به نام خدا

دانشگاه تهران

دانشکده مهندسی

برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین پنجم

e

نام و نام خانوادگی	نام نام خانوادگی – نام نام خانوادگی
شماره دانشجویی	810199357 - 810199395
تاریخ ارسال گزارش	1402.09.29

فهرست

شکل‌ها

جدول

[models test accuracy 2.1]

پاسخ ۱-۲ .سوالات تشریحی مربوط به Hubert

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

چالش های اصلی صدا برای self supervised learning

۱. در هر ادای کلمه چندین واحد صدا وجود دارد.

۲. طبقه بندی و lexicon مناسبی از واحد های صدا و معنی های هر یک از آن ها در حین آموزش وجود ندارد.

۳. واحد های صدایی طول هایی متفاوت و متغیر و با تکه بندی های نامشخص خواهند داشت .

پاسخ ۱-۲-۲ رویکرد Hubert

1. Hubert ابتدا از یک offline clustering استفاده می کند تا قسمت های متفاوت speech که شبیه هم دیگر هستند را cluster و برای آن ها target label تولید کند به این مرحله Target label offline clustering می گویند که از K-Means استفاده می کند.
2. Loss مدل Loss خاصی است به شکلی که فقط روی قسمت های masked input loss را محاسبه می کند این قسمت شبیه BERT است . این Mask باعث می شود که اساسا مدل بتواند Rule ها و قواعد زبان و Context مرتبط با آن زبان خاص را یادبگیرد
3. قسمت Teacher Network در واقع باید بیشتر label های consistent تولید کند تا اینکه target label های بهتری را generate کند و این امر بیشتر برای Hubert مهم است تا دقت و نزدیکی Target Labels به واقعیت.

Hubert از معماری اصلی bert که برای پردازش کلمات tokenize شده استفاده می شود استفاده می کند . با این تفاوت که روی داده های صدا process شده و تبدیل شده MFCC با درشت دانگی های متفاوت کار می کند و همچین ورودی هایی را برای بخش لایه Bert تولید می کند .

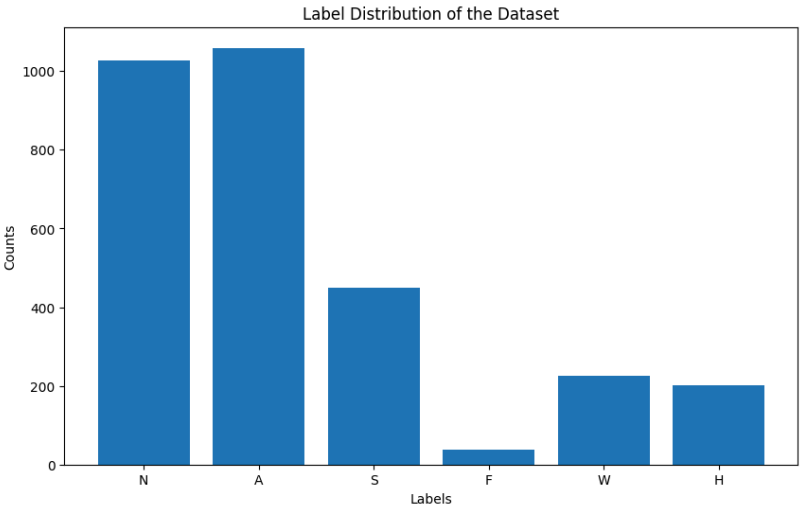
لازم به ذکر است که Hubert دقیقا از مدل Bert استفاده نمی کند بلکه قسمت ها و Component های اصلی Bert رو به ارث برده است(استفاده کردن از mask برای اینکه model generalize کند و یا اساسا استفاده کردن از یک مدل Transformer based مانند Bert) در نتیجه این سوال که ورودی این دو مدل متفاوت هستند پس چطور Hubert از Bert استفاده می کند بی معنی است .
با اینکه لایه ها و معماری کلی بسیار شبیه هم هستند ولی اساسا Hubert روی wave to vec شده سیگنال های speech audio کار می کند که با استفاده از Clustering Teacher Network خود که الگوریتم K-Means tokenize و label زده شده اند , کار می کند.

3-1 معرفی مجموعه دادگان

پیش پردازش ها

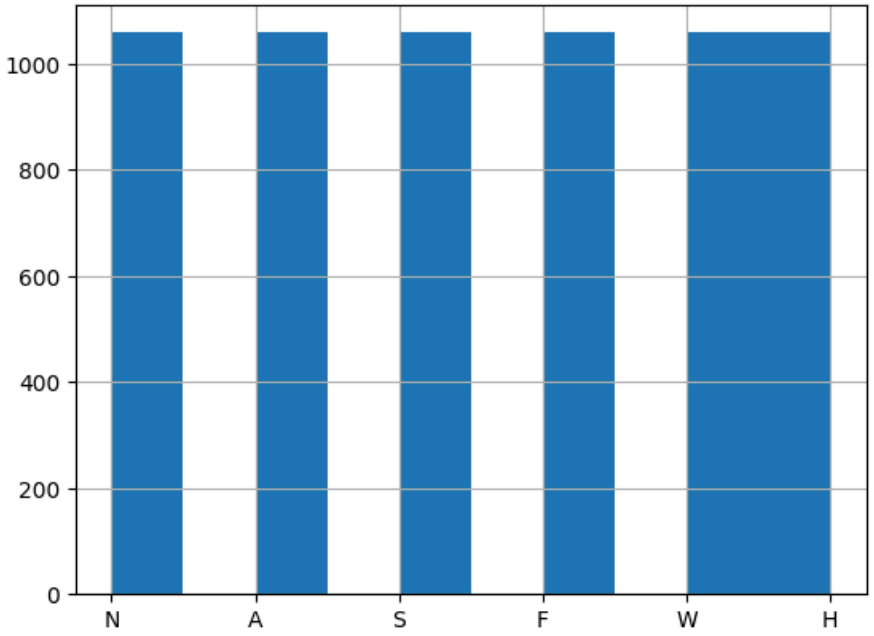
- Label زدن روی داده ها با استفاده از اسم فایل ها که در اندیس چهارم دارای برچسب هستند
- Pad کردن داده ها برای اینکه در هر Batch باید داده ها از یک سایز باشند (محدودیت های Transformer)
- مرد و زن دارای دادگان جدا هستند به همین دلیل می توانیم آن ها را با یکدگیر خلط کنیم
- Hubert ورودی را به شکل Wav2Vec میگیرد پس این Transfomration هم نیاز است
- ابتدا resample کرده و داده های identical تولید میکنیم صرفا همه داده ها را شیفت داده و به آن ها یک DC فرکانسی می توانیم اضافه کنیم (Pitch) همچنین مقداری نویز هم به صدا ها اضافه کرده و یک Resampling هم روی آن ها اجرا میکنیم)به این طریق داده های Duplicate شده رندم دقیقا مانند داده رفرنسی که از آن ها تولید شده اند نخواهند بود.

توزیع دادگان در دیتاست قبل از Augmentation



با توجه به plot مشاهده میکنیم که دیتاست Unbalanced است به این شکل که تعداد بسیاری از sample ها دارای برچسب خنثی یا عصبانی هستند و تعداد بسیار کمی از آن ها برچسب ترس را دارند.

برای حل این موضوع از Augmentation دستی استفاده میکنیم.



عکس دیتا پس از Resampling و Augmentation

```
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(df, test_size=0.2,
                                         random_state=44, stratify=df['label'])
train_data, val_data = train_test_split(train_data, test_size=0.5,
                                        random_state=44, stratify=train_data['label'])
```

```
import numpy as np
import pandas as pd
import librosa
def vectorized_shift_time(data, sampling_rate, shift_max):
    shift = np.random.randint(sampling_rate * shift_max)
    return np.roll(data, shift)

def vectorized_change_pitch_and_speed(data):
    return librosa.effects.pitch_shift(data, sr=target_sample_rate, n_steps=np.random.randint(-1, 2))

def apply_augmentation_vectorized(row):
    if row["label"] != df_majority.label.mode()[0]:
        row["signal"] = vectorized_add_noise(row["signal"])
        row["signal"] = vectorized_shift_time(row["signal"], sampling_rate=target_sample_rate, shift_max=0.2)
        row["signal"] = vectorized_change_pitch_and_speed(row["signal"])
    return row

df_train = df_train.apply(apply_augmentation_vectorized, axis=1)
```

دادگان Split شده و Data Augmentation ها همه داده ها یکبار balance شده و سپس به همه داده ها به ازای هر کدوم نوعی Noise و Shift و سرعت آن ها بعضا تغییر می کند.

۱-۳-۲ ساخت دیتالودر

در این بخش کد مربوط به کلاس CustomDataLoader دیده می شود . در این قسمت با استفاده از feature_extractor که pretrained بود و آن را import کردیم می توانیم Pad کرده و برای هر Batch Preprocessing مرتبط به آن را داشته باشیم.

با استفاده از روش max_length مزایای زیر را تجربه میکنیم:

چون در context یک batch padding انجام می شود در نتیجه efficient تر است چون دیگر نیاز نیست ماکسیمم کل dataset را برای padding استفاده کنیم و میتوانیم در یک Batch ماکسیمم استفاده کرده و در نتیجه حجم کمتری داده برای خوراندن به مدل داریم.

potentially اگر سایز batch ها بزرگ شود یک جستجو به عنوان سربار عملیاتی در هربار تولید Batch باید انجام شود در حالی که اگر از قبل Pad کنیم دیگر batch های خود را نیاز نیست pad کنیم.

```
class CustomDataLoader:

    def __init__(self, feature_extractor):
        self.feature_extractor = feature_extractor
    def __call__(self, features):
        input_features = [{"input_values": feature["input_values"]} for feature in features]
        label_features = [feature["labels"] for feature in features]
        batch = self.feature_extractor.pad(
            input_features,
            padding="longest",
            return_tensors="pt",
        )
        num_classes = len(labels)
#         one_hot_vectors = torch.nn.functional.one_hot(torch.tensor(label_features, dtype=d_type), num_classes=num_classes)
        batch["labels"] = torch.tensor(label_features,dtype=torch.long)

        return batch

data_loader = CustomDataLoader(feature_extractor)
data_loader
```

۱-۴ تعریف مسیله

با توجه به تعریف مسیله یک دیکشنری از برچسب های درست اضافه میکنیم (۶ کلید و ۶ مقدار)

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

با استفاده از کلاس ازWav2Vec2 pretrained که gradiant های آن را هم خاموش میکنیم می توانیم دنباله ورودی مناسب را تولید کنیم.

این component با Hubert ship شده و توسط ما قابل استفاده است . در واقع سیگنال خالص را دریافت کرده و دنباله ازWav2Vec ها تولید می کند . توجه شود که این ماژول را در قسمت CustomDataLoader نیز استفاده میکنیم و همچنین قبل از دادن داده ها به Hubert روی کل Dataset به عنوان Preprocessor ران میکنیم.

```
# segments_ = []
import numpy as np
def preprocess_function(segments):
    features = segments['signal']
    result = feature_extractor(features, sampling_rate=feature_extractor.sampling_rate)
    target_indices = [labels.index(segment) for segment in segments['label']]
    result['labels'] = target_indices
    return result

train_dataset = train_dataset.map(preprocess_function,batched=True,batch_size=1, num_proc=6)
val_dataset = val_dataset.map(preprocess_function,batched=True ,batch_size=1, num_proc=6)
test_dataset = test_dataset.map(preprocess_function,batched=True, batch_size=1, num_proc=6)
```

۱-۴-۲ آموزش شبکه

عکس زیر نمونه ای از فرآیند آموزش دادن شبکه را نشان می دهد.

با توجه به اینکه Validation Loss و Train Loss در حال کم شدن هستند می توانیم حدس بزنیم که فرآیند آموزش هنوز جا دارد انجام شود .

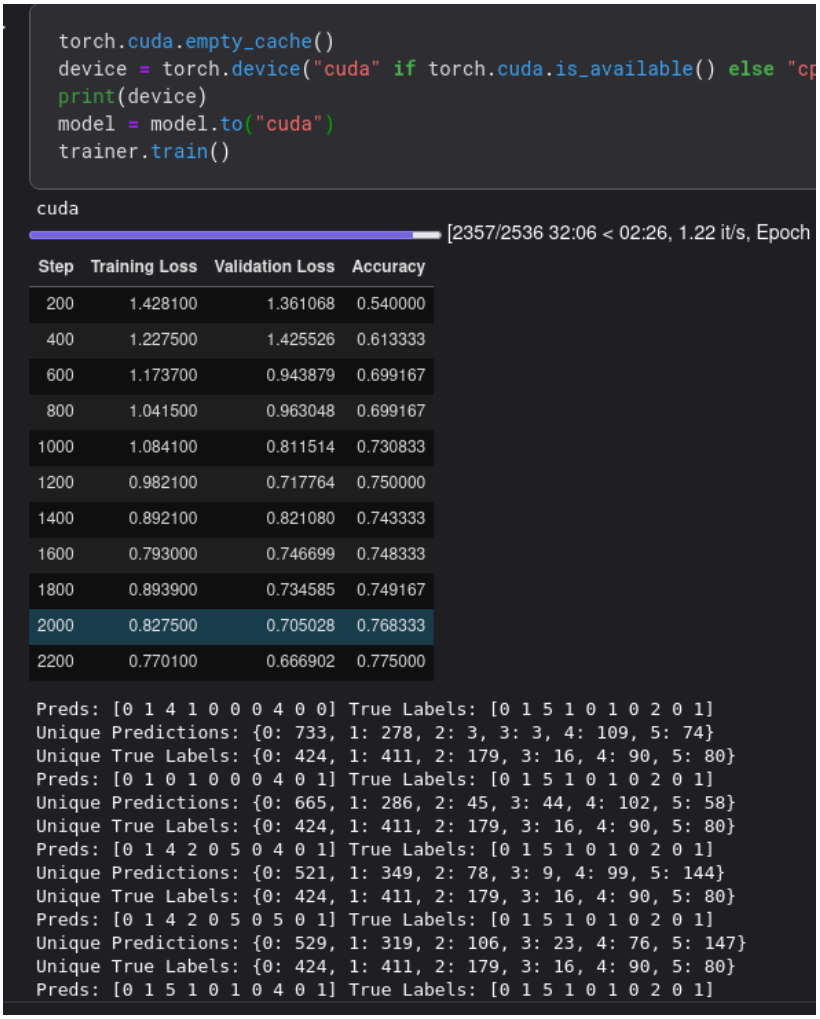
معماری Classifier استفاده شده یک MLP دو لایه است که در لایه وسط Wide تر است و به اندازه تعداد Hidden State مدل Hubert و

سپس ۴ برابر آن ورودی میگیرد و خروجی می دهد .

برای Learning rate از مقدار و برای Criterion از Cross Entropy استفاده کرده ایم .

همچنین شبکه داده ها را به شکل Batch های دوتایی دریافت می کند و به مدت ۴ Epoch آموزش دیده است .

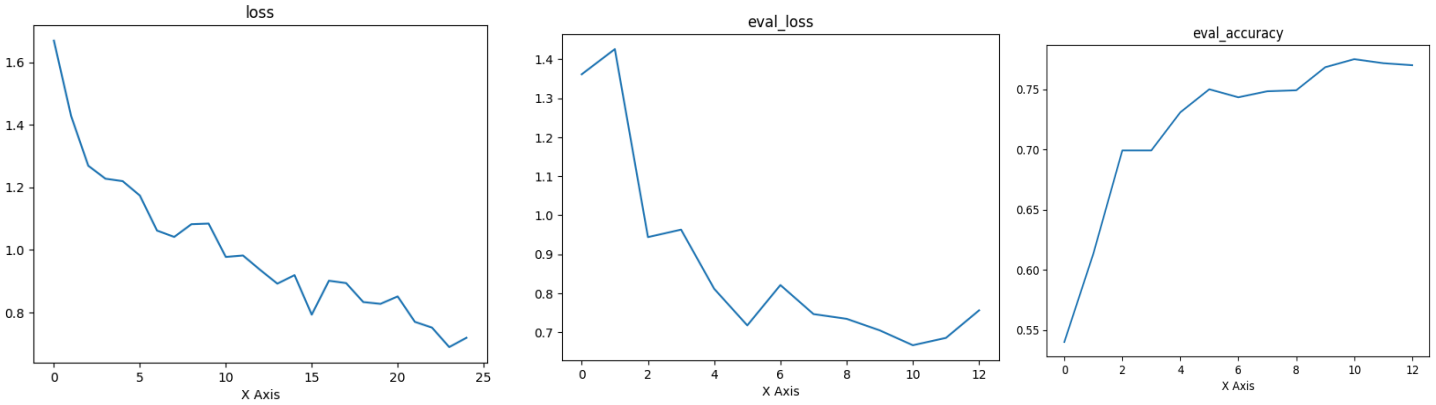
Epochs	Learning Rate	Noise Intensity	Sample Rate
4	0.00001	0.005	16000
Train Batch size	**Criterion**	**Optimizer**	
2	Cross Entropy	Adam	



حین فرآیند آموزش دیدن Hubert و لاگ های مرتبط

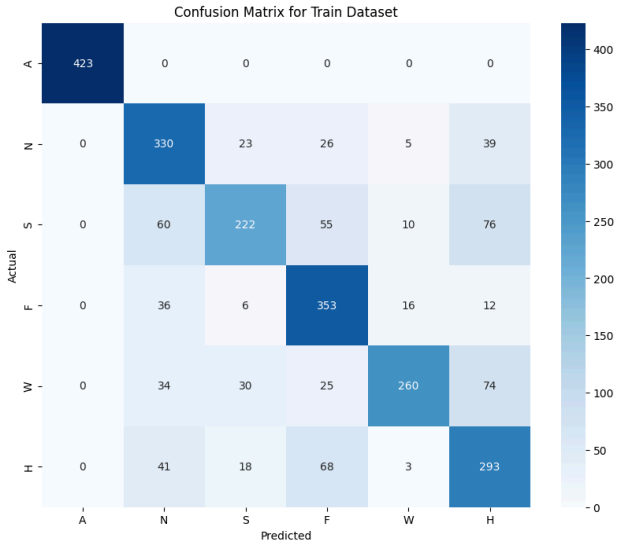
```
]:  trainer.evaluate(test_dataset)

Preds: [1 1 0 1 1 1 0 0 4 1] True Labels: [1 1 0 1 1 1 0 0 4 1]
Unique Predictions: {0: 236, 1: 226, 2: 42, 3: 12, 4: 30, 5: 54}
Unique True Labels: {0: 212, 1: 206, 2: 90, 3: 7, 4: 45, 5: 40}
{'eval_loss': 0.7562432289123535,
 'eval_accuracy': 0.7699999809265137,
 'eval_runtime': 41.9574,
 'eval_samples_per_second': 14.3,
 'eval_steps_per_second': 7.15,
 'epoch': 4.0}
```

+ Code + Markdown



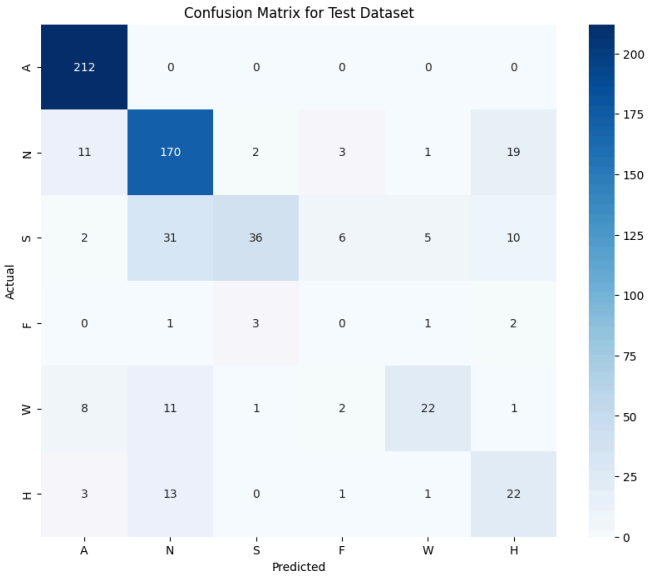
Validation Loss & Validation Accuracy

و نمودار Train Loss به خوبی نشان می دهد که فرآیند آموزش موفق امیز بوده




```
Classification Report for Train Dataset:

              precision    recall  f1-score   support

           A       1.00      1.00      1.00       423
           N       0.66      0.78      0.71       423
           S       0.74      0.52      0.61       423
           F       0.67      0.83      0.74       423
           W       0.88      0.61      0.73       423
           H       0.59      0.69      0.64       423

    accuracy                           0.74      2538
   macro avg       0.76      0.74      0.74      2538
weighted avg       0.76      0.74      0.74      2538
```

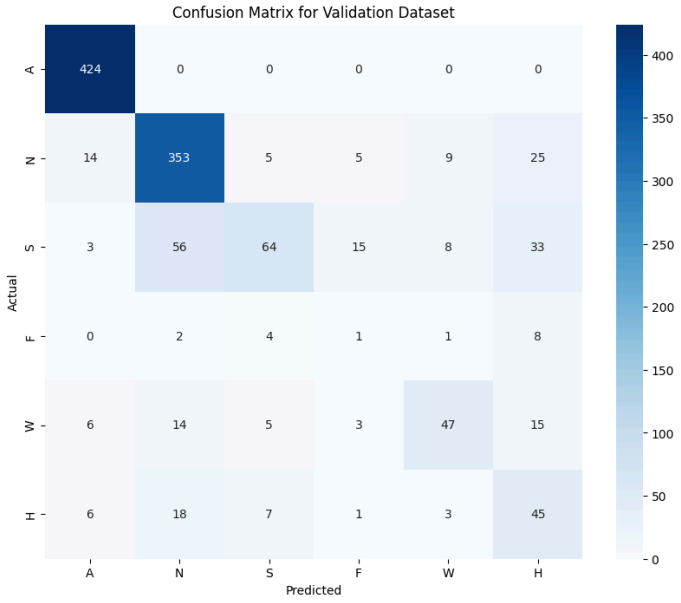
عملکرد روی دیتای تست نشان می دهد که over fitting رخ نداده است . همچنین با توجه به اینکه Datatset در ابتدا Imbalanced بود تا حدی توانسته این مورد را رفع کنیم و بیشتر Prediction ها تبدیل به Angry نشود (با اینکه مقدار خوبی Skewness به سمت Angry وجود دارد و این به دلیل ذات Augmentaiton ما است که Majority class را دست نزده بودیم)



Classification Report for Test Dataset:

	precision	recall	f1-score	support
A	0.90	1.00	0.95	212
N	0.75	0.83	0.79	206
S	0.86	0.40	0.55	90
F	0.00	0.00	0.00	7
W	0.73	0.49	0.59	45
H	0.41	0.55	0.47	40
accuracy			0.77	600
macro avg	0.61	0.54	0.56	600
weighted avg	0.79	0.77	0.76	600

عملکرد روی کلاس ترس خوب نبوده , احتمال بالا به دلیل Confuse شدن این کلاس با کلاس های دیگر و سختی این کلاس باشد .
همچنین تعداد Sample های کلاس ترس بسیار کم بوده و Augmentation ما در بهتر کردن وضع این کلاس تاثیر گذار نبوده است با توجه به اینکه روی داده های Train ترس وضعیت خوبی داشت احتمالا Under fitting شدیدی در این لیبل موقع Train رخ داده است.



Classification Report for Validation Dataset:

	precision	recall	f1-score	support
A	0.94	1.00	0.97	424
N	0.80	0.86	0.83	411
S	0.75	0.36	0.48	179
F	0.04	0.06	0.05	16
W	0.69	0.52	0.59	90
H	0.36	0.56	0.44	80
accuracy			0.78	1200
macro avg	0.60	0.56	0.56	1200
weighted avg	0.79	0.78	0.77	1200

در نهایت می توان دید که با توجه به نمودار اولیه توزیع کلاس ها , توانسته ایم صدمه imbalanced بودن را کم کنیم ولی می توان دید که کلاس های پر جمعیت Neutral و Anger بهترین عملکرد را دارند چون تعداد بسیار بیشتری Sample ازآن ها در اختیار داریم .

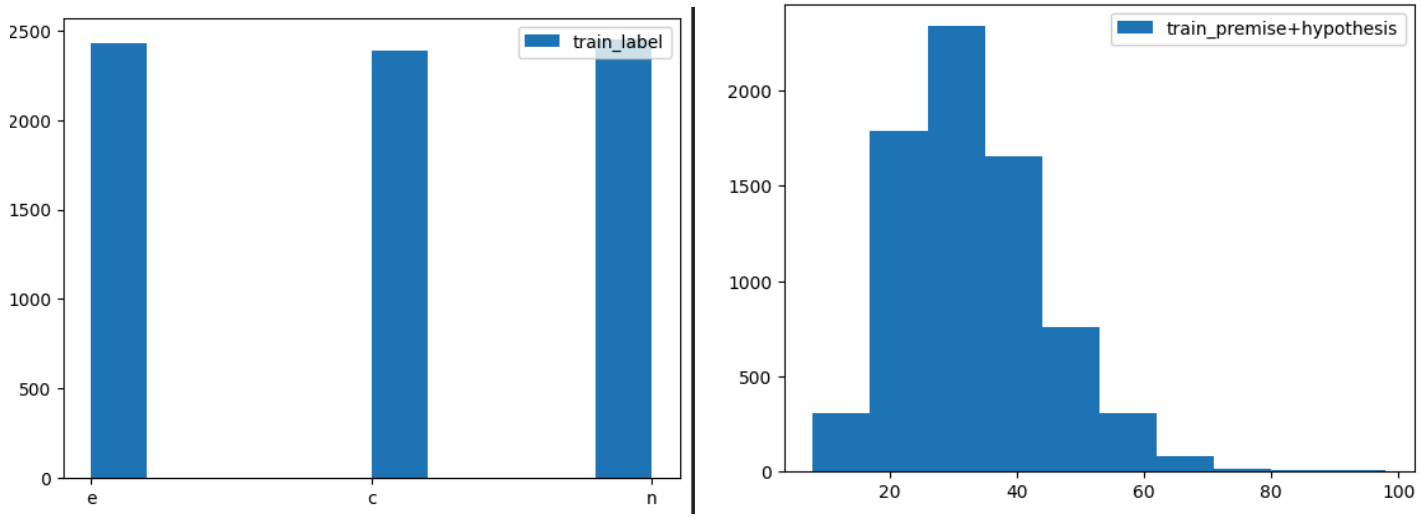
پرسش۲. تنظیم دقیق مدل BERT

۲-۱.پیش پردازش داده ها

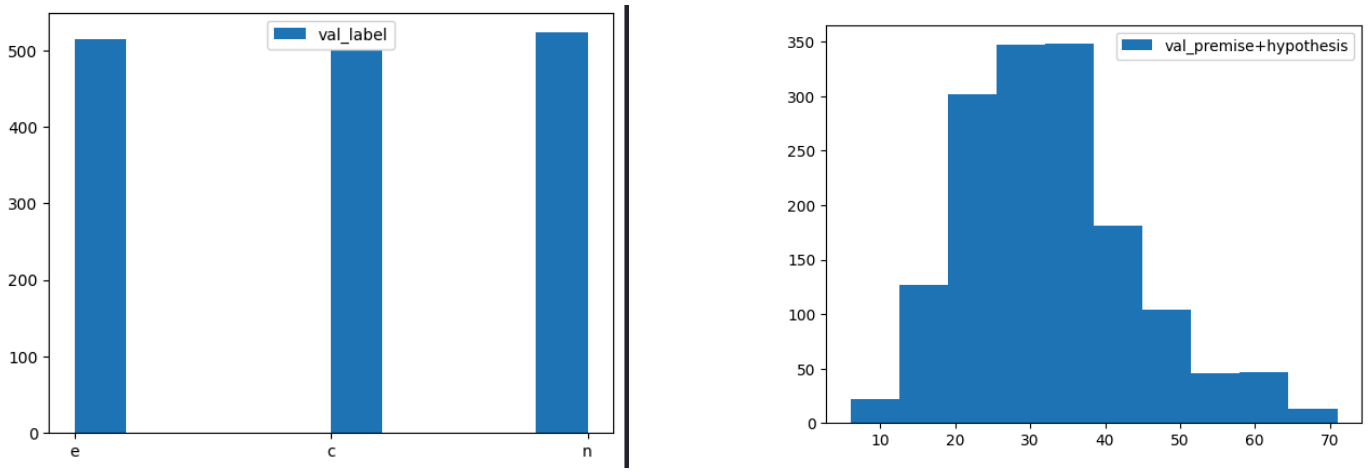
روش های پیش پردازش:

از کتابخانه hazm برای normalize کردن داده های و حذف stopword ها و همچنین lemmatize کردن توکن ها استفاده کردیم. و از AutoTokenizer مربوط به لایبری داده شده از در صورت سوال، برای توکن کردن جملات استفاده کردیم.
همچنین لیبل ها را به اندیس تبدیل کردیم، تا در بتوانیم multi-classification task از آنها استفاده کنیم.

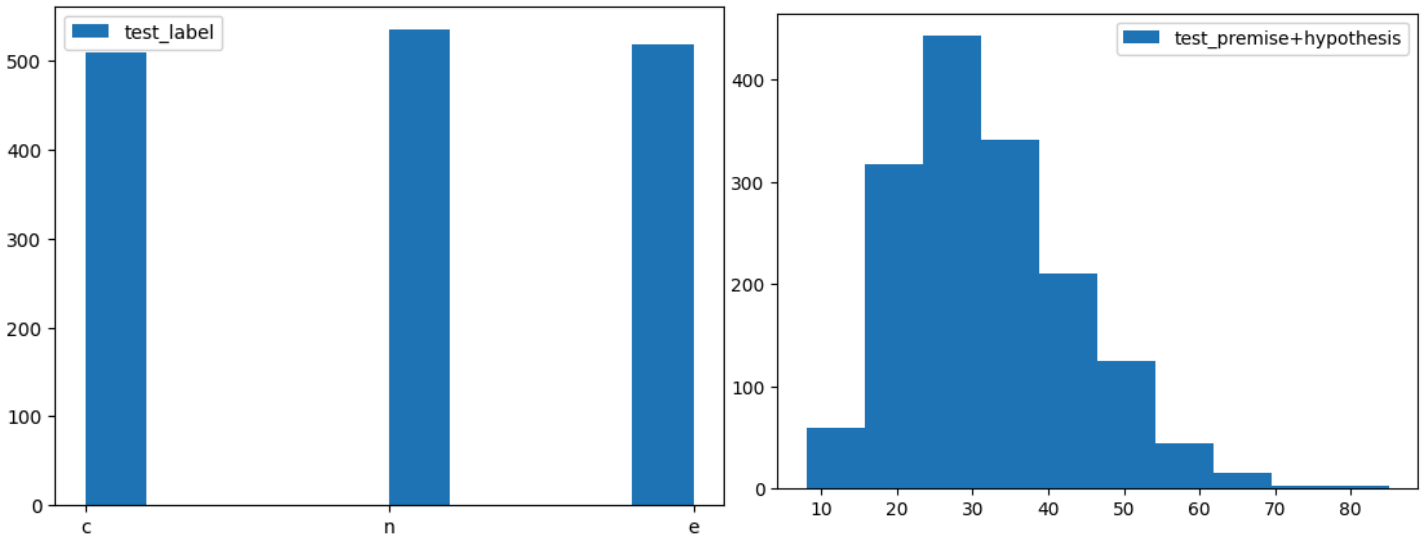
توزیع کلاس ها و طول جملات برای هر dataset train, val, test:



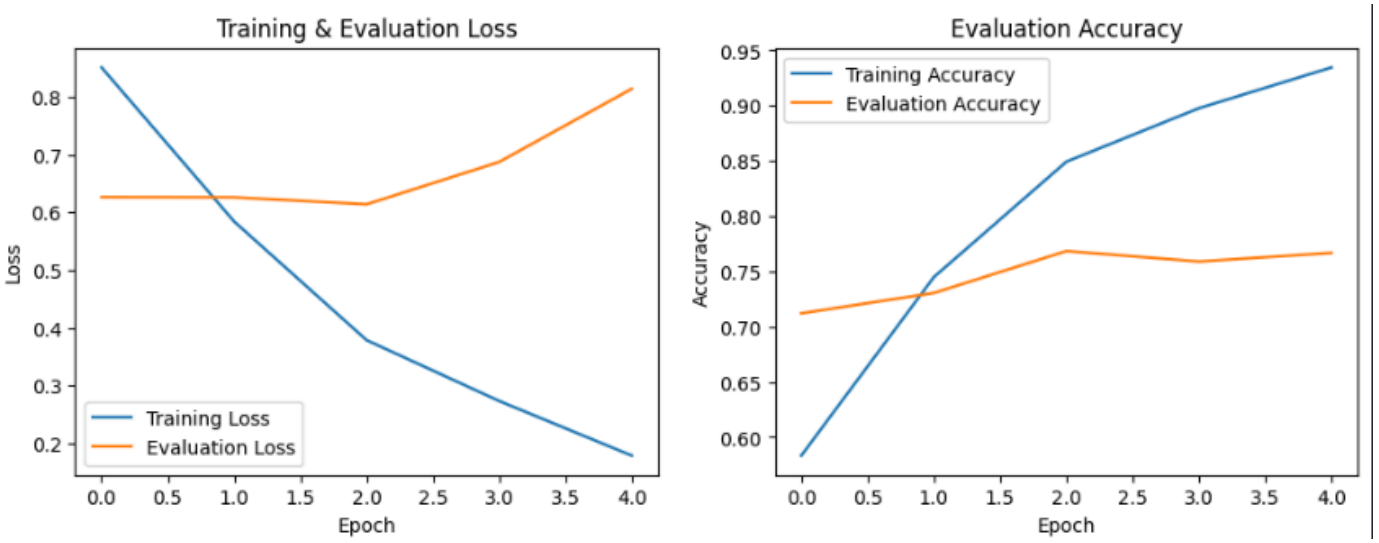
۲.اتوزیع کلاس های و طول جملات train-dataset



۲.۲توزیع کلاس های و طول جملات val-dataset

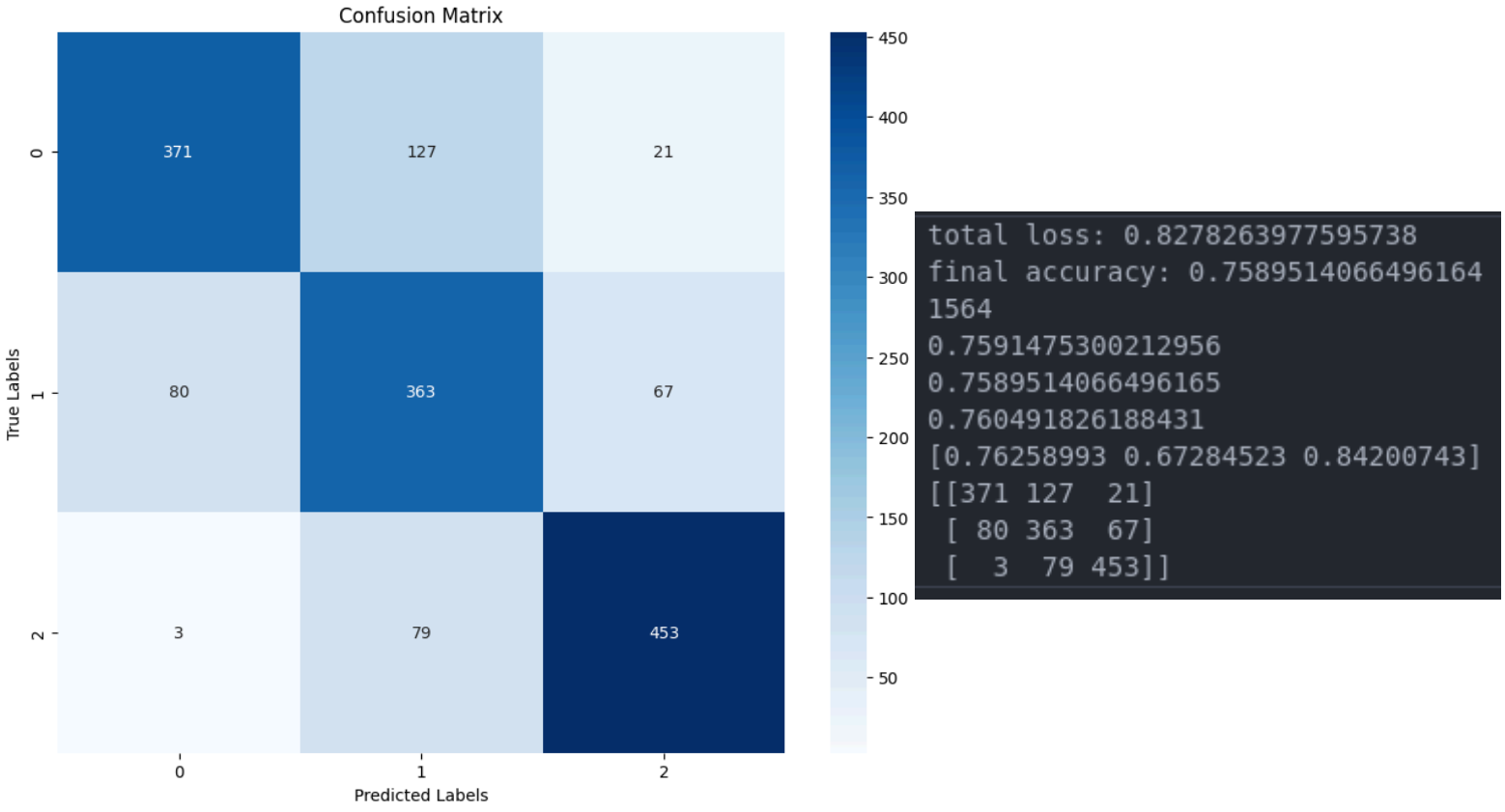


۳.۲توزیع کلاس های و طول جملات test-dataset



Fine tuning BERT .۲-۲

fine tuning val dataset loss/acc figure .۴.۲



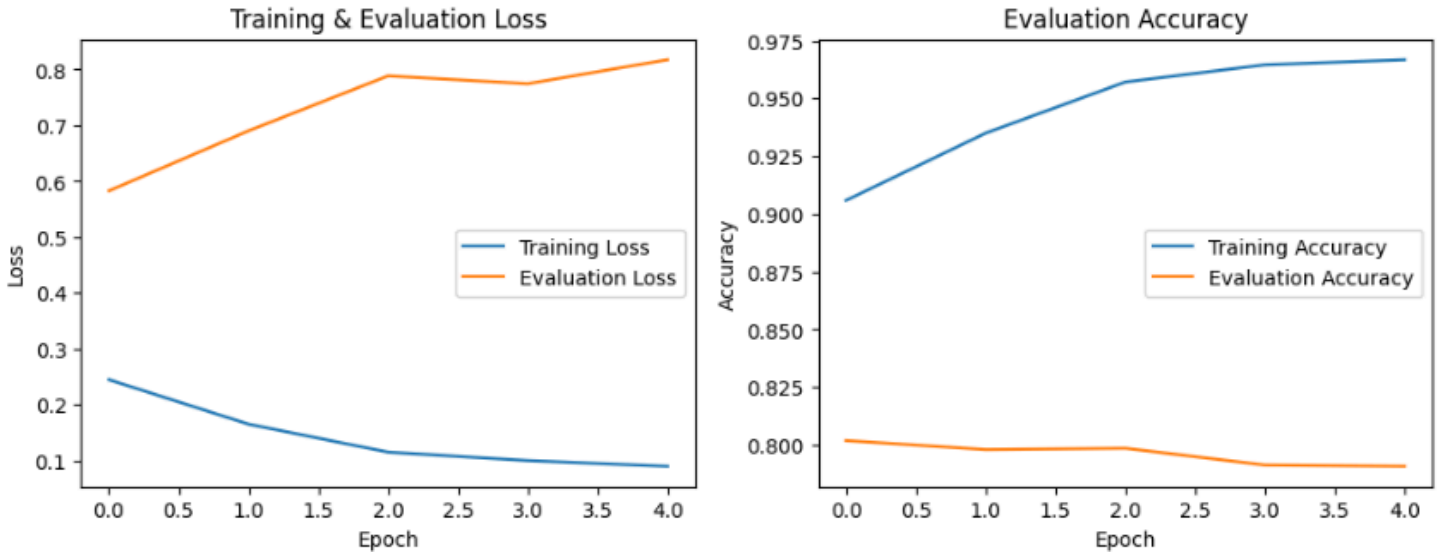

```
total loss: 0.8278263977595738
final accuracy: 0.7589514066496164
1564
0.7591475300212956
0.7589514066496165
0.760491826188431
[0.76258993 0.67284523 0.84200743]
[[371 127  21]
 [ 80 363  67]
 [  3  79 453]]
```

fine tuning test dataset metrics .۵.۲

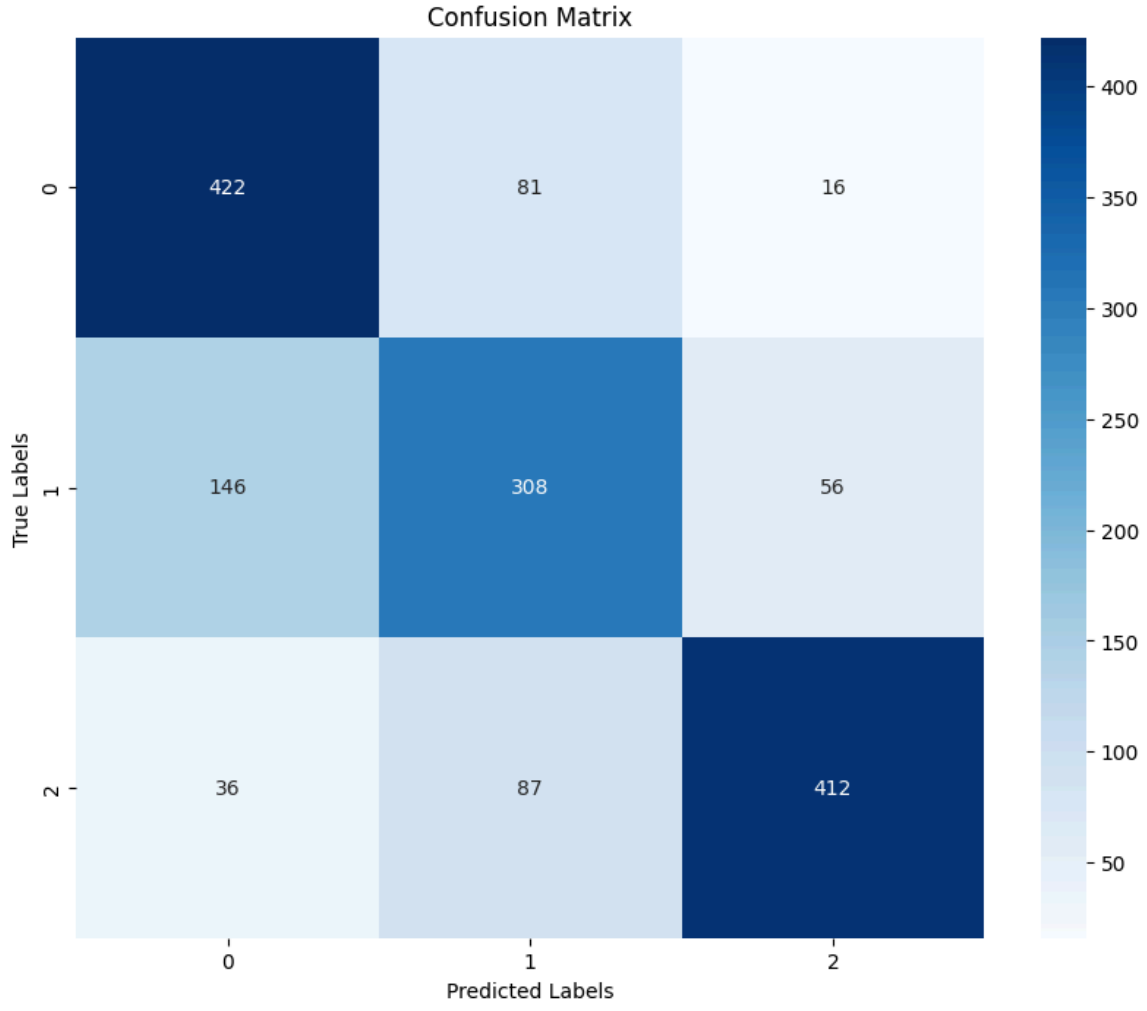
۲-۳. فریز کردن لایه های مدل

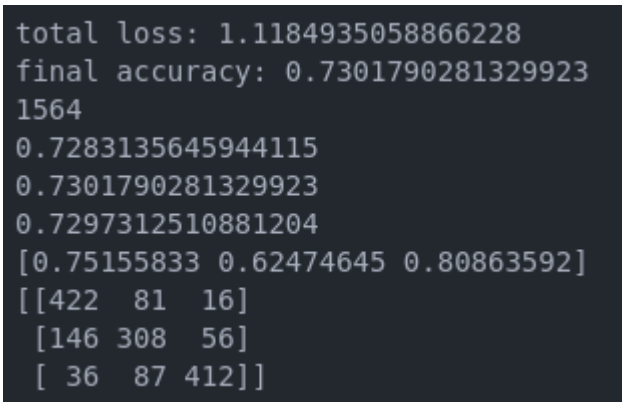
- فریز کردن ۹ لایه ابتدایی:

```
100%|██████████| 57/57 [01:09<00:00,  1.22s/it]
Epoch 1 training loss: 0.24491927066915914
Epoch 1 training accuracy: 0.9058723815700465
100%|██████████| 7/7 [00:05<00:00,  1.18it/s]
Epoch 1 validation loss: 0.5828112219418732
Epoch 1 validation accuracy: 0.8018973214285714
-----------------------
100%|██████████| 57/57 [01:09<00:00,  1.21s/it]
Epoch 2 training loss: 0.16533265445838896
Epoch 2 training accuracy: 0.9350021251460963
100%|██████████| 7/7 [00:05<00:00,  1.18it/s]
Epoch 2 validation loss: 0.6902376052644935
Epoch 2 validation accuracy: 0.7979910714285714
-----------------------
100%|██████████| 57/57 [01:09<00:00,  1.22s/it]
Epoch 3 training loss: 0.11521729892283156
Epoch 3 training accuracy: 0.9571109685981483
100%|██████████| 7/7 [00:05<00:00,  1.17it/s]
Epoch 3 validation loss: 0.7886320943299714
Epoch 3 validation accuracy: 0.7985491071428571
-----------------------
100%|██████████| 57/57 [01:09<00:00,  1.22s/it]
Epoch 4 training loss: 0.10015172400234039
Epoch 4 training accuracy: 0.9645542429204572
100%|██████████| 7/7 [00:06<00:00,  1.17it/s]
Epoch 4 validation loss: 0.7745124268923453
Epoch 4 validation accuracy: 0.7912946428571429
-----------------------
100%|██████████| 57/57 [01:09<00:00,  1.22s/it]
Epoch 5 training loss: 0.08983589852588218
Epoch 5 training accuracy: 0.9667472253765976
100%|██████████| 7/7 [00:06<00:00,  1.16it/s]
Epoch 5 validation loss: 0.817457317960881
Epoch 5 validation accuracy: 0.7907366071428571
-----------------------
```



2.6. freezing the first 9 layers val loss/acc figure

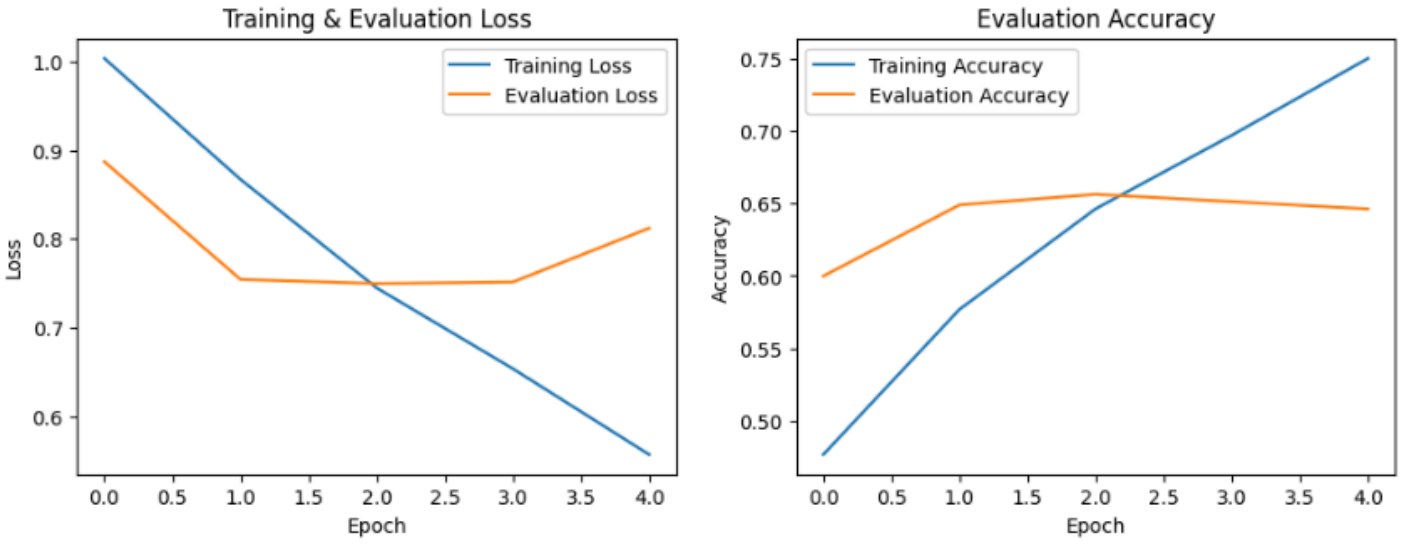




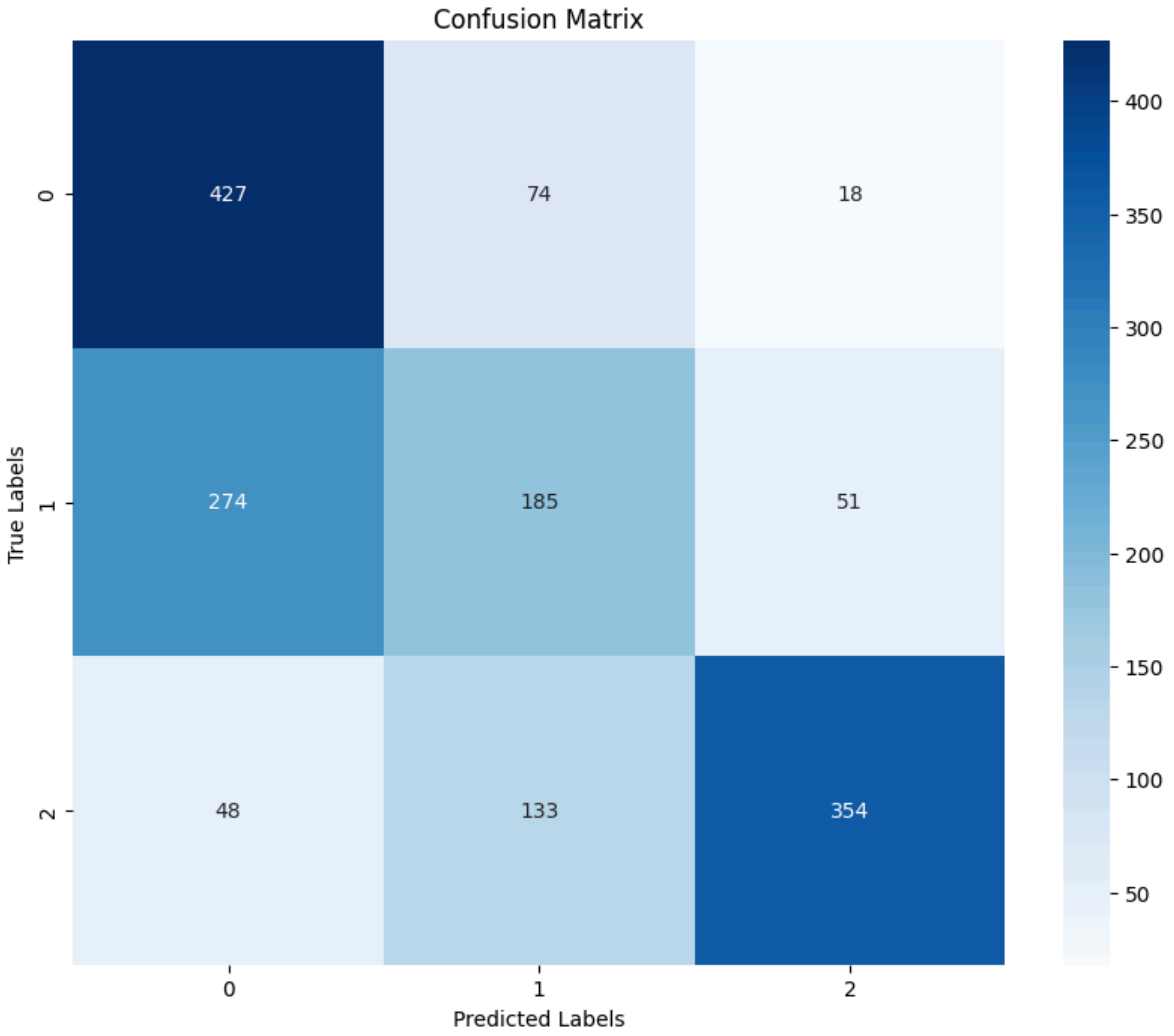
2.7. freezing the first 9 layers test metrics

- فریز کردن تمامی لایه ها به جز لایه آخر و لایه embedding در مدل

```
100%|██████████| 57/57 [01:06<00:00,  1.16s/it]
Epoch 1 training loss: 1.0040053223308765
Epoch 1 training accuracy: 0.47677228743569894
100%|██████████| 7/7 [00:05<00:00,  1.18it/s]
Epoch 1 validation loss: 0.8873195818492344
Epoch 1 validation accuracy: 0.5998883928571429
-----------------------
100%|██████████| 57/57 [01:05<00:00,  1.15s/it]
Epoch 2 training loss: 0.8672520936581126
Epoch 2 training accuracy: 0.5769222159134714
100%|██████████| 7/7 [00:05<00:00,  1.17it/s]
Epoch 2 validation loss: 0.7543151293482099
Epoch 2 validation accuracy: 0.6489955357142857
-----------------------
100%|██████████| 57/57 [01:06<00:00,  1.17s/it]
Epoch 3 training loss: 0.7446957742958739
Epoch 3 training accuracy: 0.6463032573984381
100%|██████████| 7/7 [00:06<00:00,  1.16it/s]
Epoch 3 validation loss: 0.7496507444552013
Epoch 3 validation accuracy: 0.65625
-----------------------
100%|██████████| 57/57 [01:06<00:00,  1.17s/it]
Epoch 4 training loss: 0.6534737881861234
Epoch 4 training accuracy: 0.6970047888002897
100%|██████████| 7/7 [00:05<00:00,  1.17it/s]
Epoch 4 validation loss: 0.7513848748058081
Epoch 4 validation accuracy: 0.6512276785714286
-----------------------
100%|██████████| 57/57 [01:05<00:00,  1.16s/it]
Epoch 5 training loss: 0.5565359100960848
Epoch 5 training accuracy: 0.7500363630160951
100%|██████████| 7/7 [00:06<00:00,  1.16it/s]
Epoch 5 validation loss: 0.8122083287952202
Epoch 5 validation accuracy: 0.6462053571428571
-----------------------
```



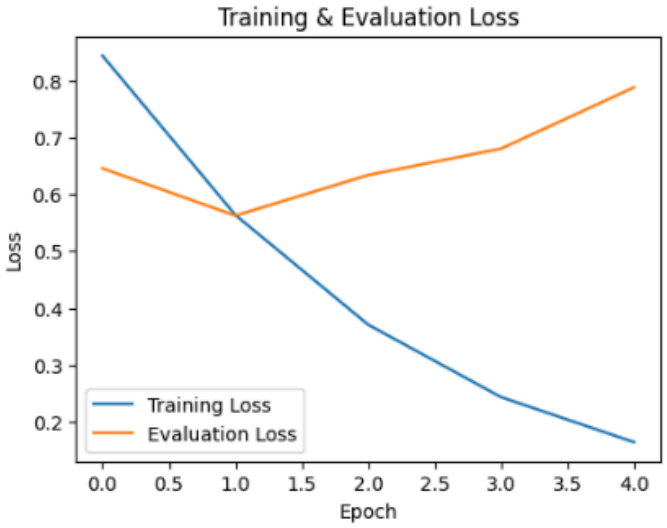
2.8. freezing all layers except the last one val loss/acc figure

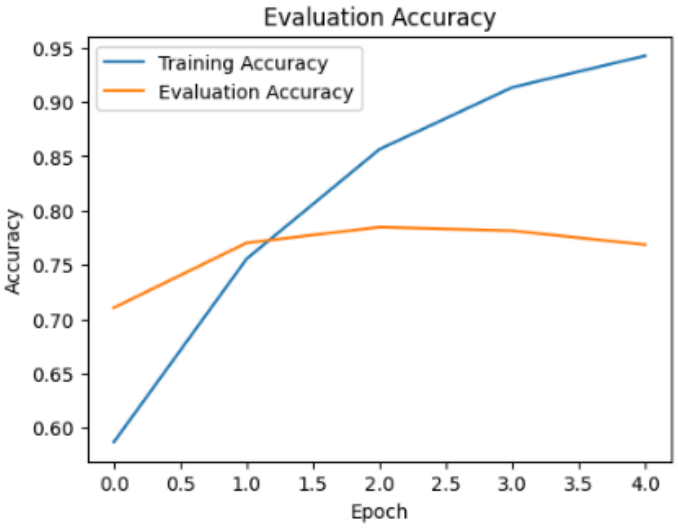



```
total loss: 0.9011879137584141
final accuracy: 0.6176470588235294
1564
0.6075802665996196
0.6176470588235294
0.6100609422904546
[0.67350158 0.41019956 0.73903967]
[[427  74  18]
 [274 185  51]
 [ 48 133 354]]
```

2.9. freezing all layers except the last one test metrics

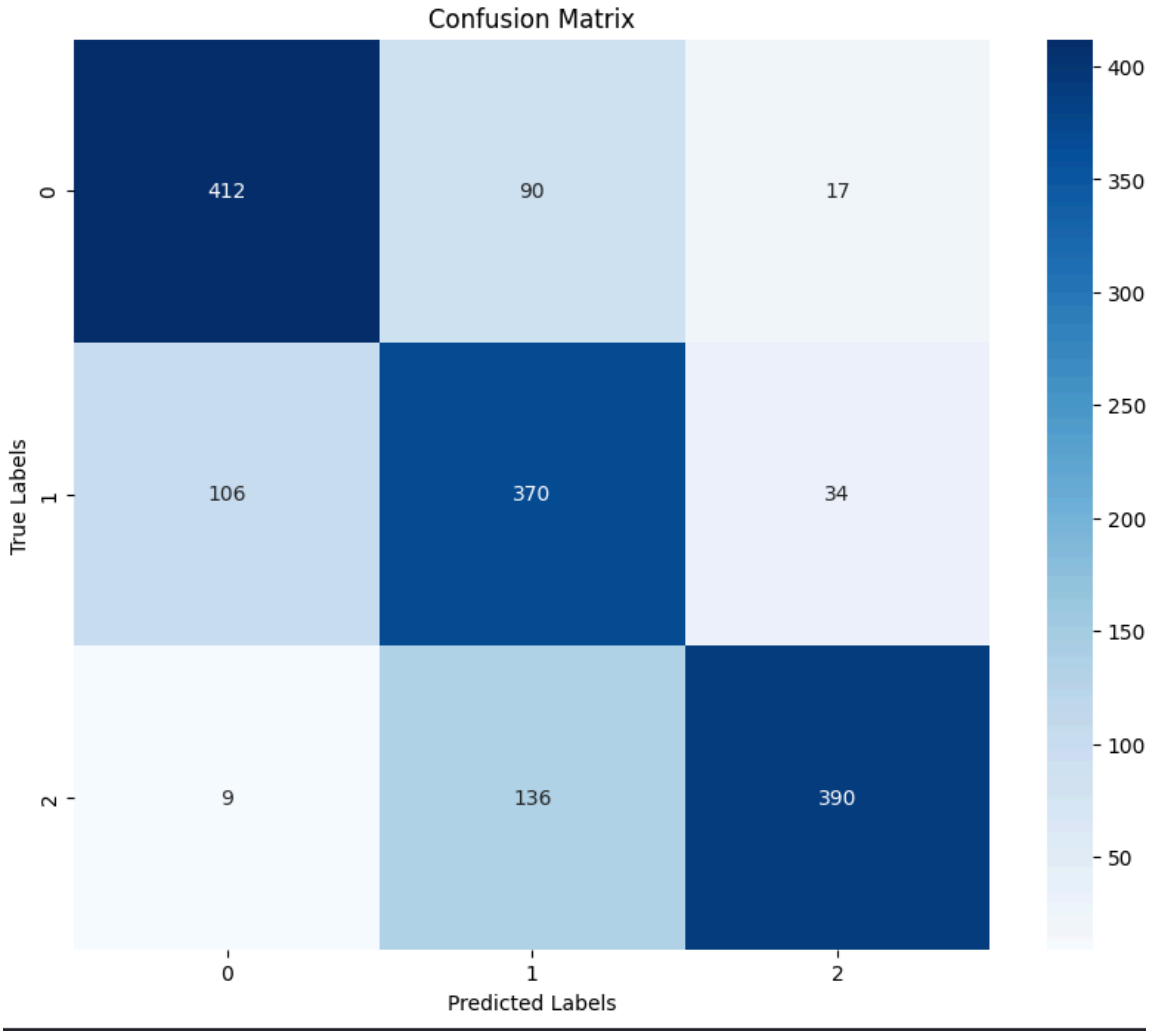
۴-۲. Fine tuning on intermediate layers






```
100%|██████████| 57/57 [01:16<00:00,  1.33s/it]
Epoch 1 training loss: 0.843661956619798
Epoch 1 training accuracy: 0.5866815468721223
100%|██████████| 7/7 [00:06<00:00,  1.01it/s]
Epoch 1 validation loss: 0.6455841703074319
Epoch 1 validation accuracy: 0.7103794642857143
-----------------------
100%|██████████| 57/57 [01:16<00:00,  1.34s/it]
Epoch 2 training loss: 0.5636748880670782
```

2.10. fine-tuning intermediate layers val loss/acc figure

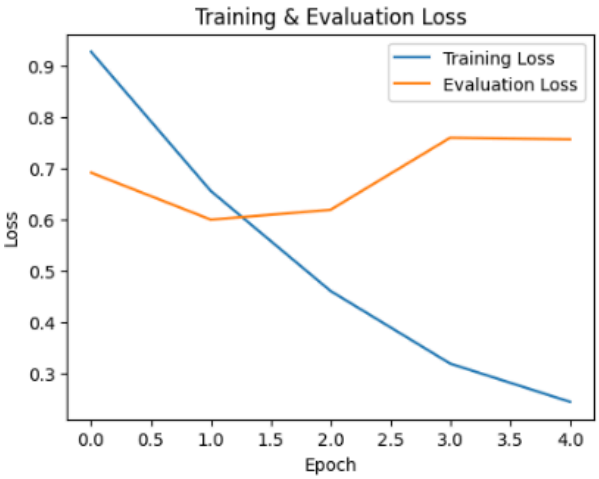


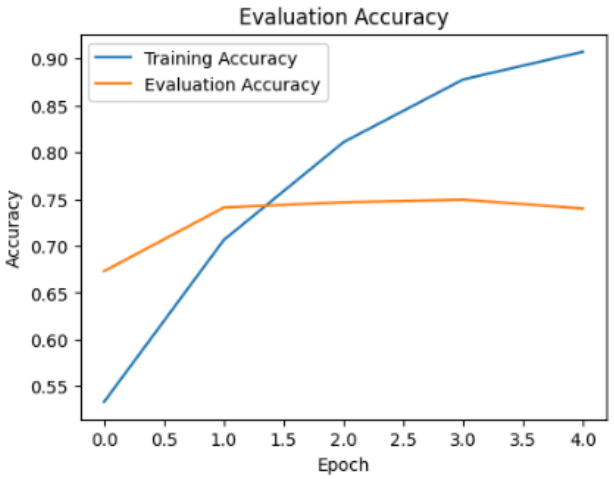

```
total loss: 0.8503413711275373
final accuracy: 0.7493606138107417
1564
0.7520069972879854
0.7493606138107417
0.7529668031995694
[0.78776291 0.66907776 0.79918033]
[[412  90  17]
 [106 370  34]
 [  9 136 390]]
```

2.11. fine-tuning intermediate layers test metrics

همان طور که نتایج نشان می دهد توانایی مدل در حل مسپله تغییری نکرده است این نشان میدهد که همان لایه های ابتدایی، قادر به حل مسئله هستند و نیاز به لایه بعدی نیست.

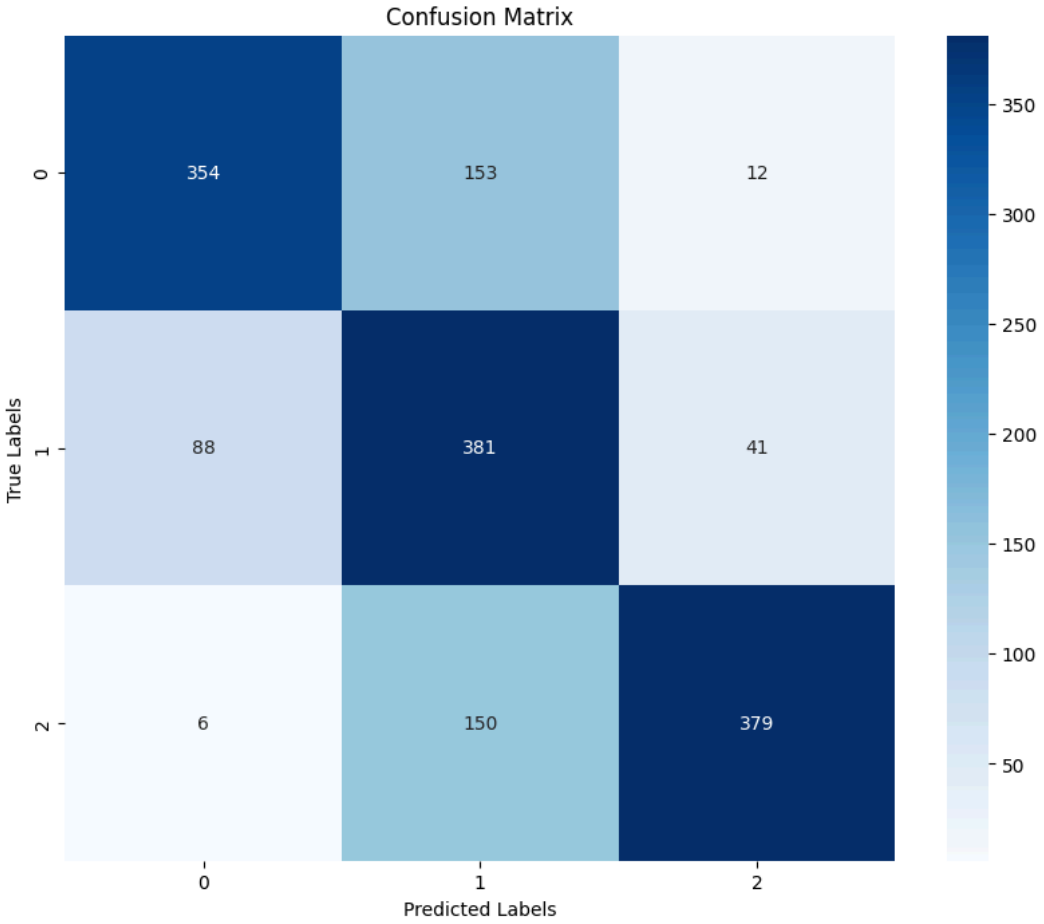
۲-۵. remove attention heads






```
100%|██████████| 57/57 [01:21<00:00,  1.43s/it]
Epoch 1 training loss: 0.9273825361017596
Epoch 1 training accuracy: 0.5332723552720589
100%|██████████| 7/7 [00:07<00:00,  1.02s/it]
Epoch 1 validation loss: 0.6910515257290432
Epoch 1 validation accuracy: 0.6729910714285714
-----------------------
100%|██████████| 57/57 [01:20<00:00,  1.42s/it]
Epoch 2 training loss: 0.6553850498115807
Epoch 2 training accuracy: 0.7063781106681154
100%|██████████| 7/7 [00:07<00:00,  1.04s/it]
```

2.12. remove attention heads val loss/acc figure




```
total loss: 0.8417168685368129
final accuracy: 0.7122762148337596
1564
0.7180733034354813
0.7122762148337596
0.7192060722287066
[0.73216132 0.63819095 0.78386763]
[[354 153  12]
 [ 88 381  41]
 [  6 150 379]]
```

2.11. remove attention heads test metrics

نتایج:

Model	Accuracy
1. fine-tuning the BERT	0.7589514066496164
2. freezing the first 9 lyaers of the model	0.7301790281329923
3. freezing all the layers except the embedding layer and the the last one	0.6176470588235294
4. Fine-tuning the intermediate layers	0.7493606138107417
5. Remove the attention heads	0.7122762148337596

2.1 models test accuracy

همان طور که مشاهده میکنید، حالت اول از همه حالت های اندکی بهتر عمل کرده است. البته که شاید در ایپاک ها بالاتر نتایج اندکی متفاوتی داشته باشند. در اینجا نتایج پس از اجرای **5 ایپاک** میباشد. با توجه به مدل ۴و۵ میتوان این نتیجه را گرفت که کاهش تعداد لایه ها (مدل۴) و کم کردن capacity مدل و همچنین حذف کردن و کاهش attention head ها، به صورت چشمگیری تاثیر منفی در accuracy مدل ندارد و همچنان مدل به خوبی به روی داده ها تست عمل میکند. میتوان نتیجه گرفت که مدل با همان لایه های ابتدایی اش قادر به حل این مسئله classification میباشد و لایه بعدی تاثیر چشمگیری در دقت مدل ندارند. همچنین با توجه به مدل ۵ همان طوری که در مقاله ذکر شده، گفته شده بود، با حذف کردن attention head ها افت قابل ملاحظه ای در دقت مدل مشاهده نمیباشد و مدل به خوبی به روی داده های تست عمل میکند.

همچنین **مدل۳** نشان میدهد که وضعیت pre-trained لایه های ابتدایی که نقش کلیدی در حل مسئله را بازی میکنند، نیاز به اندکی tuning دارند و freeze کردن این لایه ها باعث کاهش دقت شده است.