به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین پنجم

نام و نام خانوادگی	مهسا همت‌پناه – محمد هادی بابالو
شماره دانشجویی	۸۱۰۱۹۹۵۸۴ - ۸۱۰۱۹۹۳۸۰
تاریخ ارسال گزارش	۱۴۰۲,۱۰,۱۷

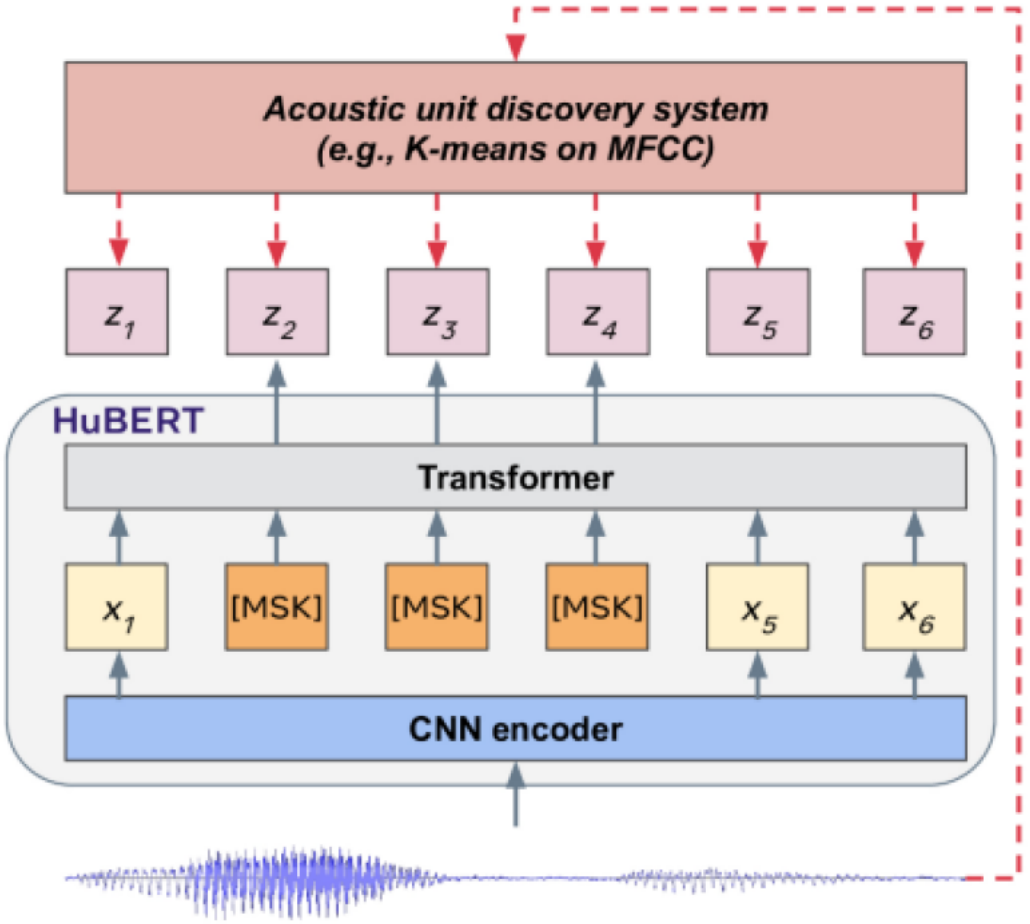
فهرست

شکل‌ها

جدول ها

پاسخ ۱. تشخیص احساسات گفتار (SER)

۱-۱. معرفی مدل HuBERT



شکل ۱ - شمایی کلی از فرایند پیش آموزش مدل **HuBERT**

۱-۲. سوالات تشریحی

۱-۲-۱. چالش های داده های صوتی در یادگیری

سه تا از مهم ترین چالش های داده های صوتی برای یادگیری خود نظارتی که در مقاله مربوط به مدل HuBERT نیز به آن ها اشاره شده است شامل وجود واحدهای صوتی چندگانه، عدم وجود لغت‌نامه در طول پیش‌آموزش و طول‌های متغیر واحدهای صوتی است.

- **واحدهای صوتی چندگانه**: بر خلاف داده‌های متنی که یک جمله را می‌توان به راحتی به کلمات یا زیرکلمات تقسیم کرد، یک داده صوتی شامل چندین واحد صوتی است که به صورت صریح و واضحی تقسیم نشده‌اند. در نتیجه شناسایی و پردازش این واحدهای صوتی مختلف در یک صوت می تواند چالش برانگیز باشد، زیرا مدل باید یاد بگیرد که بین الگوهای آکوستیک مختلف درک و تمایز قائل شود. همچنین داده‌های صوتی اغلب دارای تنوع بالایی هستند. افراد مختلف، با لهجه‌ها، سرعت‌ها، و استایل‌های متفاوت صحبت می‌کنند. این تنوع باعث می‌شود که مدل‌های یادگیری خود نظارتی بر روی داده‌های صوتی با دقت پایینی مواجه شوند. این باعث می‌شود که اعمال مدل‌هایی مانند BERT که از ابتدا برای واحدهای ورودی گسسته طراحی شده‌اند، دشوار باشد.

- **عدم وجود لغت‌نامه در طول پیش‌آموزش:** برخلاف داده‌های متنی، که در آن واژه‌ها واژگانی واضح و از پیش تعریف‌شده دارند، داده‌های صوتی ممکن است در مرحله pre training فاقد واژگان ساختاری باشند. همچنین بین مفهوم معنایی صوت و متن تفاوت وجود دارد به همین دلیل صوت نیاز به اطلاعات ترکیبی برای دست‌یابی به یادگیری بهتر و تعمیم‌پذیری بهتر دارد.
  بطور کلی عدم وجود واژگان از پیش تعریف‌شده کارهایی مانند تشخیص گفتار یا تشخیص احساسات را پیچیده تر میکند.

- **طول‌های متغیر واحدهای صوتی:** واحدهای صدا در داده های صوتی اغلب دارای طول های متغیر هستند و ممکن است هیچ تقسیم بندی صریحی بین آنها وجود نداشته باشد اما کلمات در داده‌های متنی طول‌های ثابت دارند. پرداختن به دنباله‌هایی با طول متغیر چالش‌هایی را برای مدل‌ها، به‌ویژه شبکه‌های عصبی ای که به ورودی‌هایی با اندازه ثابت نیاز دارند، ایجاد می‌کند. مدیریت طول های متغیر به طور کارآمد بدون از دست دادن اطلاعات مرتبط بسیار مهم است.

این چالش‌ها باعث می‌شوند که اعمال مستقیم مدل‌های مشابه BERT، که برای داده‌های متنی طراحی شده‌اند، به داده‌های صوتی دشوار باشد.

۱-۲-۲. رویکرد HuBERT

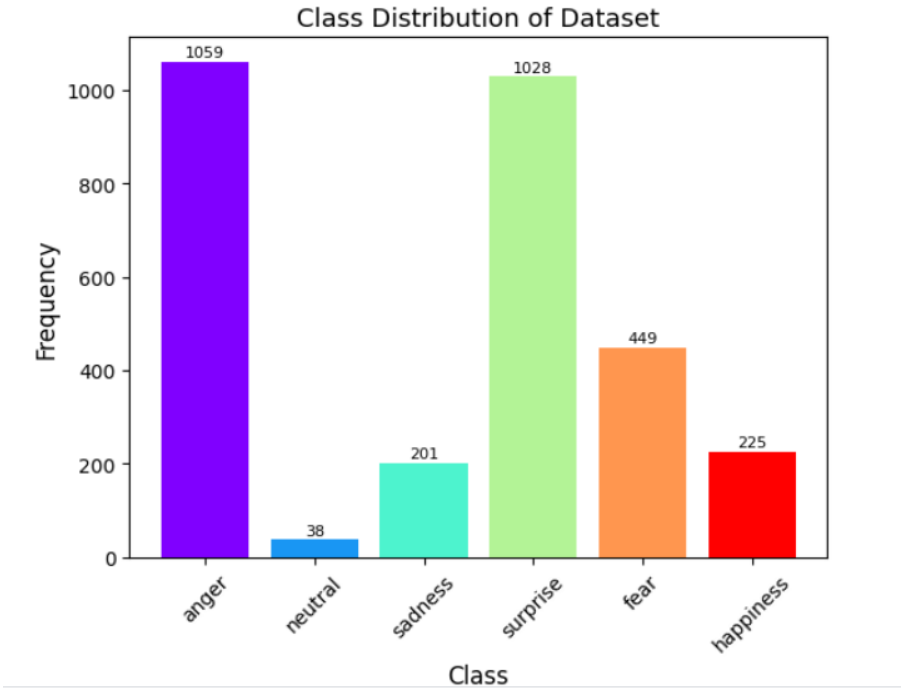
رویکردی که نویسندگان مقاله HuBERT برای مقابله با چالش های مطرح شده در سوال قبل در پیش گرفته اند، یک مرحله خوشه‌بندی آفلاین است تا برچسب‌های هدف متناظر را فراهم کند. برای خوشه بندی در این مدل از الگوریتم K-means برای اختصاص هر بخش از صدا به یکی از خوشه‌های K استفاده می‌شود. هر خوشه شناسایی شده سپس تبدیل به یک واحد پنهان می‌شود و تمام فریم‌های صوتی که به این خوشه اختصاص داده شده‌اند، با این برچسب واحد مجهز می‌شوند.

بطور کلی استفاده از خوشه بند بر روی داده صوت کمک میکند تا داده های صوتی ساختاری مشابه مدل های زبانی پیدا کنند. خوشه بندی صوت در جایگاه یک الوریتم self-supervised عمل کرده و لیبل هایی را برای داده های صوتی ایجاد میکند. لذا صوت میتواند به عنوان دنباله ای از اجزای گسسته ی لیبل دار در نظر گرفته شود و در نتیجه این امکان ایجاد میشود تا بتوان از مدل های قدرتمند حوزه پردازش زبان های طبیعی مانند Bert در کاربردهای تشخیص گفتار استفاده کرد.

۱-۳. معرفی مجموعه دادگان

۱-۳-۱. پیش پردازش داده ها

با توجه به اینکه برای این سوال داخل kaggle کد زده شده است، داده را بطور مستقیم از همین سایت داخل نوت بوک import می کنیم.



شکل ۲ – توزیع کلاس ها در مجموعه دادگان **ShEMO**

شکل ۲ نشان دهنده توزیع داده ها در هر کلاس است. برای بالانس کردن داده های می توان از روش های زیر استفاده کرد.

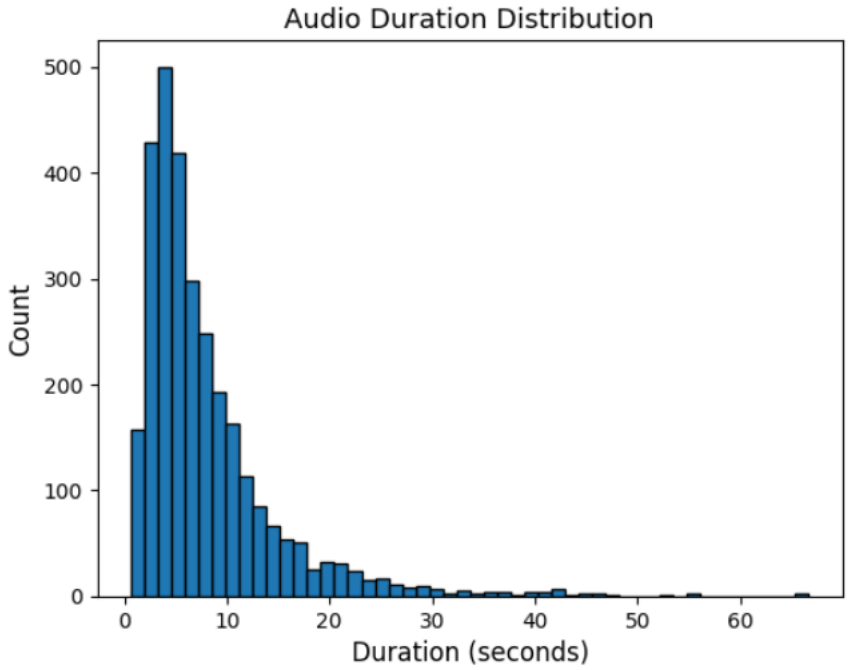
- **نمونه‌برداری مجدد**: این روش شامل افزایش نمونه‌های کلاس با تعداد کمترین تعداد داده یا کاهش نمونه‌های کلاس داده ها در کلاس با بیشترین تعداد داده است. این روش می‌تواند به تعادل بخشیدن به کلاس‌ها کمک کند، اما افزایش نمونه ممکن به دلیل تکرار داده ها منجر به overtraining شود و کاهش نمونه نیز ممکن است منجر به از دست دادن اطلاعات مهم در داده ها بشود.
- **افزایش داده با تولید داده های جدید**: این شامل ایجاد نمونه‌های جدید ترکیبی با اعمال تبدیلات مانند کشیدن زمان، شیفت زمانی و وارد کردن نویز است. این می‌تواند برای داده‌های صوتی بسیار موثر باشد. این روش می‌تواند کمک کند تا داده‌های بیشتری ایجاد بشود و از overtraining جلوگیری کند. با این حال، داده‌های ترکیبی ممکن است دقیقاً داده‌های واقعی جهان را نشان ندهند.

در ابتدا هر دو روش برای بالانس کردن داده ها استفاده شد اما در روش دوم یعنی تولید داده نتایج بهتری داشتیم پس برای آموزش مدل نهایی و کد نهایی از این روش استفاده کردیم. کد زیر نشان دهنده متد های پیاده سازی شده برای تولید داده جدید در کلاس ها با تعداد داده های کمتر است.

```python
def time_stretch(audio, factor):
    return audio.speedup(playback_speed=factor)

def pitch_shift(audio, semitones):
    return audio._spawn(audio.raw_data, overrides={
        "frame_rate": int(audio.frame_rate * (2 ** (semitones / 12.0)))
    })

def augment_audio(audio_path):
    audio = AudioSegment.from_file(audio_path)

    if random.choice([True, False]):
        audio = time_stretch(audio, random.uniform(0.8, 1.2))

    if random.choice([True, False]):
        audio = pitch_shift(audio, random.uniform(-2, 2))

    return audio
```

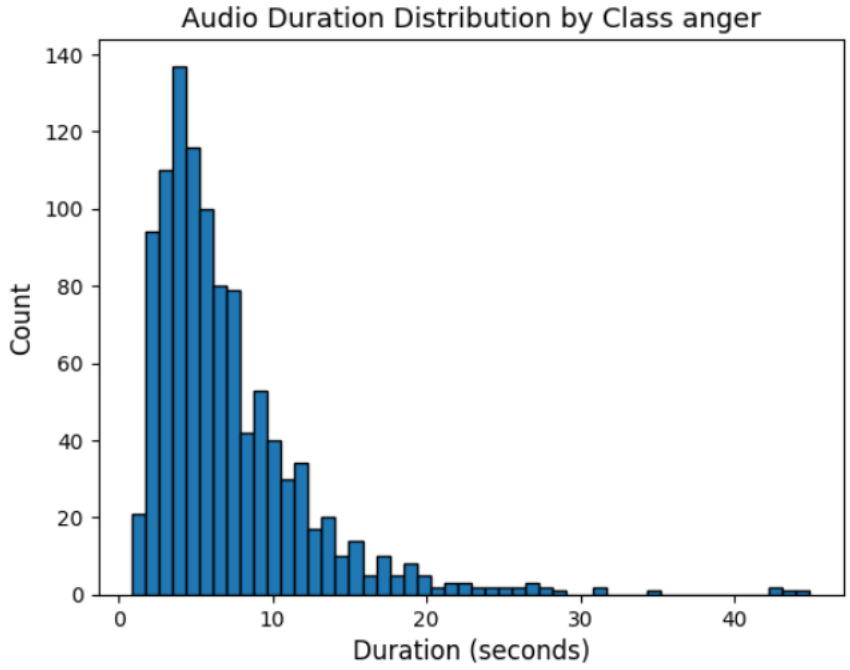
شکل ۳ – کد بالانس داده ها در کلاس ها مختلف

شکل های زیر نشان دهنده طول داده های صوتی و همچنین میانگین و انحراف معیار طول داده بطور کلی در همه داده ها و در هر کلاس است.



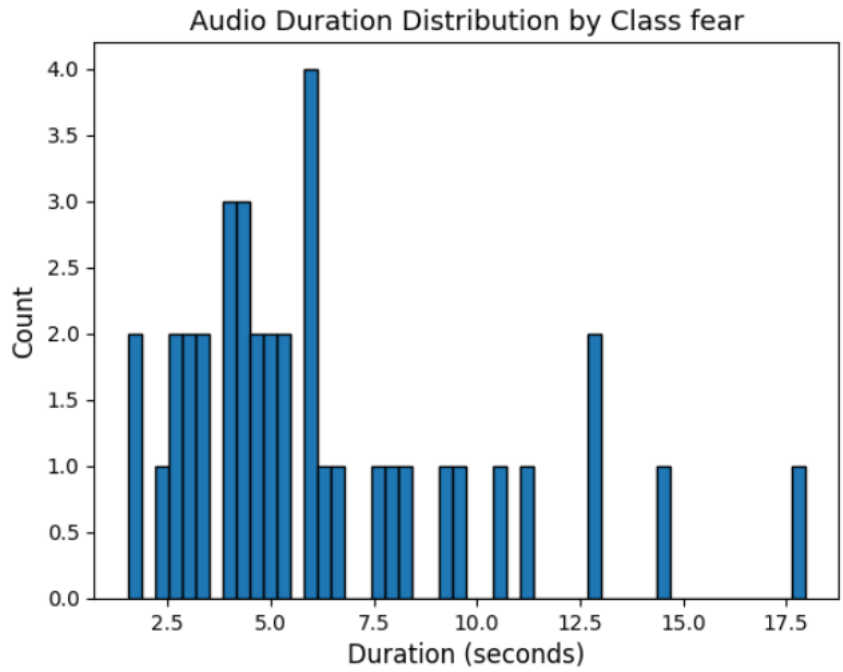

```
Mean Length: 8.03
Standard Deviation of Length: 6.82
```

شکل ۴ – اطلاعات آماری داده ها در همه کلاس ها



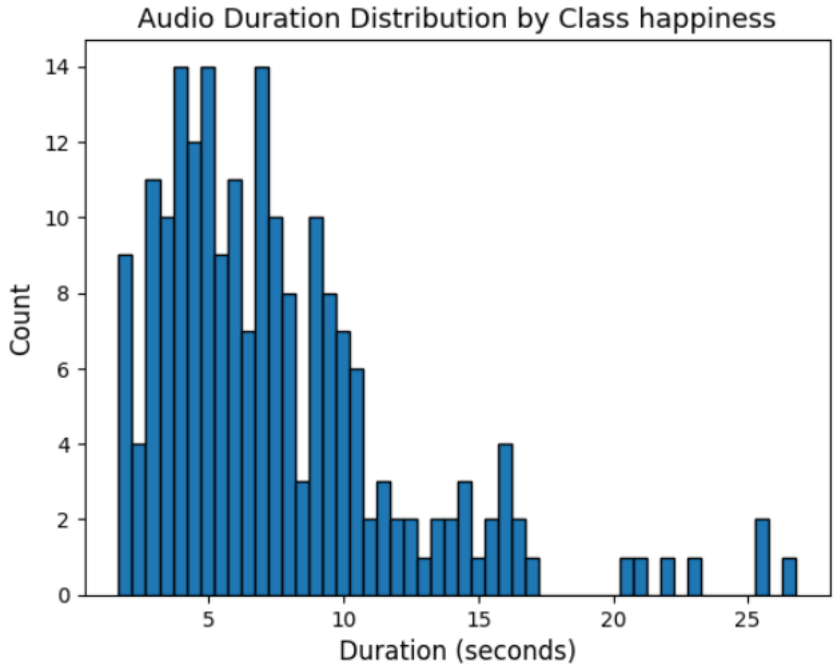

```
Mean Length: 7.19
Standard Deviation of Length: 5.27
```

شکل ۵ – اطلاعات آماری کلاس خشم



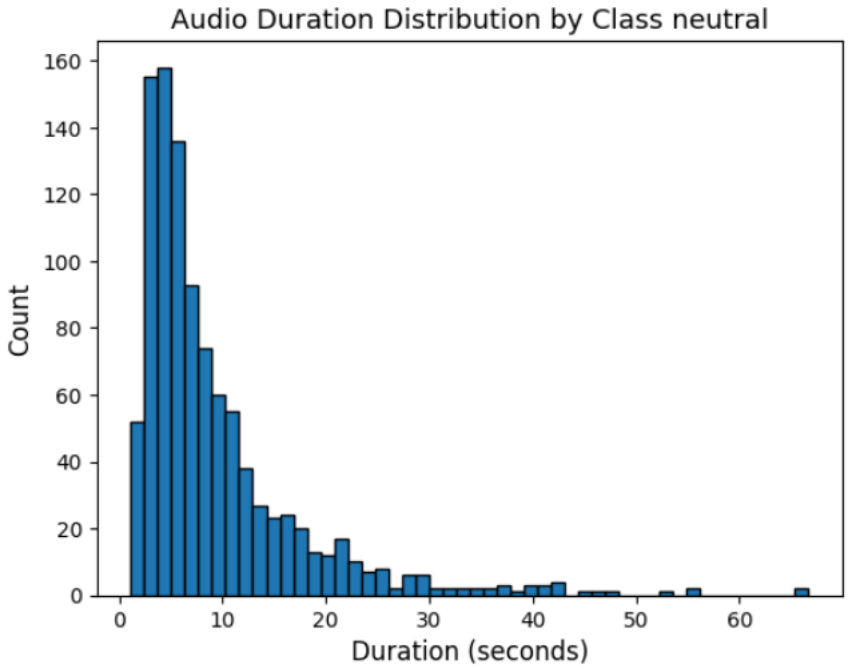

```
Mean Length: 6.28
Standard Deviation of Length: 3.68
```

شکل ۶ – اطلاعات آماری کلاس ترس



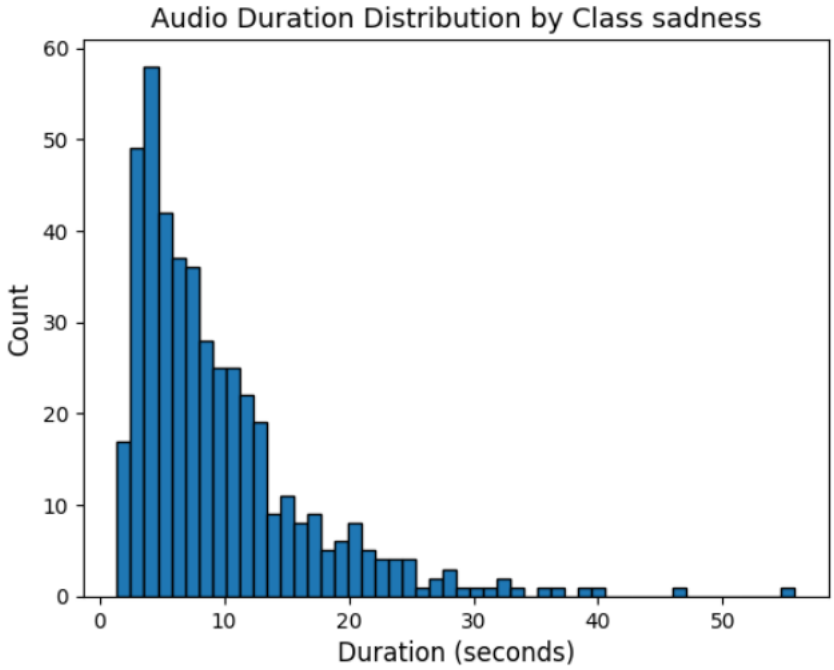

```
Mean Length: 7.59
Standard Deviation of Length: 4.72
```

شکل ۷ – اطلاعات آماری کلاس شادی



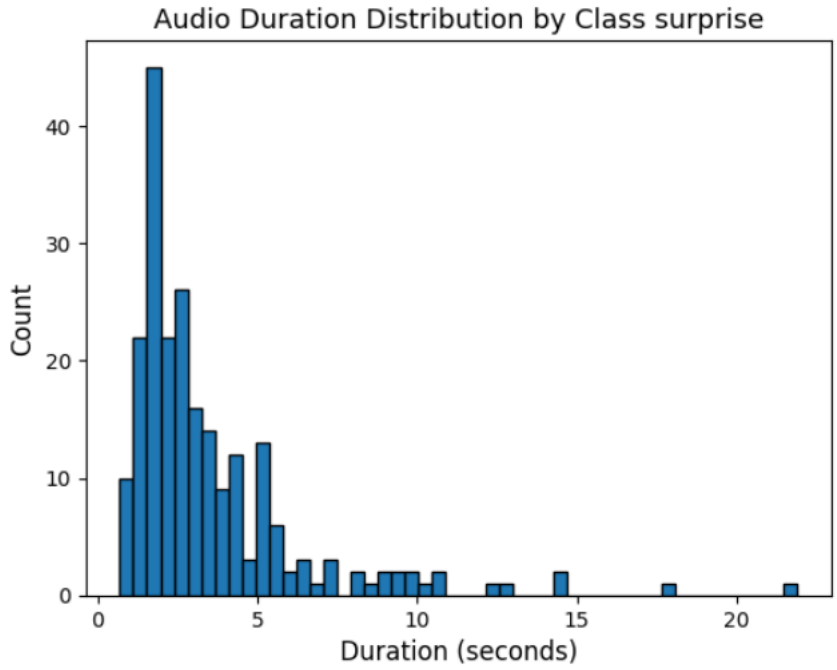

```
Mean Length: 9.34
Standard Deviation of Length: 8.28
```

شکل ۸ - اطلاعات آماری کلاس خنثی




```
Mean Length: 9.63
Standard Deviation of Length: 7.39
```

شکل ۹ - اطلاعات آماری کلاس غم



Mean Length: 3.54
Standard Deviation of Length: 2.92

شکل ۱۰ - اطلاعات آماری کلاس تعجب

در ادامه از ابزار wav2vecFeatureExtractor برای نرمالایز کردن و استخراج ویژگی از داده های صوتی و همچنین padding که در قسمت بعد توضیح داده شده است، استفاده می کنیم. از کتابخانه librosa نیز برای لود کردن داده های صوتی استفاده شده است.

۱-۳-۲. ساخت دیتالودر

مقایسه دو مدل شرح داده شده در صورت پروژه برای padding در زیر آورده شده است.

حالت اول پد کردن تمام فایل‌های صوتی در یک مجموعه داده به طول بلندترین فایل صوتی است.

مزایا:

- پیاده سازی این روش ساده و آسان است.
- تمام فایل‌های صوتی طول یکسان خواهند داشت، که می‌تواند مراحل پردازش بعدی را ساده کند.

معایب:

- پد کردن تمام فایل‌های صوتی به طول بلندترین فایل صوتی می‌تواند حافظه زیادی را به خصوص برای مجموعه داده‌های بزرگ، مصرف کند.

- این روش می‌تواند کارایی کمی داشته باشد زیرا مدل باید صفرهای پد شده را که اطلاعات مفیدی ندارند، پردازش کند.

حالت دوم پد کردن فایل‌های صوتی در یک دسته(batch) به طول بلندترین فایل صوتی در آن دسته است.

مزایا:

- این روش به طور کارآمدتری از حافظه استفاده می‌کند زیرا حداقل میزان پد لازم را فراهم می‌کند.
- این روش این امکان را فراهم می‌کند که دسته‌ها به گونه‌ای دسته بندی شوند که دسته‌هایی با طول مشابه کنار هم قرار گیرند. این می‌تواند فرآیند آموزش را بهینه سازی کند.

معایب:

- پیاده سازی این روش پیچیده‌تر از روش اول است زیرا باید طول بلندترین فایل صوتی را در تمام batch ها محاسبه کرد.
- در این حالت طول ورودی های مدل می‌تواند دسته به دسته متفاوت باشد

به طور خلاصه، در حالی که پد کردن تمام فایل‌های صوتی در یک مجموعه داده به طول بلندترین فایل صوتی ساده‌تر است، پد کردن فایل‌های صوتی در یک دسته به طول بلندترین فایل صوتی در آن دسته می‌تواند کارآمدتر باشد و به حافظه کمتری نیاز داشته باشد.

برای این کار از متد pad در Wav2Vec2FeatureExtractor استفاده کرده ایم. این متد ورودی را پد می‌کند تا همه آن‌ها طول یکسان داشته باشند. این کار زمانی که DataLoader داده ها را به صورت batch در می آورد به طور جداگانه برای داده های هر batch انجام می شود.

۱- ۴. تعریف مسئله

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

همانطور که در صورت سوال مطرح شده است، برخلاف مدل های مبتنی بر ترنسفورمر متنی مثل BERT، HuBERT یک بردار تعبیه که بازنمایی کل دنباله ورودی باشد ایجاد نمی کند و لازم است تا به نحوی این بردار تولید شود. یکی از روش های رایج برای ساخت یک بردار نمایه که کل دنباله ورودی را نمایش می دهد، استفاده ازPooling است. پس از به دست آوردن دنباله حالت های پنهان(hidden_state) ازHuBERT ، می توان یک عملیات pooling را بر روی تمام حالت های پنهان اعمال کرد تا یک بردار

واحد به دست آورد. سپس این بردار می تواند به عنوان ورودی به یک لایه طبقه بندی برای دسته بندی احساس داده صوتی، استفاده شود. در حقیقیت با اعمال pooling می توان اطلاعات را از کل دنباله به یک بردار منفرد تجمیع کرد و در نتیجه می توان از خروجی pooling برای اعمال دسته بندی استفاده کرد. از سه نوع مختلف pooling می توان استفاده کرد:

- Mean Pooling: این روش میانگین مقادیر داده های ورودی را محاسبه می کند.
- Max Pooling: این روش بیشترین مقدار را از داده های ورودی انتخاب می کند. این روش نمایشی از برجسته ترین ویژگی در دنباله را به ما می دهد.
- Attention Pooling: این روش مجموع وزن دار حالت های پنهان را محاسبه می کند.

در این تمرین ما از mean pooling استفاده کرده ایم زیرا باعث کاهش تعداد پارامترهایی که باید یاد گرفته شوند، می شود و در نتیجه بار محاسباتی در مدل را کاهش می دهد.

```
def flattened_states(self, hidden_states):
    return torch.mean(hidden_states, dim=1)
```

شکل ۱۱ – کد ایجاد بردار تعبیه

۱-۲-۲. آموزش مدل

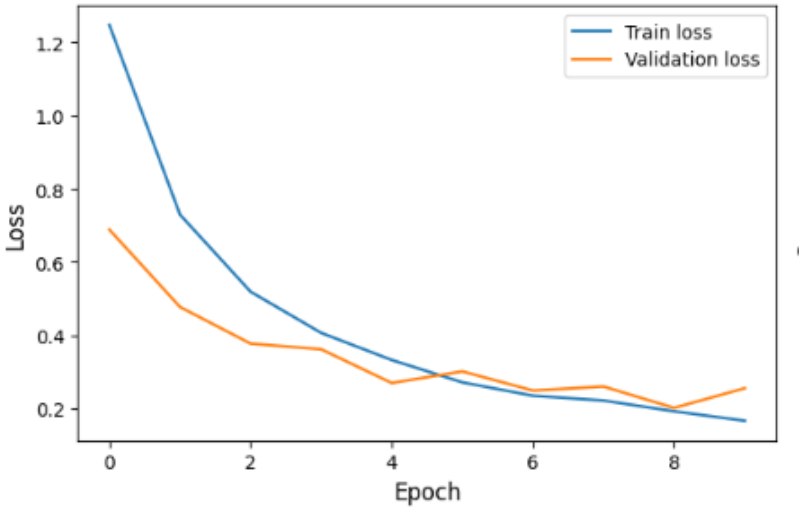
جدول زیر حاوی مقادیر انتخاب شده برای پارامترهای آموزش مدل است. مقادیر betas در optimizer مطابق با مقادیر گزارش شده در مقاله HuBERT قرار داده شده است. همچنین طبق راهنمایی برای تسریع در فرایند آموزش در صورت سوال feature_extractor فریز شده است.

جدول ۱ – پارامتر های مدل

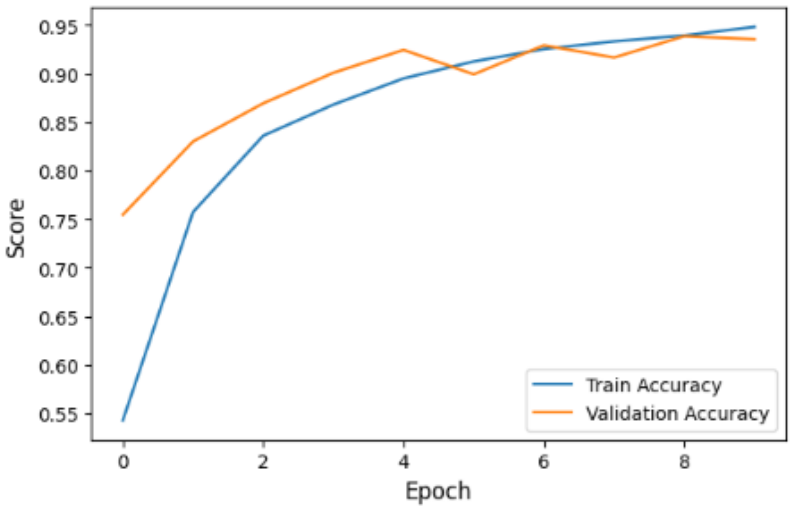
پارامتر	مقدار
Learning Rate	**1e-5**
Sampling Rate	**16kHz**
# Train Samples	**5083**
# Test Samples	**635**
# Validation Samples	**636**
Batch Size	**2**
# Train Batches	**2545**

# Test Batches	**318**
# Validation Batches	**318**
# Epochs	**10**
Optimizer	**AdamW**
Loss Function	**Cross Entropy**

شکل های زیر نشان دهنده دقت و loss در هر ایپاک برای داده های آموزش و ارزیابی است. همان طور که در نمودار دقت و loss داده آموزش مشخص است، دقت و loss هم در داده های آموزش با شیب صعودی و به صورت مستمر به ترتیب افزایش و کاهش می یابد. نمودار دقت و loss داده ارزیابی نیز با همین روند اما با سرعت کمتری در مقایسه با داده های آموزش به ترتیب افزایش و کاهش می یابد.



شکل ۱۲ – نمودار **loss** برای داده های آموزش و ارزیابی

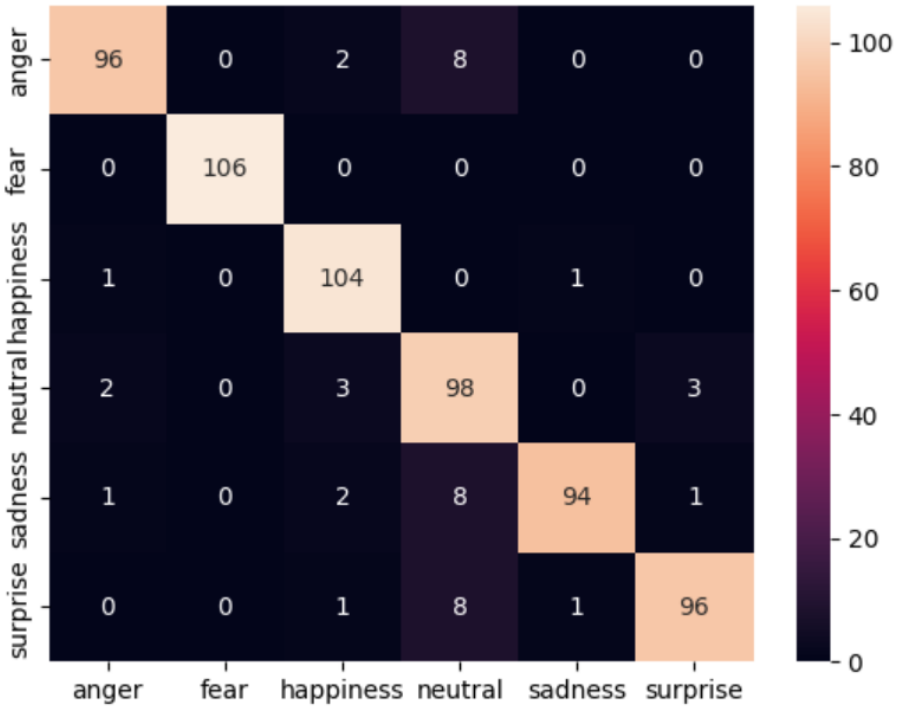


شکل ۱۳ - نمودار دقت برای داده های آموزش و ارزیابی

مقدار دقت و loss در ایپاک نهایی به شکل زیر است.

```
Training Epoch = 10 -> Loss = 0.1666, Accuracy = 0.9483
Validation Epoch = 10 -> Loss = 0.2554, Accuracy = 0.9355
```

شکل زیر نشان دهنده طبقه بندی پیش بینی شده بر روی داد تست است. همچنین مقدار دقت بطور کلی و برای هر کلاس نیز آورده شده است.



شکل ۱۴ – طبقه بندی پیش بینی شده داده تست

```
Overall Test Results -> Loss = 0.2212, Accuracy = 0.9340
Anger Accuracy = 0.9056603773584906
Fear Accuracy = 1.0
Happiness Accuracy = 0.9811320754716981
Neutral Accuracy = 0.9245283018867925
Sadness Accuracy = 0.8867924528301887
Surprise Accuracy = 0.9056603773584906
```

جدول زیر نیز نشان دهنده نتایج در ماتریس درهم ریختگی است.

جدول ۲ – جدول ماتریس درهم ریختگی

	precision	recall	f1-score	support
anger	0.96	0.91	0.93	106
fear	1.00	1.00	1.00	106
happiness	0.93	0.98	0.95	106
neutral	0.80	0.92	0.86	106
sadness	0.98	0.89	0.93	106
surprise	0.96	0.91	0.93	106
accuracy			0.93	636
macro avg	0.94	0.93	0.93	636
weighted avg	0.94	0.93	0.93	636

با توجه به نتایج می توان دریافت که بطور کلی کلاس ترس دارای بهترین و کلاس خنثی و غم دارای بدترین نتایج است. این نتایج برای کلاس خنثی به این دلیل است که همانطور که در شکل ۱۴ نیز مشخص است، ۸ داده از داده های هر کلاس غم، عصبانیت و تعجب به اشتباه در کلاس خنثی بیش بینی شده اند. کلاس غم نیز دارای کمترین تعداد داده پیش بینی شده درست است. در مقابل تمام داده های کلاس ترس به درستی پیش بینی شده اند.

پاسخ ۲ – تنظیم دقیق مدل BERT

در این بخش می‌خواهیم با استفاده از مدل ParsBERT که مدلی برگرفته شده از معماری BERT برای استفاده بر روی زبان فارسی است، یک تسک پردازش زبان طبیعی classification بر روی مجموعه داده فارسی FarsTail را انجام با استفاده از PyTorch و ابزارهای موجود انجام دهیم.

```
class BaseLineModel(nn.Module):
    def __init__(self, model_name, num_classes, num_layers_to_keep=12):
        super(BaseLineModel, self).__init__()
        self.config = AutoConfig.from_pretrained(model_name)
        self.config.num_labels = num_classes
        self.config.num_hidden_layers = num_layers_to_keep
        self.bert = AutoModel.from_pretrained(model_name, config=self.config)
        self.dropout = nn.Dropout(self.config.hidden_dropout_prob)
        self.classifier = nn.Linear(self.config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask, token_type_ids):
        outputs = self.bert(input_ids, attention_mask, token_type_ids)
        pooled_output = outputs[1]
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        return logits
```

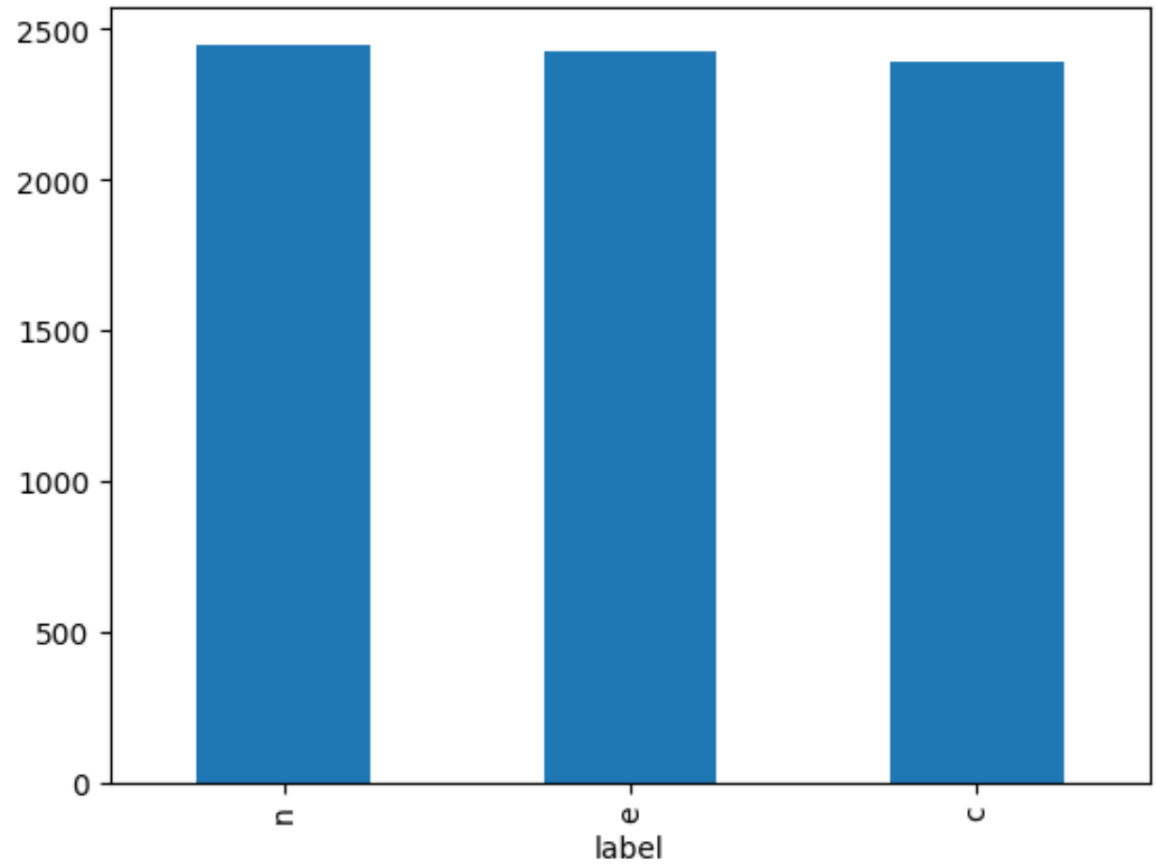
شکل ۱۵ - معماری کلی مدل استفاده شده

۲-۱. پیش‌پردازش داده‌ها

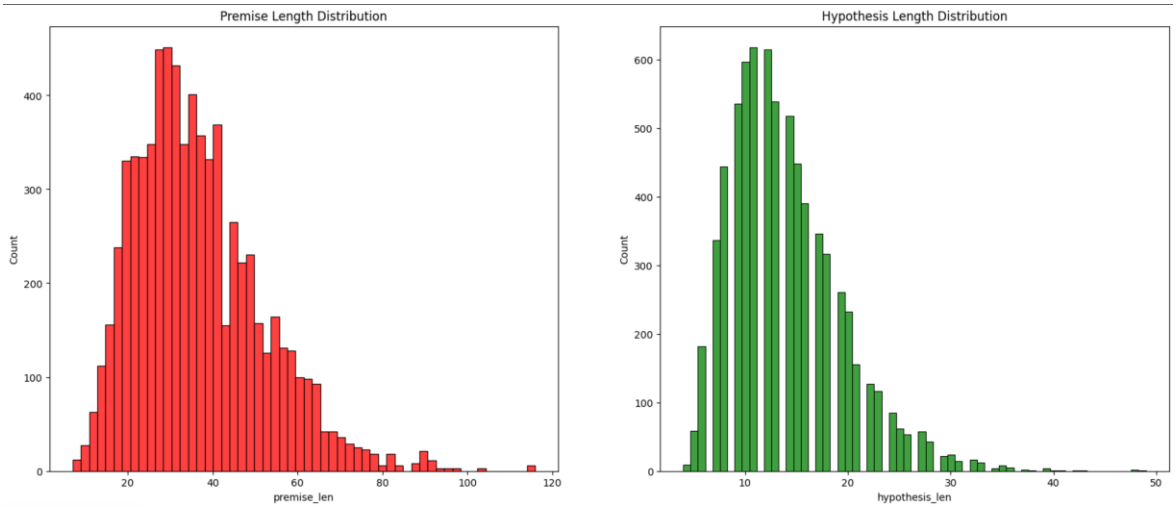
در این پروژه از مجموعه داده FarsTail استفاده شده است. این مجموعه داده مجموعه داده‌ای به زبان فارسی است. هر رکورد این مجموعه داده شامل ۳ بخش است، Premise یا همان محتوای اصلی ما، Hypothesis یا همان فرضیه که جمله‌ای است که می‌تواند از Premise برداشت شود و در نهایت label که می‌تواند e به معنای Entailment یا موافق و هم‌سو، c به معنای Contradiction یا متضاد و خلاف جهت و یا n به نشانه Neutral یا خنثی باشد. در واقع label ما در این مجموعه داده نشان‌دهنده ارتباط بین premise و hypothesis ما است.

برای پیش‌پردازش داده‌ها در این بخش از کتابخانه hazm که کتابخانه‌ای مبتنی بر NLTK برای زبان فارسی است استفاده شده است و مواردی مثل حذف موارد نامطلوب مثل لینک‌ها، تگ‌های HTML، علائم نگارشی، نرمال کردن عبارات و فواصل و lemmatization و stemming روی داده‌های ورودی و بخش

premise و hypothesis مجموعه داده اعمال شد. همچنین label های مجموعه داده هم به صورت one-hot انکود شدند.



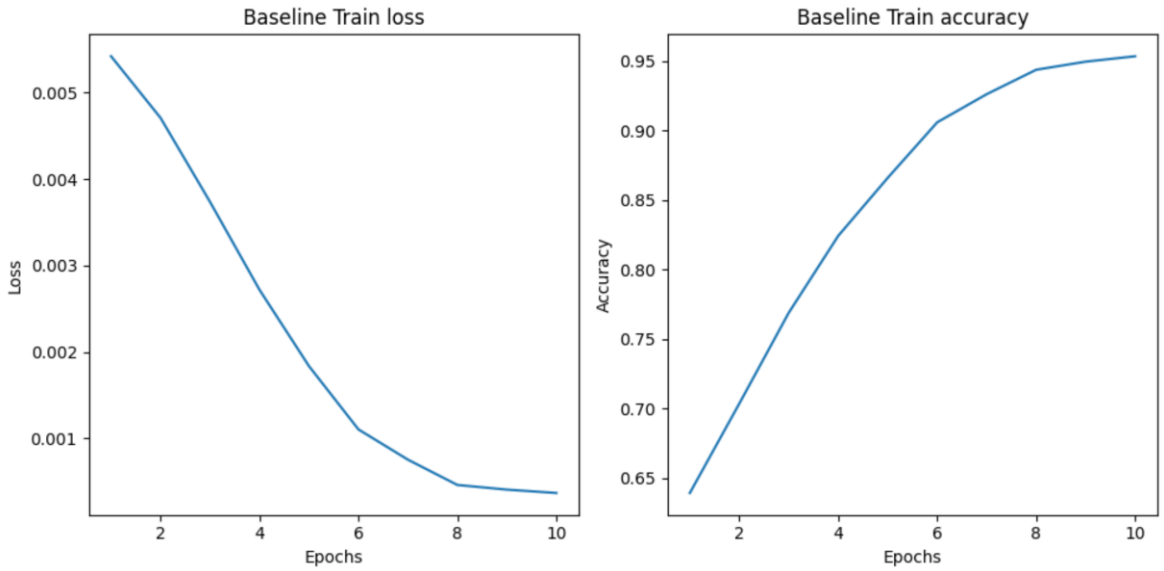
شکل ۱۶ - توزیع کلاس‌های متفاوت در مجموعه داده آموزش



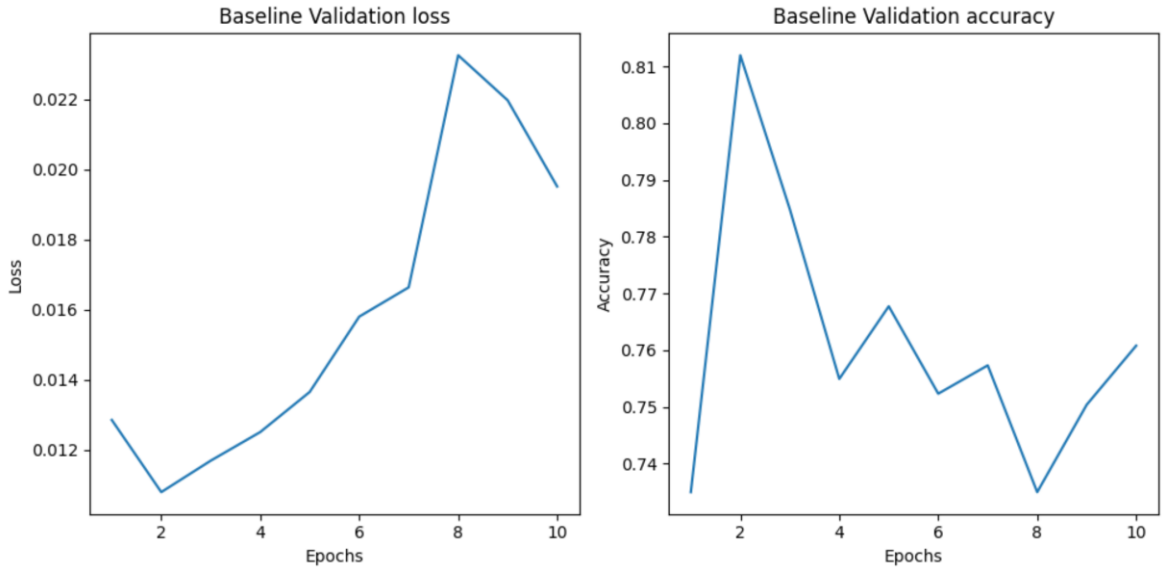
شکل ۱۷ - نمودار توزیع طول عبارات مجموعه داده آموزش

۲-۲. تنظیم دقیق مدل

در این بخش مدل ParsBERT را با استفاده از مجموعه داده آموزش FarsTail فاین-تیون می‌کنیم. مراحل و پیاده‌سازی این بخش در jupyter notebook مربوطه موجود است و در این بخش صرفا به بیان نتایج مدل و عملکرد آن حین فرایند Fine-Tuning می‌پردازیم. در این مدل در حین فرایند آموزش بر روی داده ارزیابی به دقت ۸۰ درصد می‌رسد.



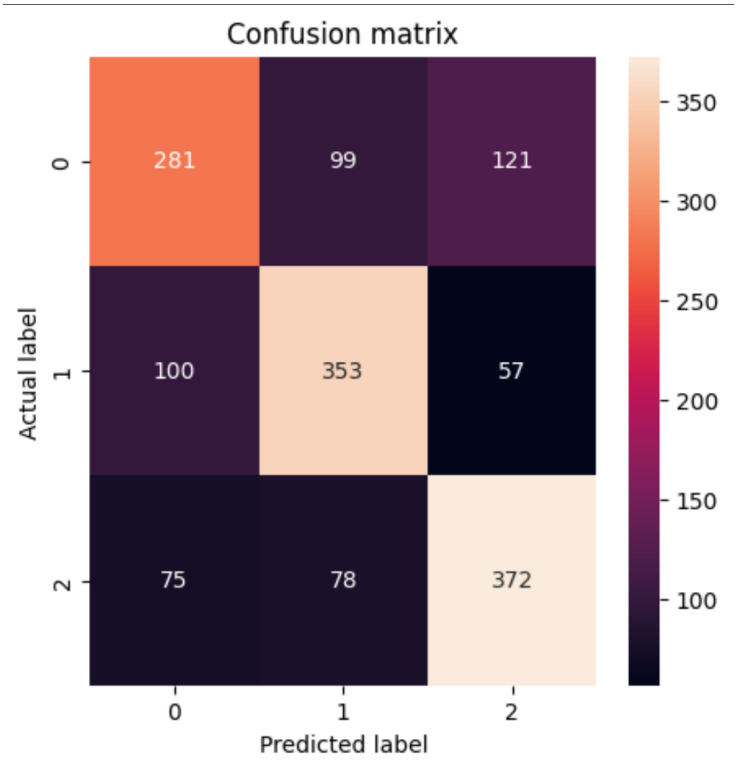
شکل ۱۸ - نمودار دقت و loss روی داده آموزش در حین آموزش



شکل ۱۹ - نمودار دقت و loss روی داده ارزیابی حین آموزش

	precision	recall	f1-score	support
c	0.62	0.56	0.59	501
e	0.67	0.69	0.68	510
n	0.68	0.71	0.69	525
accuracy			0.65	1536
macro avg	0.65	0.65	0.65	1536
weighted avg	0.65	0.65	0.65	1536

شکل ۲۰ - نتایج مدل بر روی داده‌های تست



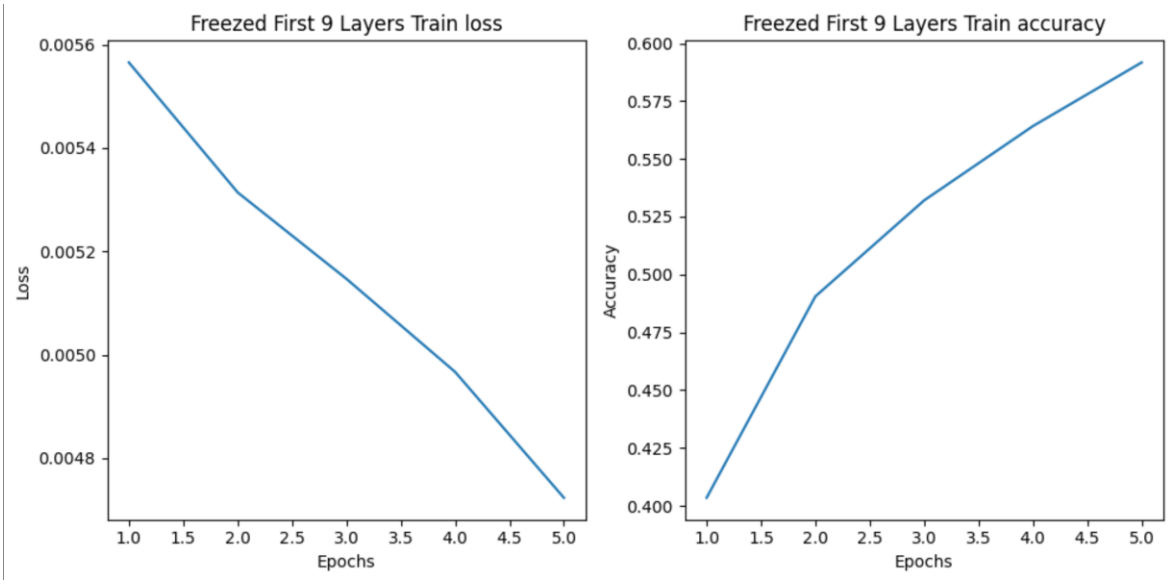
شکل ۲۱ - ماتریس آشفتگی مدل

۲-۳. فریز کردن لایه‌های مدل

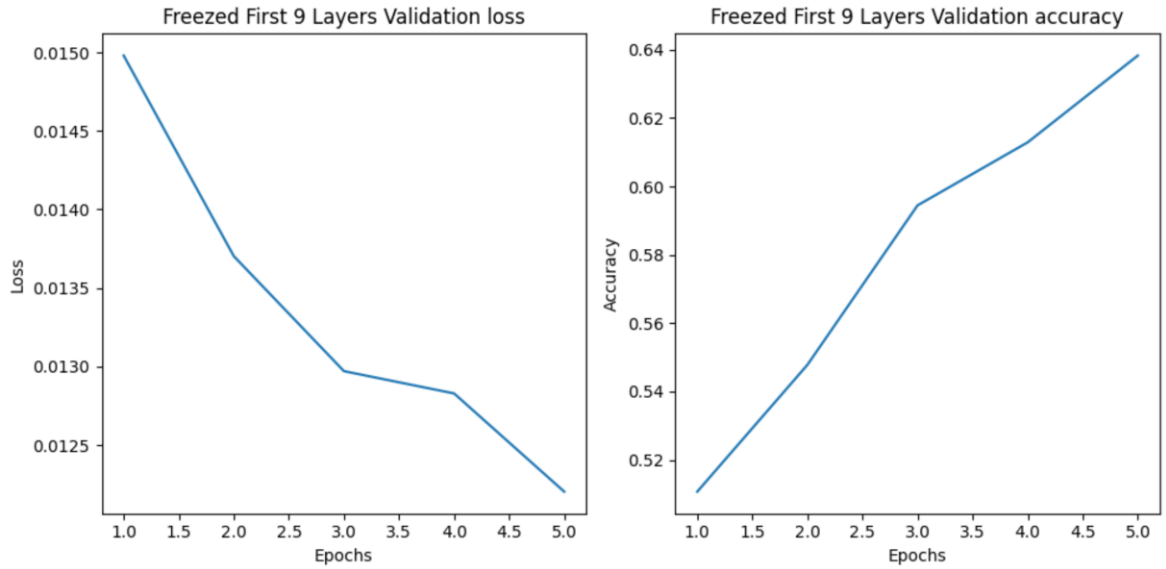
در این بخش به freeze کردن برخی از لایه‌های مدل و سپس fine-tune کردن آن می‌پردازیم. این freeze کردن به دو روش در این بخش انجام شده است.

۲-۳-۱. فریز کردن ۹ لایه ابتدایی

در این بخش طبق گفته مقاله ذکر شده، بخش embedding مدل ParsBERT به همراه ۹ لایه ابتدایی بخش encoder این مدل freeze شده و در حین فرایند fine-tuning مقادیر وزن‌های آن‌ها آپدیت نشد. نحوه پیاده‌سازی این بخش در نوت‌بوک آورده شده است و در اینجا صرفا نتایج حاصل ذکر می‌شوند.



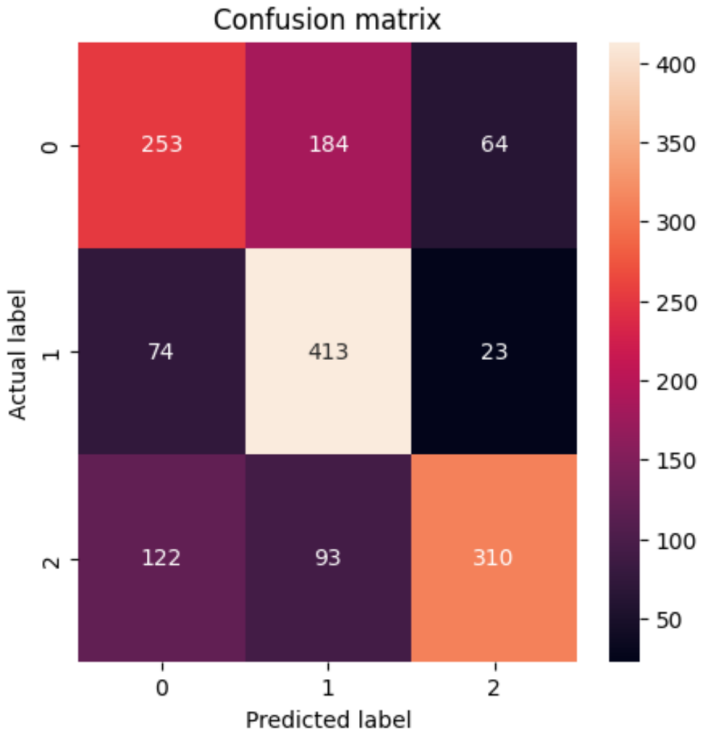
شکل ۲۲ - نمودار دقت و loss روی داده آموزش در حین آموزش



شکل ۲۳ - نمودار دقت و loss روی داده ارزیابی در حین آموزش

	precision	recall	f1-score	support
c	0.56	0.50	0.53	501
e	0.60	0.81	0.69	510
n	0.78	0.59	0.67	525
accuracy			0.64	1536
macro avg	0.65	0.64	0.63	1536
weighted avg	0.65	0.64	0.63	1536

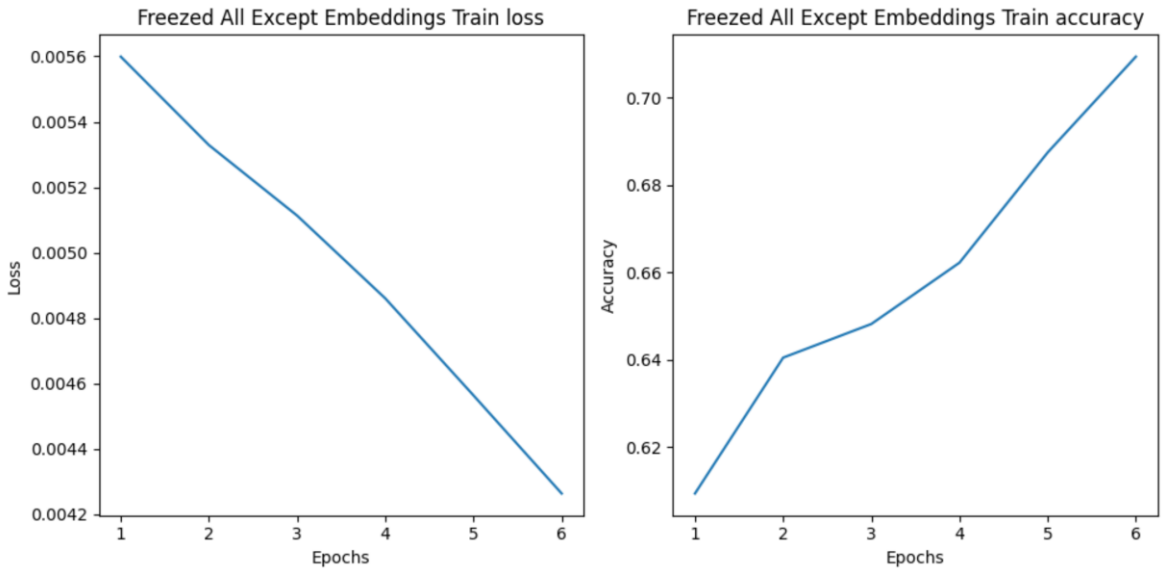
شکل ۲۴ - نتایج مدل بر روی داده‌های تست



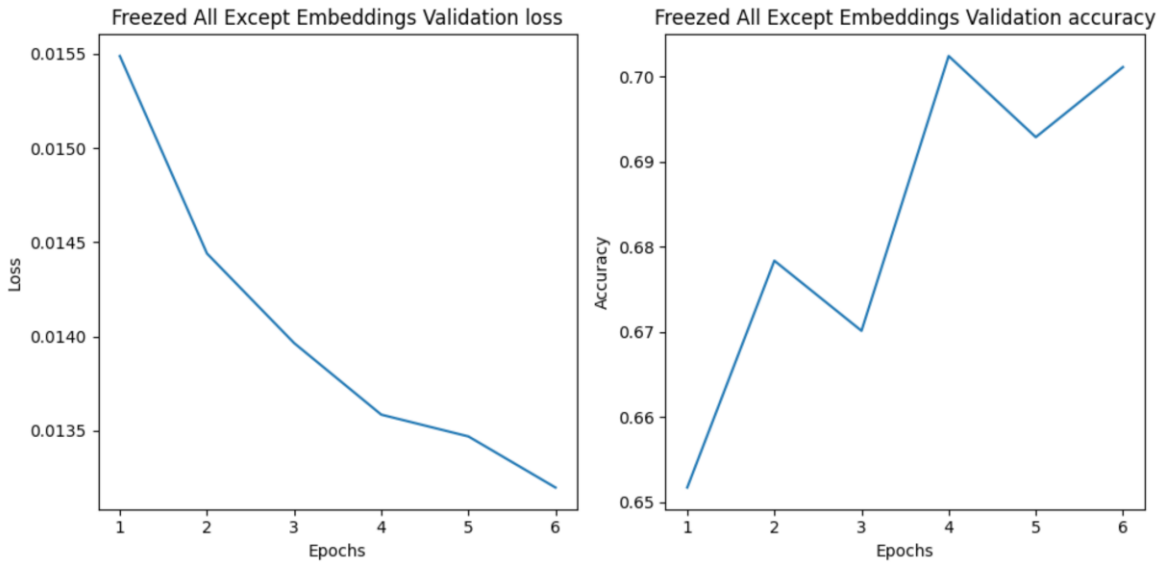
شکل ۲۵ - ماتریس آشفتگی مدل

۲-۳-۲. فریز کردن تمام لایه‌ها به جز لایه آخر

در این بخش طبق گفته صورت سوال، تمام لایه‌های بخش encoder غیر از لایه آخر و دوازدهم آن freeze شده و در حین فرایند fine-tuning مقادیر وزن‌های آن‌ها آپدیت نشد. نحوه پیاده‌سازی این بخش در نوت‌بوک آورده شده است و در اینجا صرفا نتایج حاصل ذکر می‌شوند.



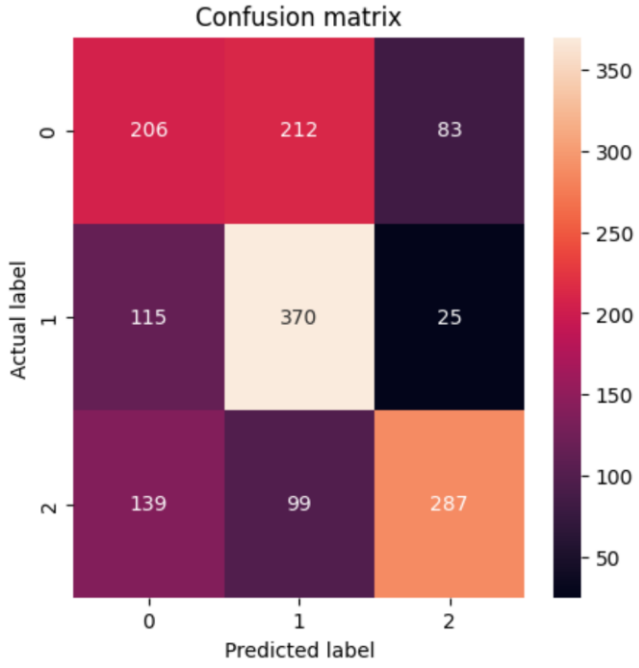
شکل ۲۶ - نمودار دقت و loss روی داده آموزش در حین آموزش



شکل ۲۷ - نمودار دقت و loss روی داده آموزش در حین آموزش

	precision	recall	f1-score	support
c	0.45	0.41	0.43	501
e	0.54	0.73	0.62	510
n	0.73	0.55	0.62	525
accuracy			0.56	1536
macro avg	0.57	0.56	0.56	1536
weighted avg	0.57	0.56	0.56	1536

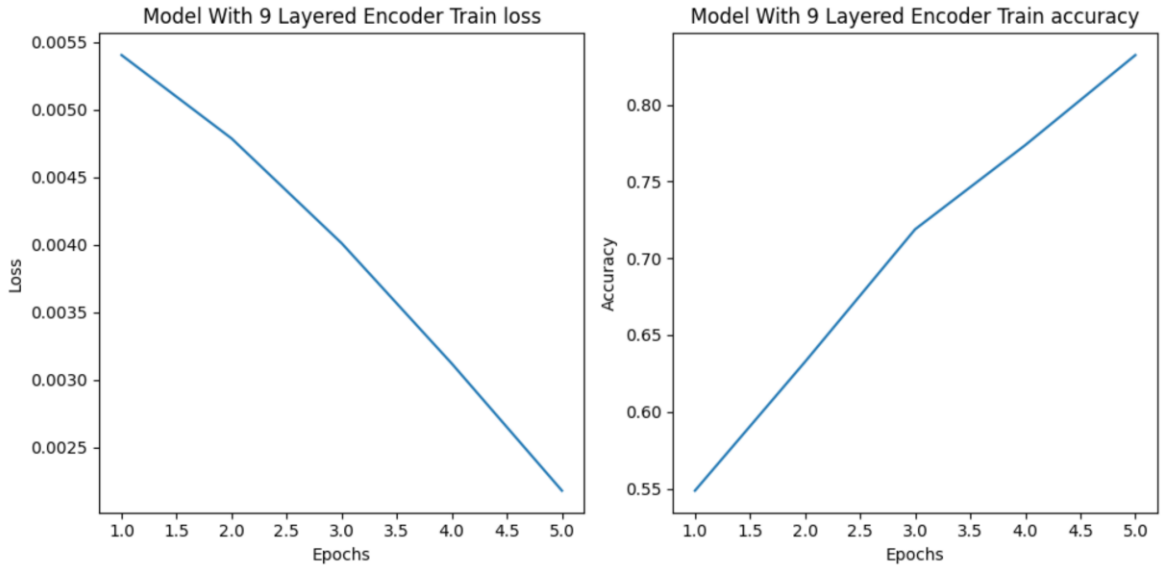
شکل ۲۸ - نتایج مدل بر روی داده‌های تست



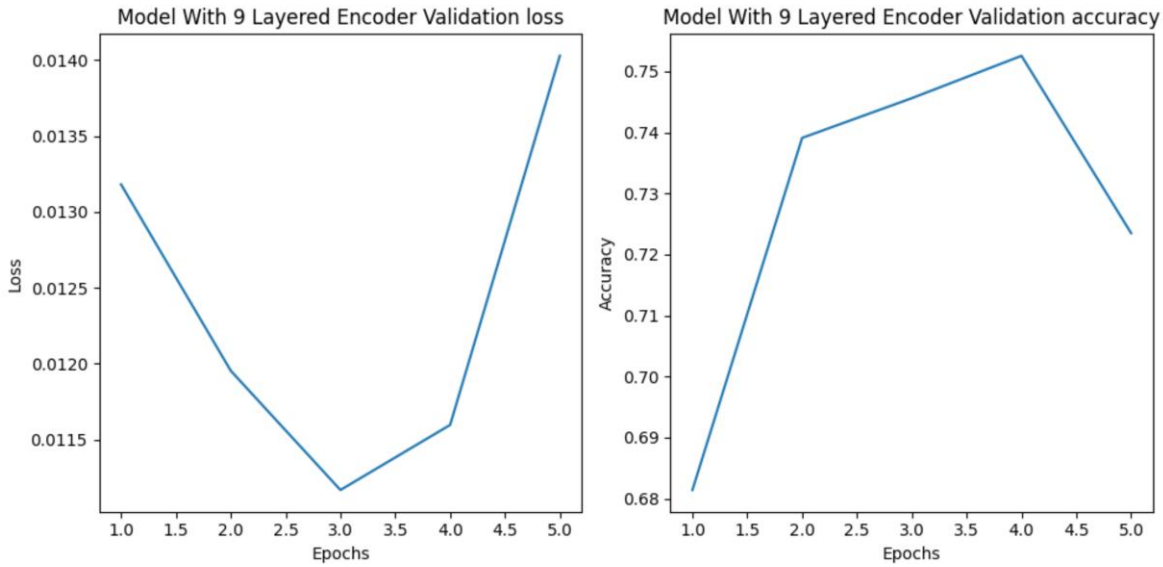
شکل ۲۹ - ماتریس آشفتگی مدل

۲-۴. تنظیم دقیق مدل بر روی لایه‌های میانی

در این بخش به منظور بررسی تاثیر عمق بخش encoder شبکه BERT بر روی عملکرد آن، این بخش را تغییر داده و صرفا ۹ لایه ابتدایی آن را نگه داشته و بقیه لایه‌ها یعنی لایه ۱۰ تا ۱۲ بخش encoder از معماری شبکه حذف شدند. نحوه پیاده‌سازی این بخش در نوت‌بوک آورده شده است و در اینجا صرفا نتایج حاصل ذکر می‌شوند. دقت مدل در این بخش بر روی داده‌های ارزیابی در حین فرایند آموزش به ۷۵ درصد نیز رسیده است.



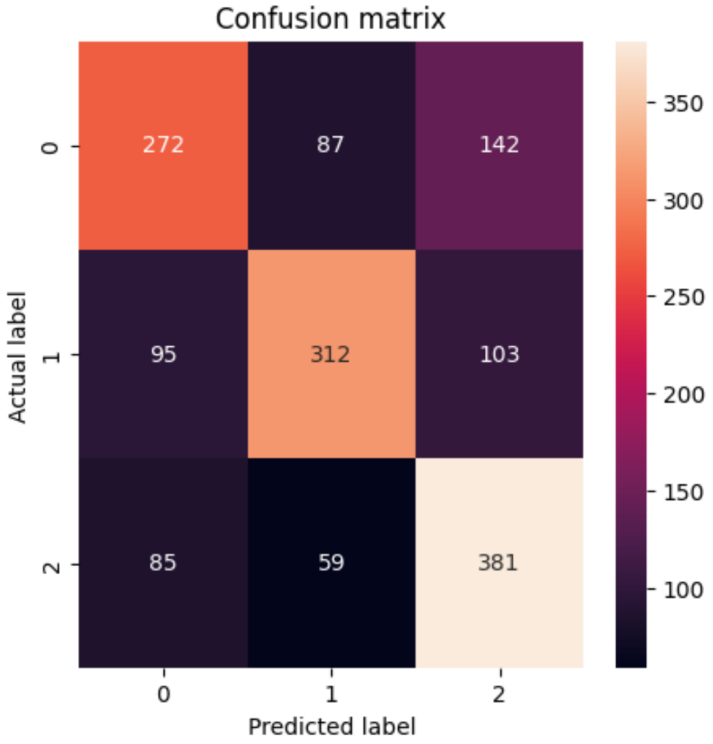
شکل ۳۰ - نمودار دقت و loss روی داده آموزش در حین آموزش



شکل ۳۱ - نمودار دقت و loss روی داده ارزیابی در حین آموزش

	precision	recall	f1-score	support
c	0.60	0.54	0.57	501
e	0.68	0.61	0.64	510
n	0.61	0.73	0.66	525
accuracy			0.63	1536
macro avg	0.63	0.63	0.63	1536
weighted avg	0.63	0.63	0.63	1536

شکل ۳۲ - نتایج مدل بر روی داده‌های تست



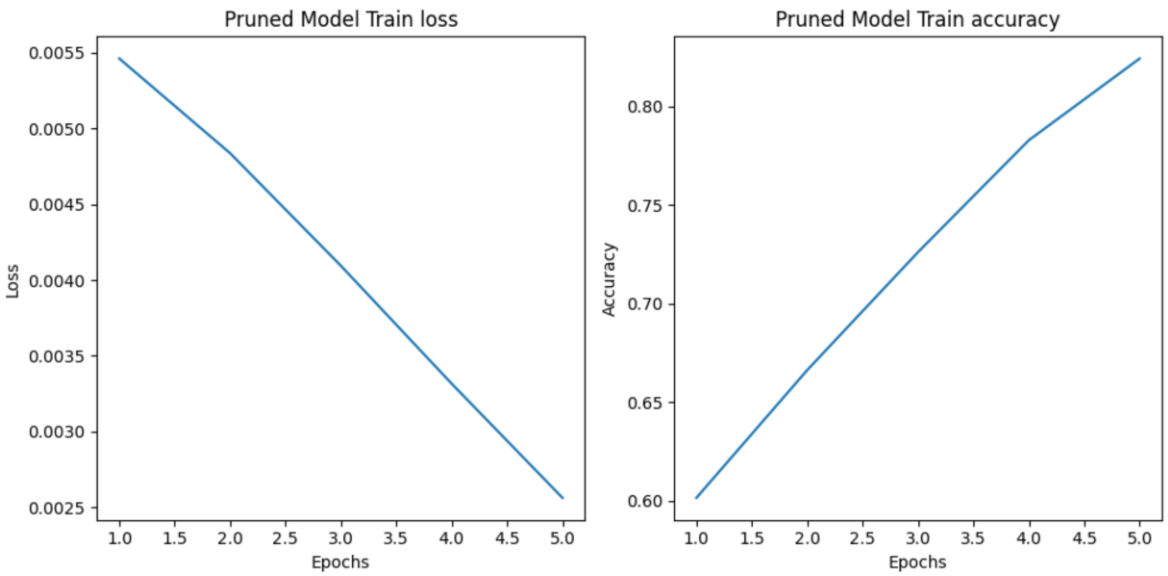
شکل ۳۳ - ماتریس آشفتگی مدل

۲-۵. حذف head های attention در مدل

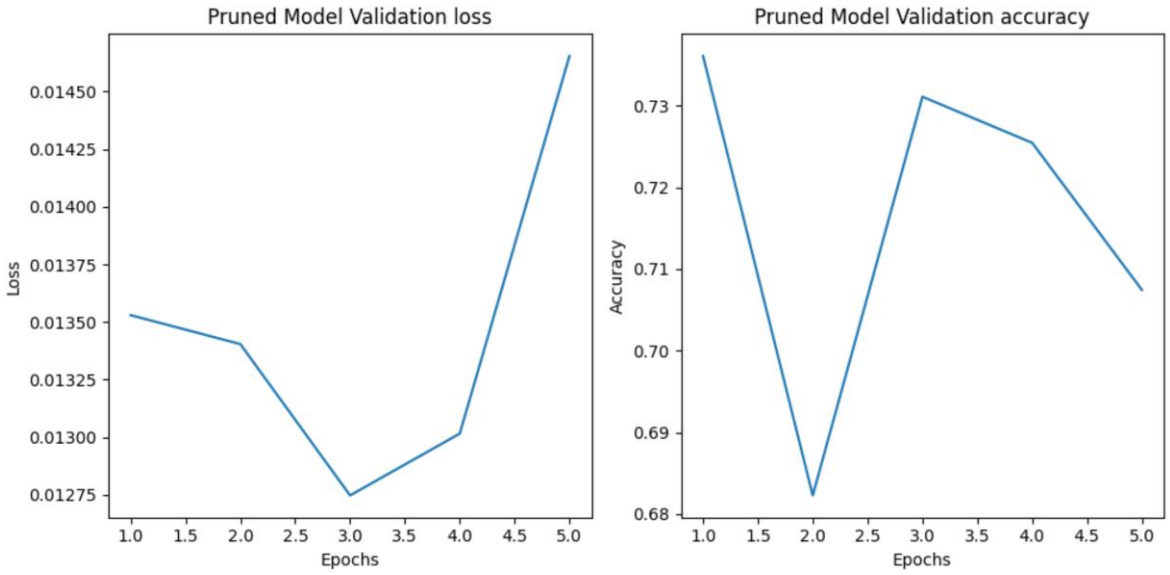
در این بخش طبق مقاله Are Sixteen Heads Really Better than One? و بررسی به عمل آمده توسط آن‌ها بر روی تاثیر تعداد attention head های لایه‌های encoder بر روی عملکرد نهایی مدل، نیمی از attention head های مدل به صورت تصادفی حذف شده و نتایج و عملکرد مدل پس از fine-tune کردن دوباره آن به دست آمد که به شرح زیر هستند. این مدل prune شده در حین فرایند fine-tune شدن روی داده ارزیابی به دقت حدود ۷۴ درصد دست پیدا می‌کند.

```python
class PrunedModel(nn.Module):
    def __init__(self, model_name, num_classes, num_layers_to_keep=12, prune_heads_percent=0.0):
        super(PrunedModel, self).__init__()
        self.config = AutoConfig.from_pretrained(model_name)
        self.config.num_labels = num_classes
        self.config.num_hidden_layers = num_layers_to_keep
        self.bert = AutoModel.from_pretrained(model_name, config=self.config)
        self.dropout = nn.Dropout(self.config.hidden_dropout_prob)
        self.classifier = nn.Linear(self.config.hidden_size, num_classes)

        self.prune_heads_dict = {}
        for i in range(num_layers_to_keep):
            num_heads = self.bert.encoder.layer[i].attention.self.num_attention_heads
            num_pruned_heads = int(num_heads * prune_heads_percent)
            pruned_heads = np.random.choice(num_heads, num_pruned_heads, replace=False)
            self.prune_heads_dict[i] = pruned_heads
        self.bert.prune_heads(self.prune_heads_dict)

    def forward(self, input_ids, attention_mask, token_type_ids):
        outputs = self.bert(input_ids, attention_mask, token_type_ids)
        pooled_output = outputs[1]
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        return logits
```

شکل ۳۴ - معماری مدل prune شده



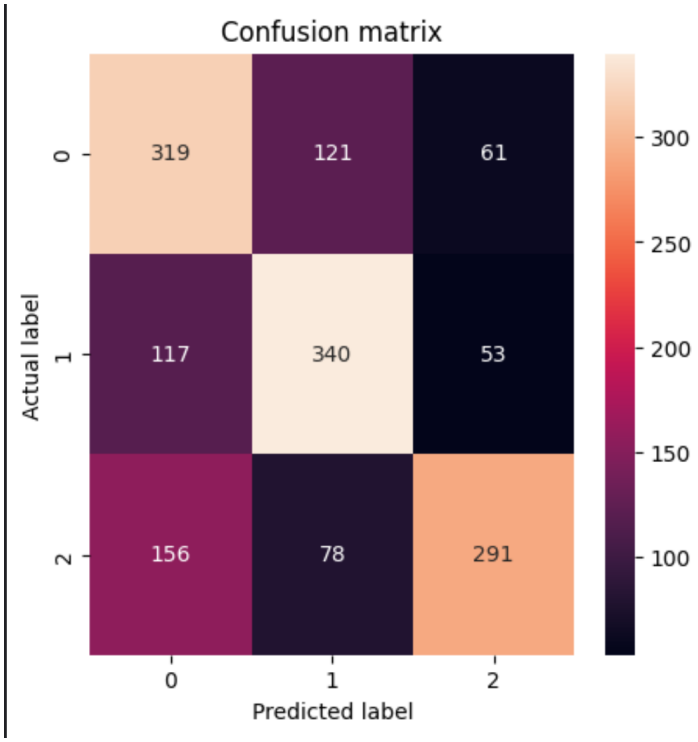
شکل ۳۵ - نمودار دقت و loss روی داده آموزش در حین آموزش



شکل ۳۶ - نمودار دقت و loss روی داده ارزیابی در حین آموزش

	precision	recall	f1-score	support
c	0.54	0.64	0.58	501
e	0.63	0.67	0.65	510
n	0.72	0.55	0.63	525
accuracy			0.62	1536
macro avg	0.63	0.62	0.62	1536
weighted avg	0.63	0.62	0.62	1536

شکل ۳۷ - نتایج مدل بر روی داده‌های تست



شکل ۳۸ - ماتریس آشفتگی مدل

همانطور که در جدول زیر هم مشاهده می‌کنیم، نتایج نهایی به دست آمده مطابق با انتظار ما از مدل‌ها و منطبق بر نتایج مقالات مرتبط با آن‌ها بوده است. Fine-Tune کردن مدل ParsBERT بر روی مجموعه داده FarsTail توانسته است که دقت قابل قبولی در مواجهه با مجموعه داده تست این دیتاست به ما ارائه دهد.

با freeze کردن ۹ لایه ابتدایی مدل، مدل همچنان توانایی بالایی برای یاد گرفتن ویژگی‌ها و در نتیجه طبقه‌بندی مناسب داده‌ها را دارد و با آموزش یافتن بقیه بخش‌های مدل، با هزینه محاسباتی بسیار کمتری نسبت به مدل اولیه (آموزش یافتن ۲۲ ملیون پارامتر به جای ۱۶۲ ملیون پارامتر) به نتیجه بسیار نزدیکی نسبت به مدل اولیه دست پیدا می‌کنیم.

اما با freeze کردن ۱۱ لایه ابتدایی دیگر این اتفاق نمی‌افتد و مدل با کاهش توانایی در عملکرد به دلیل عدم امکان آپدیت کردن مناسب وزن‌های خود مواجه می‌شود و ما شاهد افت عملکرد مدل نسبت به مدل پایه خود هستیم.

در بخش بعد با نگه داشتن تنها ۹ لایه در بخش encoder مدل ParsBERT به جای ۱۲ لایه، باز هم شاهد افت بسیار خفیف عملکرد مدل هستیم که این نتیجه انتظار می‌رفت، اما همانطور که می‌بینیم، این افت عملکرد فاحش نیست و در واقع مدل با همین ۹ لایه هم با داشتن پیچیدگی و پارامترهای کمتر نسبت به مدل اولیه می‌تواند به طور مناسبی عمل کند.

در نهایت هم نتایج مربوط به مدل prune شده که نیمی از attention head های آن به صورت تصادفی حذف شده‌اند را مشاهده می‌کنیم. در این بخش هم شاهد افت عملکرد مدل نسبت به مدل پایه هستیم اما باز هم مثل بخش قبل این تفاوت عملکرد آنچنان شدید نیست که این نتیجه نیز مطابق با نتایج مقاله Are Sixteen Heads Really Better than One? است. از طرفی مدل این بخش در صورت انتخاب نشدن تصادفی attention head ها و استفاده از روش‌های دیگر و حذف attention head های کم‌اهمیت‌تر، می‌تواند عملکرد بهتری از خود بر جای بگذارد و در واقع روش prune کردن در نتایج این بخش می‌تواند موثر واقع شود.

جدول ۳ – نتایج تجمعی مدل‌های پیاده‌سازی شده

	Accuacy	Precision	Recall	F1-Score
Baseline Model	0.65	0.65	0.65	0.65
Embedding + First 9 Layers Frozen	0.64	0.65	0.64	0.63

11 Layers Frozen	0.56	0.57	0.56	0.56
9 Layered Encoder	0.63	0.63	0.63	0.63
Pruned Half Attention Heads	0.62	0.63	0.62	0.62