|  |  |
| --- | --- |
| نام و نام خانوادگی | **بهار برادران افتخاری – علیرضا قاسمی** |
| شماره دانشجویی | **810102097 - 115198** |
| تاریخ ارسال گزارش | **۱۴۰۲.۱۰.۲۱** |

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین اول** | | |

فهرست

[پاسخ ۱. تشخیص احساسات گفتار 4](#_Toc155889445)

[۱-۱. چالش های داده های صوتی در یادگیری 4](#_Toc155889446)

[۲-۱. رویکرد HuBERT 5](#_Toc155889447)

[۳-۱. پیش پردازش داده ها 6](#_Toc155889448)

[۴-۱. ساخت دیتالودر 10](#_Toc155889449)

[۱-۴-۱. تولید بازنمایی مناسب از کل دنباله ورودی 12](#_Toc155889450)

[۲-۴-۱. آموزش مدل 13](#_Toc155889451)

[پاسخ ۲. تنظیم دقیق مدل BERT 18](#_Toc155889452)

[۱-۲. پیش‌پردازش داده‌ها 18](#_Toc155889453)

[۲-۲. تنظیم دقیق داده‌ها 19](#_Toc155889454)

[۳-۲. فریز کردن لایه‌های مدل 21](#_Toc155889455)

[۱-۳-۲. فریز کردن ۹ لایه‌ی ابتدایی 21](#_Toc155889456)

[۲-۳-۲. فریز کردن تمامی لایه‌ها بجز لایه‌ی آخر و لایه‌ی embedding 23](#_Toc155889457)

[۴-۲. تنظیم دقیق مدل بر روی لایه‌های میانی 25](#_Toc155889458)

[۵-۲. حذف head های attention در مدل 27](#_Toc155889459)

**شکل‌ها**

شکل 1 - اطلاعات اماری مربوط به فرکانس نمونه برداری 7

شکل 2 - اطلاعات اماری مربوط به فرکانس نمونه برداری 8

شکل 3 - نمودار دقت 15

شکل 4 - نمودار خطا 16

شکل 5 - ماتریس آشفتگی 17

شکل 6 - توزیع طول token ها 19

شکل 7 - خطای آزمایش ۱ 20

شکل 8 - دقت آزمایش ۱ 21

شکل 9 - خطای آزمایش ۲ 22

شکل 10 - دقت آزمایش ۲ 23

شکل 11 - خطای آزمایش ۳ 24

شکل 12 - دقت آزمایش ۳ 25

شکل 13 - دقت آزمایش ۴ 26

شکل 14 - خطای آزمایش ۴ 27

شکل 15 - دقت آزمایش ۵ 28

شکل 16 - خطای آزمایش ۵ 29

**جدول‌ها**

[جدول 1 - پارامترهای آموزش 14](#_Toc155889432)

[جدول 2 - توزیع برچسب‌ها 18](#_Toc155889433)

[جدول 3 - نتایج آزمایش ۱ 19](#_Toc155889434)

[جدول 4 - ماتریس آشفتگی آزمایش ۱ 20](#_Toc155889435)

[جدول 5 - نتایج آزمایش ۲ 22](#_Toc155889436)

[جدول 6 - ماتریس آشفتگی آزمایش ۲ 22](#_Toc155889437)

[جدول 7 - نتایج آزمایش ۳ 23](#_Toc155889438)

[جدول 8 - ماتریس آشفتگی آزمایش ۳ 24](#_Toc155889439)

[جدول 9 - نتایج آزمایش ۴ 25](#_Toc155889440)

[جدول 10 - ماتریس آشفتگی آزمایش ۴ 26](#_Toc155889441)

[جدول 11 - نتایج آزمایش ۵ 28](#_Toc155889442)

[جدول 12 - ماتریس آشفتگی آزمایش ۵ 28](#_Toc155889443)

[جدول 13 - مقایسه 30](#_Toc155889444)

# پاسخ ۱. تشخیص احساسات گفتار

## ۱-۱. چالش های داده های صوتی در یادگیری

چالش اول

در داده های ورودی هر گفتار حاوی چندین واحد صدا مختلف است. این چالش به این واقعیت اشاره دارد که در هر گفته ورودی (یک عبارت یا جمله) واحدها یا اجزای مختلف صوتی متمایز وجود دارد. واحدهای صدا را می توان به عنوان واج ها، هجاها یا سایر عناصر صوتی که سیگنال کلی گفتار را تشکیل می دهند در نظر گرفت.

در پردازش سنتی گفتار یا یادگیری بازنمایی، معمول است که با این فرض کار کنید که یک واحد صدا به خوبی تعریف شده، به طور مثال داده های ورودی مثل phoneme هایی هستند که در حالت ایده ال تولید شده اند. با این حال، واقعیت در زبان طبیعی این است که گفته ها از ترکیبی از واحدهای صوتی مختلف تشکیل شده است. اینها می توانند شامل تغییرات در زیر و بم، تن، سرعت و سایر ویژگی های آکوستیک باشند. و فرض حالت ایده ال در اینجا برقرار نیست. در یادگیری خود نظارتی بازنمود های گفتاری، برخورد با واحدهای صوتی متعدد یک چالش شناخته شده است زیرا مدل نیاز به درک و نمایش روابط و تغییرات بین این واحدها دارد. صرفاً در نظر گرفتن کل گفته به عنوان یک واحد صدا ممکن است کار یادگیری را بیش از حد ساده کند و جزئیات دقیق سیگنال گفتار را ثبت نکند.

چالش دوم

قبل از مرحله اموزش هیچ لیست و داده ای از واژگانی که در فایل های صوتی ورودی داریم در دسترس نیست. واژه‌نامه (lexicon) مجموعه‌ای از کلمات یا واحدها، اغلب واج‌هایی در زمینه پردازش گفتار است که به عنوان مرجعی برای درک و پردازش یک زبان خاص عمل می‌کند. در غیاب واژه نامه در مرحله قبل از آموزش، مدل دانش یا راهنمایی صریح در مورد واحدهای صوتی خاصی که باید در حال یادگیری نمایش آنها باشد، ندارد. این فقدان واحدهای از پیش تعریف شده می تواند تراز و سازماندهی نمایش های آموخته شده را به روشی معنادار برای مدل چالش برانگیز کند. در سناریوهای یادگیری نظارت شده سنتی، یک واژه نامه اغلب برای نظارت بر مدل در طول آموزش استفاده می‌شود که برچسب‌های صریح برای واحدهای صوتی مختلف در داده‌ها ارائه می‌کند. با این حال، در یادگیری خود نظارتی، مدل نیاز به کشف این واحدها بدون تجمل برچسب های از پیش تعریف شده دارد.

چالش سوم

هر یک از واحد های صدا دارای طول متفاوتی هستند و دارای یک طول ثابت نیستند. واحدهای صوتی در داده های گفتاری می توانند مدت زمان های متفاوتی داشته باشند و هیچ تقسیم بندی واضح و صریحی بین این واحدها وجود ندارد. در سیگنال‌های گفتاری، واج‌ها، هجاها یا سایر واحدهای صوتی می‌توانند بسته به عواملی مانند سرعت گفتار، تأکید یا زمینه، مدت زمان متفاوتی داشته باشند. علاوه بر این، ممکن است مرزها یا نشانگرهای واضحی وجود نداشته باشد که به صراحت شروع و پایان هر واحد صوتی را در سیگنال صوتی خام نشان دهد.این تنوع در طول و عدم تقسیم بندی صریح، تشخیص و درک ساختار زمانی سیگنال گفتار را برای مدل های یادگیری خود نظارتی چالش برانگیز می کند. بدون نشانه‌های واضح برای تقسیم‌بندی، مدل باید یاد بگیرد که مرزهای بین واحدهای صوتی را بر اساس الگوها و تغییرات ذاتی در داده‌های صوتی شناسایی کند.

## ۲-۱. رویکرد HuBERT

این مقاله برای حل چالش هایی بالا مدل HuBERT را معرفی کرده است.بدین منظور این مدل از یک بخش خوشه بندی افلاین برای تولید لیبل های تراز شده (aligned target labels) برای تابع هزینه پیش بینی همانند الگوریتم BERT استفاده میکند. این مرحله به مدیریت واحد های صوتی متنوع کمک میکند همچنین این روش به HuBERT این قابلیت را میدهد تا بازنمایی سطح بالایی از ورودی های که ماسک نشده اند را یادبگیرد تا بتواند با استفاده از ان به درستی اهداف ورودی های که ماسک شده اند را پیشبینی و استنباط کند. تابع هزینه فقط برای نواحی که ماسک شده اند اعمال شده است و این کار باعث میشود که مدل مجبور باشد تا ساختار متوالی داده های ورودی را به خوبی اموزش ببیند همچنین مدل مجبور میشود هم زمان هر دو بخش اوایی (acoustic) و زبانی مربوط به داده ها را هم زمان یادبگیرد. با استفاده از این روش خوش بندی بدون نظارت این الگوریتم میتواند حضور چند واحد صوتی را در هر گفتار ورودی با طول متغییر واحد های صوتی یادبگیرد

مدل Hubert یک مدل self-supervised است که میتواند کلمات را از فایل های صوتی ورودی بازنمایی کند این مدل از ساختاری همانند مدل bert برای پیش بینی ناحیه های ماسک شده فایل صوتی استفاده میکند با این تفاوت که در این مدل بجای استفاده از words یا tokens به عنوان لیبل های تارگت از hidden unit هایی که از طریق خوشه بندی ویژگی (MFCC) فایل های صوتی بدست امده است استفاده میکند. این hidden unit ها به یک تعبیری مشابه زبان انسان است و میتواند هم ویژگی های اوایی (acoustic) و زبانی مربوط به داده ها را در خود بازنمایی کند

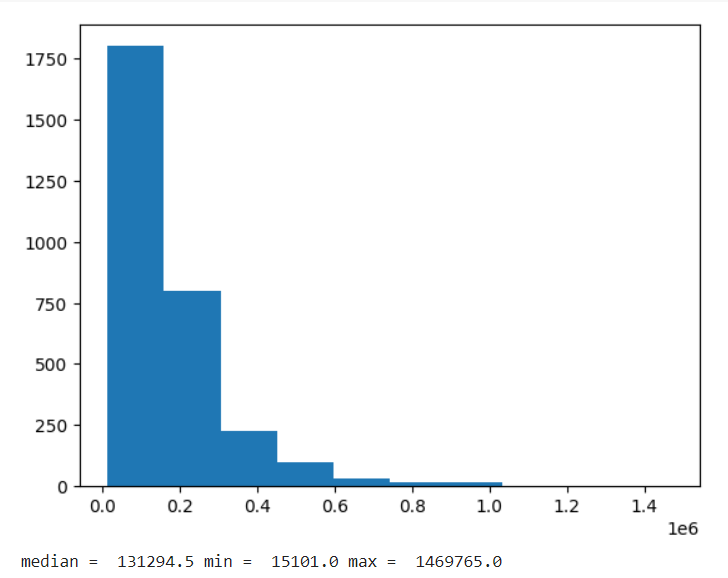
## ۳-۱. پیش پردازش داده ها

اطلاعات اماری دیتاست

این دیتاست شامل 3000 فایل صوتی میباشد. که هر یک از این فایل های صوتی به یکی از 6 دسته خشم، ترس ، شادی ، غم ، تعجب و حالت خنثی تعلق دارد. همچنین در نام این فایل ها مشخصات جنسیت گوینده و شماره مربوط به هر گوینده و لیبل فایل و شماره فایل قرار دارد که از این اطلاعات در اینجا تنها لیبل مورد استفاده قرار گرفته است در کد زیر لیبل مربوط به هر فایل استخراج شده است و به شکل عددی و onehot تبدیل شده اند

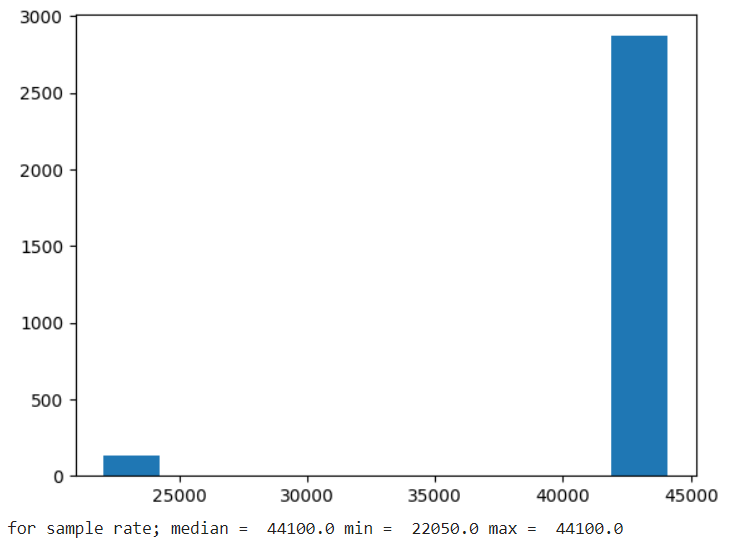


اطلاعات اماری مربوط به اندازه طول فایل های صوتی داخل دیتاست به صورت زیر است.



شکل 1 - اطلاعات اماری مربوط به فرکانس نمونه برداری

همچنین اطلاعات اماری مربوط به فرکانس نمونه برداری فایل های صوتی به صورت زیر است این داده ها با دو فرکانس نمونه برداری مختلف ضبط شده اند.



شکل 2 - اطلاعات اماری مربوط به فرکانس نمونه برداری

پیش پردازش های لازم برای انجام تسک مورد نظر شامل موارد زیر میباشد

فایل های صوتی برای اینکه برای اموزش به شبکه داده شود باید از فرمت داده های wav به یک فرمت که قابل فهم برای شبکه باشد تبدیل شود به همین منظور با استفاده از دستور زیر فایل های صوتی را میخوانیم

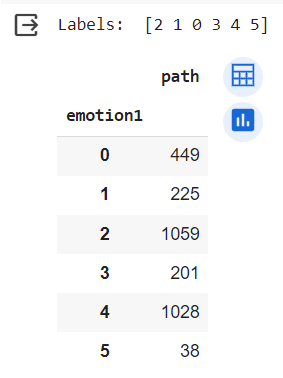
speech,sr= librosa.load(self.data["path"][idx], sr=16000, mono=True)

همچنین با استفاده از دستور زیر ویژگی های فایل های صوتی به یک بردار با طول مشخصص تبدیل میشود

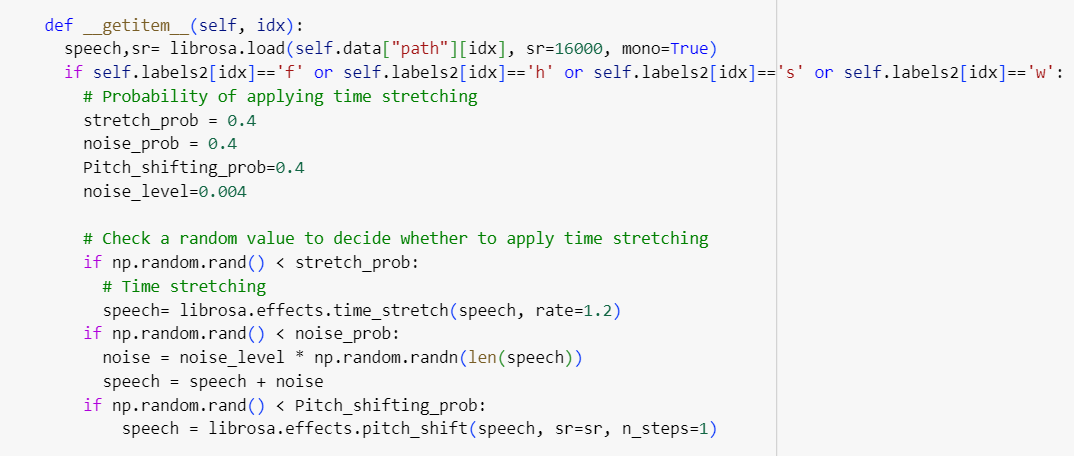
encodings = self.feature\_extractor(speech, sampling\_rate=16000, padding=True, return\_tensors="pt")

برای استخراج ویژگی از Wav2Vec2FeatureExtractor که یک مدل از پیش اموزش داده شده میباشد استفاده میکنیم این مدل را از کتابخانه transformers فراخوانی شده است.

توزیع داده های دیتاست داده شده به صورت زیر است.



با توجه به اینکه توزیع داده های هر کلاس متفاوت میباشد پس دیتاست داده شده بالانس نمیباشد. کلاس خشم دارای بیشترین تعداد داده و کلاس ترس دارای کمترین تعداد داده است. برای حل این مشکل از روش data augmentation استفاده میشود و برای کلاس هایی که دارای توزیع کمتری نسبت به بقیه کلاس ها هستند فرایند data augmentation انجام میشود. برای فایل های صوتی تقویت داده های متفاوتی میتوان انجام داد از بین همه انها در این تمرین از اضافه کردن نویز و کشیدن فایل صوتی (stretched audio) تغییر pitch\_shift استفاده شده است که نمونه هایی از انها را در کد ها اورده شده است و میتوانید تاثیر تقویت داده های بیان شده را مشاهده کنید. فرایند تقویت داده تنها برای داده های بخش ترین انجام شده است برای این کار در بخش دیتالودر در کلاس داده های ترین بخش زیر اعمال شده است. که با احتمال 0.4 بر روی کلاس هایی که توزیع کمتری دارند فرایند تقویت داده انجام میشود بدین صورت در هر ایپاک مدل داده های جدیدی از کلاس هایی که توزیع کمتری دارند مشاهده میکند و مشکل بالانس نبودن داده حل میشود



برای اینکه داده ها را با حفظ توزیع داده شده به سه دسته اموزش تست و ولیدیشن تقسیم کنیم باید پارامتر stratify را هنگام تقسیم داده ها برابر با لیبل های داده ها قرار دهیم با این کار توزیع داده ها هنگام تقسیم داده بهم نمیخورد همچنین باید تقویت داده را تنها برای بخش ترین انجام دهیم

val\_df, test\_df = train\_test\_split(test\_df1, test\_size=0.5, random\_state=101, stratify=test\_df1["emotion1"])

## ۴-۱. ساخت دیتالودر

در دیتالودر با استفاده از دستور زیر ویژگی هایی داده های ورودی استخراج میشود.

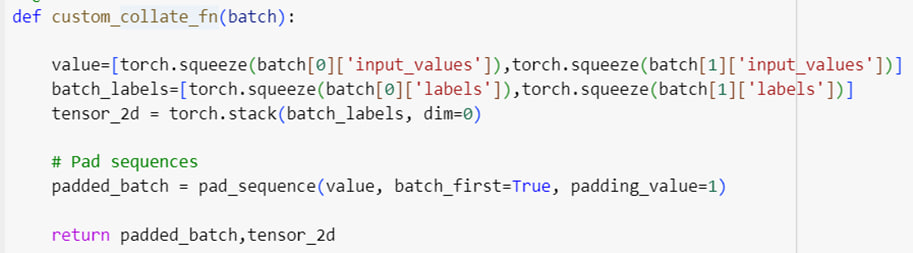
encodings = self.feature\_extractor(speech, sampling\_rate=16000, padding=True, return\_tensors="pt")

اگر از این دستور خارج از دیتالودر استفاده کنیم باتوجه به پارامتر padding ان طول همه بردار ها برابر با طول بزرگترین داده میشود اما چون حجم داده های تولید شده بیشتر از فضای گوگل کولب است نمیتوانیم از این روش استفاده کنیم و باید بخش استخراج ویژگی را در داخل دیتالودر قرار دهیم در این صورت در دیتالودر در بخش

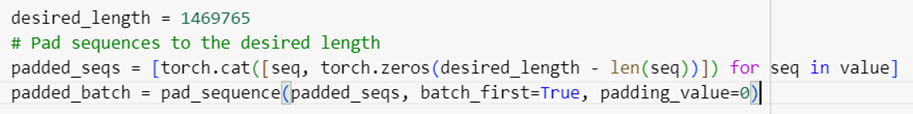
def \_\_getitem\_\_(self, idx):

در این صورت دیتالودر هر بار یک داده میگیرد و به تابع self.feature\_extractor میدهد و ویژگی های داده داده شده استخراج میشود اما چون تنها یک داده به این تابع داده شده است پارامتر padding دیگر نمیتواند طول ویژگی های داده های مختلف را با یک دیگر یکسان کند و در نتیجه در خروجی دیتالودر داده هایی با طول مختلف داریم که باعث ایجاد خطا سایز خروجی میشود برای رفع این مشکل از ویژگی collate\_fn در دیتالودر استفاده میکنیم با استفاده از این ویژگی میتوانیم یک تابع دلخواه تعریف کنیم در این تابع خروجی های مربوط به داده های داخل هر batch دیتالودر به عنوان ورودی داده میشوند و خروجی های این تابع برابر با خروجی دیتالودر است و به بیان دیگر ابتدا خروجی های داده های داخل هر batch به این تابع وارد میشود و فرایند هایی روی انها انجام میشود و سپس خروجی دیتالودر میشوند

با توجه به مطالب بیان شده در کلاس custom\_collate\_fn با استفاده از pad\_sequence پدینگ مورد نیاز برای هم طول شدن داده های انجام میشود. این پدینگ بدین صورت است که داده های با طول کمتر با مقدار صفر پر میشوند تا هم طول داده های بزرگتر شوند.



راه دوم این است که طول همه دنباله ها را برابر با طول بزرگترین داده قرار بدهیم همانگونه که قبلا گفته ایم اگر در خارج از دیتالودر این کار را انجام بدهیم باعث میشود که حجم زیادی از رم را اشغال کند به همین خاطر در داخل دیتالودر از دستور زیر استفاده میکنیم



با استفاده از این دستور میتوانیم داده های داخل هر بچ را به ازای هر طول دلخواه (طول انتخاب شده باید بزرگتر از کوچکترین طول داده ها باشد) تبدیل کنیم. اگر بجای طول دلخواه از طول بزرگترین داده ورودی استفاده کنیم خواسته مسئله را براورده کرده ایم.

حال به بررسی این دو روش میپردازیم

روش اول پد کردن داده ها به حداکثر طول داده ها در هرbatch:

مزایای:

کارایی حافظه: در این روش در مقایسه با روش دوم به حجم حافظه کمتری برای ذخیره سازی داده ها نیاز داریم

بار محاسباتی کاهش میابد: در این روش محاسبات غیر ضروری مرتبط با پردازش داده ها با طول های بسیار طولانی را کاهش می دهد. و تنها برای داده های با طول بزرگ پردازش هایی با ابعاد بزرگ داریم و برای داده های با طول کوچک پردازش هایی متناسب با طول انها خواهیم داشت

معایب:

بازده تاثیر اندازه دسته: طول ثابت باید با دقت انتخاب شود، زیرا ممکن است اندازه دسته را محدود کند. اگر دنباله ها بسیار طولانی تر از طول انتخاب شده هستند، ممکن است لازم باشد آنها را به تکه های کوچکتر تقسیم کنید.

روش دوم پد کردن دنباله داده ها به حداکثر طول داده ها

پردازش دسته ای کارآمد: در این روش تمام دنباله ها در یک دسته دارای طول یکسانی هستند که امکان پردازش موازی را برای gpu فراهم می کند که می تواند به طور قابل توجهی سرعت آموزش را افزایش دهد.

پیاده سازی ساده: پیاده سازی با حداکثر طول بسیار ساده است و توسط بسیاری از چارچوب های یادگیری عمیق پشتیبانی می شود.

معایب:

افزایش استفاده از حجم حافظه: پد کردن داده ها به طول حداکثر ممکن است منجر به استفاده بیش از حد از حافظه شود، به خصوص اگر اکثر دنباله ها بسیار کوتاهتر از حداکثر باشند. یا داده ها از نوع عکس یا فایل های صوتی باشند

بار محاسباتی: پد کردن داده ها به طول حداکثر باعث میشود تمامی محاسبات در ابعاد بالاتر اجرا شود که این کار باعث میشود حجم محاسباتی بسیار زیاد شود

### ۱-۴-۱. تولید بازنمایی مناسب از کل دنباله ورودی

مدل HuBERT با توجه به ساختار ان خروجی که بازنمایی کل دنباله ورودی باشد را تولید نمیکند. این مدل از سه بخش (a feature extractor, a feature projection layer, a BERT encoder.) تشکیل شده است

The feature extractor

این بخش شامل شبکه های کانولوشنی CNN است که داده های ورودی را به دنباله ای از feature vectors تبدیل میکند خروجی هر یک از این شبکه های کانولوشنی دارای بعد 512 است و همچنین این بخش طول دنباله ورودی را تا 320 برابر کوچک تر میکند بنابر این بعد کلی خروجی این شبکه برابر با (batch\_size, L/320, 512) است.

The feature projection layer

این لایه یک تبدیل کننده خطی است که بردار های ویژگی که در حالت قبلی بدست امده را به بردار هایی که با بعد hidden states مدل BERT encoder مطابقت داشته باشد تبدیل میکند. بعد خروجی این مرحله به اندازه مدل Hubert بستگی دارد برای مدل پایه خروجی 768 است و برای مدل بزرگ خروجی 1024 و برای مدل x-large خروجی برابر با 1280 است در مسئله از مدل پایه که دارای خروجی 768 است استفاده شده است بعد کلی خروجی این مرحله برابر با (batch\_size, L/320, 768) است

The BERT encoder

این مرحله شامل پشته ای از لایه های ترانسفورماتور است که حالت های پنهان را از لایه قبلی میگیرد و دنباله ای از نمایش های زبانی را تولید میکند در مدل پایه این انکودر شامل 12 لایه ترانسفورماتور است

بعد خروجی هر لایه ترانسفورماتور همان ابعاد ورودی است که برابر با (batch\_size, L/320, 768)

در اینجا ما از اخرین خروجی 'last\_hidden\_state' شبکه encoder به عنوان یک بازنمایی مناسب برای داده های ورودی استفاده میکنیم برای اینکه این بازنمایی به یک لایه dense وارد شوند باید با استفاده از دستور زیر به ابعاد مناسب تبدیل شوند با استفاده از این دستور از بعد L/320 میانگین میگیریم

x11=torch.mean(x1['last\_hidden\_state'], 1)

بجای دستور میانگین میتوانستیم از تابع های دیگری نیز استفاده کنیم این تابع با ازمون و خطا بدست امده است. و با استفاده از این تابع جواب بهتری گرفته ایم .بدین ترتیب برای هر داده ورودی بازنمایی با بعد 768 داریم که از این بازنمایی برای طبقه بندی داده ها استفاده میشود

### ۲-۴-۱. آموزش مدل

جدول انتخاب شده برای پارامتر های اموزش به شکل زیر است:

جدول 1 - پارامترهای آموزش

|  |  |
| --- | --- |
| پارامتر های اموزش | مقادیر استفاده شده |
| نرخ یادگیری | lr=1e-3 |
| تعداد ایپاک | 6 |
| criterion | nn.CrossEntropyLoss() |
| optimizer | AdamW |
|  |  |

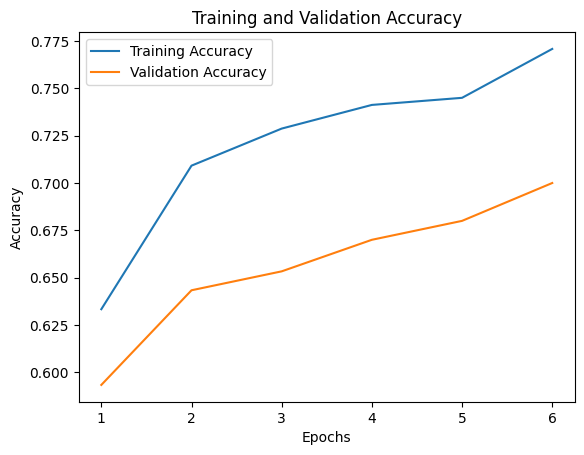
**پارامتر های بیان شده در بالا با استفاده از تجربه و ازمون و خطا بدست امده اند و ممکن است مقدار بهینه ای نداشته باشند به طور مثال با نرخ یادگری ده به توان منفی چهار سرعت یادگیری مدل و افزایش دقت پایین بود به همین دلیل نرخ یادگیری را افزایش داده ایم**

**تعداد ایپاک ها باید با توجه به نمودار های loss و دقت خروجی محاسبه شود در شکل زیر مشاهده میشود که شبکه با افزایش تعداد ایپاک ها هنوز در حال یادگیری میباشد و اگر تعداد ایپاک ها بیشتر شود شبکه میتواند به دقت بهتری برسد و نمودار loss نیز دارای شیب کاهشی است و با افزایش تعداد ایپاک ها شبکه به نتایج قابل قبول تری میرسید اما به دلیل محدودیت های gpu که توسط گوگل کولب در اختیار ما قرار داده میشود ما از این مقادیر استفاده کرده ایم**

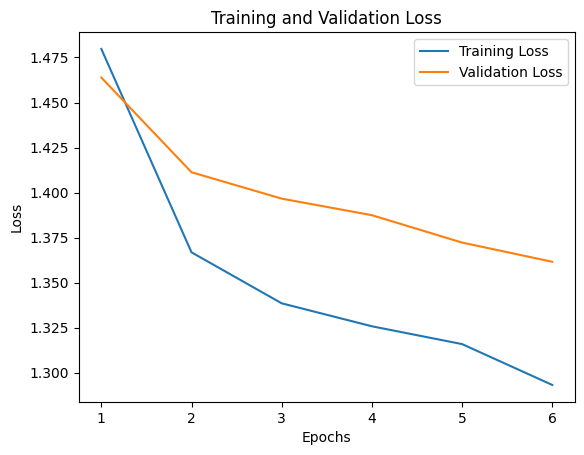
**مقدار تابع هزنیه در اینجا crossEntropyloss در نظر گرفته شده است این انتخاب بر اساس تجربه بوده است و در مسائل مشابه با این مسئله دیده شده است که از این تابع هزینه استفاده میشود**

**Optimizer استفاده شده در این مسئله AdamW این بهینه ساز یک اکستنشن از بهینه ساز adam است. که در این مسئله جواب بهتری نسبت به این بهینه ساز میدهد و به صورت ازمون و خطا انتخاب شده است.**

نمودار های loss و دقت خروجی به صورت زیر است. همانگونه که مشاهده میشود این نمودار دقت داده های ولیدیشن پایین تر از نمودار داده های تست قرار میگیرد و همراه با ان در حال افزایش یافتن است و این نشان دهنده این است که دقت در حال افزایش یافتن است و با افزایش تعداد ایپاک ها دقت شبکه بیشتر میشود نمودار های loss نیز به صورت نزولی هستند و هنوز به مرحله اورفیتینگ نرسیده اند



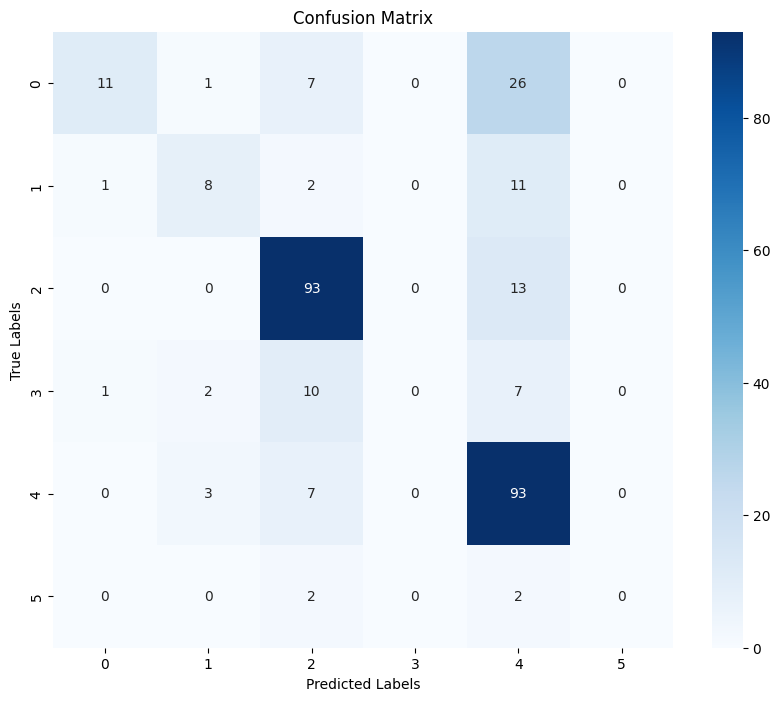
شکل 3 - نمودار دقت



شکل 4 - نمودار خطا

ماتریس درهم ریختگی برای داده های تست این مسئله برابر است با:

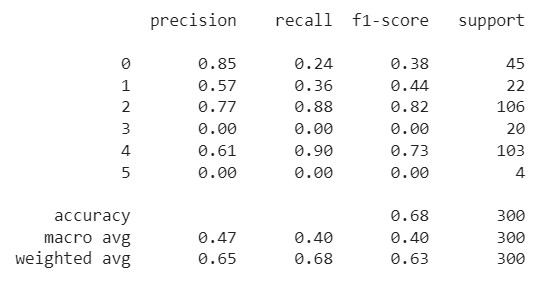
همانگونه که مشاهده میشود در این مسئله هنوز مسئله بالانس نبودن داده ها باقی مانده است اما نسبت به حالتی که از دیتا اگمنتیشن استفاده نکرده ایم نتایج بهتری گرفته ایم. در این نمودار در دو کلاس شماره دو و چهار مدل به خوبی توانسته این کلاس ها را پیش بینی کند همچنین کلاس های صفر و یک را نیز تا حد متوسطی به درستی پیش بینی کرده است اما برای کلاس های سه و پنج نتیجه خوبی از خود نشان نداده است. نتایج ضعیف در این قسمت ممکن است مربوط به تعداد کم ایپاک ها باشد و اگر تعداد ایپاک ها بیشتر شود دقت خروجی نیز بالاتر میرود و نتایج بهتری را مشاهده میکنیم



شکل 5 - ماتریس آشفتگی

گزارش طبقه بندی برای داده های تست

همانگونه که گفته شد برای دو کلاس که تعداد داده ها بیشتر است مدل نتایج خوبی از خود نشان داده است و برای دو کلاس نتایج دقت، recall،f1\_score متوسط میباشد و برای دو مدل با تعداد داده های کم مدل نتایج ضعیفی از خود نشان داده است.



# پاسخ ۲. تنظیم دقیق مدل BERT

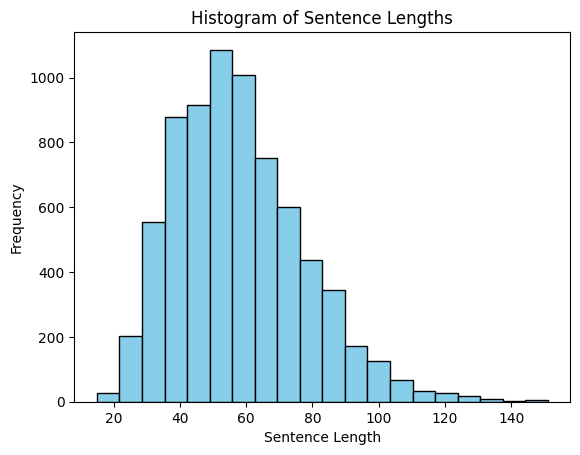
## ۱-۲. پیش‌پردازش داده‌ها

مجموعه داده‌ی FarsTail دارای سه دسته آموزش، ارزشیابی و تست است. هر کدام از این گروه داده‌ها دارای یک نمونه‌ی premise، یک نمونه‌ی hypothesis و برچسب مربوط به این دو جمله است. مجموعه داده‌های آموزش دارای ۷۲۶۶ نمونه، مجموعه داده‌های ارزیابی دارای ۱۵۳۷ نمونه و مجموعه داده‌های تست دارای ۱۵۶۴ نمونه می‌باشد. توزیع برچسب‌ها در مجموعه داده‌های آموزش به شکل زیر است:

جدول 2 - توزیع برچسب‌ها

|  |  |  |  |
| --- | --- | --- | --- |
| برچسب | n | e | c |
| تعداد | ۲۴۴۸ | ۲۴۲۹ | ۲۳۸۹ |

توزیع طول جملات موجود در مجموعه داده آموزش به شکل زیر است:



شکل 6 - توزیع طول token ها

برای اجرای پیش‌پردازش‌ها ابتدا وجود missing value در این مجموعه داده بررسی شد که خوشبختانه موردی مشاهده نشد. در مرحله بعد بر روی متن‌ها پیش‌پردازش‌های معمولی مانند حذف کدهای html، حذف اموجی‌ها و حذف اطلاعات شخصی انجام شد و هر سه دیتافریم آموزش، ارزیابی و تست به این روش تمیز شدند. خروجی این پیش‌پردازش‌ها ذخیره شد تا در مراحل بعد از آن‌ها استفاده شود.

## ۲-۲. تنظیم دقیق داده‌ها

برای تنظیم دقیق مدل ابتدا کلاس MNLIDataBert تعریف شد که Dataloader های آموزش، ارزیابی و تست را با استفاده از dataframe های تمیزشده‌ی FarsTail می‌سازد. سپس مدل هوشواره با سه کلاس ایجاد شد. این مدل با Adam optimizer به اندازه‌ی ۱۰ ایپاک بر روی مجموعه داده آموزش داده شد و به نتایج زیر دست یافت:

جدول 3 - نتایج آزمایش ۱

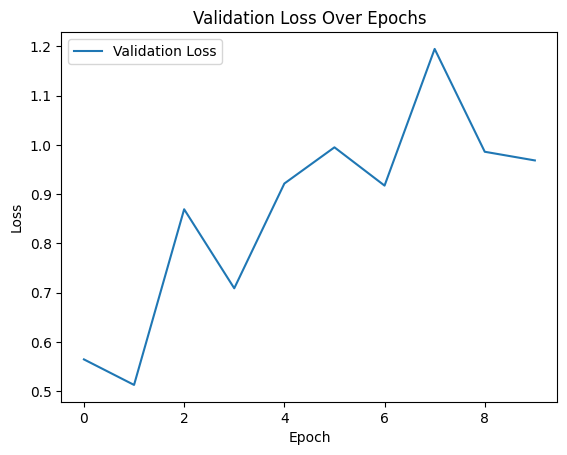
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| معیار | Accuracy | F1 Score for Class 0 | F1 Score for Class 1 | F1 Score for Class 2 |
| مقدار | 0.8087 | 0.8173 | 0.74638 | 0.8642 |

کانفیوژن ماتریس مدل نهایی به شکل زیر است:

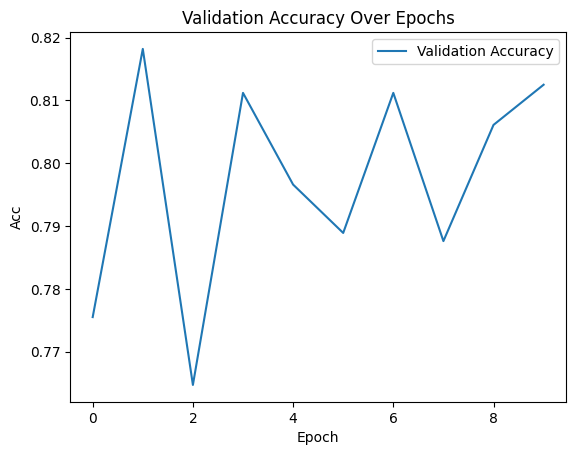
جدول 4 - ماتریس آشفتگی آزمایش ۱

|  |  |  |
| --- | --- | --- |
| 10 | 82 | 423 |
| 36 | 387 | 76 |
| 433 | 69 | 21 |

هم‌چنین نمودارهای مقدار خطا و دقت در طول روند آموزش بر روی داده‌های ارزیابی به شکل زیر است:



شکل 7 - خطای آزمایش ۱



شکل 8 - دقت آزمایش ۱

با توجه به این که مدل اولیه هوشواره برای مساله سه کلاسه طراحی نشده بوده، دقت فوق نشان می‌دهد که آموزش مدل در ایپاک اول پیش رفته است اما در ادامه آموزش مدل به سمت overfitting رفته است. استفاده از روش‌هایی مانند back translation برای داده‌افزایی یا کاهش پیچیدگی مدل یا فریز کردن بخشی از مدل می‌تواند برای حل این مساله به ما کمک کند.

## ۳-۲. فریز کردن لایه‌های مدل

تاثیر فریز کردن لایه‌ها در دو شرایط زیر بررسی شدند:

### ۱-۳-۲. فریز کردن ۹ لایه‌ی ابتدایی

پس از فریز ۹ لایه‌ی ابتدایی، مدل به اندازه‌ی ۵ ایپاک بر روی مجموعه داده آموزش ترین شد و به نتایج زیر دست یافت:

جدول 5 - نتایج آزمایش ۲

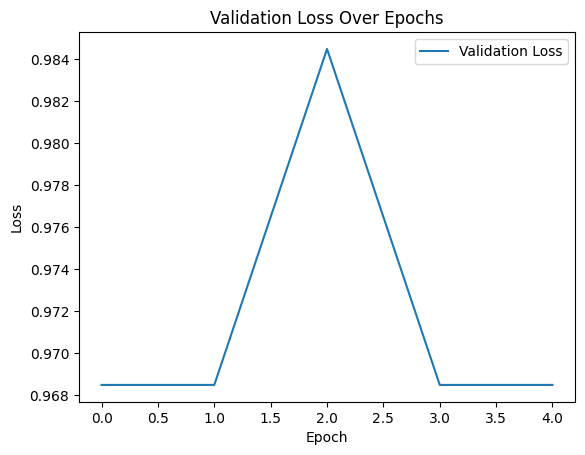
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| معیار | Accuracy | F1 Score for Class 0 | F1 Score for Class 1 | F1 Score for Class 2 |
| مقدار | 0.8087 | 0.8173 | 0.74638 | 0.8642 |

کانفیوژن ماتریس مدل نهایی به شکل زیر است:

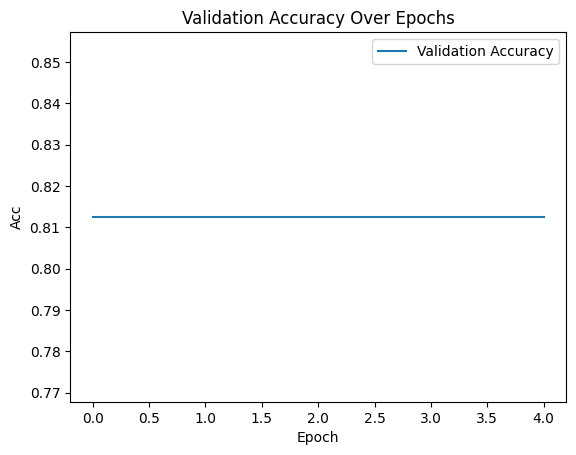
جدول 6 - ماتریس آشفتگی آزمایش ۲

|  |  |  |
| --- | --- | --- |
| 10 | 82 | 423 |
| 36 | 387 | 76 |
| 433 | 69 | 21 |

هم‌چنین نمودارهای مقدار خطا و دقت در طول روند آموزش بر روی داده‌های ارزیابی به شکل زیر است:



شکل 9 - خطای آزمایش ۲



شکل 10 - دقت آزمایش ۲

معیارهای فوق نشان می‌دهند که مدل پس از فریز ۹ لایه‌ی اول پیشرفت خاصی نداشته است و عملکردی مشابه مرحله قبل دارد.

### ۲-۳-۲. فریز کردن تمامی لایه‌ها بجز لایه‌ی آخر و لایه‌ی embedding

پس از فریز ۹ لایه‌ی ابتدایی، مدل به اندازه‌ی ۵ ایپاک بر روی مجموعه داده آموزش ترین شد و به نتایج زیر دست یافت:

جدول 7 - نتایج آزمایش ۳

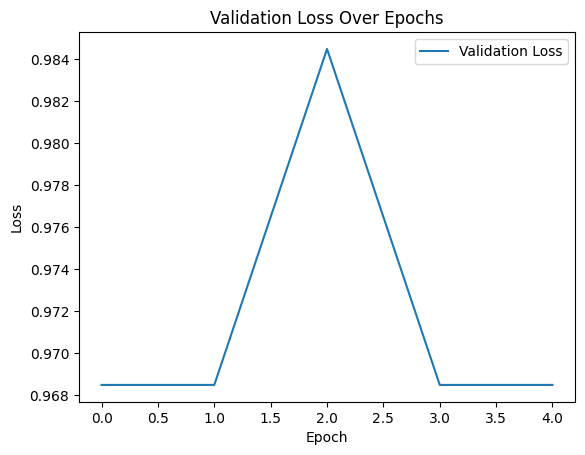
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| معیار | Accuracy | F1 Score for Class 0 | F1 Score for Class 1 | F1 Score for Class 2 |
| مقدار | 0.8087 | 0.8173 | 0.74638 | 0.8642 |

کانفیوژن ماتریس مدل نهایی به شکل زیر است:

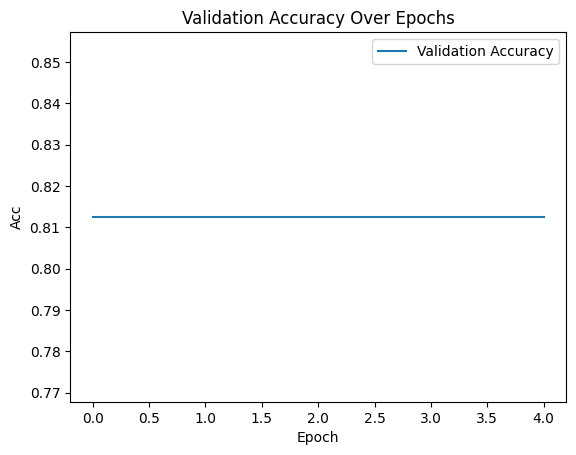
جدول 8 - ماتریس آشفتگی آزمایش ۳

|  |  |  |
| --- | --- | --- |
| 10 | 82 | 423 |
| 36 | 387 | 76 |
| 433 | 69 | 21 |

هم‌چنین نمودارهای مقدار خطا و دقت در طول روند آموزش بر روی داده‌های ارزیابی به شکل زیر است:



شکل 11 - خطای آزمایش ۳



شکل 12 - دقت آزمایش ۳

معیارهای فوق نشان می‌دهند که فریز کردن تمامی لایه‌ها به جز دو لایه‌ی گفته شده نیز باعث پیشرفت خاصی در مدل نشده است و عملکردی مشابه مرحله قبل دارد. مدل بر روی داده‌های آموزش عملکرد بسیار خوبی دارد اما بر روی داده‌های ارزیابی و تست ضعیف عمل می‌کند که به معنای پیچیدگی زیاد شبکه یا تنوع کم داده‌های آموزش است. استفاده از مجموعه داده‌های دیگر یا ساده‌تر کردن معماری شبکه می‌تواند به حل این مساله کمک کند.

## ۴-۲. تنظیم دقیق مدل بر روی لایه‌های میانی

در این مرحله ۹ لایه ابتدایی شبکه باقی مانده و لایه‌های بعدی حذف شدند. نتایج این آزمایش به شکل زیر است:

جدول 9 - نتایج آزمایش ۴

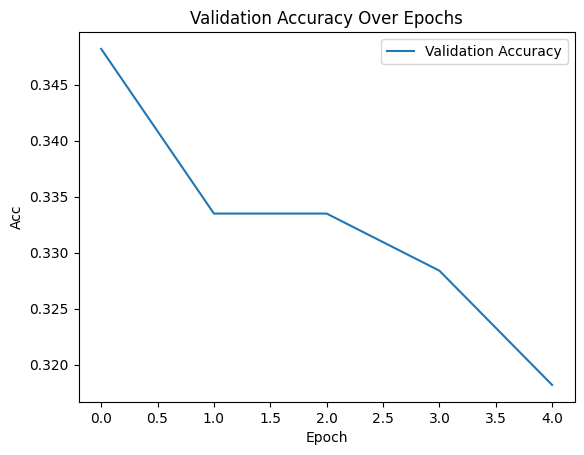
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| معیار | Accuracy | F1 Score for Class 0 | F1 Score for Class 1 | F1 Score for Class 2 |
| مقدار | 0.3246 | 0.0 | 0.4901 | 0.0 |

کانفیوژن ماتریس مدل نهایی به شکل زیر است:

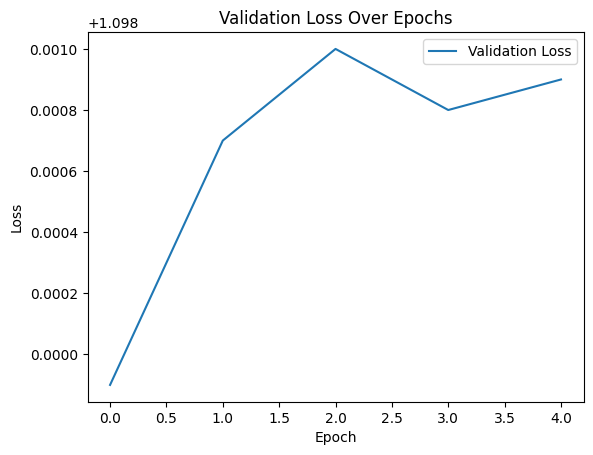
جدول 10 - ماتریس آشفتگی آزمایش ۴

|  |  |  |
| --- | --- | --- |
| 0 | 515 | 0 |
| 0 | 499 | 0 |
| 0 | 523 | 0 |

هم‌چنین نمودارهای مقدار خطا و دقت در طول روند آموزش بر روی داده‌های ارزیابی به شکل زیر است:



شکل 13 - دقت آزمایش ۴



شکل 14 - خطای آزمایش ۴

برای درک بهتر این مشکل، به جز معیارهای فوق دقت شبکه بر روی داده‌های آموزش نیز باید در نظر گرفته شود. دقت مدل در پایان هر ایپاک بر روی داده‌های آموزش نیز در حد chance level می‌باشد که نشانگر underfit بودن مدل است. دلیل این مساله می‌تواند کم بودن پیچیدگی شبکه (بسیار بعید)، مورد نیاز بودن hyperparameter tuning یا نیازمندی مدل به آموزش با تعداد ایپاک بیشتر باشد. هم‌چنین ممکن است لایه‌های حذف شده ویژگی‌های مهم‌تری در خود داشته‌اند که با حذفشان آسیب زیادی به شبکه‌ی pretrained وارد شده و نیاز به استفاده از داده آموزش بیشتری باشد.

## ۵-۲. حذف head های attention در مدل

برای انجام این مرحله از هر کدام از ۱۲ attention head موجود در شبکه نیمی از آن‌ها انتخاب شدند و به تابع prune\_heads داده شدند. خروجی‌های این مرحله به شکل زیر است:

جدول 11 - نتایج آزمایش ۵

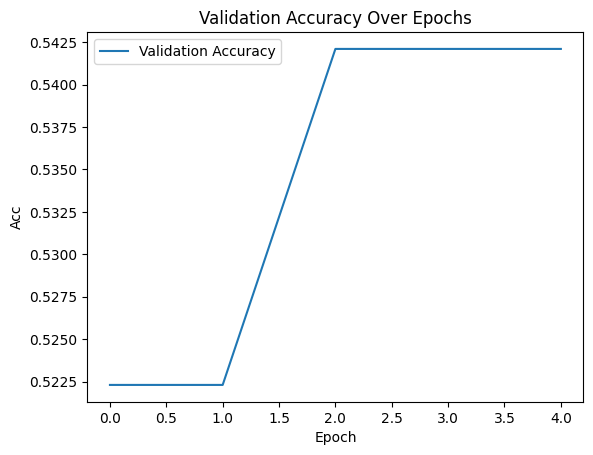
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| معیار | Accuracy | F1 Score for Class 0 | F1 Score for Class 1 | F1 Score for Class 2 |
| مقدار | 0.5328 | 0.5851 | 0.5640 | 0.3908 |

کانفیوژن ماتریس مدل نهایی به شکل زیر است:

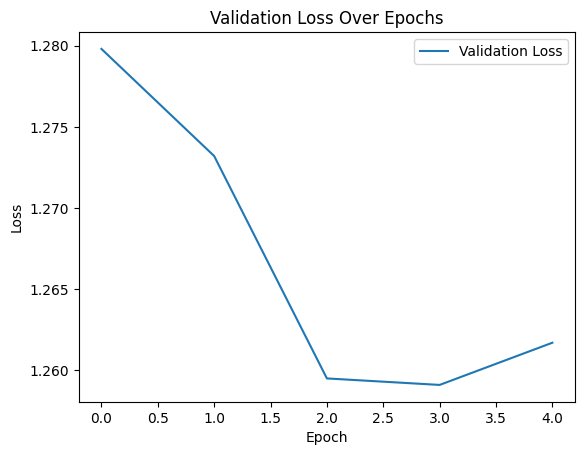
جدول 12 - ماتریس آشفتگی آزمایش ۵

|  |  |  |
| --- | --- | --- |
| 1 | 270 | 244 |
| 3 | 447 | 49 |
| 128 | 369 | 26 |

هم‌چنین نمودارهای مقدار خطا و دقت در طول روند آموزش بر روی داده‌های ارزیابی به شکل زیر است:



شکل 15 - دقت آزمایش ۵



شکل 16 - خطای آزمایش ۵

با توجه به نمودارهای فوق می‌توان گفت که روند ترین فوق احتمالا با ادامه پیدا کردن آموزش به نتایج بهتری می‌رسیده و از این رو برای سبک‌تر کردن مدل حذف head ها روش بهتری از حذف برخی لایه‌ها است.

در نهایت، جدول زیر خلاصه ای از دقت مدل در آزمایش‌های مختلف را نشان می‌دهد:

جدول 13 - مقایسه

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| معیار | Accuracy | F1 Score for Class 0 | F1 Score for Class 1 | F1 Score for Class 2 |
| Finetune اولیه | **0.8087** | **0.8173** | **0.74638** | **0.8642** |
| فریز ۹ لایه | **0.8087** | **0.8173** | **0.74638** | **0.8642** |
| فریز همه لایه‌ها به جز دو مورد | **0.8087** | **0.8173** | **0.74638** | **0.8642** |
| حذف بقیه‌ی لایه‌ها به جز ۹ لایه ابتدایی | 0.3246 | 0.0 | 0.4901 | 0.0 |
| حذف نیمی از head ها | 0.5328 | 0.5851 | 0.5640 | 0.3908 |