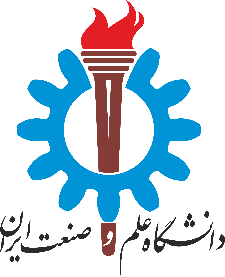
بسمه تعالی



**گزارش پروژۀ بینایی کامپیوتر**

**عنوان:**

طراحی مدل کلاسیک و عمیق برای الگوریتم anti-spoofing

**استاد درس:**

دکتر محمدی

**اعضای گروه:**

هلیا شمس زاده 400521486

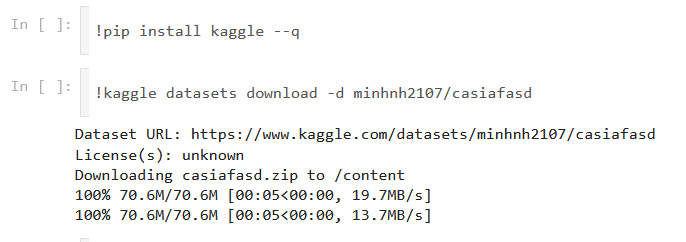
بهاره کاووسی نژاد 99431217

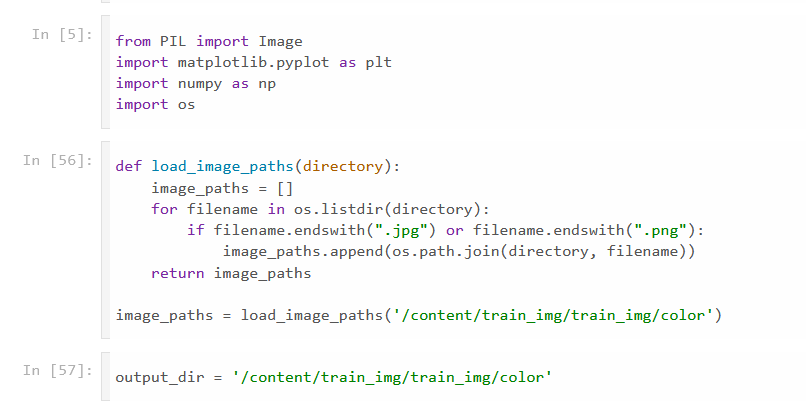
**تهیۀ دیتاست:**

دیتاست این پروژه با ترکیب دو دیتاست [CASIA-FASD](https://paperswithcode.com/dataset/casia-fasd) و برخی تصاویر [anti-spoofing-face-fake](https://universe.roboflow.com/huu-thinh-muem8/anti-spoofing-face-fake/dataset/1/download) به دست آمده است.

**پیش‌پردازش داده‌ها:**

ابتدا داده‌های موجود در دیتاست را در وبسایت [Hugging Face](https://huggingface.co/) آپلود کرده، و با استفاده از کد زیر داخل گوگل کولب load می‌کنیم.



با قطعه کد زیر، مسیرهای مربوط به تصاویر را در لیست image\_paths قرار می‌دهیم:

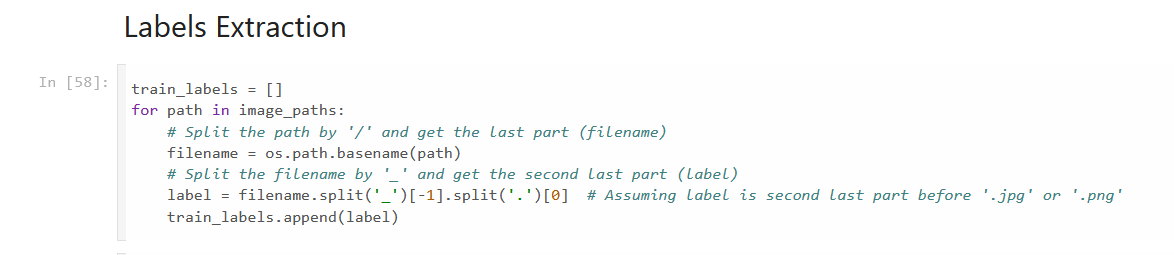
چند مورد از تصاویر موجود در دیتاست در ادامه نمایش داده شده است:

****

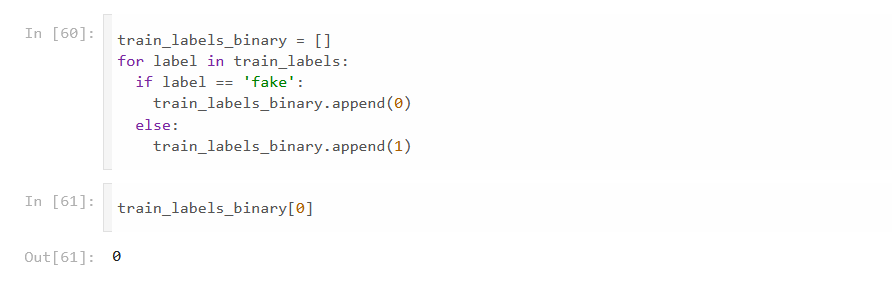
با کد زیر، label های تصاویر را براساس اسم فایل‌ها پیدا می‌کنیم، زیر فرمت نام فایل‌ها به صورت زیر است:

/.. /\*\*\*\_fake.jpg

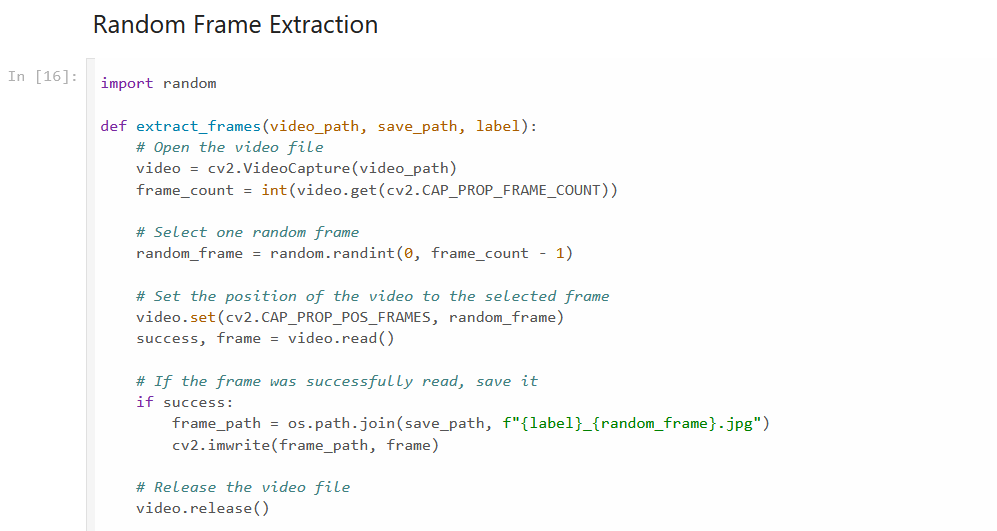
/\_\*\*\*/..real.jpg

****

چون در classification دو کلاس داریم، label تصاویر fake را به 0 و label تصاویر real را به 1 تغییر می‌دهیم تا بعدا به صورت categorical در آوریم:

****

برای مرحلۀ تست، نیاز داریم تا از ویدیوهایمان فریم استخراج کنیم، به این منظور تابع زیر را نوشته که ورودی آن دیرکتوری ویدیوها، دیرکتوری مقصد برای ذخیرۀ فریم‌ها و label مربوطه است. از هر ویدیو یک فریم تصادفی استخراج کرده و در آدرس‌های نوشته شده در آخر کد (با نام فایل ایجاد شده) ذخیره می‌کند. چون ویدیوهای real و fake در دیرکتوری‌های جدایی نگهداری شده‌اند براساس نام دیرکتوری مربوطه ابتدای نام فریم‌ها صفر (یعنی fake) یا 1 (یعنی real) قرار می‌دهیم:

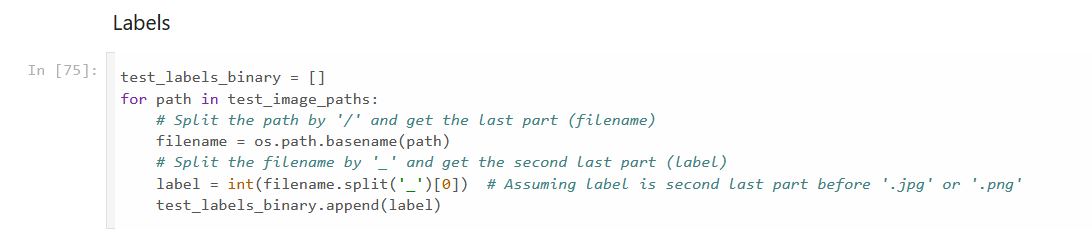
****

**منابع**



نتیجۀ کد بالا، ذخیرۀ تمامی فریم‌ها در دیرکتوری save\_frames\_path می‌باشد. با کمک تابع load\_images\_path که بالاتر نوشته شده است، مسیر تمام فریم‌ها را در متغیر paths قرار می‌دهیم:

حال لیبل فریم‌های استخراج شده را با کد زیر پیدا کرده و در لیست test\_labels\_binary ذخیره می‌کنیم:

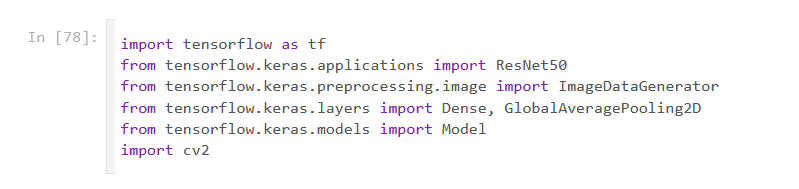


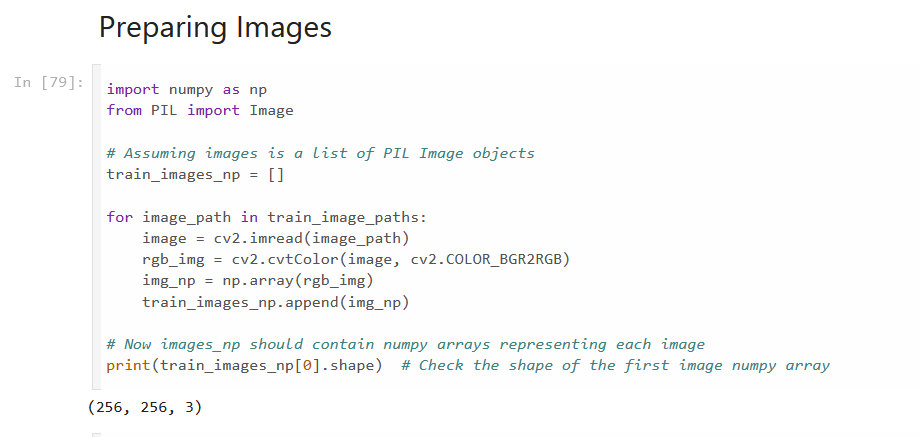
**طراحی مدل برای یادگیری ویژگی (deep learning)**

برای این بخش از مدل‌های pretrained استفاده شده است. اولین مدل مورد استفاده، مدل ResNet50 با وزن‌های اولیۀ imagenet است. مدل دوم که مورد بررسی قرار گرفت، مدل google/vit-base-patch16-224 است.

**آموزش مدل ResNet50:**

ابتدا کتابخانه‌های لازم را ایمپورت می‌کنیم:



در بخش قبلی، تمام path های داده‌های آموزش را در train\_image\_paths ذخیره کردیم. حال باید هر کدام را با دستور cv2.imread خوانده و به np.array تبدیل کرده و در لیست train\_images\_np قرار دهیم:

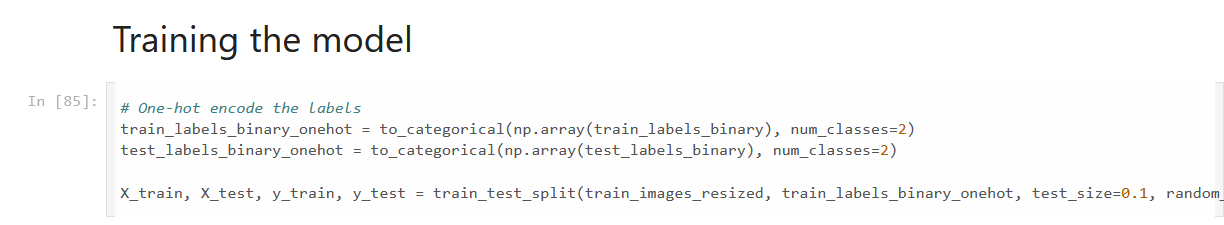
برای داده‌های تست هم روند بالا را انجام داده و هر فریم را از مسیر مربوطه خوانده و به آرایه تبدیل می‌کنیم:



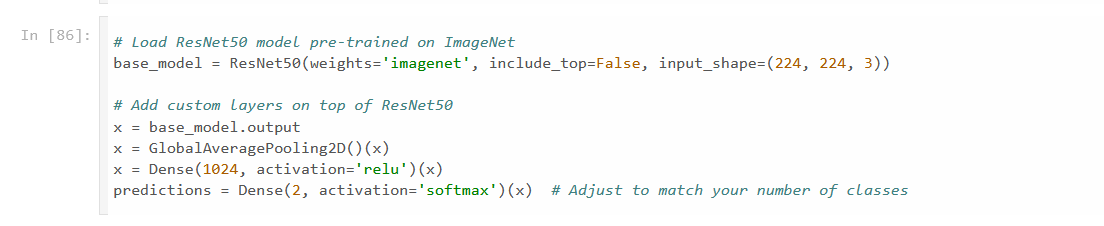
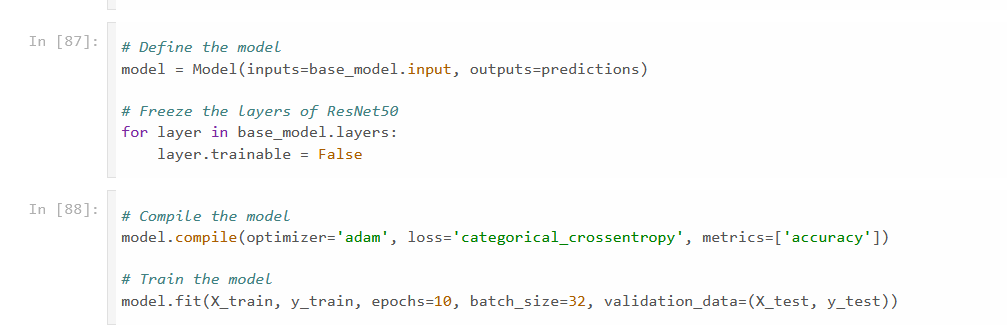
همانطور که می‌دانیم ورودی مدل ResNet یک تصویر سه کانالۀ 224 x 224 است، برای همین لازم است که داده‌های آموزش و تست resize شوند. با استفاده از دستور cv2.resize تمام آرایۀ تصاویر آموزش و تست را resize می‌کنیم:



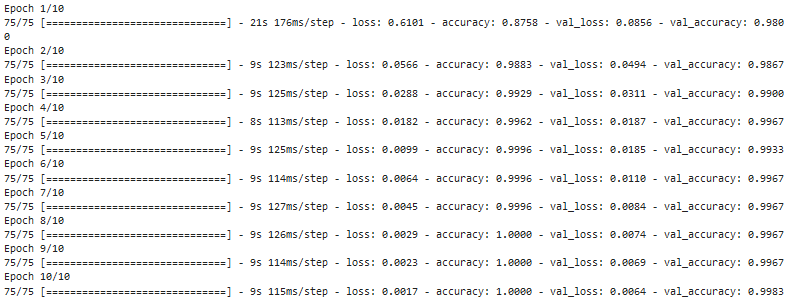
برای آموزش مدل، ابتدا لیبل‌های داده‌های آموزش و تست (فریم ویدیوها) را به حالت one-hot (categorical) در می‌آوریم. همچنین، داده‌های آموزش را به دو بخش آموزش و تست اولیه تقسیم می‌کنیم:



مدل ResNet را Load کرده و به جای لایۀ FC آن، یک لایۀ GAP، یک لایۀ dense با 1024 نورون و نهایتا یک لایۀ FC با دو نورون (چون دو کلاس داریم) و تابع فعالسازی sofmax اضافه می‌کنیم. لایه‌های base که مربوط به ResNet هستند را فریز کرده تا فقط لایه‌های جدید با وزن‌های تصادفی آموزش ببینند:



**نتایج آموزش:**



**نتایج تست‌های اولیه (بخشی از داده‌های دیتاست استفاده شده):**



**آموزش مدل ResNet50:**

**1) نتایج ارزیابی روی داده‌های خام تست (فریم ویدیو‌ها)**

به منظور استفاده از فریم های تصادفی انتخاب شده برای تست مدل، از تابع resize\_image استفاده می کنیم تا تصاویر را به numpy array تبدیل کرده و آنها را به همراه برچسب هایشان در دو لیست ذخیره کنیم.



دقت مدل را بر روی داده تست بدست می آوریم:



با استفاده از کد زیر می‌توان فریم‌هایی که prediction اشتباه داشته اند را مشاهده کرد:

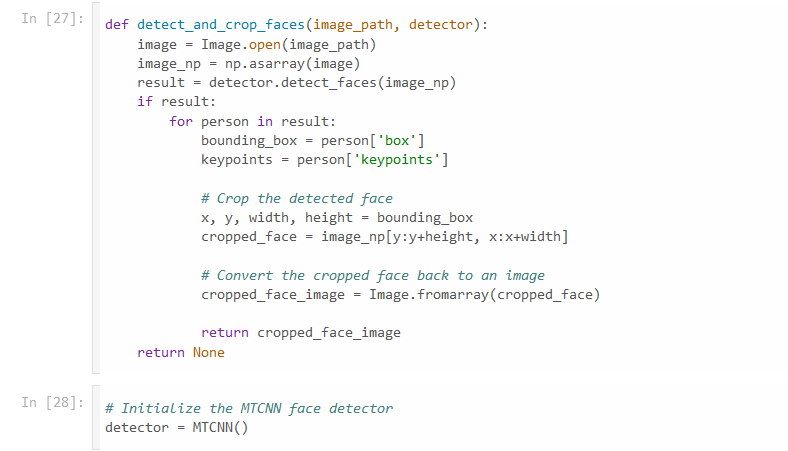


در این قسمت پیش بینی های مدل برای real بودن تصاویر نرمالایز شده و در predicted\_scores\_normalized ذخیره می شوند که در ادامه این مقادیر را در فایل .csv ذخیره می کنیم.

****

**2) نتایج ارزیابی روی داده‌های تست کراپ شده (فریم ویدیو‌ها)**

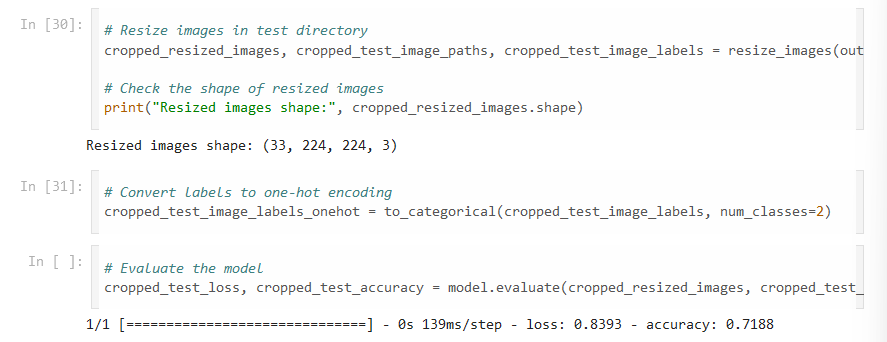
به منظور برش صورت در تصاویر از MTCNN استفاده شده است. بدین منطور تابع detect\_and\_crop\_faces قرار داده شده است که با تعیین bounding\_box دور صورت برش زده می شود.

****

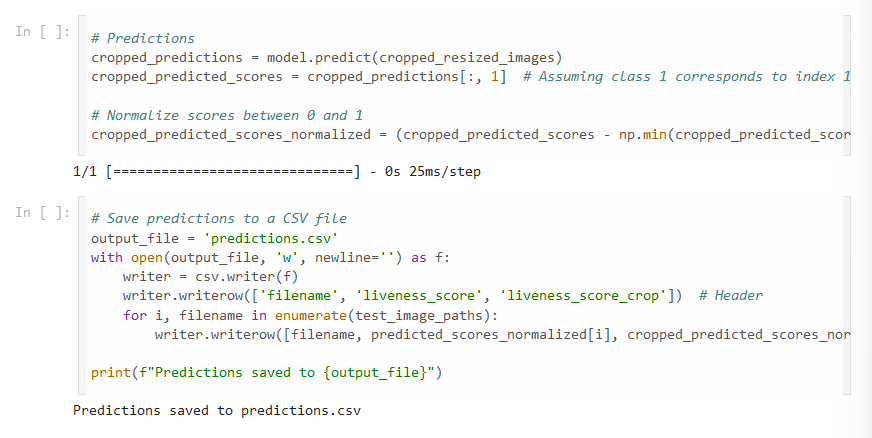
تصویر روبرو یک نمونه از عملکرد برش صورت را نشان می دهد:

****

مدل را بر روی تصاویر برش خورده نیز تست می کنیم. همان طور که در تصویر پیداست دقت در حدود 72 درصد می باشد.

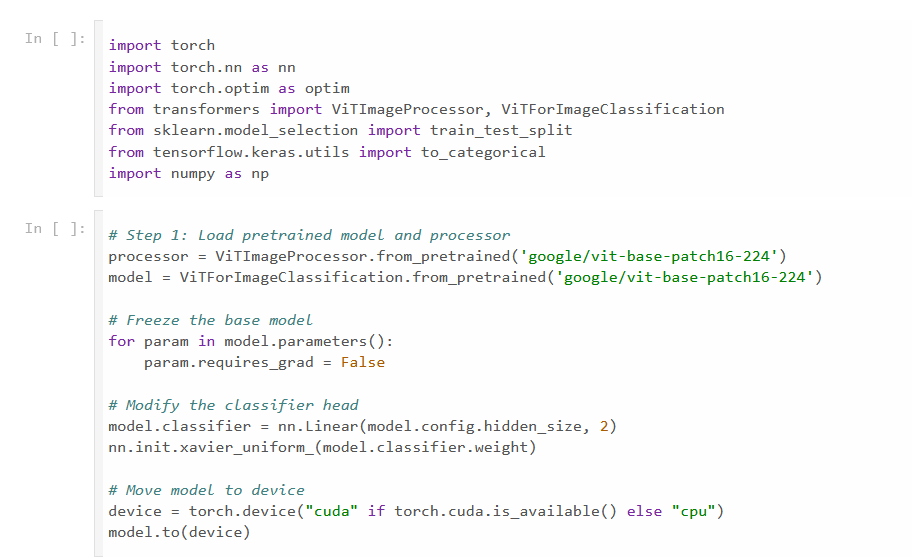
****

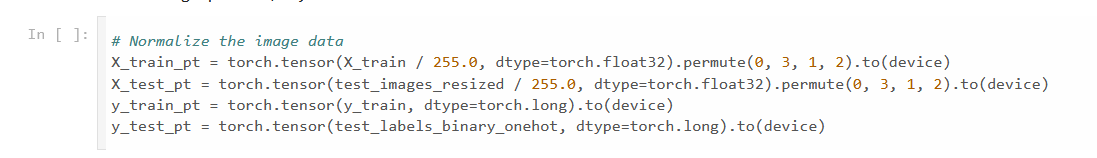
مانند قسمت بالا پیش بینی های مدل را برای real بودن تصویر به دست آورده و در predictions.csv ذخیره می کنیم. توجه کنید که این فایل در پوشه پروژه به صورت predictions\_ResNet.csv ذخیره شده است تا از تداخل نام آن با مدل گوگل جلوگیری به عمل آید.

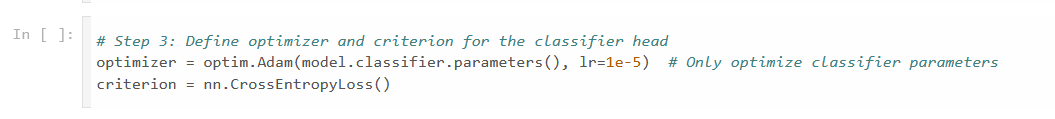
****

**آموزش و ارزیابی مدل google/vit-base-patch16-224:**

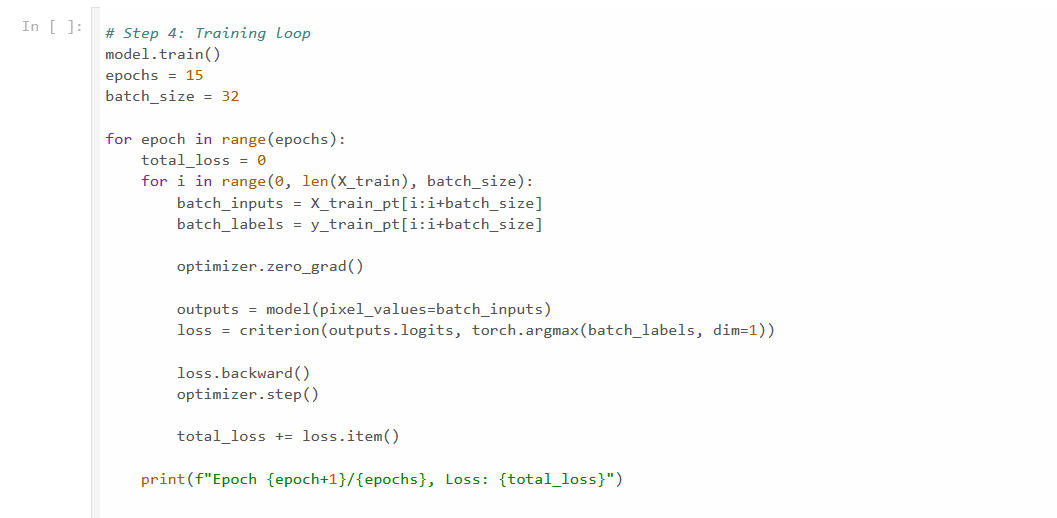
این مدل، یک مدل pretrained با وزن‌های اولیۀ خود است.

ابتدا مدل را load می‌کنیم (در این بخش از کتابخانۀ torch استفاده شده است):

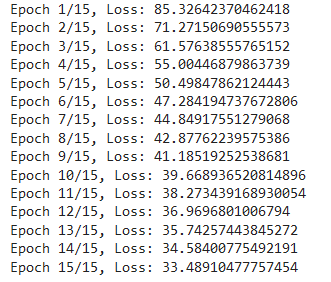
سپس داده‌های ورودی (چه آموزش چه تست) را نرمالیزه می‌کنیم. (این داده‌ها در cell های قبلی تعریف شدند) یعنی آرایۀ تصاویر را به 255 تقسیم کرده تا هر پیکسل مقداری بین صفر و یک داشته باشد:

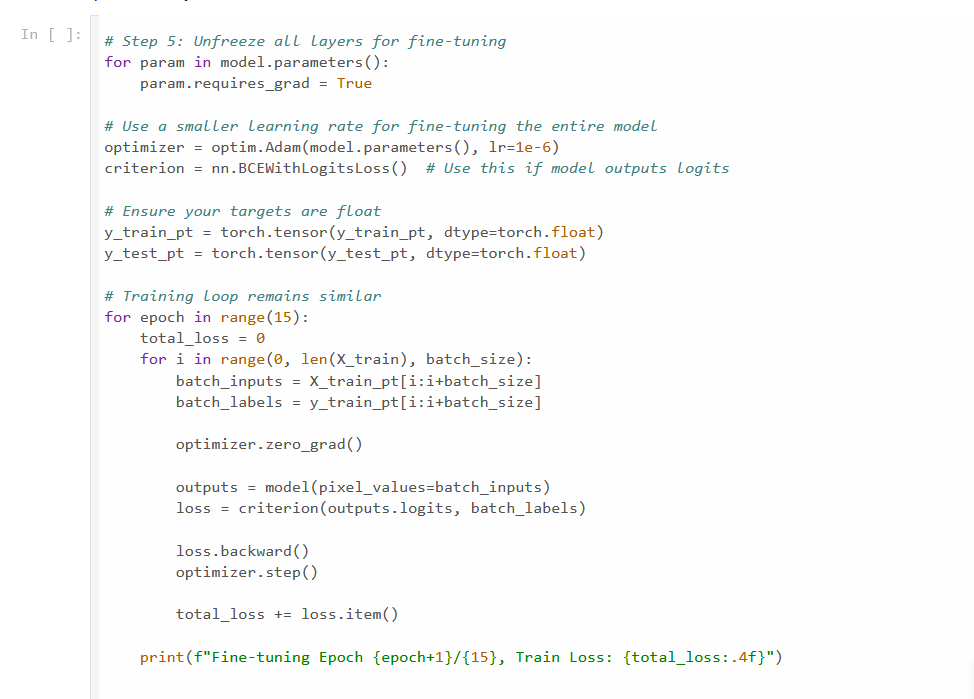
تابع بهینه‌ساز را Adam انتخاب کرده و از تابع ضرر Cross Entropy استفاده می‌کنیم:

با قطعه کد زیر، مدل را با داده‌های تست آموزش می‌دهیم. در این مرحله شبکۀ پایه فریز است:

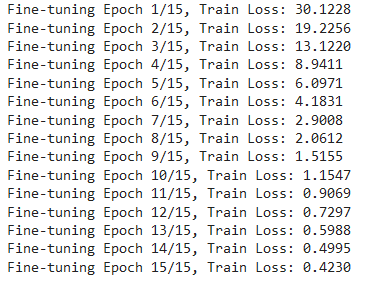


**نتیجۀ آموزش:**

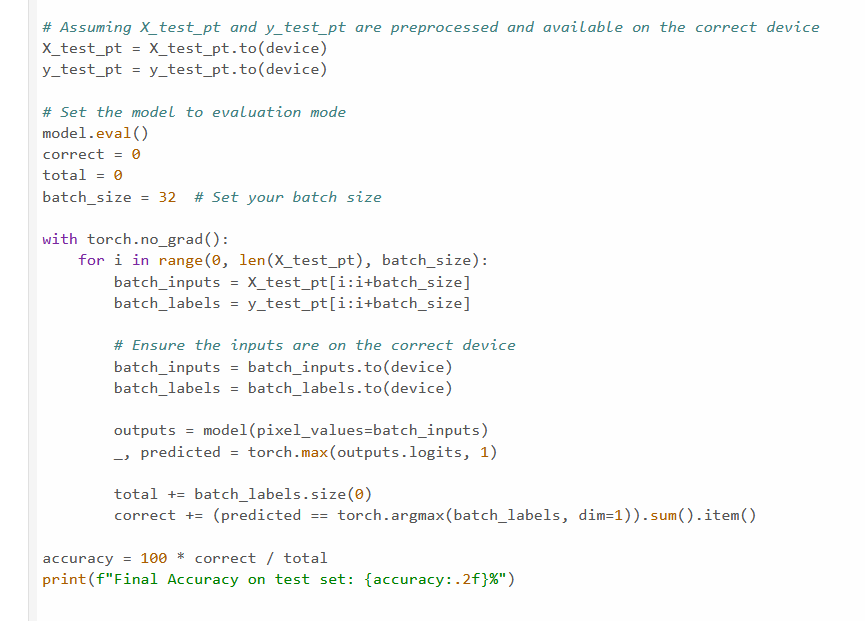


حال برای کم کردن loss و رسیدن به دقت بهتر، fine tuning انجام می‌دهیم تا وزن‌های همۀ لایه‌ها بهینه شود:

**نتایج fine tuning:**



حال با قطعه کد زیر، مدل با وزن‌های به دست آمده را روی داده‌های تست (فریم‌ها) ارزیابی می‌کنیم:



**نتایج روی داده‌های تست:**



**نتایج روی داده‌های تست (فریم های تصادفی):**

****

**نتایج روی داده‌های تست (فریم های تصادفی برش خورده):**

****

مانند قسمت قبل پیش بینی های مدل را برای real بودن تصویر به دست آورده و در predictions.csv ذخیره می کنیم. توجه کنید که این فایل در پوشه پروژه به صورت predictions\_Google.csv ذخیره شده است تا از تداخل نام آن با مدل ResNet جلوگیری به عمل آید.

**طراحی مدل برای استخراج ویژگی (classic learning)**

**مدل CNN**

برای این قسمت از یک مدل CNN که فقط شامل چند لایۀ کانوولوشنی و FC است استفاده شده است.

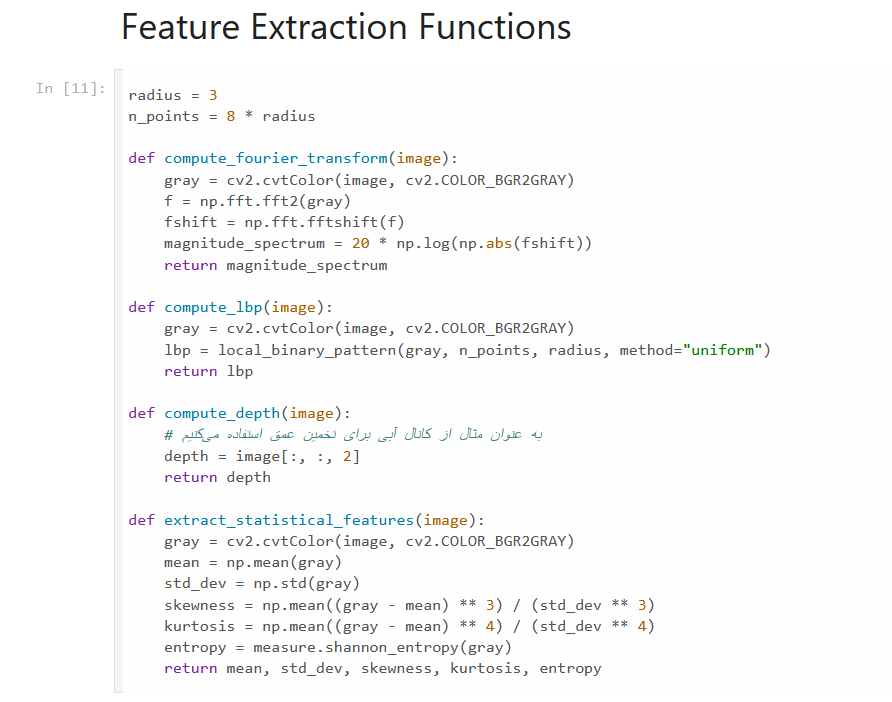
ابتدا دیتاست را از Hugging Face لود می‌کنیم. این دیتاست به صورت دیکشنری‌ای از دیکشنری‌ها می‌باشد:



سپس کتابخانه‌های مورد نیاز را import می‌کنیم:



برای بخش استخراج ویژگی، ویژگی‌هایی همانند فرکانس، lbp، عمق و یک سری ویژگی‌های آماری استخراج می‌شود. توابع مربوط به هر یک در ادامه آورده شده است:



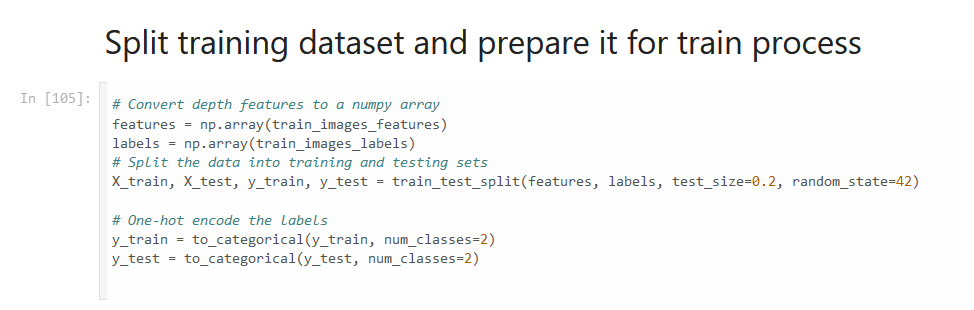
حال به تابعی نیاز داریم که از دیکشنری Dataset عکس را پیدا کرده و ویژگی‌های آن را با توابع نوشته شده در بالا استخراج کند. هر ویژگی ابتدا به 64 در 64 resize شده و سپس با هم concat شده و به عمق 3 می‌رسد (3 ویژگی):

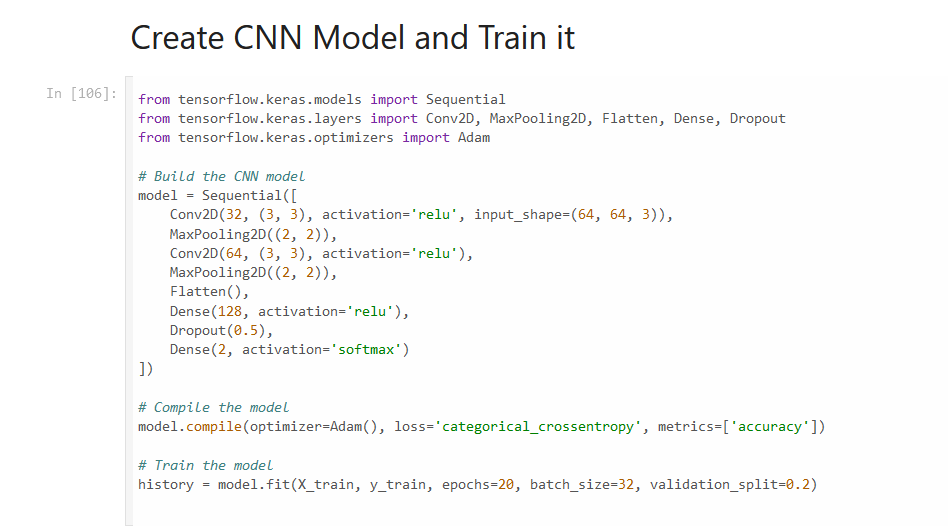


چون تعداد داده‌های آموزشی زیاد است (حدود 6000 تا) فقط 3000 تای آن را برای استخراج ویژگی استفاده می‌کنیم:

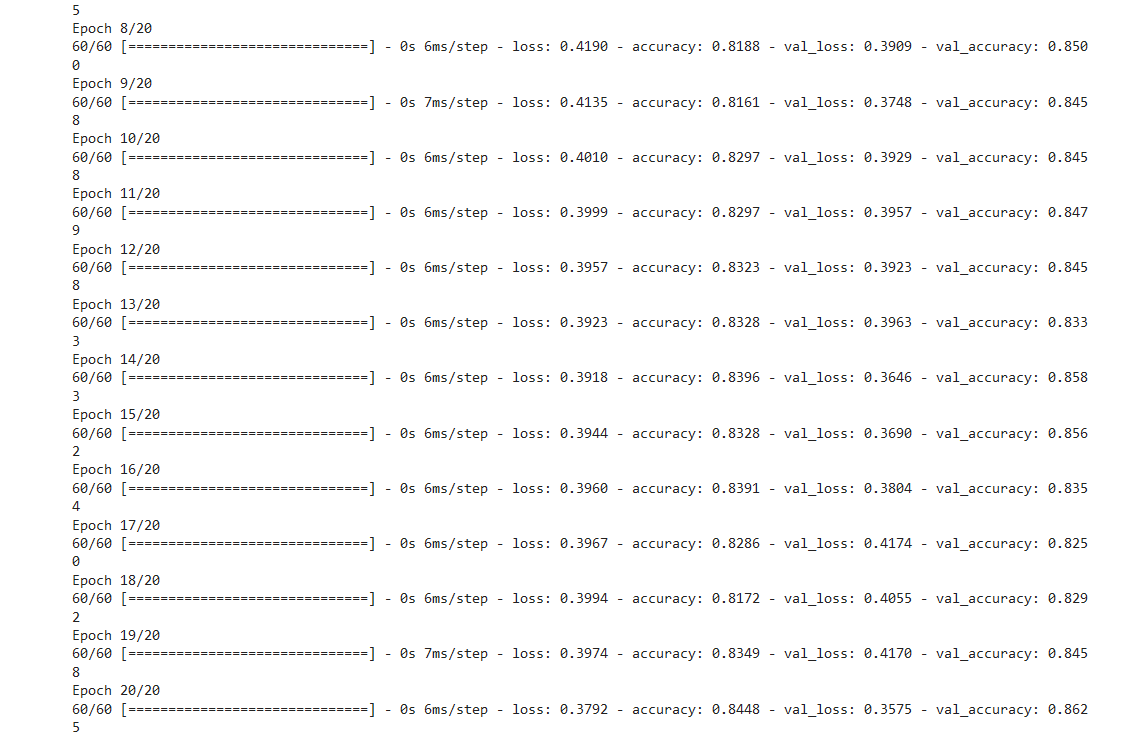


حال پس از استخراج ویژگی از هر 3000 داده آموزشی و نگهداری آنها در لیست train\_images\_features و لیبل‌های آنها در train\_images\_labels، باید این داده‌ها را برای ورودی به مدل آماده کنیم. داده‌ی پردازش شده را به دو بخش train و test تقسیم می‌کنیم. همچنین label های آنها را به صورت one-hot در می‌آوریم:

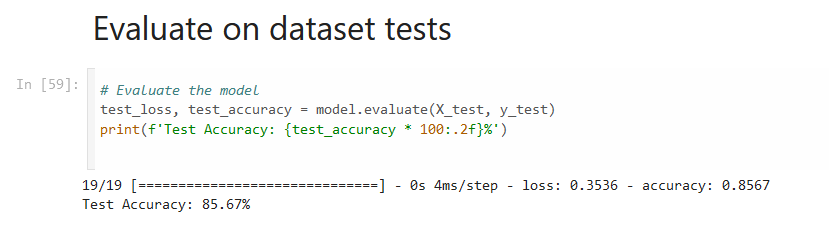


مدل CNN را به شکل زیر ساخته که شامل لایه‌های کانوولوشنی و max\_pooling است و در ادامه چند لایۀ FC داریم:

نتایج آموزش مدل:

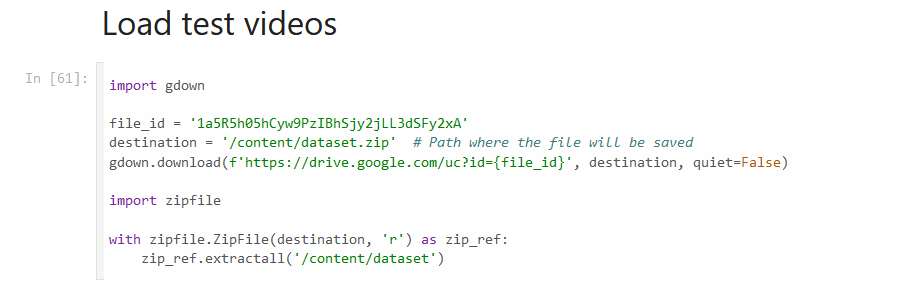


**نتیجۀ تست روی بخشی از داده‌های دیتاست:**



1. **ارزيابي با فريم‌هاي خام.........................................................................................................................**

حال ویدیوها را لود كرده و پس از استخراج فريم براي هر يك ويژگي استخراج مي‌كنيم:



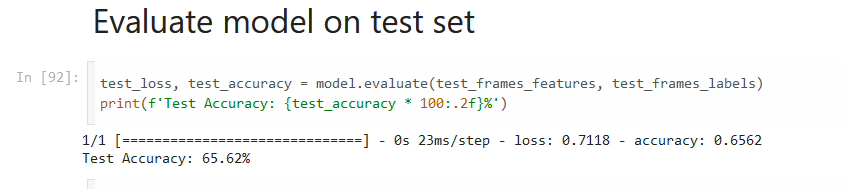




اين تابع براي استخراج ويژگي‌هاي فريم‌ها است:



حال فريم‌ها را روي مدل تست مي‌كنيم و به دقت 65 درصد مي‌رسيم:



1. **ارزيابي با فريم‌هاي كراپ شده..............................................................................................................**

**///////////////////////////**

**مدل InceptionV3**

ما يك مدل ديگر به اسم InceptionV3 نيز آموزش داده و تست كرديم كه پس از طي كردن مراحل بالا (ولي با ورودي‌هاي 75 در 75) به دقت زير رسيد:

