

فاز دوم پروژه

**پیشبینی رده سنی بر اساس خلاصه فیلم**

پردازش زبان‌ و گفتار

**استاد درس**

دکتر مینایی

**دانشجویان**

امیرحسین احمدی – ملیکا احمدی رنجبر

بهار 1401

**فهرست مطالب**

[بخش اول 3](#_Toc108331267)

[بخش دوم 7](#_Toc108331268)

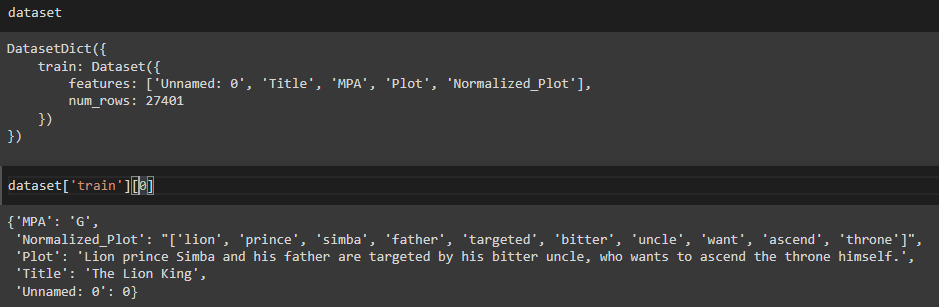
[بخش سوم 9](#_Toc108331269)

# بخش اول

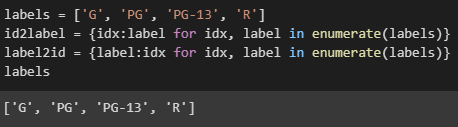
در بخش اول با استفاده از Bert Model و داده‌ها جمع‌آوری شده از فاز اول پروژه، باید جملات جدیدی را تولید می‌کردیم. برای انجام این کار در ابتدا داده‌های Clean شده را با ساختار Dataset و با استفاده از کتابخانه خاص Load می‌کنیم و به شکل زیر آن را خواهیم داشت:



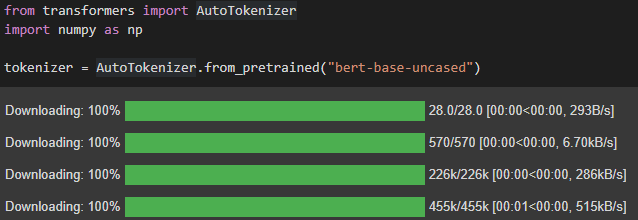
داده کنونی به شکل زیر است:



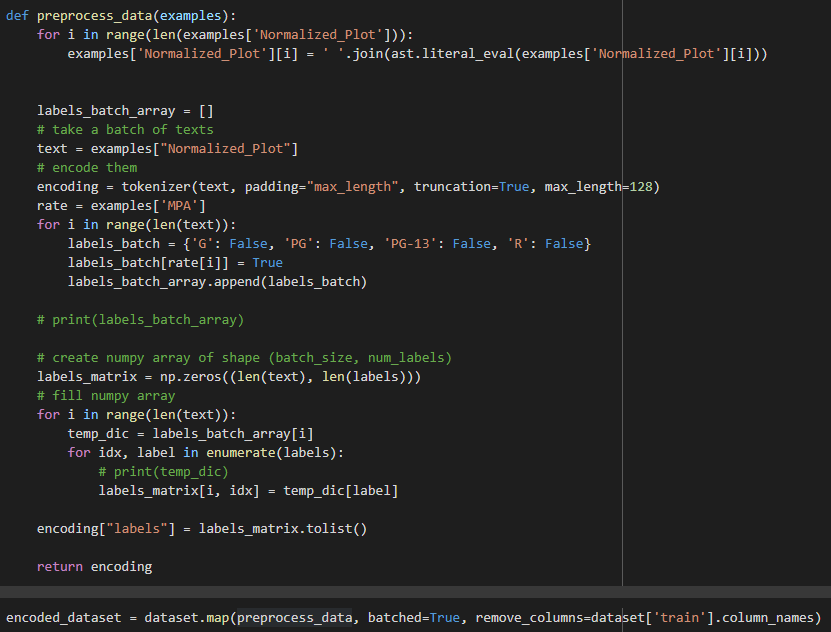
سپس با توجه به تعداد کلاس‌هایی که برای این Classification داریم یک آرایه درست می‌کنیم و Indexing آن را نیز انجام می‌دهیم:



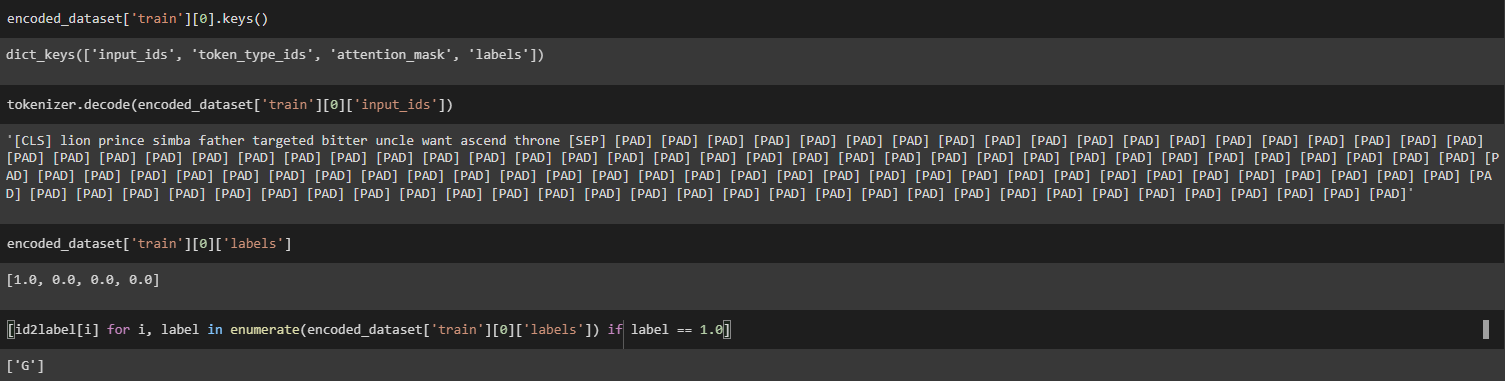
سپس با استفاده از کتابخانه Transformers طبق سوال، Bert Model را خواهیم داشت و از Tokenizer آن استفاده می‌کنیم که به ما یک Vector باز می‌گرداند. ورودی آن نیز تمامی جملات ما به شکل String خواهد بود.



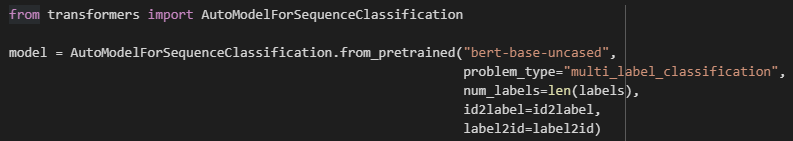
سپس مرحله پیش‌پردازش را بر روی داده‌ها داریم تا بتوانیم از آن برای Fine Tune گردد این مدل استفاده کنیم. خروجی پیش‌پردازش ما یک Embedding خواهد بود.

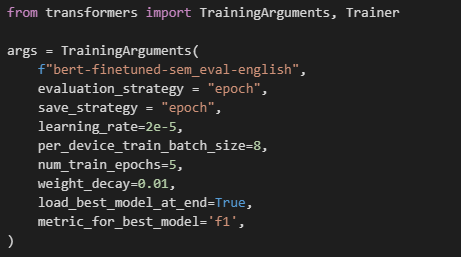


این مرحله یکی از مراحل مهم است. ورودی به این تابع شامل تمامی داده‌های ما خواهد‌بود که به شکل Batch وارد می‌شوند. بنابراین پیاده‌سازی به شکل Vectorized است. برای مثال Examples را که می‌گیریم و Nomalized\_Plot آن را می‌خوانیم به ما یک Array برمی‌گرداند. پس از آن، Rate و یا همان MPA, Label را نیز می‌خوانیم. با توجه به همان Text گرفته از Tokenizerای که با Bert تعریف کردیم استفاده می‌کنیم و Padding نیز به آن می‌دهیم تا Vector هایی که برمی‌گرداند با توجه به طول مختلف ورودی‌ها اندازه یکسانی داشته‌باشند. سپس از یک آرایه Dictionary استفاده می‌کنیم و برای هر کدام از این داده‌ها مشخص می‌کنیم کدام دسته برای آن مقدار True دارد. در ادامه مثالی را بررسی خواهیم کرد. سپس با توجه به این آرایه از دیکشنری‌ها یک ماتریس برای تمامی ورودی‌ها و Labelها میسازیم. اگر فرض کنیم که ورودی‌ها اندازه Batch = 32 داشته‌باشند و 4 دسته داریم بنابراین ماتریس 32\*4 خواهد بود که مقادیر 0 و 1 دارند تا کلاس درست را مشخص کنند. در انتها نیز این Encoding را باز می‌گردانیم.



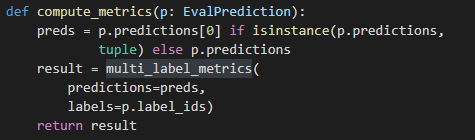
پس از آماده کردن این Embedding ها، دوباره با توجه به Bert یک مدل برای Sequence Classification انتخاب می‌کنیم و پارامترهای مختلف آن را نیز تعیین می‌کنیم:

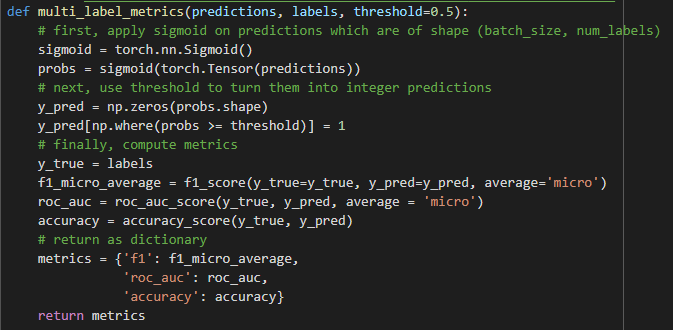


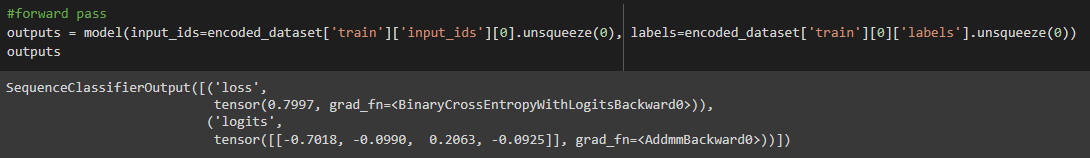


پارامترهایی نظیر Learning\_Rate و … در بالا مشاهده می‌شوند.

برای دسته‌بندی در نظر گرفته شده که می‌تواند Multi-Label باشد و یک فیلم دارای چند MPA باشد. اما به طور منطقی باشد یک حالت باشد. باز با اینکه Multi-Label حالتی کلی‌تر است آن را در نظر گرفتیم. پس از آن نیز برای محاسبه مقادیر و Metricهایی مثل Accuracy یک تابع جداگانه تعریف می‌کنیم:

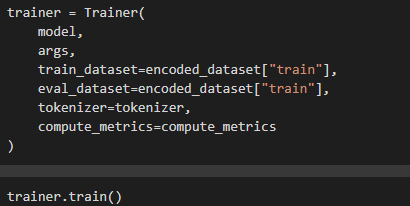




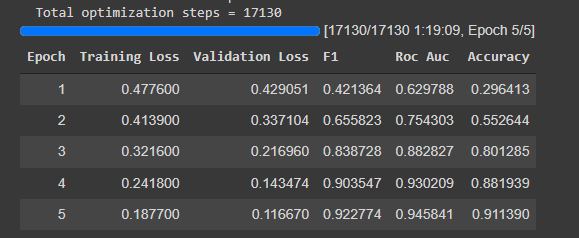


خروجی به این شکل مشخص می‌شود که پس از آن Trainer مدل را تعریف می‌کنیم و تمامی داده‌های Train را به آن می‌دهیم. حال با توجه به همان Encoderای که در ابتدا نیز زده‌بودیم آن را صدا می‌زنیم.

نوبت آموزش مدل است:

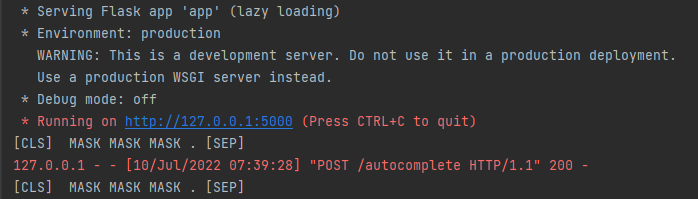


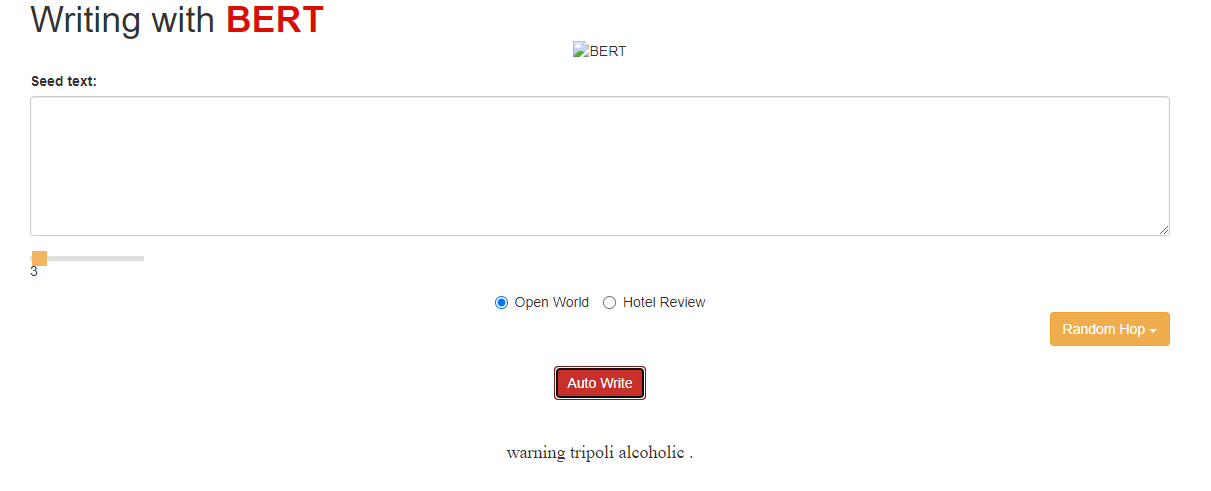
در 5 مرحله آموزش می‌بیند:



نتیجه نهایی نیز به این شکل است که بسیار قابل قبول است و دقت بالایی دارد و Loss کم. در انتها نیز مدل را Save می‌کنیم تا بتوانیم برای تولید جملات با استفاده از Prediction از آن استفاده کنیم.

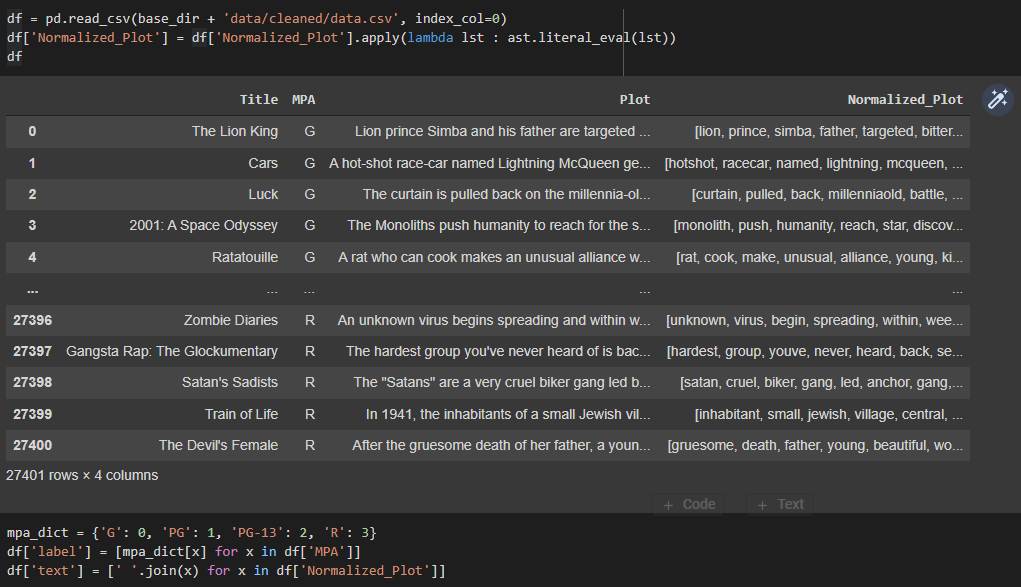
برای تولید جمله فقط لازم است app.py را راه اندازی کرده و مکان ذخیره شده مدل را به آن بدهیم. سپس با باز کردن index.html و تعیین پارامتر های دلخواه جمله تولید شده جدید را دریافت کنید.



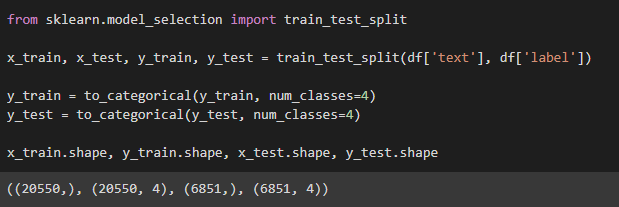


# بخش دوم

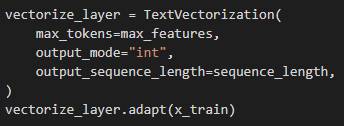
برای این بخش ابتدا داده های تولید شده در فاز اول پروژه را دریافت کرده و ستون های label که Index نمرات MPA فیلم هاست و text که جمله Normalize شده در فاز قبل است را به آن اضافه میکنیم.



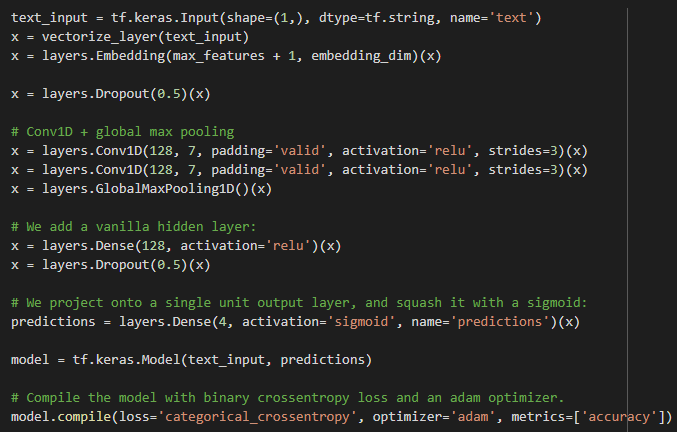
داده های آموزشی و تست را از دو ستون اضافه شده ساخته و y ها را به صورت Categorical در میاوریم.



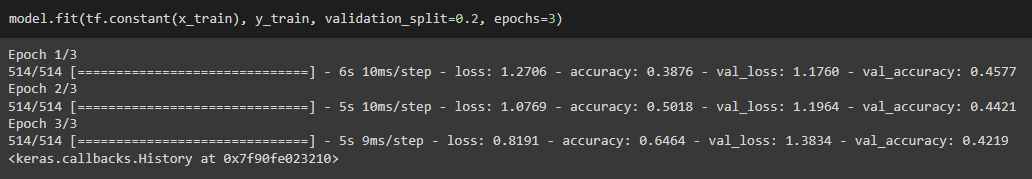
برای مدل این بخش، ابتدا یک لایه به نام vectorize\_layer تعریف کرده که وظیفه دارد جملات را به صورت دنباله ای با طول ثابت و تشکیل شده از Index کلمات در بیاورد. آن را با استفاده از داده های آموزشی ابتدا adapt کرده تا بتوانیم در مدل استفاده کنیم.



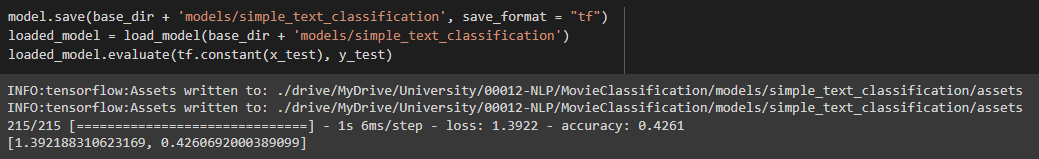
برای ساخت مدل، بعد از لایه vectorize، یک لایه embedding میگذاریم. سپس بعد از یک لایه Dropout، دو لایه Conv گذاشته و GlobalMaxPooling زده تا بتوان داده های بدست آمده را به لایه های Dense داد. در نهایت بعد از یک لایه Dense 128 تایی و تعدادی Dropout لایه آخر که شامل 4 نورون به ازای هر MPA قرار میدهیم تا مدل کامل شود. برای loss از categorical\_crossentropy و برای optimizer از adam استفاده میکنیم.



سپس در 3 epoch مدل را آموزش میدهیم.(وقتی با epoch های بیشتر fit کردم، مدل overfit میشد و دقت داده های validation ثابت میموند، به همین دلیل فقط 3 epoch آموزش دادیم.) همانطور که میبینید دقت داده های validation برابر با 42 درصد و loss برابر با 1.38 است.

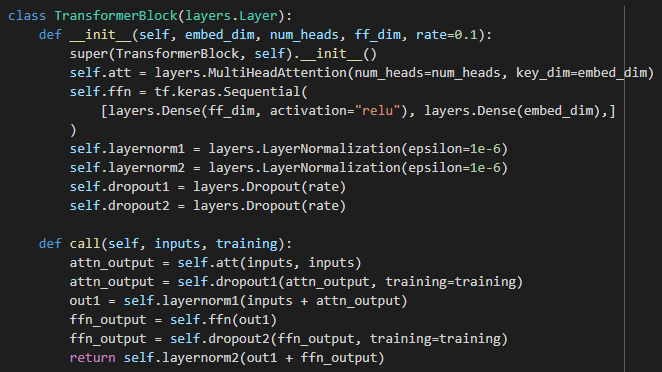


در نهایت مدل را save کرده و آن را با داده های تست evaluate میکنیم که به دقتی شبیه به داده های validation میرسیم.

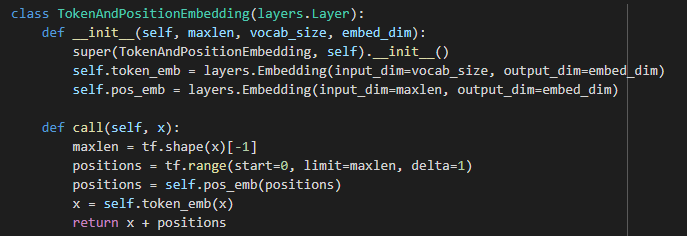


# بخش سوم

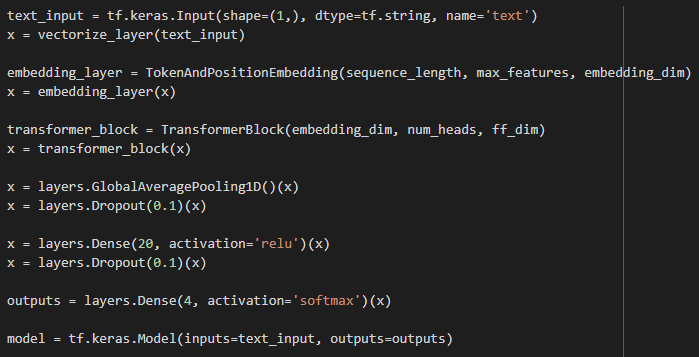
در این بخش سه مدل مختلف را آموزش داده تا بتوانیم به دقت بهتری از قسمت قبل برسیم. در مدل اول میخواهیم با یک لایه Transformers به بهبود مدل کمک کنیم. لایه Transformers را به صورت زیر تعریف میکنیم که شامل یک لایه Attention دو لایه برای Normalization و دو لایه Dropout است.



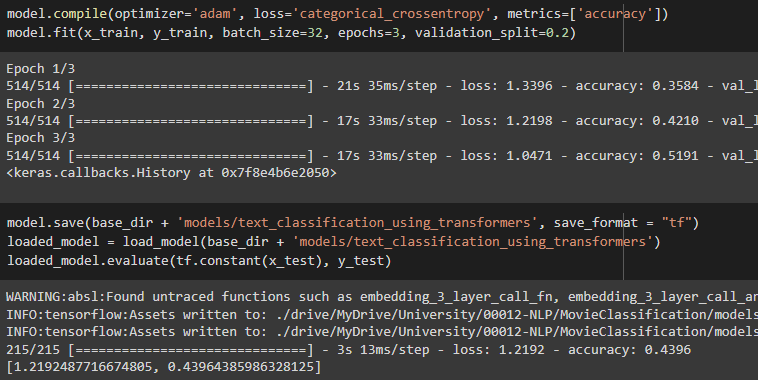
همچنین برای پیش پردازش Transformers یک لایه Embedding برای زمان هر کلمه نیاز داریم. برای این کار یک لایه TokenAndPositionEmbedding تعریف کرده که در آن هر دو Embedding انجام شود.



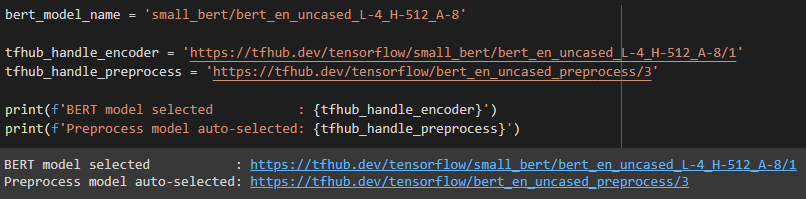
سپس مدل را تقریبا مانند مدل قسمت قبل ساخته با این تفاوت که به جای لایه های Conv از Transformers استفاده میکنیم.



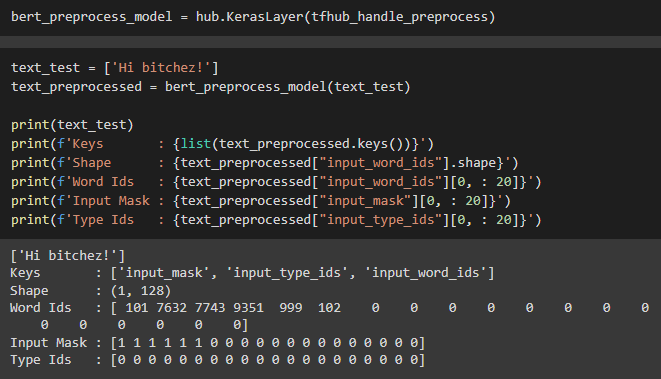
نتیجه آموزش مدل و save و evaluate آن را نیز در زیر مشاهده میکنید که هم دقت و هم loss کمی (نه خیلی) نسبت به حالت قبل بهتر شده است.



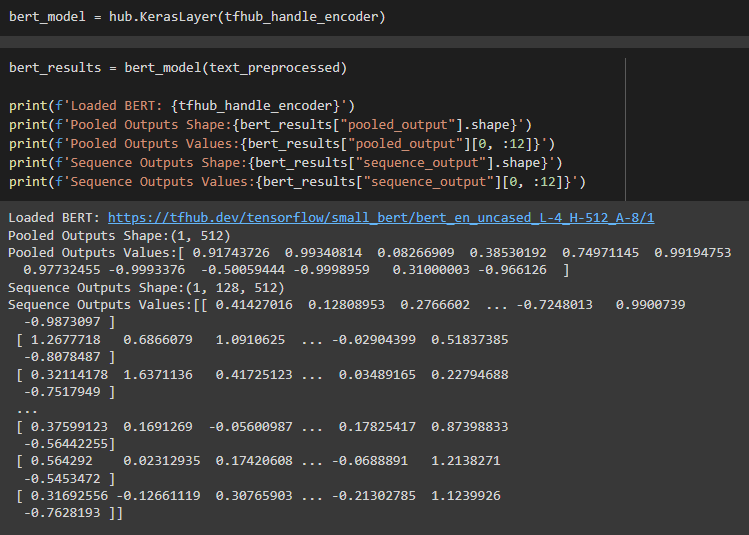
برای مدل دوم این بخش به سراغ استفاده از Bert رفتیم. برای این کار ابتدا یک نسخه از Bert را در نظر گرفته و لینک لایه Encoder و Preprocess آن را بدست آورده تا بتوانیم از آن ها استفاده کنیم.



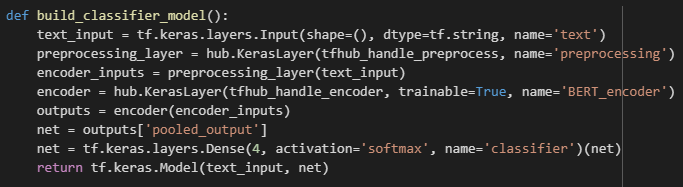
حال لایه Preprocess را دانلود کرده و با یک مثال دستی آن را تست میکنیم.



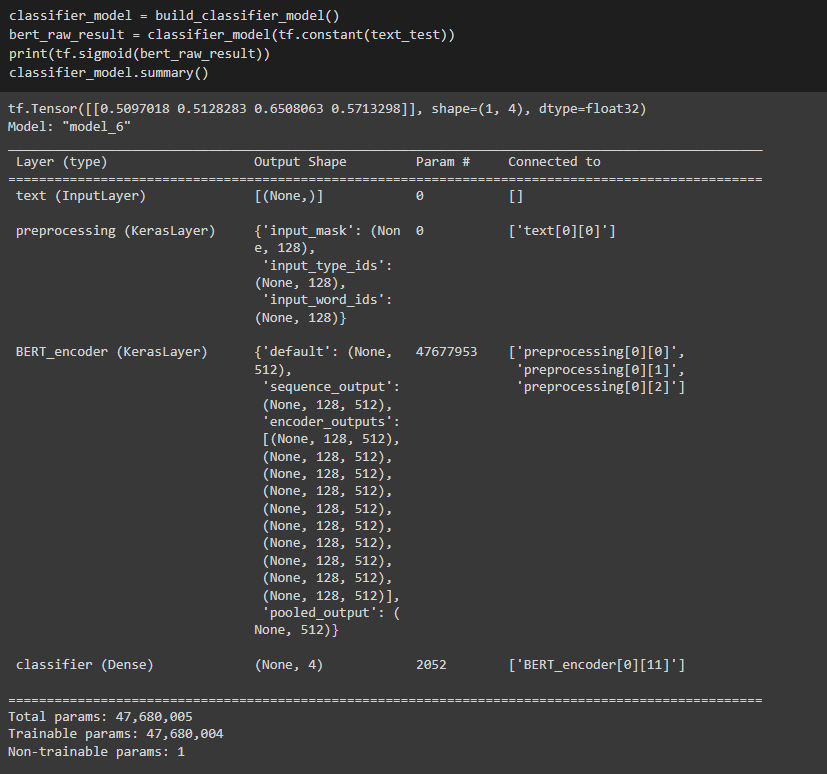
برای لایه Encoder نیز همین کار را تکرار کرده و خروجی لایه Preprocess را به عنوان ورودی نمونه به لایه داده تا عملکرد آن را بررسی کنیم.



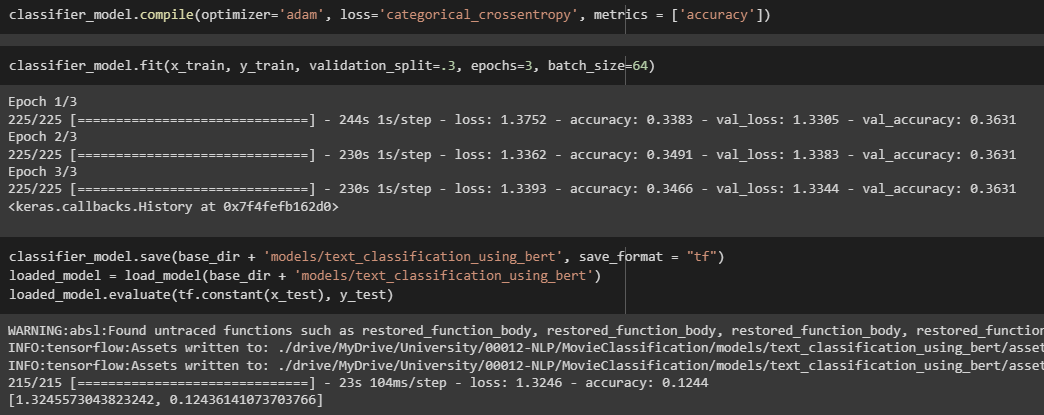
برای ساخت مدل این لایه های Bert را به هم متصل کرده و در نهایت به یک Dense 4 تایی وصل میکنیم تا با داده های ما سازگار شود.



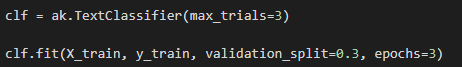
در ادامه معماری مدل ساخته شده و اجرای یک مثال رو آن را میبینیم.



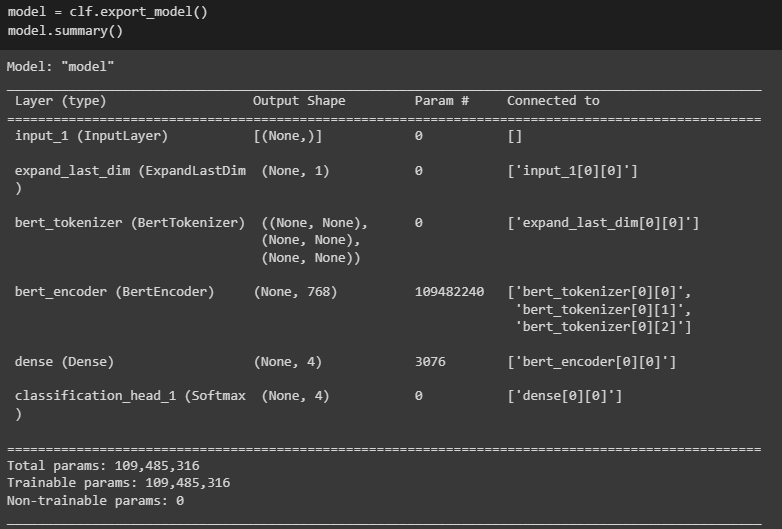
در آخر نیز مدل را Compile کرده و bert را fine tune میکنیم. همانطور که میبینید دقت نهایی نسبت به مدل های پیشین پایین تر است. من چندین بار با عوض کردن مدل bert، اضافه کردن لایه Dense بالا پایین کردن hyperparameter ها سعی کردم که دقت مدل را بالا ببرم ولی نتوانستم به بالا تر 36 درصد برسم.



در آخر با استفاده از کتابخانه AutoKeras سعی کردیم بهترین مدل ممکن Text Classification را بر روی داده های خود پیدا کنیم. این کار را در 3 Trial انجام داده و برای هر کدام مدل را 3 epoch آموزش دادیم.



در نهایت معماری بهترین مدل بدست آمده را در زیر میبینید که بسیار نزدیک به معماری مدل بخش قبلی است، ولی با جزئیاتی که من متوجه آن نشدم توانسته دقت بالا تری را بدست بیاورد.



در نهایت مدل با دقت نزدیک به 50 درصد و loss 1.15 توانست روی داده های test، evaluate کند.



دقت پایین در تمام مدل ها احتمالا ناشی از آن است که تعداد label ها 4 تاست و شاید Accuracy معیار مناسبی برا ارزیابی نباشد. از طرفی تعداد داده های موجود کم بود که در پایین آمدن دقت بی تاثیر نیست. البته ما با استفاده از تعدادی data augmentation تلاش در زیاد کردن داده ها کردیم ولی با تاثیر چندانی روی دقت مدل ها نگذاشت و از آن ها استفاده نکردیم.

# منابع

<https://github.com/prakhar21/Writing-with-BERT>

<https://keras.io/examples/nlp/text_classification_from_scratch/>

<https://keras.io/examples/nlp/text_classification_with_transformer/>

<https://colab.research.google.com/github/tensorflow/text/blob/master/docs/tutorials/classify_text_with_bert.ipynb#scrollTo=6IwI_2bcIeX8>

<https://towardsdatascience.com/text-classification-made-easy-with-autokeras-c1020ff60b17>