا فرکانس‌های پایین در مرکز قرار گیرند.
- طیف اندازه محاسبه می‌شود و با یک مقدار کوچک (eps) از صفر یا مقادیر منفی جلوگیری می‌شود.
- طیف اندازه تخت (flatten) شده و به عنوان ویژگی نهایی ذخیره می‌شود.

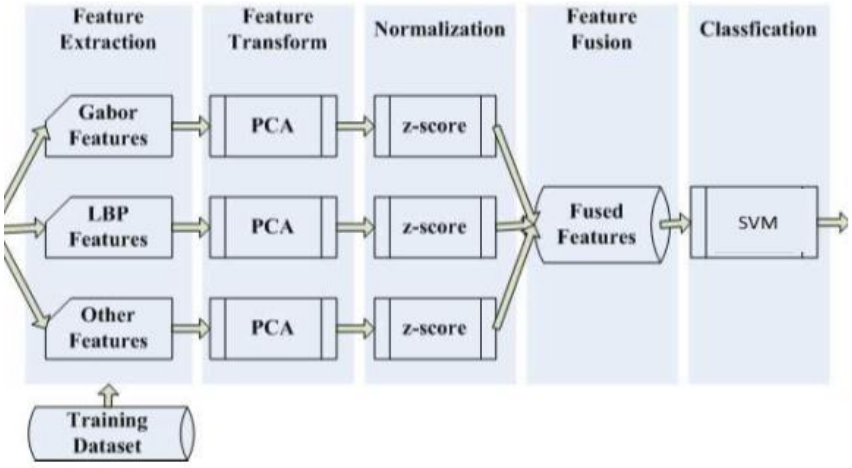
دسته‌بند: برای دسته‌بند، از سه دسته‌بند SVM، ETC و نیز معماری FC استفاده شده است.

- SVM: یک الگوریتم یادگیری نظارت شده است که عمدتاً برای مسائل طبقه‌بندی استفاده می‌شود. این الگوریتم با یافتن یک ابر صفحه که بهترین جداکننده بین کلاس‌ها در فضای ویژگی‌ها است، کار می‌کند. با توجه به فضای نسبتاً بزرگ ویژگی‌ها ورودی این دسته‌بند، به عنوان یک انتخاب مناسب ظاهر شده است. کرنل‌های مختلفی برای این مدل طراحی شده است، که با توجه به بررسی و مطالعه، کرنل RBF انتخاب شد.
- ETC: این الگوریتم مربوط به ایجاد چندین درخت تصمیم‌گیری تصادفی و در پایان انتخاب بهترین مورد است که بر روی داده‌های ورودی فیت شده است. بنابراین احتمال بیش‌برازش با توجه به این ویژگی این الگوریتم بسیار بالا می‌رود که شاهد این مورد در فرآیند آموزش مدل بودیم.
- FC: این الگوریتم سختی بالا و نیز داده‌های بسیار خوب و تعداد بالایی نیاز دارد تا بتواند جواب خوبی را به عنوان خروجی بیرون دهد. با توجه به داده‌های اندک و نیز ناهمگنی‌ای که در این داده‌ها موجود بود، این مدل به خوبی نتوانست که به این داده‌ها فیت شود.

نگاهی کلی به معماری استفاده شده: با توجه به نکات گفته شده در رابطه با SVM و ویژگی‌های تعریف شده، نیاز بر این بود تا بهترین ترکیب از ویژگی‌ها را انتخاب کنیم. در ابتدا مجموعه‌ای از پایپ‌لاین‌ها را به ازای هر یک ویژگی‌ها تعریف کردیم. این

پروژه پایانی – Anti-Spoofing

طراحی مطابق با تصویر زیر از مقالهٔ Fusing Gabor and LBP Feature Sets for Kernel-Based Face Recognition است:



(تصویر مطابق با معماری استفاده اندکی تغییر کرده است)

بنابراین در ابتدا هر یک از ویژگی‌ها را بر روی تصویر ورودی اعمال می‌کنیم و در ادامه با استفاده از PCA که به منظور کاهش ابعاد ویژگی‌های بدست آمده استفاده می‌شود، ابعاد را کاهش می‌دهیم و در پایان با استفاده از z-score عملیات نرمال‌سازی مقادیر بدست آمده را انجام می‌دهیم. در پایان نیز، نیاز است که تمام بردارهای ویژگی را با همدیگر پیوست داد تا ویژگی‌های نهایی بدست آید. ویژگی‌های بدست آمده را نیز از SVM گذر می‌دهیم تا نتیجهٔ نهایی را از مدل کسب کنیم.

به منظور پیاده‌سازی هر یک از مراحل از Pipeline استفاده شده است. این کلاس مربوط به کتابخانهٔ Sklearn بوده که این کمک را می‌کند که کلاس‌ها را بتوانیم به صورت مرحله به مرحله بر روی داده‌های ورودی اعمال کنیم. هر کلاسی یک پایپ‌لاین مخصوص خود دارد که هر سه مرحلهٔ (اعمال ویژگی، PCA و سپس نرمال‌سازی) بر روی آن‌ها انجام می‌شود. تنها در ویژگی gabor و LBP هستند که در ابتدا چهره از تصویر جدا شده و سپس ویژگی‌ها بر روی چهره اعمال می‌شوند. همچنین تعداد کامپوننت‌ها برای LBP وGabor ۳۰ مورد و برای باقی ویژگی‌ها ۵۰ مورد در نظر گرفته شده است.

همچنین برای Gabor دو پایپ‌لاین در نظر گرفته شده است. یک مورد مربوط به کل تصویر، و دیگری برای حالتی که چهره از تصویر جدا شده است. بنابراین پایپ‌لاین اول با توجه به سایز بلاک تعریف شده و سایر پارامترها، ویژگی‌های سراسری‌تری را از تصویر استخراج می‌کند.

آزمون و خطای ویژگی‌ها: در گام بعدی برای انتخاب بهترین ویژگی‌ها، مجموعه‌ای از پایپ‌لاین مربوط به تمام ویژگی‌ها تشکیل دادیم. این مجموعه شامل ترکیبات یک عضوی تا ۴ عضوی می‌باشد. همچنین برای ارزیابی هر ترکیب از سه متریک Accuracy، Precision و Recall استفاده شده است. به طور کلی ترکیبات تک عضوی قدرت لازم برای دریافت نتایج مناسب را در مقایسه با سایر ترکیبات نداشته و در نتیجهٔ نهایی آورده نشده‌اند. با این حال نتایج در نوت‌بوک قرار گرفته است. همچنین برای هر یک از ترکیبات از کرنل RBF استفاده شده است. دیتاست استفاده شده نیز، دیتاست CASIA می‌باشد.

علت انتخاب کرنل RBF برآمده از استفاده از GridSearch جهت انتخاب بهترین مدل است.

مشکل اصلی این روش از آزمون و خطا کردن ترکیبات، زمان‌بر بودن فرآیند آن می‌باشد که برای همین منظور از اکانت‌های مختلف کولب استفاده شد و نتایج به طور جدا گانه در فایل‌های CSV قرار گرفت. در پایان تمام این نتایج مرج شده و بهترین ترکیبات مشخص شده‌اند. ده مورد اول در زیر آورده شده‌اند:

ترکیب استفاده شده	Precision	Recall	Accuracy	امتیاز میانگین	رتبه
lbp_pipeline + gabor_pipeline + dog_pipeline +...	0.875000	0.758037	0.914037	0.849025	1.0
lbp_pipeline + gabor_pipeline + hog_pipeline +...	0.834646	0.717428	0.895764	0.815946	2.0
lbp_pipeline + gabor_pipeline + dog_pipeline	0.838323	0.710660	0.895349	0.814777	3.0
lbp_pipeline + gabor_full_image_pipeline + dog...	0.823301	0.717428	0.892857	0.811195	4.0
lbp_pipeline + gabor_full_image_pipeline + dog...	0.796262	0.720812	0.886213	0.801095	5.0
lbp_pipeline + gabor_full_image_pipeline + hog...	0.800774	0.700508	0.883721	0.795001	6.0
lbp_pipeline + gabor_pipeline + hog_pipeline +...	0.795802	0.705584	0.883306	0.794897	7.0
lbp_pipeline + dog_pipeline	0.834409	0.656514	0.883721	0.791548	8.0
lbp_pipeline + gabor_pipeline + gabor_full_ima...	0.808333	0.656514	0.877492	0.780780	9.0
gabor_pipeline + gabor_full_image_pipeline + d...	0.823144	0.637902	0.877492	0.779513	10.0

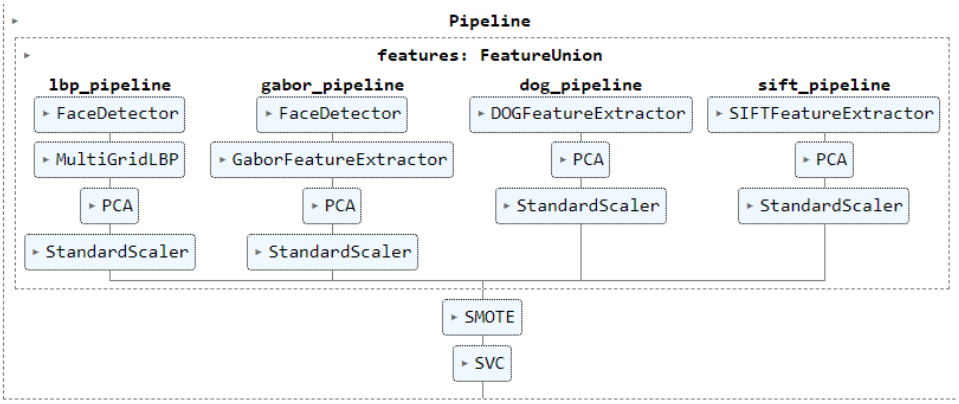
اطلاعات کامل این جدول در results.csv موجود است.

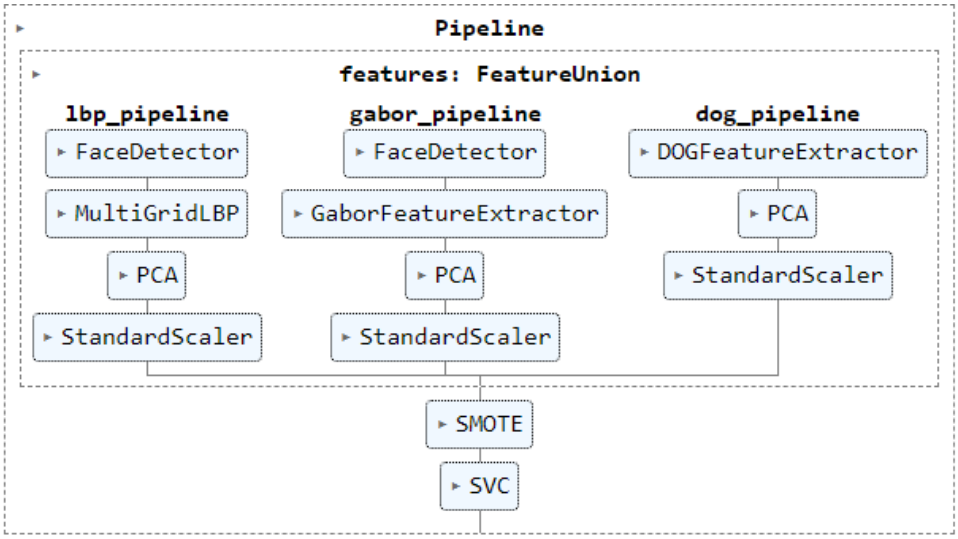
انتخاب مدل‌ها و اعمال آموزش: دو مدل موجود در رتبهٔ اول و سوم برای آموزش انتخاب شدند. چرا که مدل اول و دوم دارای چهار ویژگی و مدل سوم دارای سه ویژگی بوده و نسبت به مدل دوم تفاوت زیادی در امتیازات ندارد. در ابتدا دیتاست بزرگتر برای آموزش در نظر گرفته شد. اما به دلیل استفاده زیاد از مموری چندین مرتبه کولب ری‌استارت شد و در نتیجه از استفاده از این دیتاست صرف نظر شد. (هر چند یک بار به اندازهٔ ۴۰۰۰ داده از آن در نظر گرفته شد که نتیجهٔ خوبی بر روی پایپ‌لاین‌های تعریف شده با SVM نداشتند.)

با انتخاب دیتاست اول و دو مدل گفته شده، آموزش مدل انجام شد و در پایان هر یک از مدل‌ها برای استفاده‌های بعدی در کولب ذخیره شدند. از جمله مشکلات ذخیره‌سازی مدل، می‌توان به نبود امکان برای ذخیره‌سازی داده‌های مربوط به open-cv که در کانستراکتور هر کلاس تعریف شده‌اند، اشاره کرد. که برای رفع این مشکل دو تابع get_state و set_state برای دو کلاس SIFTFeatureExtractor و FaceDetector تعریف شدند.

برای کمرنگ کردن مشکل مربوط به ناهمگنی موجود در دیتاست نیز از SMOTE استفاده شده است که یک Upsampler می‌باشد.

مدل‌های آموزش دیده دارای معماری‌های زیر هستند:





در پایان نیز مدل‌ها ذخیره شدند. با توجه به اینکه مدل اول باز هم نتیجهٔ بهتری از خود نشان داد و تفاوت زیادی از نظر مدت زمان پاسخگویی نداشته است، همین مدل برای آموزش نهایی که در آن SVM دارای ویژگی احتمالات است، استفاده شده است.

Model: "sequential_43"

Layer (type)	Output Shape	Param #
dense_256 (Dense)	(None, 1024)	164864
dropout_204 (Dropout)	(None, 1024)	0
dense_257 (Dense)	(None, 512)	524800
dropout_205 (Dropout)	(None, 512)	0
dense_258 (Dense)	(None, 512)	262656
dropout_206 (Dropout)	(None, 512)	0
dense_259 (Dense)	(None, 256)	131328
dropout_207 (Dropout)	(None, 256)	0
dense_260 (Dense)	(None, 64)	16448
dropout_208 (Dropout)	(None, 64)	0
dense_261 (Dense)	(None, 32)	2080
dropout_209 (Dropout)	(None, 32)	0
dense_262 (Dense)	(None, 1)	33

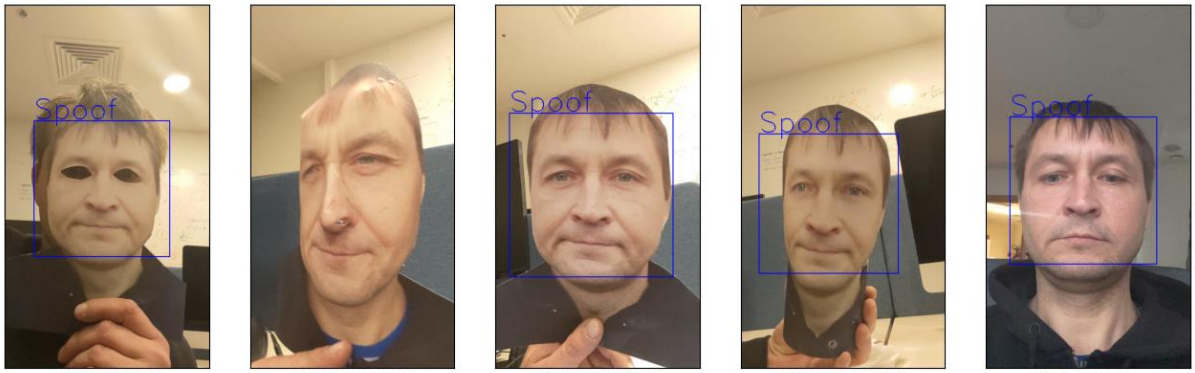
Total params: 1102209 (4.20 MB)
Trainable params: 1102209 (4.20 MB)
Non-trainable params: 0 (0.00 Byte)

تلاش برای استفاده از FC: در ادامه تلاش شد که از FC استفاده شود که با شکست روبه‌رو شدیم. در این مرحله تلاش کردیم که بر روی دیتاست که بزرگتر بوده و تمام ویژگی‌های تعریف شده یک مدل FC با ساختار مقابل را آموزش دهیم. اما با توجه به مشکلاتی نظیر دادهٔ کم موفق به آموزش موفق مدل نشدیم و اگر چه که به خوبی بر روی داده‌های آموزشی عمل کرد اما بر روی داده‌های تست با شکست روبه‌رو شد.

برای این مدل از بهینه‌ساز AdamW استفاده شده است.

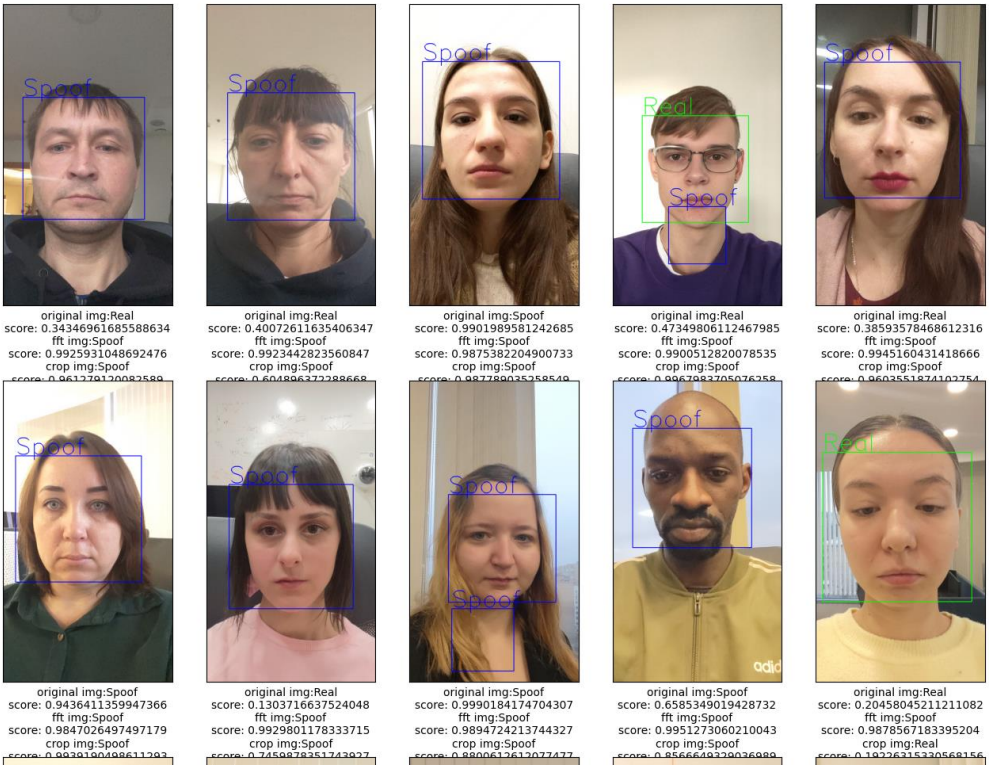
همچنین برای جلوگیری از تاثیر ناهمگنی داده‌ها به طور دستی وزن‌های اولیه به طریقی تنظیم شد تا مدل به سمت داده‌های کمتر سوق داشته باشد که البته این موضوع باعث بایاس شدن مدل به این سمت می‌شود.

تست مدل بر روی داده‌های تست: در پایان آخرین مدل آموزش دیده را بر روی داده‌های تست که مجموعه داده‌ای متشکل از ویدئوها است ارزیابی می‌کنیم. توضیحات مربوط به طریقهٔ تست کردن مدل در بخش «ب» که مربوط به یادگیری عمیق است آورده شده است. در اینجا صرفاً خروجی‌ها نمایش داده می‌شوند.

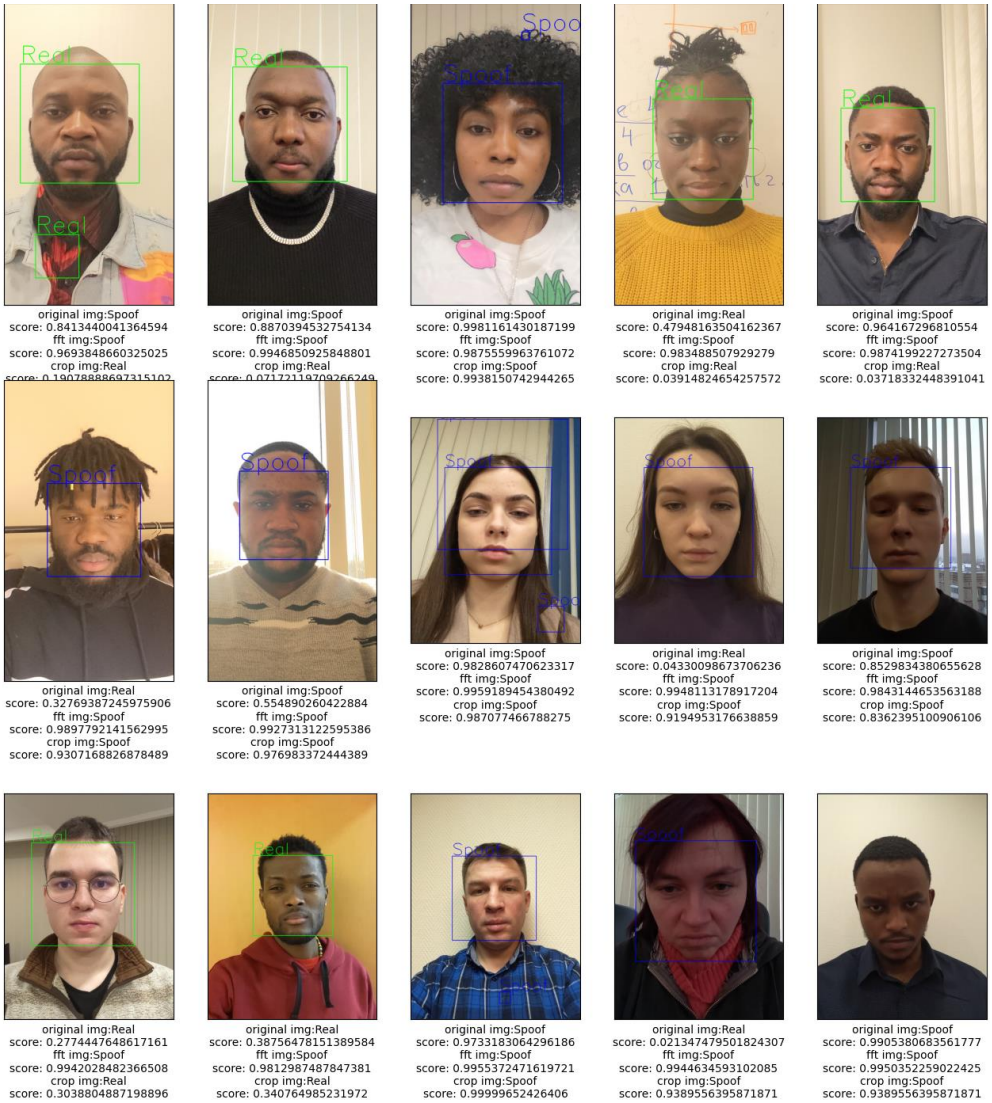


در بالا داده‌های Spoof به مدل داده شده است. امتیاز موجود در زیر هر تصویر مربوط به میزان spoof بودن آن تصویر است. همانطور که مشخص است هر پنج تصویر به عنوان spoof در نظر گرفته شده‌اند.

همچنین در زیر نتایج مربوط به تصاویری که به عنوان تصاویر واقعی هستند به همراه احتمالات و نیز پیش‌بینی مدل آورده شده است.



پروژه پایانی – Anti-Spoofing



خروجی در فایل با عنوان predictions_feature.csv قرار گرفته است. ده مورد از ردیف‌های موجود در این فایل به صورت زیر هستند:

index	video_file	Ground Truth	liveness_score	liveness_score_crop	liveness_score_frequency
20	user046.mp4	real	0.7225552351382839	0.6961195112801104	0.005797151763349162
21	user047.mp4	real	0.6124352184861042	0.6592350147680279	0.018701251215261894
22	user048.mp4	real	0.026681693570381415	3.4757359399506527e-06	0.004462752838027884
23	user049.mp4	real	0.9786525204981757	0.061044360412812915	0.00553654068979148
24	user050.mp4	real	0.009461931643822252	0.061044360412812915	0.004964774097757529
25	mask.mp4	spoof	0.7366582266209307	0.01093179112306153	0.010242538080244445
26	mask3d.mp4	spoof	0.0016344478472075652	0.01093179112306153	0.009347714515962946
27	outline.mp4	spoof	0.01010185161677113	0.018264648380300286	0.0045623853114992174
28	outline3d.mp4	spoof	0.002339587936921239	0.061538490615719255	0.007216046171718671

29	real.mp4	spoof	0.6565303831441136	0.03872087991741102	0.007406895130752433

مقاله‌ها و وب‌سایت‌ها استفاده شده برای این بخش عبارتند از:

https://github.com/ee09115/spoofing_detection

https://github.com/juan-csv/face_liveness_detection-Anti-spoofing/tree/master

https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

https://www.researchgate.net/publication/225122513_Fusing_Gabor_and_LBP_Feature_Sets_for_Kernel-Based_Face_Recognition

https://liris.cnrs.fr/Documents/Liris-5004.pdf

روش دوم – یادگیری عمیق

دیتاستی که در این روش مورد استفاده قرار گرفته است ترکیبی از دیتاست های CASIA و LCC FASD میباشد که در لینک زیر قابل مشاهده است:

https://www.kaggle.com/datasets/ahmedruhshan/lcc-fasd-casia-combined

در ابتدا سه مجموعه train, val, test از این دیتاست دانلود و آماده سازی میشود

```
[2]  !kaggle datasets download -d ahmedruhshan/lcc-fasd-casia-combined

Dataset URL: https://www.kaggle.com/datasets/ahmedruhshan/lcc-fasd-casia-combined
License(s): CC0-1.0
Downloading lcc-fasd-casia-combined.zip to /content
100% 4.88G/4.89G [01:12<00:00, 129MB/s]
100% 4.89G/4.89G [01:12<00:00, 72.0MB/s]

[3]  !unzip -q lcc-fasd-casia-combined.zip -d /content/lcc-fasd-casia-combined
```

با بررسی های صورت گرفته بر روی این مجموعه داده، مشخص گردید که تعداد نمونه های spoof یا جعلی بیشتر از تعداد نمونه های زنده میباشند. به همین خاطر بخشی از نمونه های spoof حذف میگردند تا دیتاست balance شود و آموزش مدل به درستی صورت گیرد.

پروژه پایانی – Anti-Spoofing

```
[5]  def balance_dataset(directory):

         all_files = os.listdir(directory)

         image_extensions = {'.jpg', '.jpeg', '.png', '.gif', '.bmp'}
         images = [file for file in all_files if os.path.splitext(file)[1].lower() in image_extensions]

         # Calculate the number of images to keep
         num_images_to_keep = len(images) // 2

         # Select the first one-third of the images
         images_to_keep = images[:num_images_to_keep]
         images_to_delete = set(images) - set(images_to_keep)

         # Delete unselected images
         for image in images_to_delete:
             os.remove(os.path.join(directory, image))
```

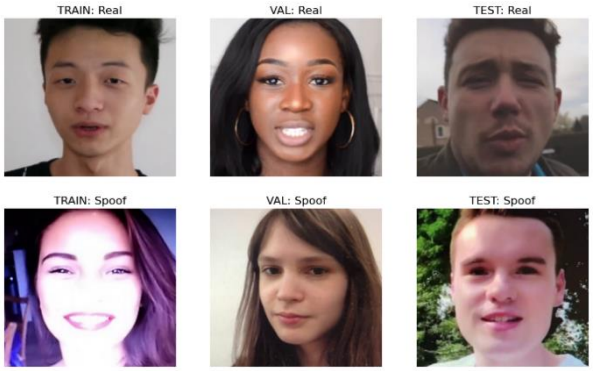
درگام بعد تعدادی از نمونه های متفاوت نمایش داده میشوند:

```
# Visualizing some of the data set
num_classes = len(label_name)
num_dataset = 0
for key, val in set_length.items():
  num_dataset += 1 if val > 0 else 0

f, ax = plt.subplots(num_classes, num_dataset, figsize=(num_dataset*5, 9))

for k in range(num_classes*num_dataset):
    j, i = k//num_dataset, k%num_dataset  # Image indexing

    img = imread(img_disp_df.iloc[j, i])
    ax[j, i].imshow(img, cmap='gray')
    ax[j, i].set_title(f"{img_disp_df.columns[i].upper()}: {img_disp_df.index[j].capitalize()}", fontsize=16)
    ax[j, i].axis('off')
    ax[j, i].set_aspect('auto')
plt.show()
```



برای آموزش بهتر مدل، داده افزایی از قبیل (چرخش تصویر، قرینه کردن تصویر، زوم تصویر و ...) روی دیتای آموزشی انجام می‌شود. برای این کار از تابع ImageDataGenerator استفاده شده است.

```
[9]  # Instantiate data generator for training procedure
     train_datagen = ImageDataGenerator(rescale = 1./255,
                                        rotation_range = 20,
                                        width_shift_range = 0.2,
                                        height_shift_range = 0.2,
                                        shear_range = 0.15,
                                        zoom_range = 0.15,
                                        horizontal_flip = True,
                                        fill_mode="nearest")

     val_datagen = ImageDataGenerator(rescale = 1./255)
     test_datagen = ImageDataGenerator(rescale=1./255) if set_length["test"] > 0 else None
```

پس از اعمال پیش پردازش های اشاره شده به ساختار زیر خواهیم رسید:

```
[12] # Displaying the dataset generator information
     print(f'Train set batch shape\t: {next(train_gen)[0].shape}')
     print(f'Val set batch shape\t: {next(val_gen)[0].shape}')
     print(f'Test set batch shape\t: {next(test_gen)[0].shape}') if test_gen is not None else None

     # Print the mapping of class labels to numerical values
     print(train_gen.class_indices)
     print(val_gen.class_indices)
     print(test_gen.class_indices)

     Train set batch shape   : (32, 224, 224, 3)
     Val set batch shape     : (32, 224, 224, 3)
     Test set batch shape    : (1, 224, 224, 3)
     {'real': 0, 'spoof': 1}
     {'real': 0, 'spoof': 1}
     {'real': 0, 'spoof': 1}
```

برای backbone شبکه ابتدا مدل آموزش دیده mobilenetv2 فراخوانی شده و وزن های آن ذخیره میشود، سپس یک لایه کانولوشنی و در ادامه دولایه Dense قرار میگیرد. همچنین برای جلوگیری از بیش برازش روی دیتای آموزشی از لایه های dropout و GAP نیز استفاده شده است. ساختار کلی شبکه به صورت زیر خواهد بود:

```
[14] # Adding extra layer for our problem
     x = pretrain_net.output
     x = Conv2D(32, (3, 3), activation='relu')(x)
     x = Dropout(rate=0.2, name='extra_dropout1')(x)
     x = GlobalAveragePooling2D()(x)
     x = Dense(units=128, activation='relu', name='extra_fc1')(x)
     x = Dropout(rate=0.2, name='extra_dropout2')(x)
     x = Dense(1, activation='sigmoid', name='classifier')(x)

     model = Model(inputs=pretrain_net.input, outputs=x, name='mobilenetv2_spoof')
     print(model.summary())
```

پس از آزمون و خطا بین چند مدل pretrained مانند resnet50، efficientnet و mobilenetv2 این مدل نتیجه بهتری داشت.

در ادامه، تابع بهینه سازی، تابع ضرر و معیار اندازه گیری مدل تعریف میگردد. همچنین با کمک کتابخانه callbacks، آدرس ذخیره سازی checkpoint ها حین آموزش تعریف می‌گردد.

```
model.compile(optimizer = AdamW(lr=learning_rate),
              loss = 'binary_crossentropy',
              metrics = ['acc'])

# Define model callback
save_dir = os.path.join("./", train_id)
if not os.path.isdir(save_dir):
  os.makedirs(save_dir)

cont_filepath = "mobilenetv2-epoch_{epoch:02d}.hdf5"
cont_checkpoint = ModelCheckpoint(os.path.join(save_dir, cont_filepath))

best_filepath = "mobilenetv2-best.hdf5"
best_checkpoint = ModelCheckpoint(os.path.join(save_dir, best_filepath),
                                  save_best_only=True,
                                  save_weights_only=True)
```

برای پویایی بیشتر نرخ یادگیری، از scheduler نیز استفاده شده تا در epoch های جلوتر نرخ یادگیری کاهش یابد. با توجه به تفاوت تعداد نمونه های spoof وreal وزن هر کلاس نیز محاسبه می‌شود.

```
# Instantiate learning rate scheduler with Plateau method
plateau_scheduler = ReduceLROnPlateau(factor=0.2, patience=2, verbose=1,
                                      min_delta= 0.005, min_lr=5e-7)


# Define class weight
train_length = len(train_gen.classes)
weight0 = train_length / case_count_df['train'][label_name[0]] * (1 / len(label_name))
weight1 = train_length / case_count_df['train'][label_name[1]] * (1 / len(label_name))
class_weight = {0: weight0, 1: weight1}

print(f"Class weight\t: {class_weight}")
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use t
Training config of 'lcc-train04b-weight_all'...
Number of epoch : 15
Initial LR      : 5e-05
Class weight    : {0: 1.9293394777265744, 1: 0.6749059645351961}
```

با فراخوانی تابع model.fit() آموزش شبکه شروع شده و نتایج هر epoch در checkpoint متناظر با آن ذخیره سازی میگردد. فایل mobilenetv2-best.hdf5 نیز حاوی پارامتر های شبکه میباشد که منجر به بهترین نتیجه شده اند. در تصویر زیر نتایج epoch های نهایی بر روی مجموعه داده train و validation مشخص گردیده است:

```
# Perform training
history = model.fit(train_gen,
                    epochs = num_epochs,
                    steps_per_epoch = set_length['train'] // train_batch_size,
                    validation_data = val_gen,
                    validation_steps = 1,
                    callbacks = [best_checkpoint,
                                 cont_checkpoint,
                                 plateau_scheduler],
                    class_weight=class_weight)

history_df = pd.DataFrame.from_dict(history.history)
history_df.to_csv(os.path.join(save_dir, "history.csv"), index=False)
```

```
Epoch 12: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
157/157 [==============================] - 127s 810ms/step - loss: 0.0292 - acc: 0.9900 - val_loss: 1.3235 - val_acc: 0.8438 - lr: 8.0000e-06
Epoch 13/15
157/157 [==============================] - 127s 811ms/step - loss: 0.0246 - acc: 0.9918 - val_loss: 0.5024 - val_acc: 0.8750 - lr: 1.6000e-06
Epoch 14/15
157/157 [==============================] - ETA: 0s - loss: 0.0210 - acc: 0.9934
Epoch 14: ReduceLROnPlateau reducing learning rate to 5e-07.
157/157 [==============================] - 127s 803ms/step - loss: 0.0210 - acc: 0.9934 - val_loss: 0.5609 - val_acc: 0.8750 - lr: 1.6000e-06
Epoch 15/15
157/157 [==============================] - 127s 806ms/step - loss: 0.0212 - acc: 0.9922 - val_loss: 0.3966 - val_acc: 0.9375 - lr: 5.0000e-07
```

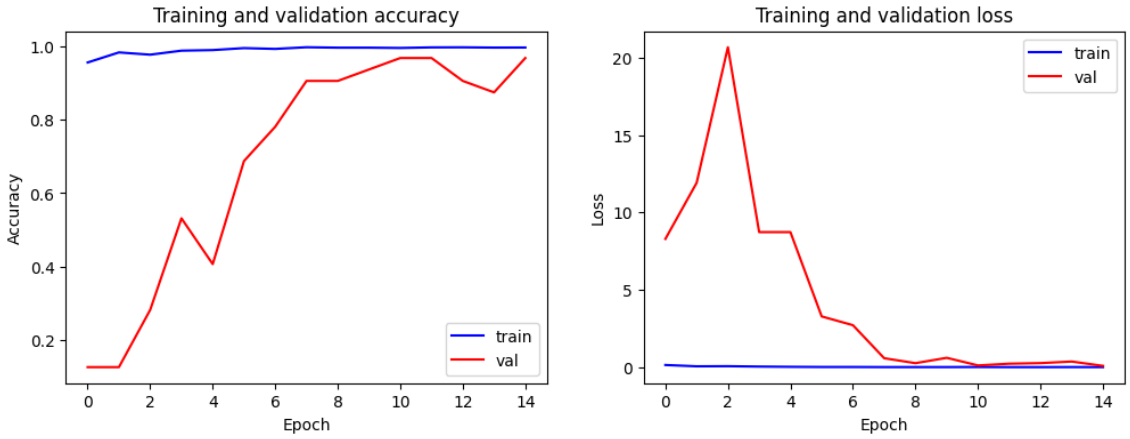
جهت استفاده راحت تر از مدل آموزش دیده، پارامتر ها در درایو گوگل نیز ذخیره می‌شوند. (اجرای این بخش اختیاری بوده و برای استفاده مجدد نوشته شده است)

```
drive.mount('/content/drive') # Mount Google Drive

source_dir = '/content/lcc-train04b-weight_all_balance2'
target_dir = '/content/drive/MyDrive/lcc-fasd-train-weight_all_balance2/'

# Copy the directory contents to Google Drive
shutil.copytree(source_dir, target_dir, dirs_exist_ok=True)
```

در ادامه نمودار های دقت و مقدار ضرر در طول epoch های متفاوت بررسی می‌شوند:



دقت آموزشی از ابتدا بالا شروع می‌شود و در طول فرآیند آموزش تقریباً ثابت و نزدیک به ۱.۰ (۱۰۰٪) باقی می‌ماند. این نشان می‌دهد که مدل داده‌های آموزشی را بسیار خوب طبقه‌بندی می‌کند و این عملکرد را حفظ می‌کند. دقت اعتبارسنجی از مقدار پایین شروع می‌شود و به طور قابل توجهی افزایش می‌یابد، که نشان دهنده بهبود عملکرد مدل بر روی داده‌های نادیده است. این دقت به اوج می‌رسد و نوساناتی را نشان می‌دهد اما به طور کلی یک روند یادگیری خوب را نشان می‌دهد، که نشان می‌دهد مدل به خوبی به داده‌های اعتبارسنجی عمومی‌سازی می‌کند.

برای ارزیابی مدل روی داده های تست به صورت زیر عمل می‌کنیم:

```
[ ]  # Test set accuracy and loss
     test_scores = model.evaluate(test_gen, steps=set_length['test'])
     print("Test results Accuracy: {:.2f}% and Loss: {:.2f}".format(test_scores[1]*100, test_scores[0]))

     # Calculate prediction
     threshold = 0.5  # Define the sigmoid threshold for True or False
     y_pred_value = np.squeeze(model.predict(test_gen, steps=set_length['test'], verbose=1))

     y_pred = np.zeros(y_pred_value.shape).astype(np.int32)  # Sigmoid
     y_pred[y_pred_value > threshold] = 1

     # y_pred = np.argmax(y_pred_value, axis=-1).astype(np.int32)  # Softmax

     y_true = test_gen.classes

     # Sanity check on the y_pred and y_true value
     print(f"Label\t\t: {y_true[:30]}")
     print(f"Prediction\t: {y_pred[:30]}")

     3979/3979 [==============================] - 198s 49ms/step - loss: 0.1786 - acc: 0.9739
     Test results Accuracy: 97.39% and Loss: 0.18
```

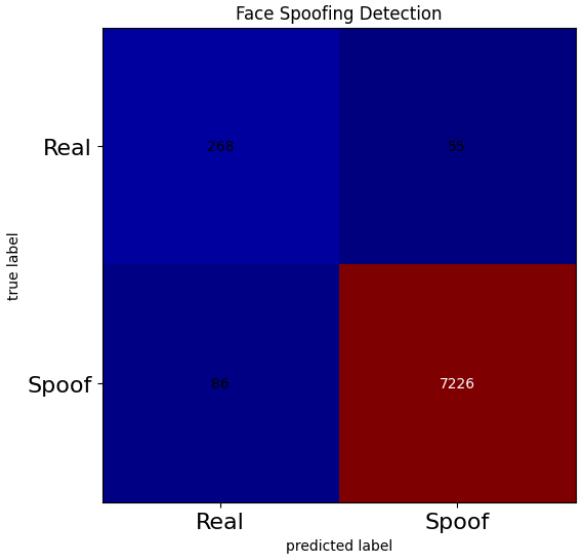
مطابق تصویر بالا دقت مدل روی داده های تست به 97 درصد میرسد. پیش بینی مدل روی داده های تست را انجام داده و با مقادیر واقعی مقایسه می نماییم. ماتریکس confusion زیر بیانگر معیار های متفاوت شبکه میباشد:

```python
# Confusion matrix result
confusion_matrix_result = confusion_matrix(y_true, y_pred)
plot_confusion_matrix(confusion_matrix_result,
                      figsize=(10,6),
                      hide_ticks=True,
                      cmap=plt.cm.jet)
plt.title("Face Spoofing Detection")
plt.xticks(range(2), ['Real', 'Spoof'], fontsize=16)
plt.yticks(range(2), ['Real', 'Spoof'], fontsize=16)
plt.show()

# Precision and Recall metrics
tn, fp, fn, tp = confusion_matrix_result.ravel()
precision = tp / (tp+fp)
recall = tp / (tp+fn)
f1_score = 2 * precision * recall / (precision+recall)

print("Report Summary:")
print("Precision\t: {:.2f}%".format(precision*100))
print("Recall\t\t: {:.2f}%".format(recall*100))
print("F1 Score\t: {:.2f}%".format(f1_score*100))

print("\nNotes: ")
print("True labels\t: Spoof")
print("False labels\t: Real")
```



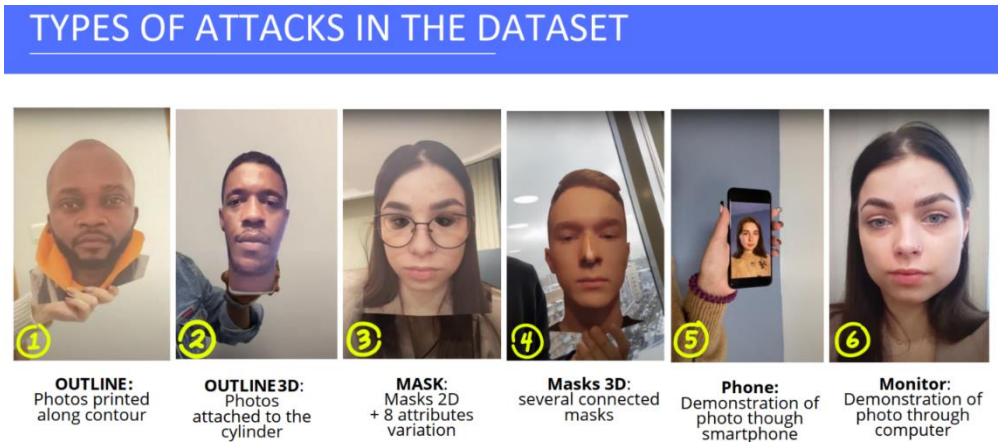
با توجه به ماتریس بالا مقادیر precision, recall و F1 Score محاسبه می‌شوند:

```
Report Summary:
Precision       : 91.81%
Recall          : 91.66%
F1 Score        : 91.73%
```

برای ارزیابی مدل توسط ویدئو از دیتاست دیگری با عنوان iBeta 1 - 42,280 Liveness Detection استفاده شده است:

https://www.kaggle.com/datasets/trainingdatapro/ibeta-level-1-liveness-detection-dataset-part-1/data

در این دیتاست چندین ویدئو کوتاه از افراد و یا از تصاویر غیر زنده افراد وجود دارد. به طور کلی ۶ نوع حمله در این دیتاست مشاهده می‌شود:



ابتدا دیتاست را دانلود کرده و سپس از هر ویدئو یک فریم استخراج می‌گردد.

پروژه پایانی – Anti-Spoofing

```
[ ] !kaggle datasets download -d trainingdatapro/ibeta-level-1-liveness-detection-dataset-part-1

Dataset URL: https://www.kaggle.com/datasets/trainingdatapro/ibeta-level-1-liveness-detection-dataset-part-1
License(s): Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0)
Downloading ibeta-level-1-liveness-detection-dataset-part-1.zip to /content
 99% 2.78G/2.80G [00:36<00:00, 58.6MB/s]
100% 2.80G/2.80G [00:36<00:00, 82.3MB/s]

[ ] !unzip -q ibeta-level-1-liveness-detection-dataset-part-1.zip -d /content/liveness-detection-dataset
```

سپس سه روش برای تشخیص زنده بودن فرد در فریم بررسی میشود:

- فریم بدون هیچ پیش پردازشی به عنوان ورودی مدل داده میشود
- ابتدا تصویر صورت شخص جدا شده و برای مدل فرستاده شود
- تبدیل فوریه رو تصویر اعمال میشود و به عنوان ورودی مدل ارسال میشود

برای تشخیص صورت افراد از تابع آماده cascade calssifier استفاده میشود:

```python
# Load face detection model (e.g., Haar Cascade)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

پس از تشخیص تصویر صورت، ابعاد تصویر مطابق با ورودی مدل ۲۲۴*۲۲۴ تنظیم میگردد و نتیجه پیش بینی از مدل دریافت میشود:

```python
def detect_faces(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    return faces

def preprocess_face(face):
    face = cv2.resize(face, (img_width, img_height))
    face = img_to_array(face)
    face = preprocess_input(face)
    face = np.expand_dims(face, axis=0)
    return face

def predict_spoof(face_image):
    processed_face = preprocess_face(face_image)
    pred = anti_spoof_model.predict(processed_face)
    return pred[0][0]
```

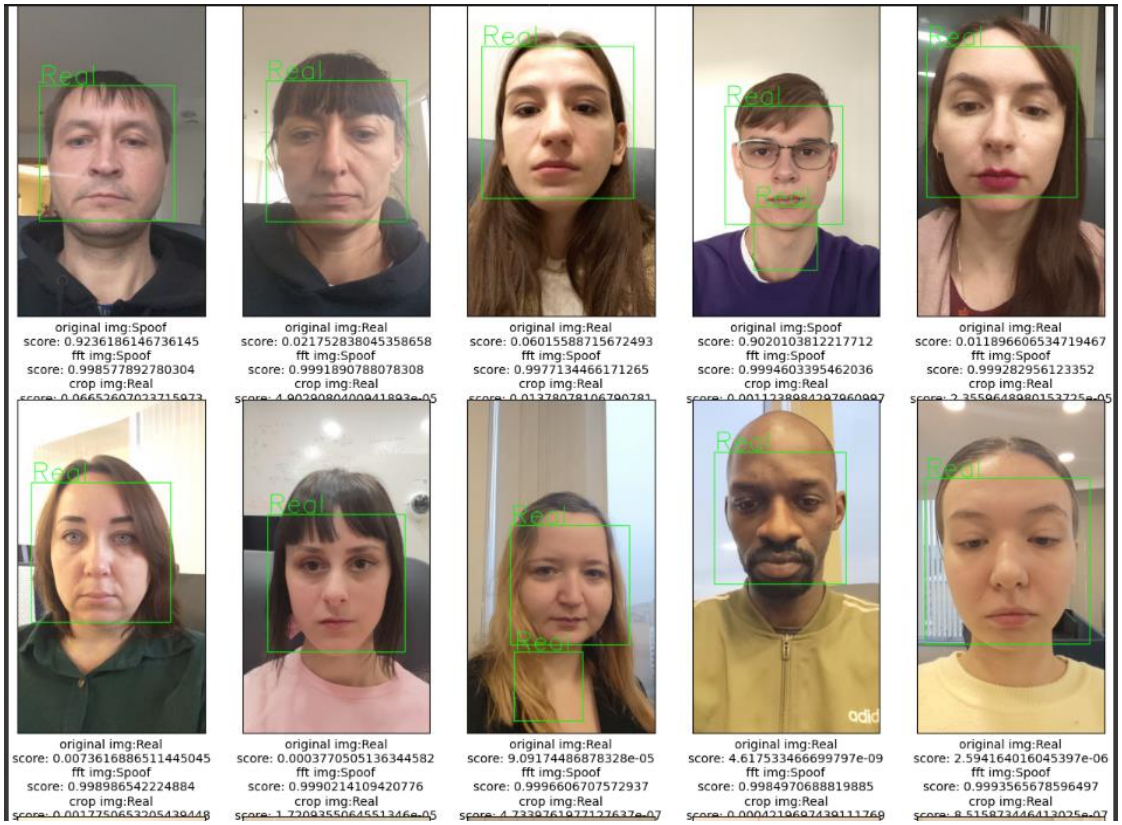
همچنین برای بخش تبدیل فوریه و دریافت دامنه فرکانس های متفاوت تصویر از تابع زیر استفاده میگردد:

```python
# Function to compute the Fourier Transform of an image
def compute_fourier_transform(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    f_transform = np.fft.fft2(gray_image)
    f_shift = np.fft.fftshift(f_transform)
    magnitude_spectrum = 20 * np.log(np.abs(f_shift) + 1)  # Adding 1 to avoid log(0)
    # Convert single channel to 3 channels
    magnitude_spectrum = cv2.merge([magnitude_spectrum, magnitude_spectrum, magnitude_spectrum])
    return magnitude_spectrum
```

در نهایت به کمک کد زیر یک فریم از ویدئو های دایرکتوری مورد نظر استخراج شده و به سه روش متفاوت تصویر به مدل داده میشود و هر سه نتیجه در لیست prediction_results ذخیره میشوند. این لیست در آخر به شکل فایل predictions_deep.csv ذخیره میگردد.

پروژه پایانی – Anti-Spoofing

نمونه خروجی به دست آمده بر روی داده های real:



نمونه خروجی به دست آمده بر روی داده های spoof:

یک نمونه از فایل csv خروجی:

video_file	Ground Truth	liveness_score	liveness_score_crop	liveness_score_frequency
user026.mp4	real	0.076381385	0.93347393	0.001422107
user027.mp4	real	0.978247162	0.999950971	0.000810921
user028.mp4	real	0.939844113	0.986219219	0.002286553
user029.mp4	real	0.097989619	0.998876102	0.00053966
user030.mp4	real	0.988103393	0.99997644	0.000717044
user050.mp4	real	0.333397985	0.14242053	0.001081109
mask.mp4	spoof	0.999065534	0.336023152	0.003079474
mask3d.mp4	spoof	5.19E-05	0.336023152	0.001186907
outline.mp4	spoof	3.40E-05	6.50E-06	0.000784934
outline3d.mp4	spoof	0.300865412	0.997709857	0.000470042

با توجه به اطلاعات بالا و نتایج به دست آمده، اکثر اوقات برش زدن صورت فرد میتواند موثر باشد و حتی برخلاف کل تصویر بتواند عملکرد درستی از زنده یا غیر زنده بودن تصویر داشته باشد. اما گاهی اوقات نیز برش زدن تصویر باعث گمراهی مدل و ایجاد اشتباه در نتیجه شده است.

همچنین با توجه به اینکه مدل هیچ آموزشی روی داده های تبدیل فوریه نداشته است، معمولا این تبدیل را به عنوان spoof درنظر میگیرد.