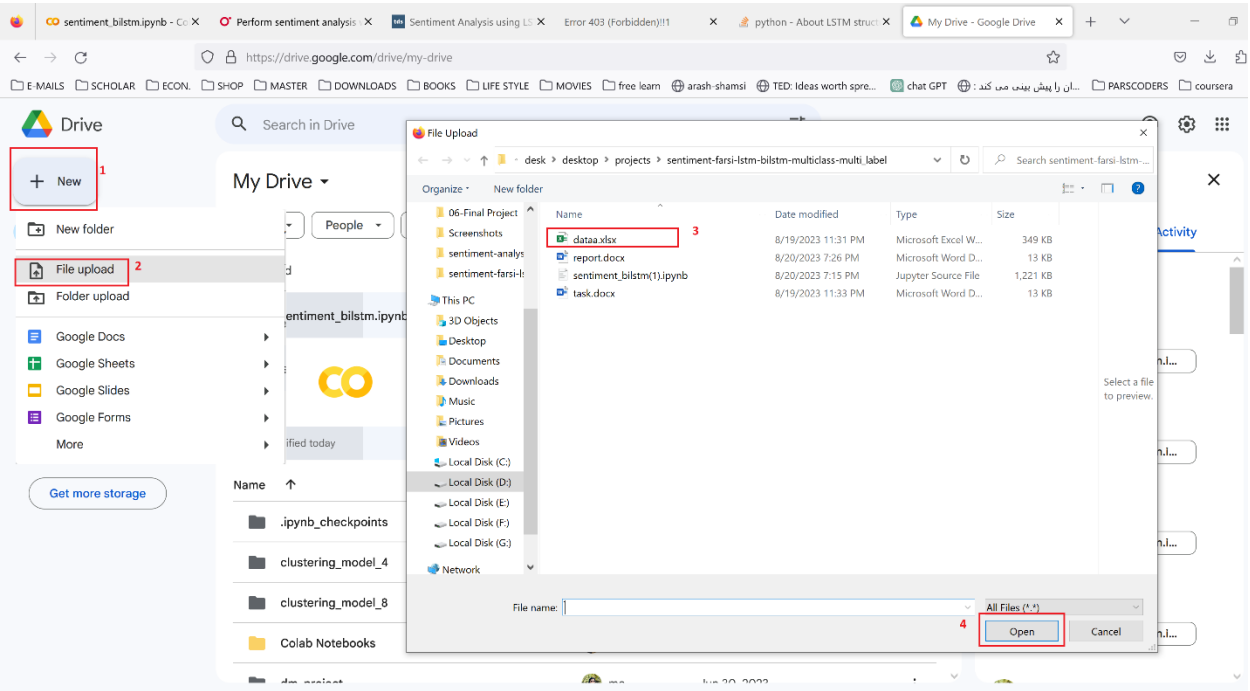
پروژه‌ی تحلیل احساسات زبان فارسی با ۵ برچسب چند کلاسه توسط lstm و bilstm

به منظور اجرای سریع برنامه از محیط گوگل کوبی و محیط پردازش گرافیک T4 استفاده شد تا مراحل آموزش و تست مدل روند سریع‌تری پیدا کند. همچنین از محیط گوگل درایو استفاده شد که مجموعه داده در آن ذخیره شود و نیاز به بارگزاری مجدد آن در هر اجرا نباشد.

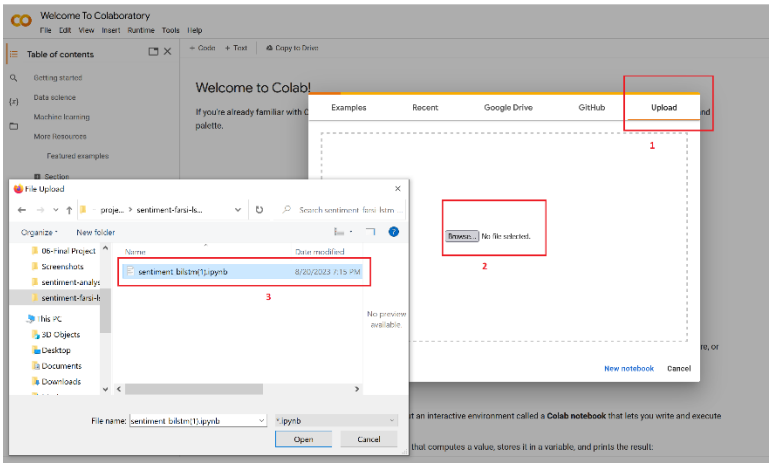
گوگل درایو:

https://drive.google.com/drive/my-drive



گوگل کولب:

https://colab.research.google.com



مرحله‌ی نخست: تحلیل اکتشافی و پیش پردازش مجموعه داده

پس از بارگزاری داده ها در گوگل درایو و اتصال به محیط کولب، برای خواندن مجموعه داده از کد زیر استفاده شد:

```
import pandas as pd

# Specify the file path of the CSV file
file_path = '/content/gdrive/MyDrive/dataa.xlsx'

# Read the CSV file into a pandas DataFrame
try:
    df = pd.read_excel(file_path)
    # Do further processing or analysis on the DataFrame as needed

    df.head()  # Print the first few rows of the DataFrame for
verification

except pd.errors.ParserError as e:
    print(f"Error occurred while parsing the CSV file: {e}")

except FileNotFoundError:
    print(f"CSV file not found at the specified file path: {file_path}")
```

	reviews	Tangibles	Reliability	Empathy	Assurance	Responsiveness
0	...اتاق ها خیلی معمولی، پرسنل بی اعصاب، صبحانه مع	0	2.0	2	-1	2
1	...هتل با توجه به قیمت و تبلیغات بالا راضی کننده	-1	-1.0	2	-1	2
2	کارکنان بسیار خوب و مودب	2	2.0	2	1	2
3	...اتاقش خیلی خوب بود دلباز و تمیز، من چون گرمایی	1	2.0	2	2	2
4	... برخورد پرسنل هتل عالی بود.موسیقی زنده در کافی	1	2.0	2	1	2

این کد یک قطعه از کد پایتون با استفاده از کتابخانه Pandas برای خواندن و پردازش یک فایل اکسل (XLSX) توضیح می‌دهد. این کد از Pandas برای خواندن داده‌ها از یک فایل اکسل استفاده می‌کند و سپس اقدام به پردازش یا تجزیه و تحلیل بیشتر اطلاعات درون آن می‌کند. در ادامه توضیحاتی در مورد هر بخش از کد آورده شده است:

۱. import pandas as pd: در این خط، کتابخانه Pandas با نام مخفف pdوارد می‌شود تا از توابع و امکانات آن استفاده شود.

۲. file_path = '/content/gdrive/MyDrive/dataa.xlsx': مسیر فایل اکسل که باید خوانده شود، در اینجا مشخص شده است.

۳. ::tryاین بخش به کد اصلی یک بلاک تلاش می‌دهد تا هر خطای ممکن را کشف کند.

۴. :(df = pd.read_excel(file_path)این خط از کد با استفاده از تابع pd.read_excel()داده‌های موجود در فایل اکسل مشخص شده توسط file_pathرا به صورت یک DataFrame در متغیر dfخوانده و ذخیره می‌کند.

۵. ::except pd.errors.ParserError as eاین بخش مشخص می‌کند چه کاری انجام شود اگر خطا در فرآیند خواندن و تجزیه و تحلیل داده‌ها رخ دهد. اگر خطایی از نوع ParserError(خطای تجزیه و تحلیل داده) رخ دهد، پیام خطا به همراه جزئیات آن نمایش داده می‌شود.

۶. ::except FileNotFoundErrorاین بخش به دنبال خطای فایل پیدا نشدن در مسیر مشخص شده می‌گردد. اگر فایل مورد نظر پیدا نشود، پیام خطا مناسب نمایش داده می‌شود.

۷. :(df.head()در این خط، ابتدا پنج ردیف اول از DataFrame موجود در متغیر dfبا استفاده از تابع head() نمایش داده می‌شود. این کار به عنوان یک تایید اولیه برای صحیح خواندن داده‌ها انجام می‌شود.

بنابراین، این کد ابتدا فایل اکسل مشخص شده را با استفاده از Pandas خوانده و در یک DataFrame ذخیره می‌کند. سپس، اگر خطای تجزیه و تحلیل داده یا خطای پیدا نشدن فایل رخ دهد، پیام‌های خطا مناسب نمایش داده می‌شوند. در نهایت، اولین پنج ردیف از داده‌ها برای تایید صحیح بودن خواندن، نمایش داده می‌شوند

با دستور زیر تعداد ردیف هر ستون شمارش می‌شود:

```
df.count()
```

```
reviews           2558
Tangibles         2558
Reliability       2557
Empathy           2558
Assurance         2558
Responsiveness    2558
dtype: int64
```

کتابخانه‌ی هضم برای پیش پردازش آتی با دستور زیر نصب می‌شود:

```
if 'google.colab' in str(get_ipython()):
  !pip install hazm numpy==1.23
```

با دستورات زیر تعداد داده‌های null و تکراری شمارش می‌شوند:

```
df.duplicated().sum()
```

```
5
```

```
[8] df.isna().sum()
```

```
reviews           0
Tangibles         0
Reliability       1
Empathy           0
Assurance         0
Responsiveness    0
dtype: int64
```

```
# Display the number of duplicated records
num_duplicates = df.duplicated().sum()
print("Number of duplicate records:", num_duplicates)

# Display the number of NaN values in each column
nan_counts = df.isna().sum().sum()
print("NaN value counts in each column:", (nan_counts))
```

```
Number of duplicate records: 5
NaN value counts in each column: 1
```

۱ داده‌ی نال و ۵ داده‌ی تکرار وجود دارد. داده های نال با مقدار مود جایگزین شدند و داده‌های تکراری حذف شدند. این بخش توسط خطوط هایلایت شده در کدهای زیر انجام شده است.

```
# Remove rows with NaN values
#df = df.dropna()

# Remove duplicate records
import pandas as pd

# Assuming you have already loaded your DataFrame as 'df'
# df = ...

# Calculate the mode of the 'Reliability' column
reliability_mode = df['Reliability'].mode()[0]
```

```
# Fill NaN values in the 'Reliability' column with its mode
df['Reliability'].fillna(reliability_mode, inplace=True)
df=df.drop_duplicates()
# Display the number of duplicated records
num_duplicates = df.duplicated().sum()
print("Number of duplicate records:", num_duplicates)

# Display the number of NaN values in each column
nan_counts = df.isna().sum().sum()
print("NaN value counts in each column:", (nan_counts))
```

```
Number of duplicate records: 0
NaN value counts in each column: 0
```

پیش پردازش:

```
alphabet = set()

for _, row in df.iterrows():
    text = row['reviews']
    for char in text:
        alphabet.add(char)
```

توسط کد بالا تمام کارکترهای مورد استفاده در متون دیتاست استخراج می‌شوند، این کارکترها ۱۶۶ عدد هستند.

این کد برای ساخت یک مجموعه حروف منحصر به فرد (الفبا) از حروف مورد استفاده در متن‌های موجود در ستون "reviews"یک DataFrame استفاده می‌شود. در زیر، هر بخش از کد توضیح داده شده است:

۱. alphabet = set():یک مجموعه (set) تهی به نام alphabetایجاد می‌شود. این مجموعه برای ذخیره حروف منحصر به فردی استفاده می‌شود که در متن‌های موجود در ستون "reviews" ظاهر می‌شوند.

۲. for _, row in df.iterrows()::یک حلقه forشروع می‌شود که بر روی هر ردیف از DataFrame با استفاده از تابع iterrows()حرکت می‌کند. این تابع هر بار یک تاپل با دو عنصر تولید می‌کند: اندیس ردیف و اطلاعات آن ردیف.

۳. text = row['reviews']:در هر تکرار حلقه، متن موجود در ستون "reviews" ردیف فعلی در متغیر text ذخیره می‌شود.

۴. for char in text::در این داخلی‌ترین حلقه for، بر روی هر کاراکتر (حرف یا نویسه) در متن textحرکت می‌شود.

۵. alphabet.add(char):در هر تکرار حلقه داخلی، حرف مورد نظر به مجموعه alphabetاضافه می‌شود. از این طریق، تمام حروف موجود در متن‌ها به صورت یکتا در مجموعه ذخیره می‌شوند.

در نهایت، پس از اجرای این کد، مجموعه alphabetشامل تمام حروف منحصر به فردی خواهد بود که در متن‌های موجود در ستون DataFrame "reviews" ظاهر شده‌اند. این کار ممکن است برای تجزیه و تحلیل نوعی از داده‌ها مفید باشد که نیاز به دست‌یابی به مجموعه یا الفبای حروف مورد استفاده دارند

در ادامه فراوانی حروف الفبای اصلی محاسبه می‌شود:

```
import numpy as np

concatdf = df['reviews']
total_length = concatdf.str.len().sum()

alphdf = pd.DataFrame(list(alphabet), columns=['alphabet'])
alphdf['dist'] = np.zeros(len(alphdf))

for index, row in alphdf.iterrows():
    char = row['alphabet']
    if char in '.^$*+?()[{\|^-]\\':
        char = f'\\{char}' # escape special regex character
    alphdf.loc[index, 'dist'] = concatdf.str.count(char).sum() /
total_length
```

این کد برای محاسبه توزیع حروف مختلف (تعداد تکرار) در متن‌های موجود در ستون "reviews" یک DataFrame استفاده می‌شود و از کتابخانه‌های Pandas و NumPy برای پردازش استفاده می‌کند. در زیر، هر بخش از کد توضیح داده شده است:

۱. import numpy as np: کتابخانه NumPy با نام مخفف npوارد می‌شود NumPy .برای پردازش عددی در پایتون استفاده می‌شود.

۲. concatdf = df['reviews']: محتوای ستون "reviews" از DataFrame با نام dfدر متغیر concatdf ذخیره می‌شود. این متغیر حاوی تمام متن‌ها از این ستون به صورت یک سری Pandas (Series) خواهد بود.

۳. total_length = concatdf.str.len().sum(): طول کلی متن‌های موجود در concatdfبا استفاده از تابع str.len()بدست آمده و سپس این طول‌ها جمع می‌شوند تا مجموع طول کلی متن‌ها بدست آید.

۴. alphdf = pd.DataFrame(list(alphabet), columns=['alphabet']): یک DataFrame با نام alphdfایجاد می‌شود. این DataFrame حاوی حروف یکتا از مجموعه alphabetاست و یک ستون به نام "alphabet"دارد.

۵. alphdf['dist'] = np.zeros(len(alphdf)): یک ستون با نام "dist" به alphdfاضافه می‌شود و همه مقادیر آن با صفر پر می‌شوند. این ستون برای ذخیره توزیع (تعداد تکرار) هر حرف در متن‌ها است.

۶. for index, row in alphdf.iterrows():: یک حلقه for شروع می‌شود که بر روی هر ردیف از DataFrame alphdf با استفاده از تابع iterrows() حرکت می‌کند. این حلقه برای محاسبه توزیع هر حرف در متن‌ها اجرا می‌شود.

۷. char = row['alphabet']: حرف مورد نظر از ستون "alphabet" در هر ردیف از alphdf در متغیر char ذخیره می‌شود.

۸. if char in '.^$*+?()[{\|^-]\\':: اگر حرف مورد نظر یکی از حروف ویژه در عبارت‌های منظم (Regular Expression) باشد، با استفاده از کاراکتر اسکیپ (backslash) از حرف ویژه درست پس از آن فرار می‌شود.

۹. alphdf.loc[index, 'dist'] = concatdf.str.count(char).sum() / total_length: تعداد تکرار حرف مورد نظر در تمام متن‌ها با استفاده از تابع str.count() بدست آمده و به توزیع آن حرف در ستون "dist" مرتبط در alphdf افزوده می‌شود. سپس مقدار توزیع تقسیم بر طول کلی متن‌ها جهت نرمالیزه کردن محاسبه می‌شود.

بنابراین، این کد به تعداد تکرار هر حرف در متن‌های موجود در ستون "reviews" می‌پردازد و مقدار توزیع هر حرف را به صورت نرمال شده در alphdf ذخیره می‌کند. این اطلاعات می‌توانند برای تحلیل توزیع حروف در متن‌ها مفید باشند

این توزیع نهایتا توسط یک جدول نشان داده می‌شود، جدول زیر ۱۰ مورد پرتکرار آن را نمایش می‌دهد.

	alphabet	dist
23		0.206629
72	ا	0.100971
82	ی	0.063770
157	و	0.056599
165	ر	0.055766
123	ه	0.050385
44	ت	0.049472
8	د	0.048949
84	ب	0.047237
52	ن	0.046329

در ادامه توسط کد زیر دو لیست حروف الفبای اصلی و حروف دیگر از حروف استخراج شده در مراحل قبل استخراج می‌شوند:

حروف الفبای مورد قبول (ok_alphabet) دستی تعریف شده و با تفاضلش از کل کارکترها سایر حروف به دست می‌آیند.

```python
import copy

ok_alphabet = ' ' 'ابـپـتـثـجـچـحـخـدذرزژسـشـصـضـطـظـعـغـفـقـکـگـلـمـنـوهـی' 'ئـة ؤإأآءۀ' '#'
# '0123456789' 'abcdefghijklmnopqrstuvwxyz' 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
'.?!,،:;'

other_alphabet = copy.copy(alphabet)

for char in ok_alphabet:
    if char in other_alphabet:
        other_alphabet.remove(char)
```

سایر حروف:

–+()'٪.!

۰۱۲۳۴۵۶□□□□ٍـکیی؟؛،°¥{}ACEFGIKMOPRSTVWX_abcdefghiklmnoprstuvwxy=:۰۱۲۳۴۵۶۷۸۹/.

🙌🙄🙂😳😭😕😍😊😅😂😁💫💐👍👌👈🏻🌺🌹🌸🌷...۷۸۹٪۰۱۲۳۴۵۶۷۸۹

🥴🥰🙏

در نهایتا حروف زیر به عنوان الفبای موثر در تحلیل احساسات در نظر گرفته می‌شود:

ابـپـتـثـجـچـحـخـدذرزژسـشـصـضـطـظـعـغـفـقـکـگـلـمـنـوهـیـۀ ء آ أ ؤ إ ئـة#ACEFGIKMOPRSTVWXabcdefghiklmnoprstuv
wxy0123456789🌷🌸🌹🌺🏻👈👌👍💐💫😁😂😅😊😍😕😭😳🙂🙄🙌🙏🥰🥴

برای پیش پردازش متون از کتابخانه هضم استفاده شد:

```python
# import nltk
# nltk.tokenize()
import hazm
import re

stop_words = hazm.stopwords_list()
normalizer = hazm.Normalizer()
lemmatizer = hazm.Lemmatizer()
```

در کتابخانه‌ی هضم استاپ ورد ها و توابع نرمالیزر و لماتایزر به صورت بالا قابل استفاده هستند.

```python
# mappings
# EXTENDED ARABIC-INDIC DIGIT
mappings1 = [('۰', '0'), ('۱', '1'), ('۲', '2'), ('۳', '3'), ('۴', '4'),
('۵', '5'), ('۶', '6'), ('۷', '7'), ('۸', '8'), ('۹', '9')]

# ARABIC-INDIC DIGIT
mappings2 = [('٠', '0'), ('١', '1'), ('٢', '2'), ('٣', '3'), ('٤', '4'),
('٥', '5'), ('٦', '6'), ('٧', '7'), ('٨', '8'), ('٩', '9')]

# Equivalent letters
```

```
mappings3 = [
    ('ە', 'ه'), # ARABIC LETTER AE, ARABIC LETTER HEH
    ('ہ', 'ه'), # ARABIC LETTER HEH GOAL, ARABIC LETTER HEH
    ('ئ', 'ی'), # ARABIC LETTER HIGH HAMZA YEH, ARABIC LETTER FARSI YEH
    ('ھ', 'ه'), # ARABIC LETTER HEH DOACHASHMEE, ARABIC LETTER HEH
    ('ى', 'ی'), # ARABIC LETTER ALEF MAKSURA, ARABIC LETTER FARSI YEH
    ('ں', 'ن'), # ARABIC LETTER NOON GHUNNA, ARABIC LETTER NOON
    ('ے', 'ی'), # ARABIC LETTER YEH BARREE, ARABIC LETTER FARSI YEH
]
ok_alphabets_list = list(ok_alphabet)

def preprocess(text, mappings):

    # replace with mappings
    for x, y in mappings:
        text = text.replace(x, y)

    # Find and separate numbers attached to valid alphabets
    pattern = re.compile(r'(\d+)([' + ''.join(ok_alphabets_list) + r']+)')
    text = pattern.sub(r'\1 \2', text)

    # filter out stop words
    text = " ".join([t for t in text.split()]) #if t not in stop_words])


    # filter out characters not in ok_alphabet
    text = "".join([t for t in text if t in alphabets])

    text = normalizer.normalize(text)

    text = lemmatizer.lemmatize(text)

    return text


# method2, use apply method
df['reviews1'] = df['reviews'].apply(lambda text: preprocess(text=text,
mappings=mappings1+mappings2+mappings3))
```

این کد برای پیش‌پردازش متن‌ها در ستون "reviews" یک DataFrame استفاده می‌شود. هدف اصلی از این کد، تبدیل و تنظیم متن‌ها به یک شکل استاندارد است تا در مراحل بعدی تحلیل داده‌ها بهتر انجام شود. در زیر، هر بخش از کد توضیح داده شده است:

۱. mappings1, mappings2, mappings3: این تاپل‌ها حاوی مپینگ‌هایی از کاراکترهای یونیکد به کاراکترهای استاندارد لاتین (اعداد) یا به یکدیگر هستند. به عبارت دیگر، این مپینگ‌ها اعداد عربی به اعداد لاتین و همچنین برخی از حروف عربی به حروف لاتین یا به یکدیگر را انجام می‌دهند.

۲. ok_alphabets_list = list(ok_alphabet): لیستی از حروفی که در ok_alphabet موجود هستند، ایجاد می‌شود. این لیست برای تعیین حروف مجاز برای استفاده در متن‌ها استفاده می‌شود.

۳. preprocess(text, mappings): این تابع متن و مپینگ‌های مشخص شده را به عنوان ورودی می‌گیرد و متن را با توجه به این مپینگ‌ها پیش‌پردازش می‌کند. مراحل پیش‌پردازش عبارتند از:

- جایگزینی حروف یا اعداد با مپینگ‌های مشخص شده.
- جدا کردن اعداد از حروف مجاز. به عبارت دیگر، اگر یک عدد به یک حرف مجاز وصل شده باشد، این کد این دو را جدا می‌کند.
- حذف کلمات متوقف.(stop words)
- حذف کاراکترهای غیرمجاز با توجه به ok_alphabet.
- نرمال‌سازی متن (normalization) با استفاده از توابع معین.
- لماتایز کردن متن (lemmatization) با استفاده از توابع معین.

۴. df['reviews1'] = df['reviews'].apply(lambda text: preprocess(text=text, mappings=mappings1+mappings2+mappings3)): از این بخش، تابع preprocess بر روی هر متن در ستون "reviews" اجرا می‌شود. نتیجه این پیش‌پردازش در یک ستون جدید به نام "reviews1" در DataFrame df ذخیره می‌شود.

در کل، این کد برای تبدیل و تنظیم متن‌ها به یک شکل مشخص و قابل تحلیل در مراحل بعدی استفاده می‌شود. این مراحل شامل جدا کردن اعداد از حروف، حذف کلمات متوقف، تبدیل حروف یونیکد به حروف لاتین یا به یکدیگر، و اعمال نرمالیزه و لماتایزه به متن‌ها می‌شود

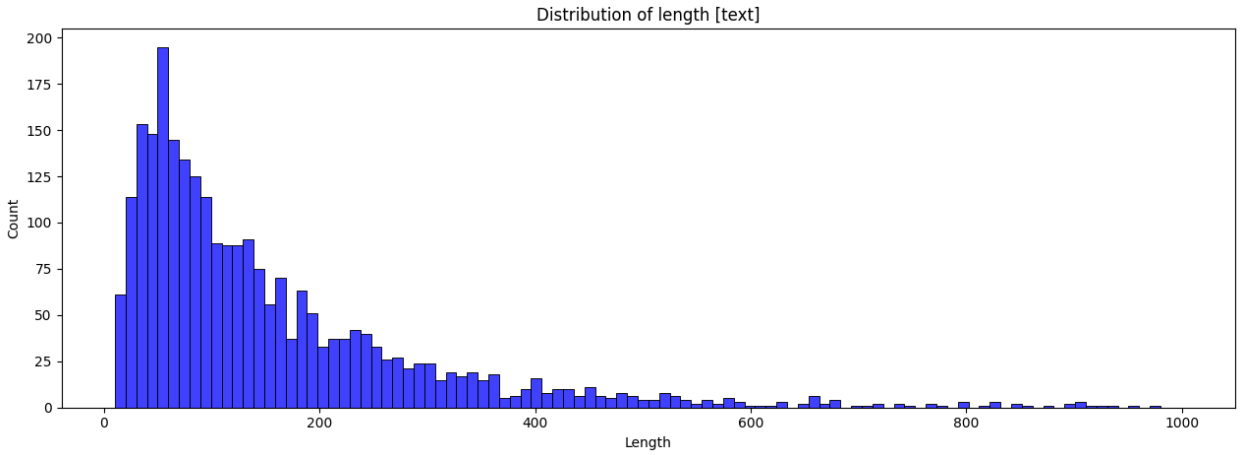
در زیر یک نمونه از اعمال پیش پردازش بر متن نظر نشان داده شده است:

متن اصلی: اتاقش خیلی خوب بود دلباز و تمیز، من چون گرمایی هستم اتاق بالکن دار گرفتیم، به صبحانش ۷از ۱۰میدم در کل راضی بودم کیفیت و تنوع صبحانه قابل قبول بود.

متن پیش پردازش شده: اتاقش خیلی خوب بود دلباز و تمیز من چون گرمایی هستم اتاق بالکن دار گرفتیم به صبحانش ۷ از ۱۰ میدم در کل راضی بودم کیفیت و تنوع صبحانه قابل قبول بود

در ادامه دیتاست از نظر توزیع حروف استفاده شده در نظرات بررسی شد:

حدقال کارکتر یک نظر ۱۰ و حداکثر ۱۹۸۷ بود. نمودار زیر توزیع نظرات شامل بین ۱۰ تا ۱۰۰۰ کارکتر را نشان می‌دهد.



در ادامه‌ی پیش پردازش، ستون های عددی که همان برچسب ها هستند به اعداد استاندارد تبدیل شدند، چرا که فرمت بعضی از آن ها اینتجر و برخی فلوت بود و لازم بود پردازش شوند:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Assuming you have already loaded your DataFrame as 'df'
# df = ...

# List of output columns you want to convert
output_columns = ['Tangibles', 'Reliability', 'Empathy', 'Assurance',
'Responsiveness']

# Create a LabelEncoder instance
label_encoder = LabelEncoder()

# Apply label encoding to each output column
for col in output_columns:
    df[col] = label_encoder.fit_transform(df[col])

# Print the updated DataFrame
df.head()
```

	reviews	Tangibles	Reliability	Empathy	Assurance	Responsiveness	reviews1
0	...اتاق ها خیلی معمولی، پرسنل بی اعصاب، صبحانه مع	1	3	4	0	3	...اتاق‌ها خیلی معمولی پرسنل بی اعصاب صبحانه معمو
1	...هتل با توجه به قیمت و تبلیغات بالا راضی کننده	0	0	4	0	3	... هتل با توجه به قیمت و تبلیغات بالا راضی کننده
2	کارکنان بسیار خوب و مودب	3	3	4	2	3	کارکنان بسیار خوب و مودب
3	...اتاقش خیلی خوب بود دلباز و تمیز، من چون گرمابی	2	3	4	3	3	... اتاقش خیلی خوب بود دلباز و تمیز من چون گرمابی
4	... برخورد پرسنل هتل عالی بود.موسیقی زنده در کافی	2	3	4	2	3	...برخورد پرسنل هتل عالی بودموسیقی زنده در کافی ش

در نهایت بررسی شد که مقادیر منحصر به فرد برا هر برچسب به چه صورت است:

```
import pandas as pd

# Assuming you have already loaded your DataFrame as 'df'
# df = ...

# List of output columns
output_columns = ['Tangibles', 'Reliability', 'Empathy', 'Assurance',
'Responsiveness']

# Count distinct values in each output column
distinct_counts = df[output_columns].nunique()

# Print the distinct value counts for each column
print("Distinct Value Counts in Each Output Column:")
for col, count in distinct_counts.items():
    print(f"{col}: {count}")
```

```
Distinct Value Counts in Each Output Column:
Tangibles: 4
Reliability: 4
Empathy: 5
Assurance: 4
Responsiveness: 5
```

امبدینگ

با استفاده از کد زیر یک امبدینگ ورد۲وک بر روی مجموعه‌ی داده آموزش داده شد:

```
from gensim.models import Word2Vec

# Combine all the sentences from the 'article1' and 'summary1' columns
train_sentences = df['reviews1'].tolist()
# Convert the sentences to lists of words
train_sentences = [sentence.split() for sentence in train_sentences]

# Create a Word2Vec model
model = Word2Vec(train_sentences, vector_size=300, window=5, min_count=1,
workers=4)

# Train the model
model.train(train_sentences, total_examples=model.corpus_count, epochs=30)

# Save the model
#model.save("word2vec.model")
```

سایز بردار امبدینگ ۳۰۰ در نظر گرفته شد.

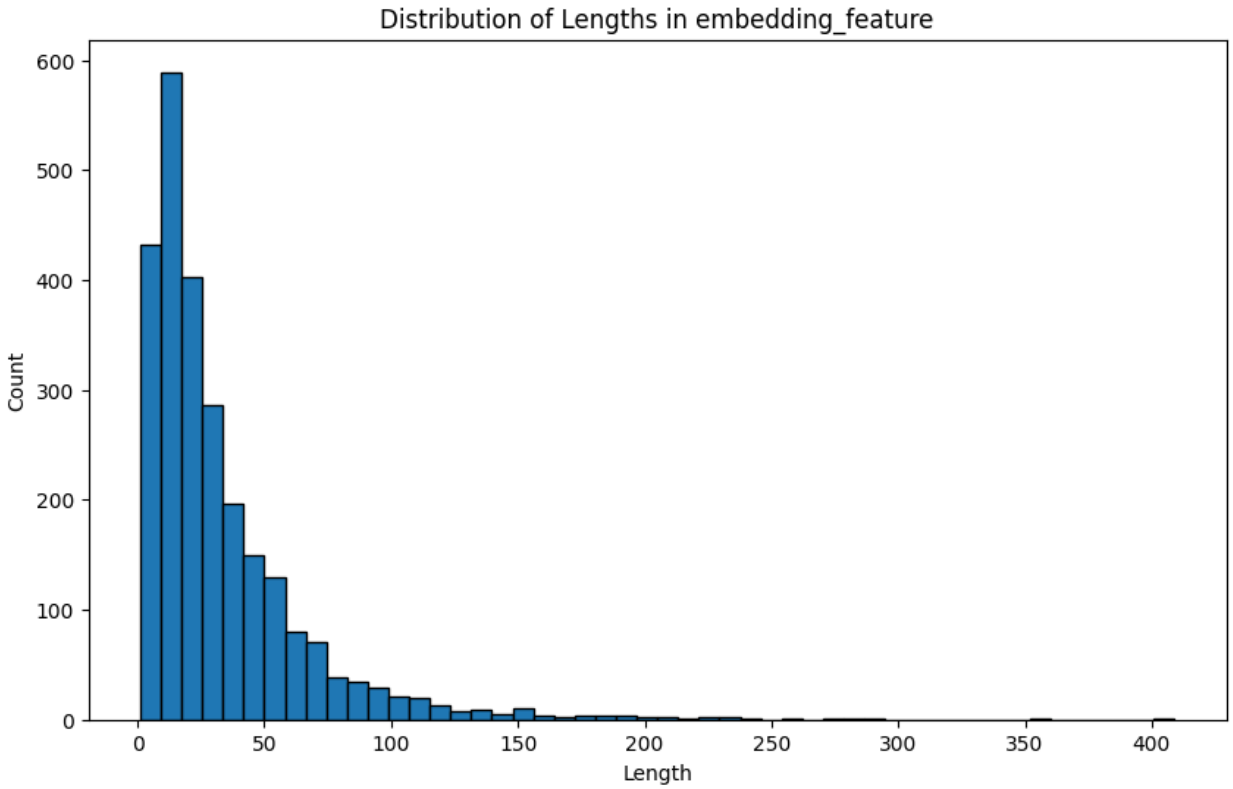
سپس با استفاده از توکنایزر تنسور فلو، نظرات به صورت توکن و توکن ها به ایندکس مرتبط در واژگان تبدیل شدند. بدین ترتیب مطابق کد زیر هر نظر که لیستی از کلمات است به لیستی از اعداد تبدیل می‌شود که هر عدد شماره‌ی مرتبط به هر کلمه در واژگان تهیه شده توسط توکنایزر است. (ستون آخر در تصویر زیر)

```
from tensorflow.keras.preprocessing.text import Tokenizer

# ساخت واژگان (vocabulary)
oov_tok='<oov>'
oov_token=oov_tok
tokenizer = Tokenizer(oov_token=oov_tok)#,num_words = vocab_size)
#tokenizer = Tokenizer(num_words = vocab_size)

tokenizer.fit_on_texts(df['reviews1'])
df['reviews_indices'] = tokenizer.texts_to_sequences(df['reviews1'])
sequences = tokenizer.texts_to_sequences(df['reviews1'])

word_index = tokenizer.word_index
df.head()
```

	reviews	Tangibles	Reliability	Empathy	Assurance	Responsiveness	reviews1	reviews_indices
0	اتاق ها خیلی معمولی، پرسنل بی اعصاب، صبحانه مع...	1	3	4	0	3	اتاق‌ها خیلی معمولی پرسنل بی اعصاب صبحانه معمو...	[51, 16, 131, 18, 98, 1007, 14, 131, 5, 50, 3,...
1	هتل با توجه به قیمت و تبلیغات بالا راضی کننده...	0	0	4	0	3	هتل با توجه به قیمت و تبلیغات بالا راضی کننده ...	[3, 12, 67, 6, 56, 2, 2588, 116, 79, 295, 28, ...
2	کارکنان بسیار خوب و مودب	3	3	4	2	3	کارکنان بسیار خوب و مودب	[19, 13, 7, 2, 82]
3	اتاقش خیلی خوب بود دلباز و تمیز، من چون گرمابی...	2	3	4	3	3	اتاقش خیلی خوب بود دلباز و تمیز من چون گرمابی ...	[1332, 16, 7, 4, 771, 2, 32, 26, 108, 3596, 48...
4	برخورد پرسنل هتل عالی بود.موسیقی زنده در کافی ...	2	3	4	2	3	برخورد پرسنل هتل عالی بودموسیقی زنده در کافی ش...	[21, 18, 3, 11, 3597, 1091, 5, 112, 174, 2, 33...

با استفاده از کد زیر تعداد کلمات موجود در هر نظر شمارش شد:

```
import matplotlib.pyplot as plt

# محاسبه طول هر مقدار در ستون
lengths = df['reviews_indices'].apply(len)
# رسم نمودار توزیع طول مقادیر
plt.figure(figsize=(10, 6))
plt.hist(lengths, bins=50, edgecolor='black')
plt.xlabel('Length')
plt.ylabel('Count')
plt.title('Distribution of Lengths in embedding_feature')
plt.show()
```



همانطور که دیده می‌شود اغلب نظرات شامل کمتر از ۲۰۰ کلمه هستند. این تعداد در مراحل جلوتر به عنوان حداکثر طول جملات در نظر گرفته شد.

تا به اینجا هر نظر به لیستی از اعداد تبدیل شده است که هر عدد نمایانگر یک کلمه است. اما هر نظر دارای طول متفاوتی است و برای آموزش شبکه عصبی باید همه‌ی ورودی‌ها اندازه‌ی یکسان داشته باشند. روش مقابله با این مسئله پدینگ کردن است. بدین صورت طول همه نظرات به حداکثر سایز تعریف شده (۲۰۰) میرسد. اگر نظری کلمات بیشتر از ۲۰۰ کلمه داشته باشد، کلمات اضافه تر از ۲۰۰ آن حذف می‌شوند و اگر نظری کمتر از ۲۰۰ داشته باشد، طول آن با افزودن مقادیر خالی (pad) به ۲۰۰ میرسد کد زیر این کار را انجام میدهد. نوع post استفاده شده است بنابراین نظراتی که طول بیش از ۲۰۰ داشته باشند از انتهایشان کلمه حذف میشود و نظراتی که طول کمتر از ۲۰۰ دارند به انتهایشان اضافه میشود.

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
padding_type = 'post'  # Pad sequences at the end
truncating_type = 'post'  # Truncate sequences at the end
max_length = 200
# Padding sequences
padded_sequences = pad_sequences(sequences, maxlen=max_length,
padding=padding_type, truncating=truncating_type)
```

```
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
train_df, test_df = train_test_split(df, test_size=0.05, random_state=42)
train_df.head()
```

با دستور بالا دیتاست به دو مجموعه آموزش و تست تقسیم می‌شود. با توجه به اینکه شبکه عصبی به تعداد زیادی داده برای آموزش lstm نیاز دارد و تعداد داده‌ی موجود کم است، تنها ۵ درصد از داده ها برای تست انتخاب شدند. که ۱۲۸ نظر می‌شوند.

مدل اولیه به صورت زیر ساخته شد:

```
from keras.layers import Input, Embedding, LSTM, Dense, Bidirectional
from keras.models import Model
# encoder input model
inputs = Input(shape=(max_length,))
#encoder1 = Embedding(input_dim=num_words,
output_dim=embedding_dim)(inputs)
encoder1 = Embedding(input_dim=num_words, output_dim=embedding_dim,
weights=[word_vectors.vectors], trainable=False)(inputs)
encoder2 = LSTM(256, return_sequences=True)(encoder1)  # Add
return_sequences=True
# Bidirectional LSTM layer
encoder2_bilstm = Bidirectional(LSTM(128))(encoder2)

#outputs= Dense(5, activation='softmax')
# Define output layers for each sentiment column
output_layers = []
for col in output_columns:
    if col=='Tangibles' or col=='Reliability'or col=='Assurance' :
      x=4
    elif col=='Empathy' or col=='Responsiveness':
      x=5
    output_layer1 = Dense(64, activation='relu')(encoder2_bilstm)
    output_layer2 = Dense(32, activation='relu')(output_layer1)
    output_layer = Dense(x, activation='softmax', name=col)(output_layer2)
    #output_layer = Dense(x, activation='softmax',
name=col)(encoder2_bilstm) #label_encoder.classes_.shape[0]

    output_layers.append(output_layer)

# Tie it together
model = Model(inputs=inputs, outputs=output_layers)
model.summary()
```

این مدل شامل یک لایه انکودر یعنی بازگشایی محتوای متن است که این کار توسط شبکه‌ی lstm و bilstm انجام می‌شود و سپس خروجی توسط لایه‌های دیگر تولید می‌شود که به آن دیکودر نیز می‌توان گفت. لایه‌ی انکودر برای پیش بینی همه ۵ ستون یکسان است ولی پس از آن برای هر ستون دیکودر مجزا در نظر گرفته شده است.

این کد یک معماری شبکه عصبی بازگشتی (RNN) با لایه‌های LSTM و Bidirectional برای ایجاد یک مدل پیش‌بینی احساسات (sentiment analysis) از متن‌ها با استفاده از فریم‌ورک Keras را توصیف می‌کند. در ادامه، هر بخش از کد توضیح داده شده است:

۱. from keras.layers import Input, Embedding, LSTM, Dense, Bidirectional: در این بخش، از Keras ، لایه‌های مختلف مورد نیاز برای ساخت مدل شبکه عصبی معرفی می‌شوند. این لایه‌ها شامل ورودی (Input)، لایه تعبیه (Embedding) ، لایه LSTM ، لایه تمام متصل (Dense) و لایه Bidirectional هستند.

۲. inputs = Input(shape=(max_length,)): یک ورودی مدل به نام inputs با ابعاد (max_length,) ایجاد می‌شود max_length .طول بیشینه دنباله ورودی را نشان می‌دهد.

۳. encoder1 = Embedding(input_dim=num_words, output_dim=embedding_dim, weights=[word_vectors.vectors], trainable=False)(inputs): لایه تعبیه (Embedding) ایجاد می‌شود که برای تبدیل واژه‌ها به بردارهای تعبیه‌شده (embeddings) با ابعاد embedding_dim استفاده می‌شود. این لایه از بردارهای تعبیه‌شده قبل‌آموزش‌دیده word_vectors.vectors استفاده می‌کند و وزن‌های آن غیرقابل آموزش (trainable=False) تنظیم می‌شوند.

۴. encoder2 = LSTM(256, return_sequences=True)(encoder1): لایه LSTM با ۲۵۶ واحد حافظه ایجاد می‌شود. این لایه به عنوان لایه اول در شبکه از دنباله خروجی‌های زمانی (sequences) خروجی تولید می‌کند.(return_sequences=True)

۵. encoder2_bilstm = Bidirectional(LSTM(128))(encoder2): لایه Bidirectional LSTM ایجاد می‌شود که دو لایه LSTM با ۱۲۸ واحد حافظه را به صورت دوطرفه پیوسته می‌کند. این لایه به طور معمول از جمع‌شدن خروجی‌های دو طرفه LSTM برای ساخت نمایشی چگال از دنباله ورودی استفاده می‌کند.

۶. output_layers = []: یک لیست خالی برای ذخیره لایه‌های خروجی ایجاد می‌شود.

۷. for col in output_columns: ...: یک حلقه for شروع می‌شود که بر روی اعمدهٔ مختلف خروجی پیمانه‌های مختلف اجرا می‌شود. این حلقه تمام مراحل برای ایجاد لایه‌های خروجی مورد نیاز را اجرا می‌کند.

۸. output_layer1 = Dense(64, activation='relu')(encoder2_bilstm): لایه تمام متصل (Dense)ایجاد می‌شود که به عنوان لایه اولیه خروجی در نظر گرفته می‌شود. این لایه ۶۴ نورون دارد و از تابع فعال‌سازی ReLU استفاده می‌کند.

۹. output_layer2 = Dense(32, activation='relu')(output_layer1): لایه تمام متصل دیگری با ۳۲ نورون و تابع فعال‌سازی ReLU ایجاد می‌شود.

۱۰. output_layer = Dense(x, activation='softmax', name=col)(output_layer2): لایه تمام متصل نهایی برای خروجی مورد نظر ایجاد می‌شود. تعداد نورون‌ها در این لایه با توجه به نوع خروجی مشخص می‌شود (با توجه به مقادیر xکه به تعداد دسته‌ها مرتبط با هر خروجی است). تابع فعال‌سازی Softmax برای تبدیل خروجی به احتمالات مربوط به دسته‌ها استفاده می‌شود. هر لایه خروجی با نام مشخص در لیست output_layersقرار می‌گیرد.

۱۱. model = Model(inputs=inputs, outputs=output_layers): مدل با ورودی‌ها و خروجی‌های مشخص ساخته می‌شود.

۱۲. model.summary(): خلاصه‌ای از معماری مدل به صورت مشخصات لایه‌ها و تعداد پارامترهای قابل آموزش نمایش داده می‌شود

معماری مدل در تصویر صفحه بعد نمایش داده شده است. حدود ۳ میلیون و ۵۰۰ هزار پارامتر در این مدل وجود دارد که تنها ۱ میلیون از آن‌ها پارامترهای قابل آموزش شبکه عصبی هستند و مابقی مربوط به امبدینگ ورد۲وک می‌باشند.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

با کد بالا مدل کامپایل می‌شود و از بهینه ساز اَدَم و تابع هزینه‌ی اسپارس کتگوریکال کراس آنتروپی استفاده شده است. دلیل این انتخاب این است که برچسب ها به صورت وان هات نمی‌باشند پس باید هزینه اسپارس باشد و همچنین به دلیل وجود چندین برچسب از به جای باینری از کتگوریکال کراس آنتروپی استفاده می‌شود.

همچنین ۱۰ درصد از مجموعه داده‌ی آموزش به عنوان validation برای اعتبار سنجی مدل استفاده شده است. و ۱۰۰ ایپاک (با شرط توقف افزایش خطای مجموعه داده‌ی اعتبارسنجی به تعداد ۵ ایپاک) آموزش داده خواهد شد.

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 200)]        0           []

 embedding_1 (Embedding)        (None, 200, 300)     2487600     ['input_2[0][0]']

 lstm_2 (LSTM)                  (None, 200, 256)     570368      ['embedding_1[0][0]']

 bidirectional_1 (Bidirectional  (None, 256)         394240      ['lstm_2[0][0]']
 )

 dense_10 (Dense)               (None, 64)           16448       ['bidirectional_1[0][0]']

 dense_12 (Dense)               (None, 64)           16448       ['bidirectional_1[0][0]']

 dense_14 (Dense)               (None, 64)           16448       ['bidirectional_1[0][0]']

 dense_16 (Dense)               (None, 64)           16448       ['bidirectional_1[0][0]']

 dense_18 (Dense)               (None, 64)           16448       ['bidirectional_1[0][0]']

 dense_11 (Dense)               (None, 32)           2080        ['dense_10[0][0]']

 dense_13 (Dense)               (None, 32)           2080        ['dense_12[0][0]']

 dense_15 (Dense)               (None, 32)           2080        ['dense_14[0][0]']

 dense_17 (Dense)               (None, 32)           2080        ['dense_16[0][0]']

 dense_19 (Dense)               (None, 32)           2080        ['dense_18[0][0]']

 Tangibles (Dense)              (None, 4)            132         ['dense_11[0][0]']

 Reliability (Dense)            (None, 4)            132         ['dense_13[0][0]']

 Empathy (Dense)                (None, 5)            165         ['dense_15[0][0]']

 Assurance (Dense)              (None, 4)            132         ['dense_17[0][0]']

 Responsiveness (Dense)         (None, 5)            165         ['dense_19[0][0]']

==================================================================================================
Total params: 3,545,574
Trainable params: 1,057,974
Non-trainable params: 2,487,600
__________________________________________________________________________________________________
```

با استفاده از کدهای زیر مدل آموزش داده می‌شود:

```python
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Convert your text data to sequences and pad them
padding_type = 'post'  # Pad sequences at the end
truncating_type = 'post'  # Truncate sequences at the end
```

```
max_length = 200
# Padding sequences
# Assuming you have already defined a tokenizer
input_column="reviews1"
train_sequences = tokenizer.texts_to_sequences(train_df[input_column])
train_padded_sequences = pad_sequences(train_sequences, maxlen=max_length,
padding=padding_type, truncating=truncating_type)
test_sequences = tokenizer.texts_to_sequences(test_df[input_column])
test_padded_sequences = pad_sequences(test_sequences, maxlen=max_length,
padding=padding_type, truncating=truncating_type)

# Train the model
from keras.callbacks import EarlyStopping

# Define the EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Train the model with the EarlyStopping callback
history = model.fit(train_padded_sequences, [train_df[col] for col in
output_columns],
                    epochs=100, batch_size=512, validation_split=0.10,
                    callbacks=[early_stopping])

# Evaluate the model
results = model.evaluate(test_padded_sequences, [test_df[col] for col in
output_columns])
```

```
5/5 [==============================] - 1s 220ms/step - loss: 2.4736 - Tangibles_loss: 0.6616 - Reliability_loss: 0.5632 - Empathy_loss: 0.1488 - Assurance_loss
Epoch 24/100
5/5 [==============================] - 1s 222ms/step - loss: 2.3914 - Tangibles_loss: 0.6283 - Reliability_loss: 0.5578 - Empathy_loss: 0.1453 - Assurance_loss
Epoch 25/100
5/5 [==============================] - 1s 230ms/step - loss: 2.3242 - Tangibles_loss: 0.6105 - Reliability_loss: 0.5471 - Empathy_loss: 0.1457 - Assurance_loss
Epoch 26/100
5/5 [==============================] - 1s 226ms/step - loss: 2.2912 - Tangibles_loss: 0.5914 - Reliability_loss: 0.5439 - Empathy_loss: 0.1472 - Assurance_loss
Epoch 27/100
5/5 [==============================] - 1s 230ms/step - loss: 2.2406 - Tangibles_loss: 0.5758 - Reliability_loss: 0.5390 - Empathy_loss: 0.1410 - Assurance_loss
Epoch 28/100
5/5 [==============================] - 1s 231ms/step - loss: 2.1929 - Tangibles_loss: 0.5620 - Reliability_loss: 0.5325 - Empathy_loss: 0.1426 - Assurance_loss
Epoch 29/100
5/5 [==============================] - 1s 227ms/step - loss: 2.1312 - Tangibles_loss: 0.5329 - Reliability_loss: 0.5244 - Empathy_loss: 0.1423 - Assurance_loss
Epoch 30/100
5/5 [==============================] - 1s 222ms/step - loss: 2.0853 - Tangibles_loss: 0.5129 - Reliability_loss: 0.5162 - Empathy_loss: 0.1408 - Assurance_loss
Epoch 31/100
5/5 [==============================] - 1s 226ms/step - loss: 2.0753 - Tangibles_loss: 0.5298 - Reliability_loss: 0.5111 - Empathy_loss: 0.1408 - Assurance_loss
Epoch 32/100
5/5 [==============================] - 1s 228ms/step - loss: 2.1075 - Tangibles_loss: 0.5531 - Reliability_loss: 0.5119 - Empathy_loss: 0.1434 - Assurance_loss
Epoch 33/100
5/5 [==============================] - 1s 233ms/step - loss: 2.0080 - Tangibles_loss: 0.5052 - Reliability_loss: 0.4970 - Empathy_loss: 0.1365 - Assurance_loss
Epoch 34/100
5/5 [==============================] - 1s 238ms/step - loss: 1.9357 - Tangibles_loss: 0.4577 - Reliability_loss: 0.4914 - Empathy_loss: 0.1368 - Assurance_loss
4/4 [==============================] - 0s 25ms/step - loss: 2.6701 - Tangibles_loss: 0.7932 - Reliability_loss: 0.5695 - Empathy_loss: 0.3036 - Assurance_loss:
```

همانطور که دیده می‌شود، مدل ۳۴ ایپاک آموزش دیده و به دلیل افزایش هزینه در مجوعه داده‌ی ولیدیشن متوقف شده است و بهترین مدل با بهترین وزن‌ها بازگشت داده شده است.

با استفاده از کد زیر نتیجه‌ی معیار دقت بر روی مجموعه‌ی داده‌ی تست به دست آمده است:

```
# Evaluate the model
evaluation_results = model.evaluate(test_padded_sequences, [test_df[col]
for col in output_columns])

# Print the evaluation results
print("Evaluation Results:")
print(f"Total Loss: {evaluation_results[0]}")
for i, col in enumerate(output_columns):
    print(f"{col} Accuracy: {round(evaluation_results[i+6]*100,2)}%")
```

```
Evaluation Results:
Total Loss: 2.670078992843628
Tangibles Accuracy: 72.66%
Reliability Accuracy: 78.12%
Empathy Accuracy: 93.75%
Assurance Accuracy: 80.47%
Responsiveness Accuracy: 84.38%
```

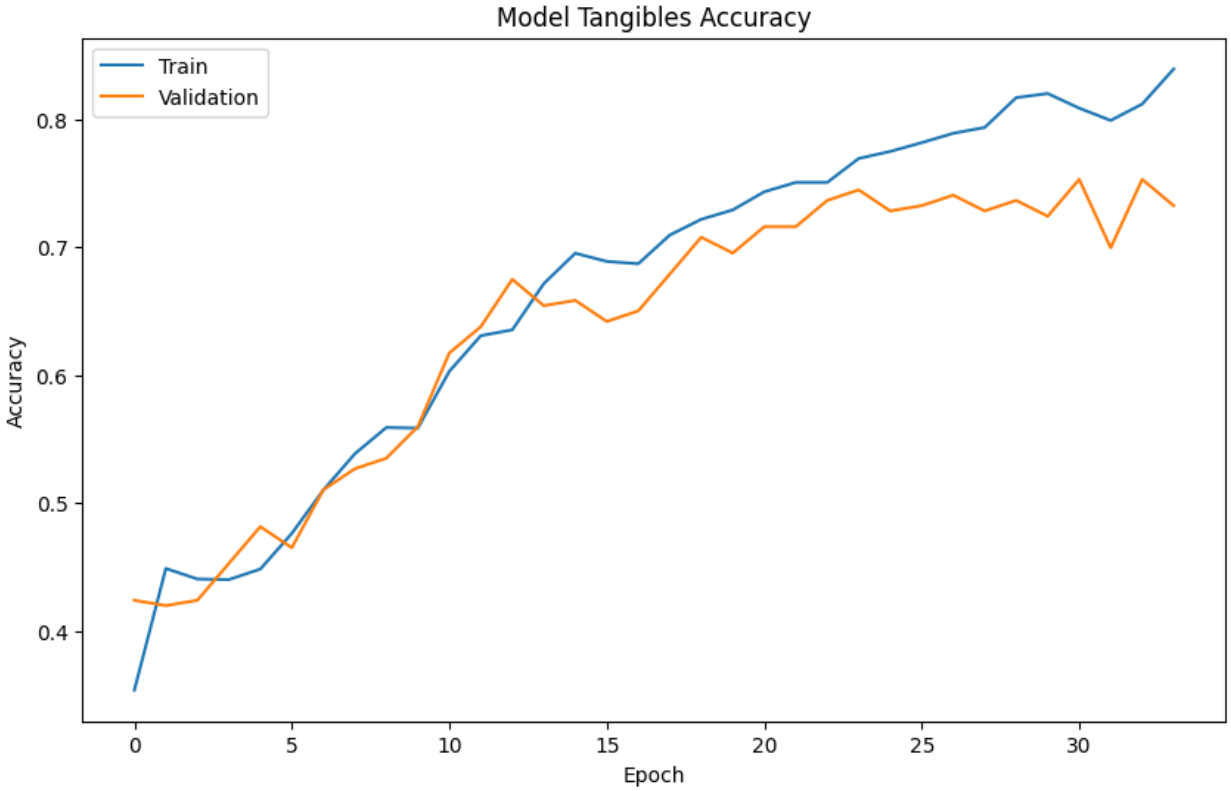
سایر معیارها نیز به صورت زیر می‌باشند:

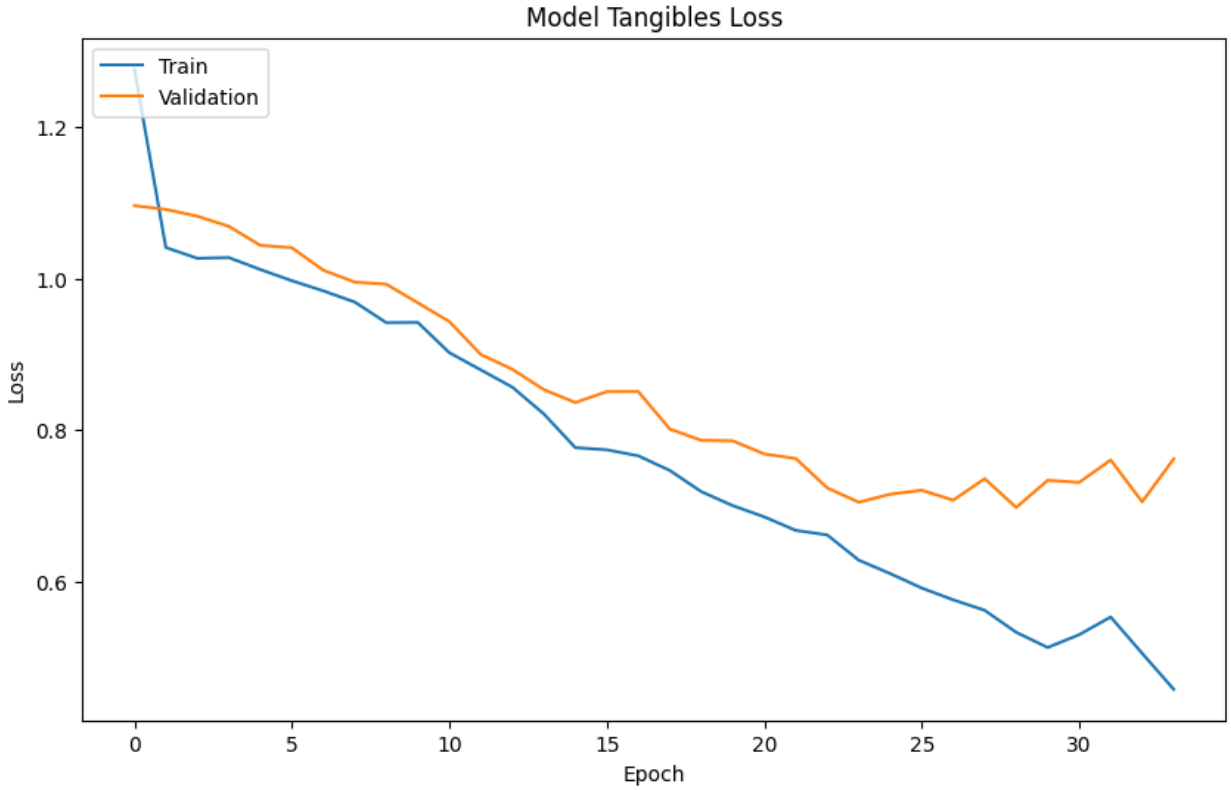
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Predict on test data
predictions = model.predict(test_padded_sequences)

# Iterate through output columns
for i, col in enumerate(output_columns):
    true_labels = test_df[col]
    pred_labels = np.argmax(predictions[i], axis=1)  # Convert
probabilities to class labels
    accuracy = accuracy_score(true_labels, pred_labels)
    precision = precision_score(true_labels, pred_labels,
average='weighted')
    recall = recall_score(true_labels, pred_labels, average='weighted')
    f1 = f1_score(true_labels, pred_labels, average='weighted')

    print(f"Metrics for {col}:")
    print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall:
{recall:.4f}, F1-score: {f1:.4f}")
```

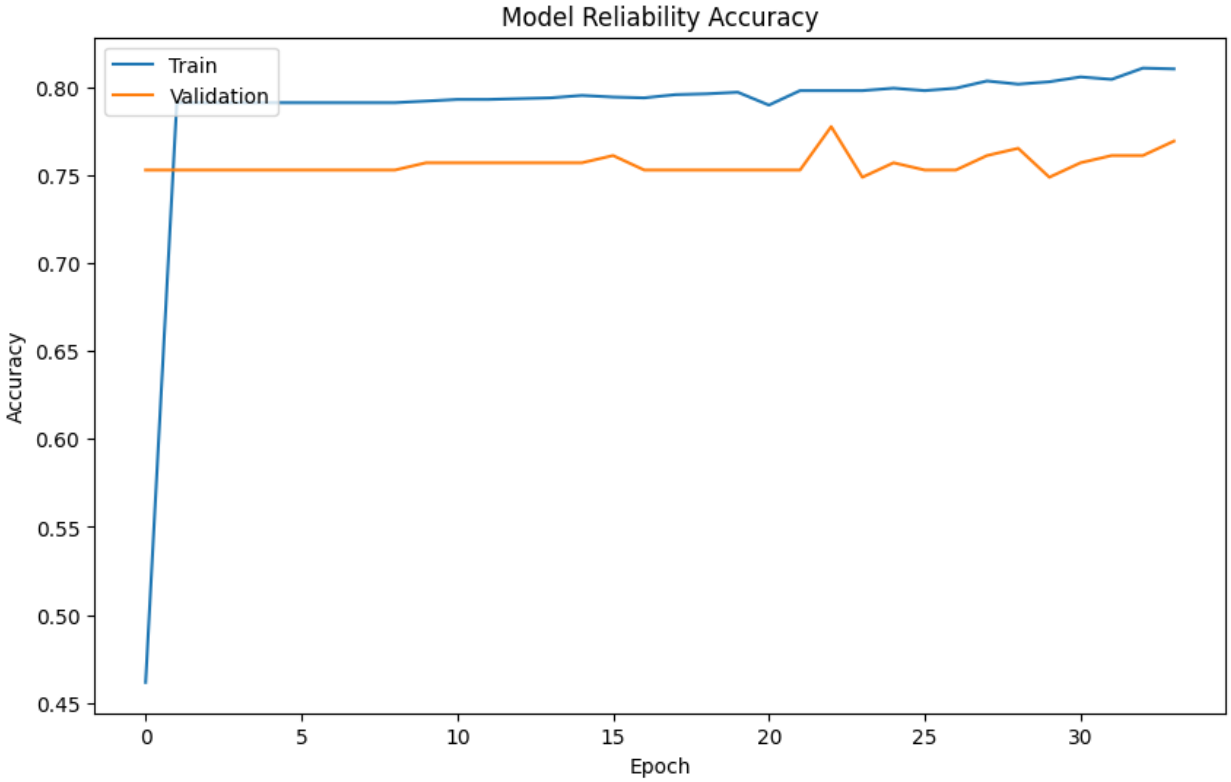
```
Metrics for Tangibles:
Accuracy: 0.7266, Precision: 0.7092, Recall: 0.7266, F1-score: 0.7026
Metrics for Reliability:
Accuracy: 0.7812, Precision: 0.6213, Recall: 0.7812, F1-score: 0.6922
Metrics for Empathy:
Accuracy: 0.9375, Precision: 0.8789, Recall: 0.9375, F1-score: 0.9073
Metrics for Assurance:
Accuracy: 0.8047, Precision: 0.7947, Recall: 0.8047, F1-score: 0.7846
Metrics for Responsiveness:
Accuracy: 0.8438, Precision: 0.8259, Recall: 0.8438, F1-score: 0.7862
```

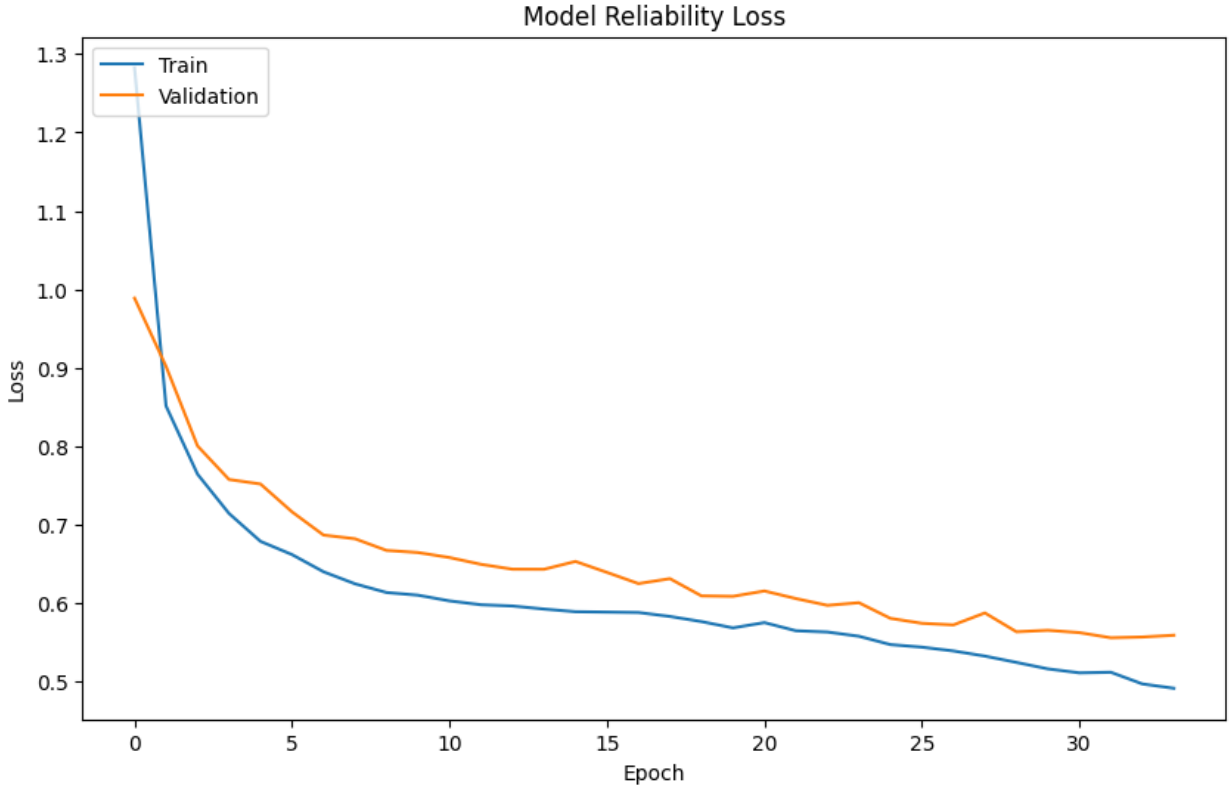
با کد زیر نمودارهای لازم رسم می‌شوند:

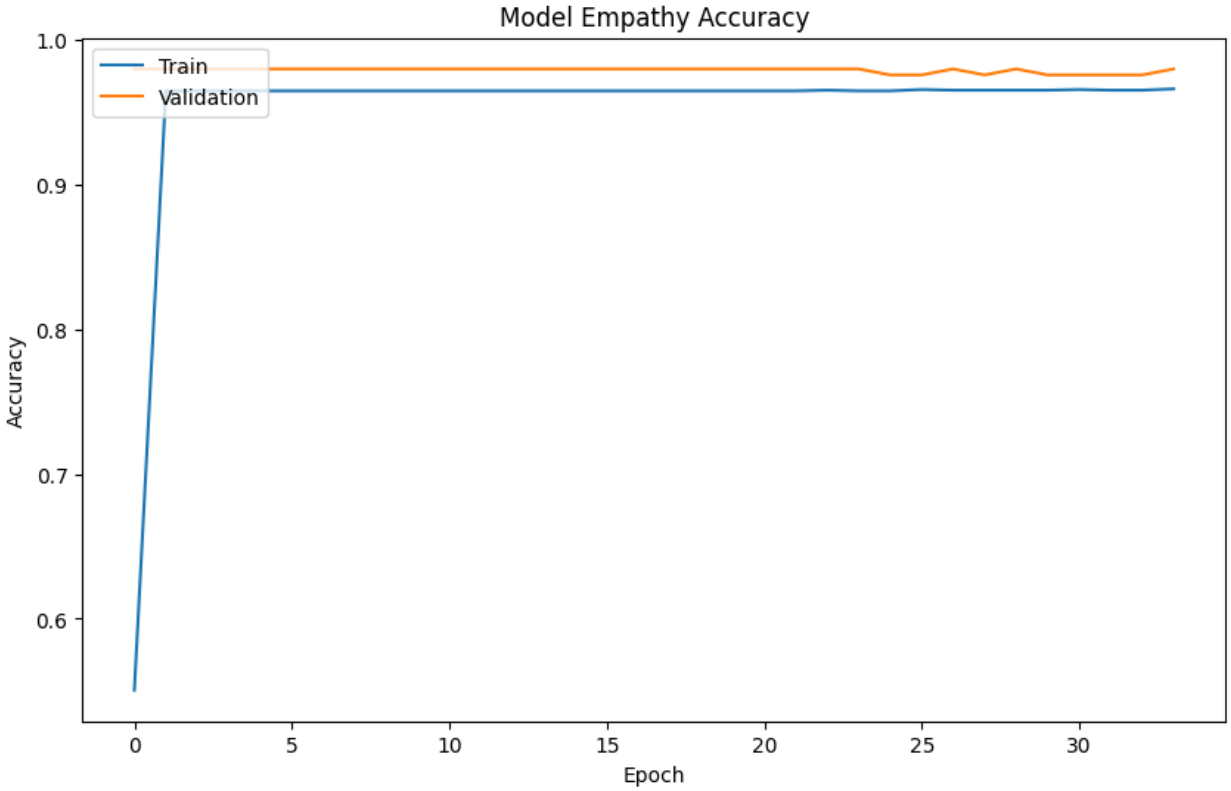
```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values for each output column
for i, col in enumerate(output_columns):
    plt.figure(figsize=(10, 6))
    plt.plot(history.history[f'{col}_accuracy'])
    plt.plot(history.history[f'val_{col}_accuracy'])
    plt.title(f'Model {col} Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.show()

    # Plot training & validation loss values for each output column
    plt.figure(figsize=(10, 6))
    plt.plot(history.history[f'{col}_loss'])
    plt.plot(history.history[f'val_{col}_loss'])
    plt.title(f'Model {col} Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.show()
```

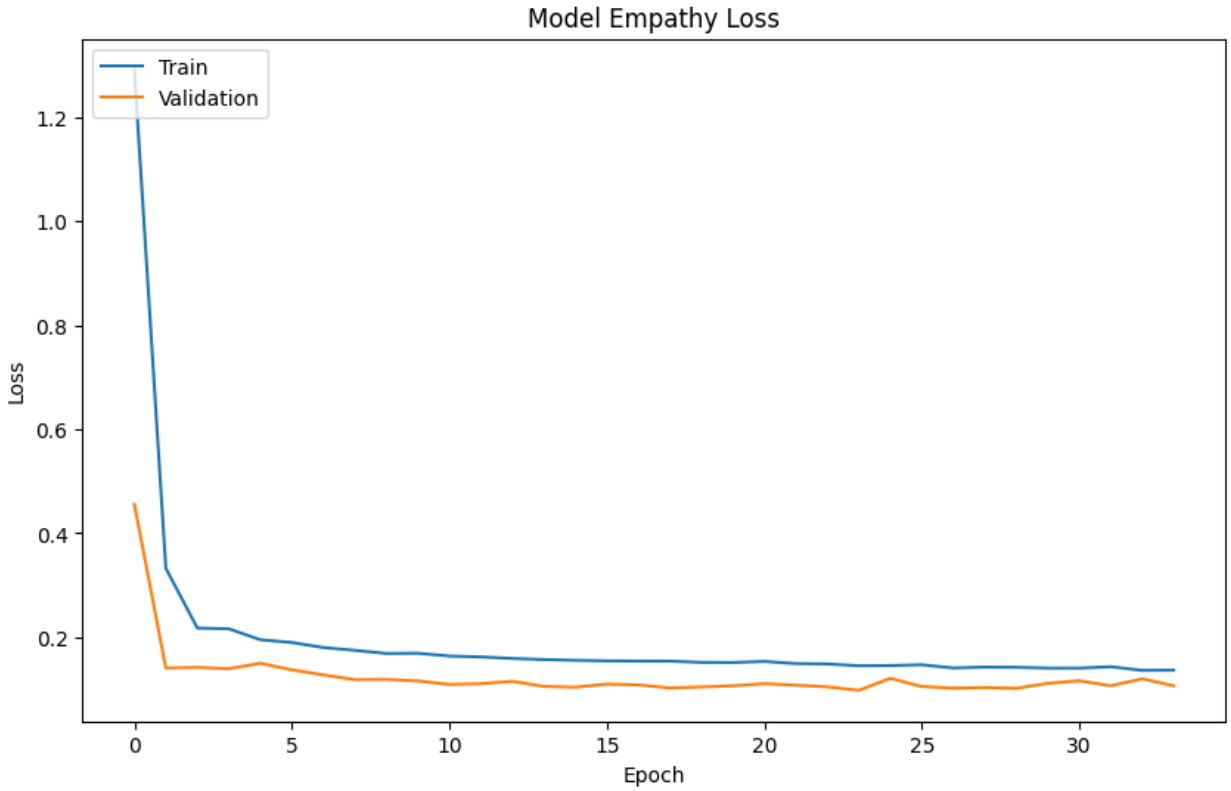

Model Tangibles Accuracy
Train
Validation
Accuracy
Epoch
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30

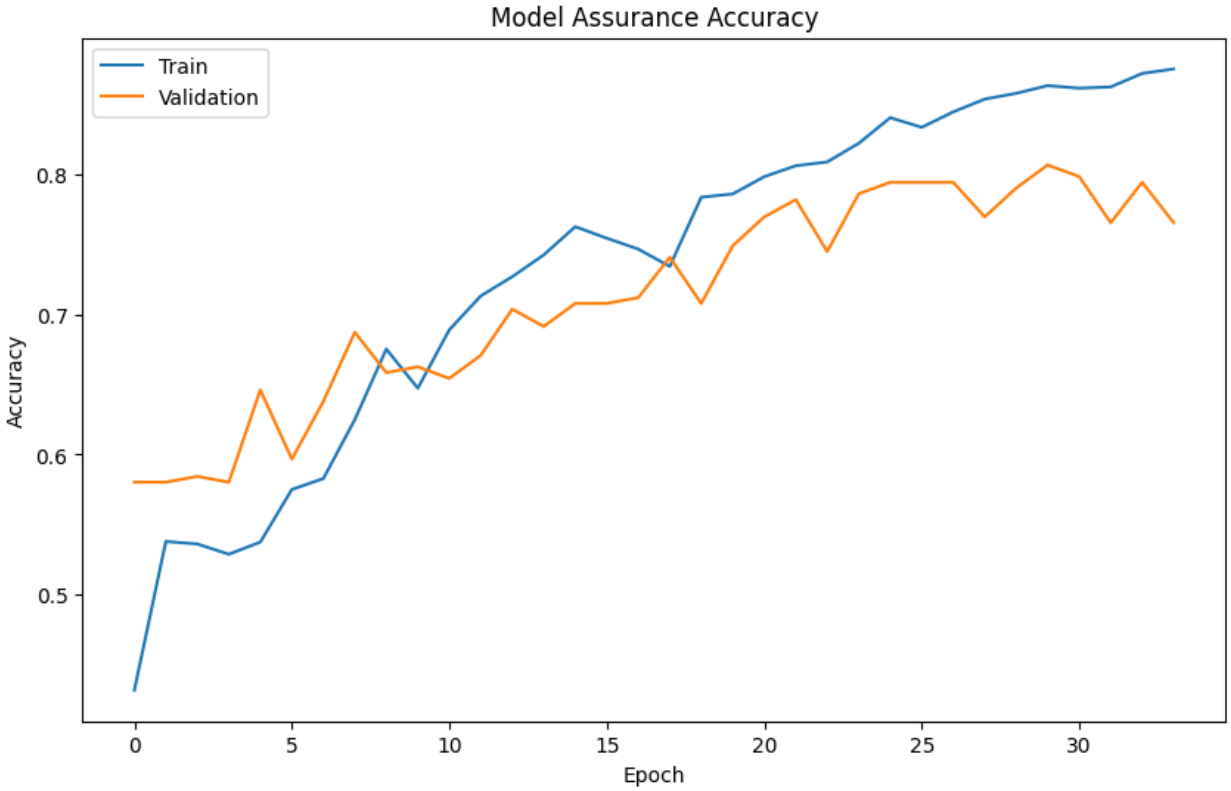

Model Tangibles Loss
Train
Validation
Loss
Epoch
0.6
0.8
1.0
1.2
0
5
10
15
20
25
30

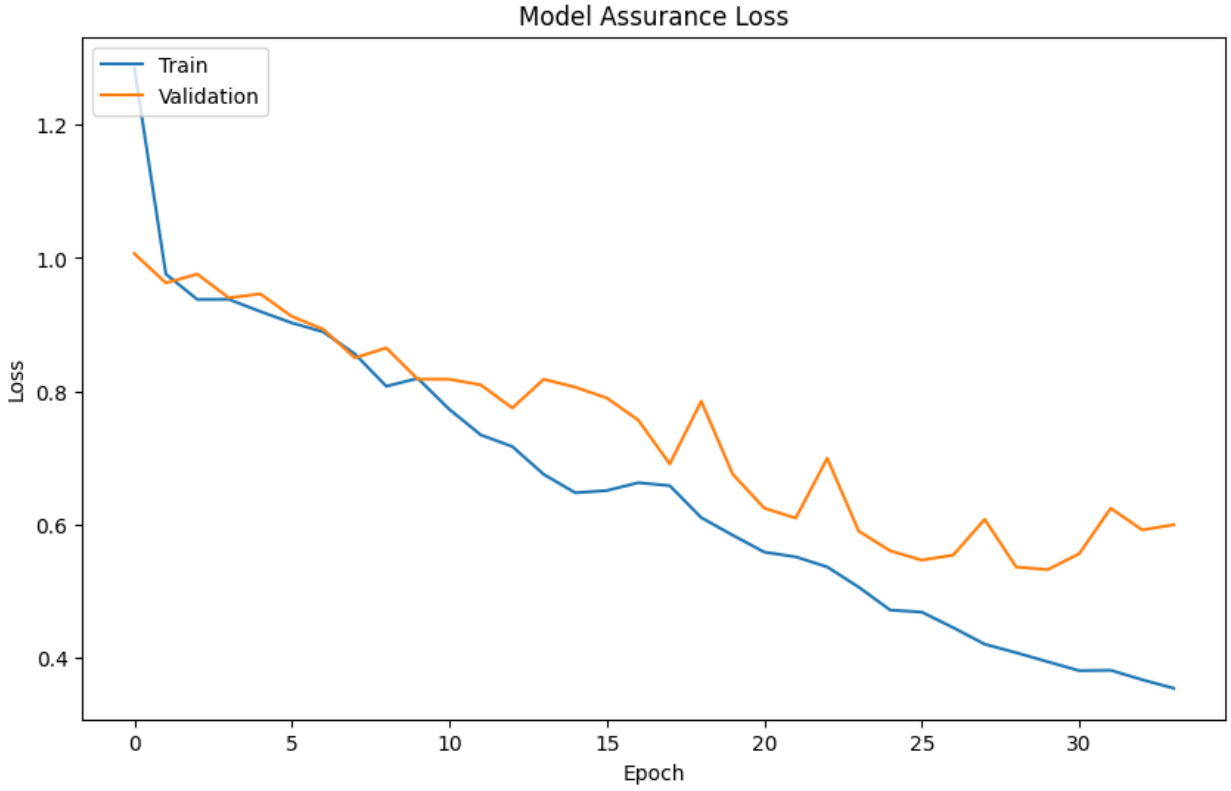

Model Reliability Accuracy
Train
Validation
Accuracy
Epoch
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0
5
10
15
20
25
30

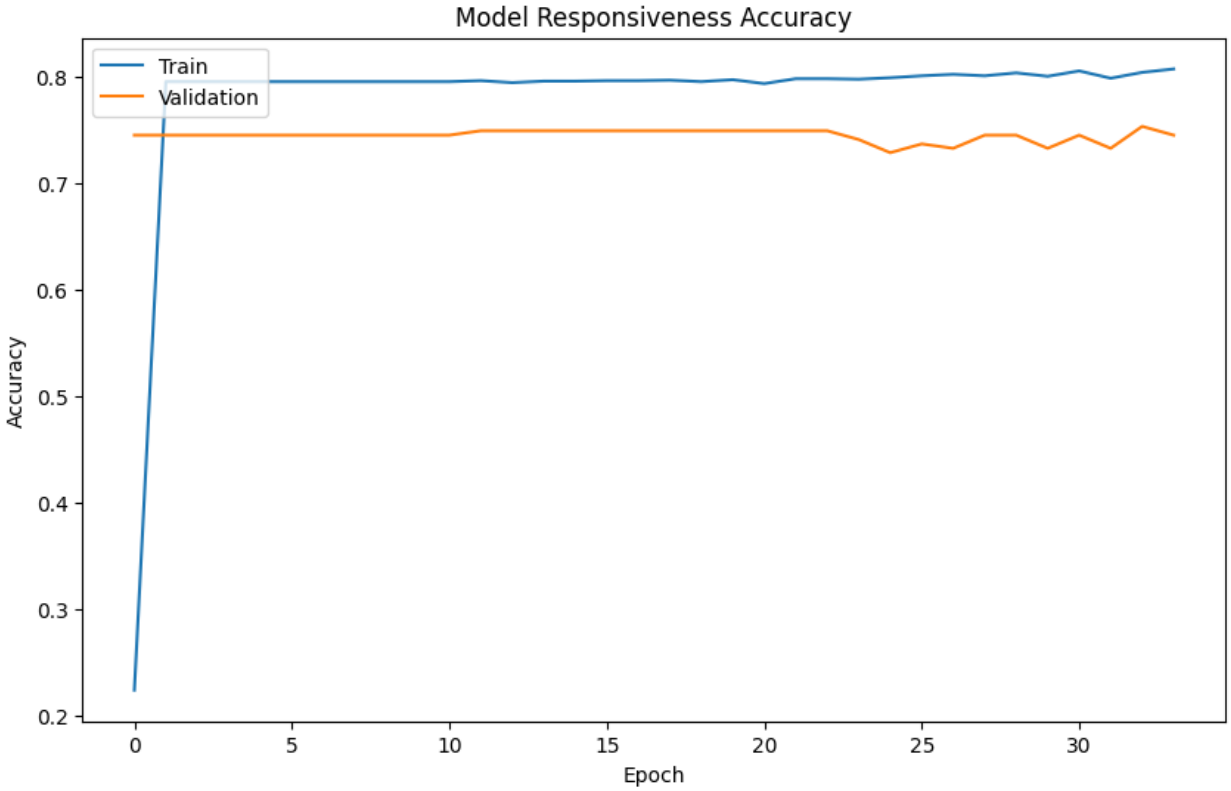

Model Reliability Loss
Train
Validation
Loss
Epoch
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30

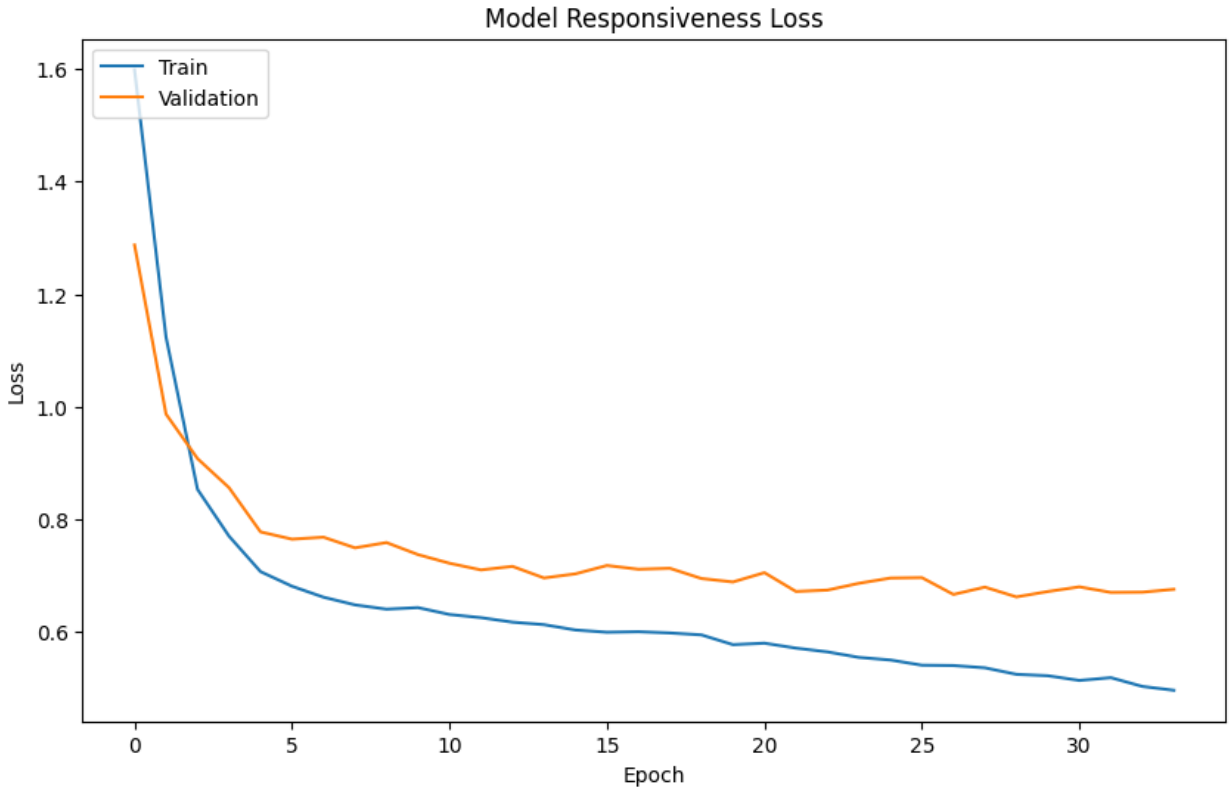

Model Empathy Accuracy
Train
Validation
Accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0
5
10
15
20
25
30


Model Empathy Loss
Train
Validation
Loss
Epoch
1.2
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30


Model Assurance Accuracy
Train
Validation
Accuracy
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30
Epoch


Model Assurance Loss
Train
Validation
Loss
0.4
0.6
0.8
1.0
1.2
0
5
10
15
20
25
30
Epoch


Model Responsiveness Accuracy
Train
Validation
Accuracy
Epoch
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30


Model Responsiveness Loss
Train
Validation
Loss
Epoch
0.6
0.8
1.0
1.2
1.4
1.6
0
5
10
15
20
25
30

در مرحله بعد سعی شد مدل دقیق تری برای پیش بینی هر ویژگی در نظر گرفته شود و پس از امتحان و ازمون خطا کردن چندین مدل و امتحان پارامترهای مختلف نظیر بهینه ساز و نرخ یادگیری و ... و افزودن دراپ اوت به معماری، مدل نهایی توسط کدهای زیر به دست آمد که بهبود نسبی نسبت به مدل اولیه و سایر مدل‌ها دارد.

```
from keras.layers import Input, Embedding, LSTM, Dense,
Bidirectional,Dropout
from keras.models import Model
# encoder input model
inputs = Input(shape=(max_length,))
#encoder1 = Embedding(input_dim=num_words,
output_dim=embedding_dim)(inputs)
encoder1 = Embedding(input_dim=num_words, output_dim=embedding_dim,
weights=[word_vectors.vectors], trainable=False)(inputs)
encoder2 = LSTM(128, return_sequences=True)(encoder1)  # Add
return_sequences=True
# Bidirectional LSTM layer
encoder2_bilstm = Bidirectional(LSTM(64))(encoder2)

#outputs= Dense(5, activation='softmax')
# Define output layers for each sentiment column
output_layers = []
for col in output_columns:
    if col=='Tangibles' or col=='Reliability'or col=='Assurance' :
      x=4
    elif col=='Empathy' or col=='Responsiveness':
      x=5
    if col=='Tangibles':
      output_layer1 = Dense(64, activation='relu')(encoder2_bilstm)
      dropout_layer1 = Dropout(0.1)(output_layer1)  # Add dropout
      output_layer2 = Dense(32, activation='relu')(dropout_layer1)
      #dropout_layer2 = Dropout(0.1)(output_layer2)
      #output_layer3 = Dense(16, activation='relu')(dropout_layer2)
      #dropout_layer3 = Dropout(1/16)(output_layer3)
      #output_layer4 = Dense(8, activation='relu')(dropout_layer3)
      last_layer = Dropout(1/8)(dropout_layer1)
    elif col=='Empathy':
      output_layer1 = Dense(32, activation='relu')(encoder2_bilstm)
      dropout_layer1 = Dropout(0.25)(output_layer1)  # Add dropout
      output_layer2 = Dense(16, activation='relu')(dropout_layer1)
      last_layer = Dropout(0.25)(output_layer2)
      #output_layer3 = Dense(16, activation='relu')(dropout_layer2)
      #dropout_layer3 = Dropout(1/16)(output_layer3)
      #output_layer4 = Dense(8, activation='relu')(dropout_layer3)
      #dropout_layer4 = Dropout(1/8)(output_layer4)
```

```
    elif col=='Reliability':
      output_layer1 = Dense(32, activation='relu')(encoder2_bilstm)
      dropout_layer1 = Dropout(0.25)(output_layer1)  # Add dropout
      output_layer2 = Dense(16, activation='relu')(dropout_layer1)
      last_layer = Dropout(0.25)(output_layer2)
      #output_layer3 = Dense(16, activation='relu')(dropout_layer2)
      #dropout_layer3 = Dropout(1/16)(output_layer3)
      #output_layer4 = Dense(8, activation='relu')(dropout_layer3)
      #dropout_layer4 = Dropout(1/8)(output_layer4)
    elif col=='Responsiveness':
      output_layer1 = Dense(32, activation='relu')(encoder2_bilstm)
      dropout_layer1 = Dropout(0.25)(output_layer1)  # Add dropout
      output_layer2 = Dense(16, activation='relu')(dropout_layer1)
      dropout_layer2 = Dropout(0.25)(output_layer2)
      output_layer3 = Dense(16, activation='relu')(dropout_layer2)
      dropout_layer3 = Dropout(1/16)(output_layer3)
      output_layer4 = Dense(8, activation='relu')(dropout_layer3)
      dropout_layer4 = Dropout(1/8)(output_layer4)
    else:
      output_layer1 = Dense(64, activation='relu')(encoder2_bilstm)
      dropout_layer1 = Dropout(1/16)(output_layer1)  # Add dropout
      output_layer2 = Dense(32, activation='relu')(dropout_layer1)
      last_layer = Dropout(1/8)(output_layer2)
      #output_layer3 = Dense(16, activation='relu')(dropout_layer2)
      #dropout_layer3 = Dropout(1/16)(output_layer3)
      #output_layer4 = Dense(8, activation='relu')(dropout_layer3)
      #dropout_layer4 = Dropout(1/8)(output_layer4)

    output_layer = Dense(x, activation='softmax', name=col)(last_layer)
    #output_layer = Dense(x, activation='softmax',
name=col)(encoder2_bilstm) #label_encoder.classes_.shape[0]

    output_layers.append(output_layer)

# Tie it together
model = Model(inputs=inputs, outputs=output_layers)
model.summary()
```

```
Model: "model_8"
__________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==================================================================================
 input_9 (InputLayer)        [(None, 200)]             0           []
```

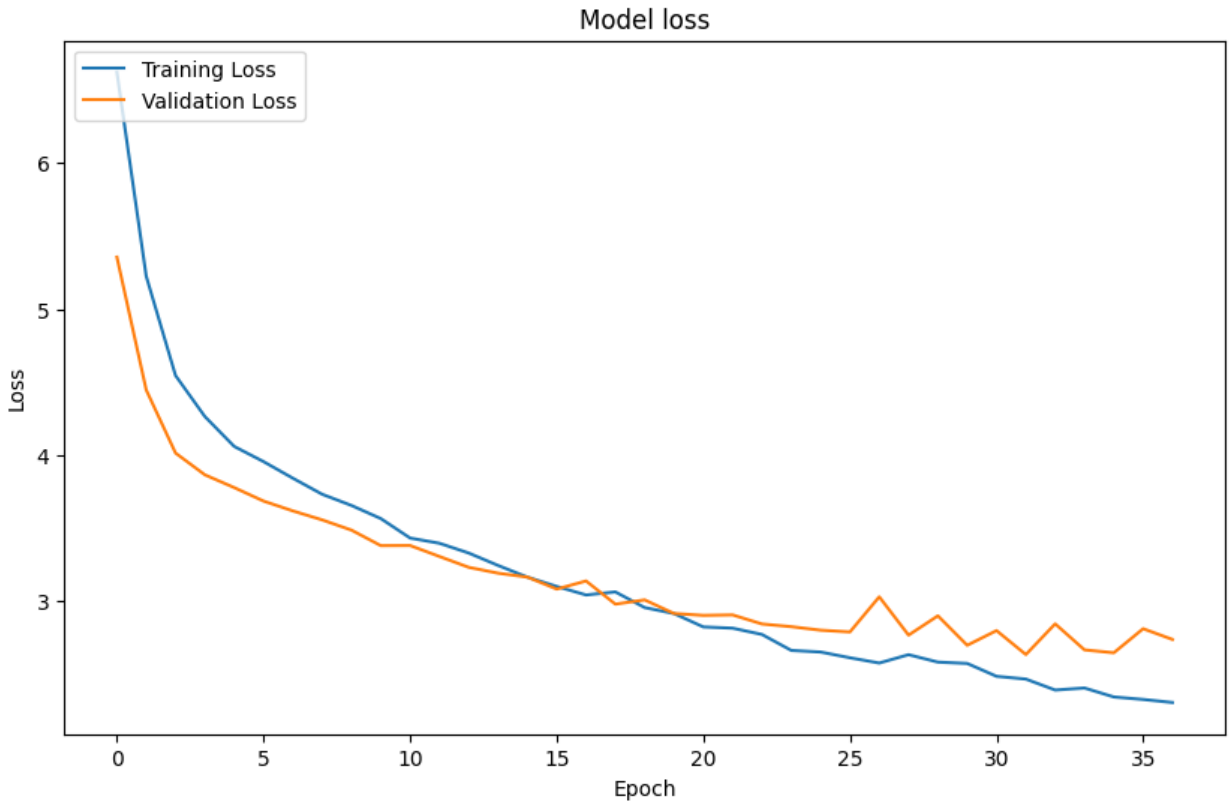
```
 embedding_8 (Embedding)        (None, 200, 300)     2487600
['input_9[0][0]']

 lstm_16 (LSTM)                 (None, 200, 128)     219648
['embedding_8[0][0]']

 bidirectional_8 (Bidirectional  (None, 128)         98816
['lstm_16[0][0]']
 )

 dense_97 (Dense)               (None, 32)           4128
['bidirectional_8[0][0]']

 dense_99 (Dense)               (None, 32)           4128
['bidirectional_8[0][0]']

 dense_101 (Dense)              (None, 64)           8256
['bidirectional_8[0][0]']

 dense_95 (Dense)               (None, 64)           8256
['bidirectional_8[0][0]']

 dropout_77 (Dropout)           (None, 32)           0
['dense_97[0][0]']

 dropout_79 (Dropout)           (None, 32)           0
['dense_99[0][0]']

 dropout_81 (Dropout)           (None, 64)           0
['dense_101[0][0]']

 dropout_75 (Dropout)           (None, 64)           0
['dense_95[0][0]']

 dense_98 (Dense)               (None, 16)           528
['dropout_77[0][0]']

 dense_100 (Dense)              (None, 16)           528
['dropout_79[0][0]']

 dense_102 (Dense)              (None, 32)           2080
['dropout_81[0][0]']

 dropout_76 (Dropout)           (None, 64)           0
['dropout_75[0][0]']

 dropout_78 (Dropout)           (None, 16)           0
['dense_98[0][0]']

 dropout_80 (Dropout)           (None, 16)           0
['dense_100[0][0]']

 dropout_82 (Dropout)           (None, 32)           0
['dense_102[0][0]']

 Tangibles (Dense)              (None, 4)            260
['dropout_76[0][0]']
```

```
 Reliability (Dense)             (None, 4)            68
['dropout_78[0][0]']

 Empathy (Dense)                 (None, 5)            85
['dropout_80[0][0]']

 Assurance (Dense)               (None, 4)            132
['dropout_82[0][0]']

 Responsiveness (Dense)          (None, 5)            165
['dropout_82[0][0]']

==================================================================================
====================
Total params: 2,834,678
Trainable params: 347,078
Non-trainable params: 2,487,600
__________________________________________________________________________________
____________________
```

این مدل ۳۷ ایپاک آموزش داده شد و به نتایج زیر رسید:

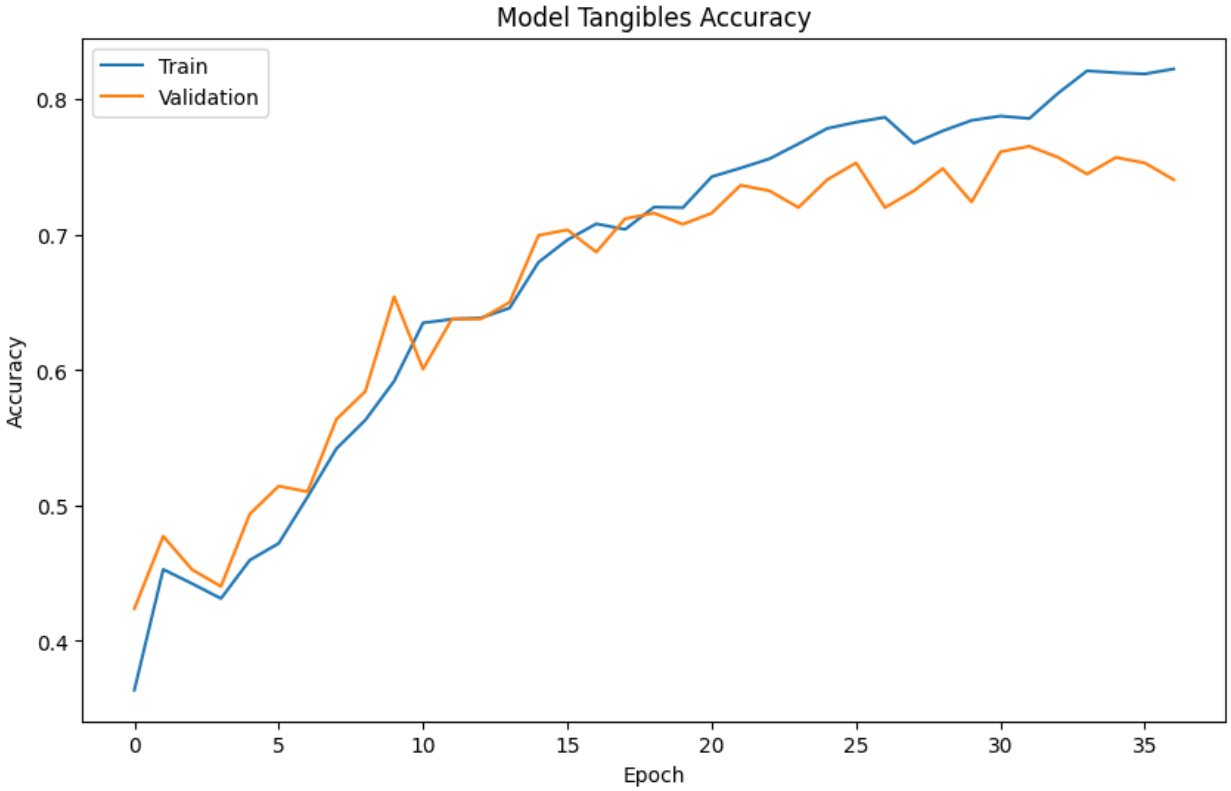
```
Evaluation Results:
Total Loss: 2.8060719966888428
Tangibles Accuracy: 72.66%
Reliability Accuracy: 78.91%
Empathy Accuracy: 93.75%
Assurance Accuracy: 82.03%
Responsiveness Accuracy: 84.38%

Metrics for Tangibles:
Accuracy: 0.7266, Precision: 0.6974, Recall: 0.7266, F1-score: 0.7066
Metrics for Reliability:
Accuracy: 0.7891, Precision: 0.6226, Recall: 0.7891, F1-score: 0.6960
Metrics for Empathy:
Accuracy: 0.9375, Precision: 0.8789, Recall: 0.9375, F1-score: 0.9073
Metrics for Assurance:
Accuracy: 0.8203, Precision: 0.8016, Recall: 0.8203, F1-score: 0.8021
Metrics for Responsiveness:
Accuracy: 0.8438, Precision: 0.7119, Recall: 0.8438, F1-score: 0.7722
```

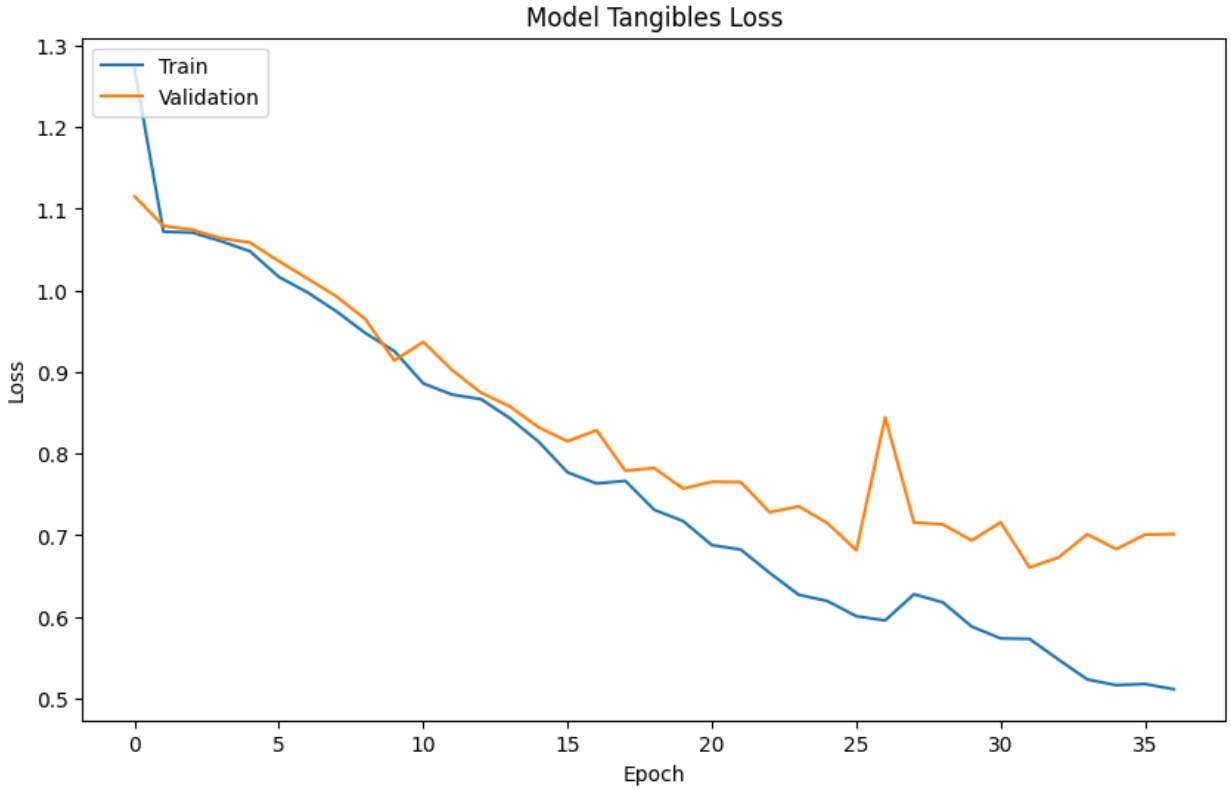
تابع زیان کلی به صورت زیر می‌باشد:

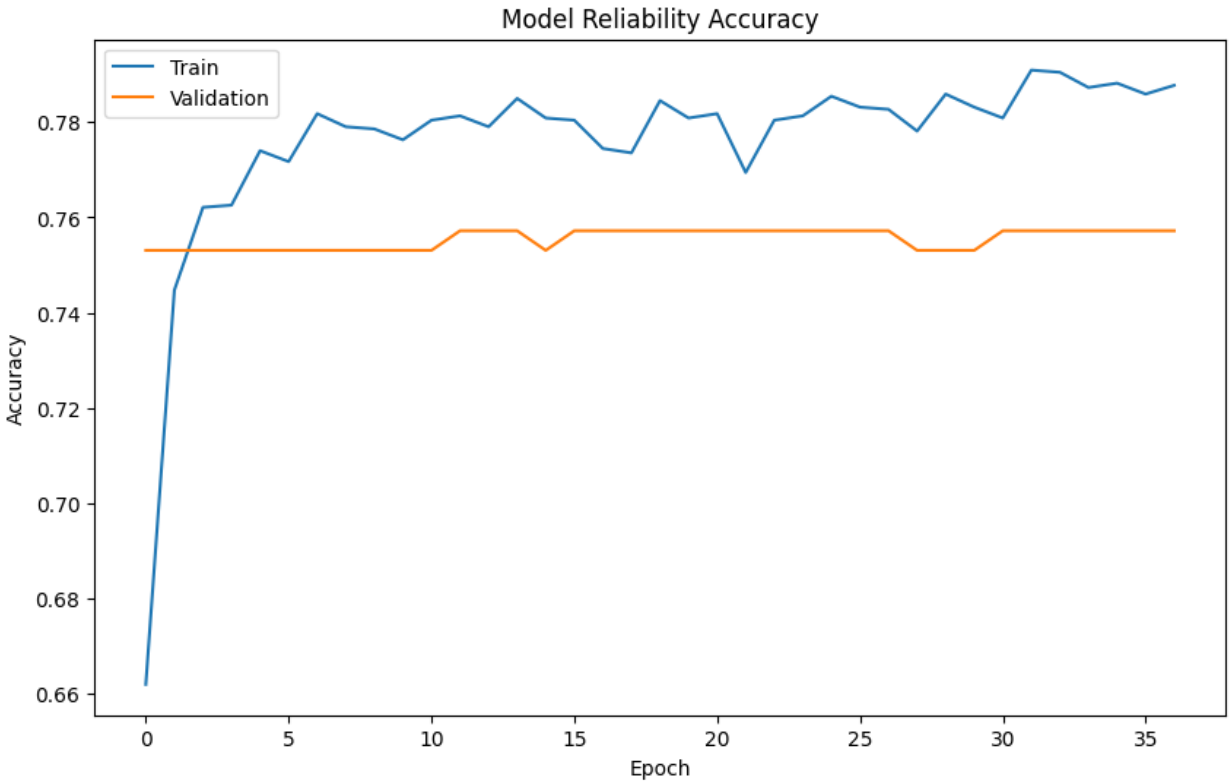


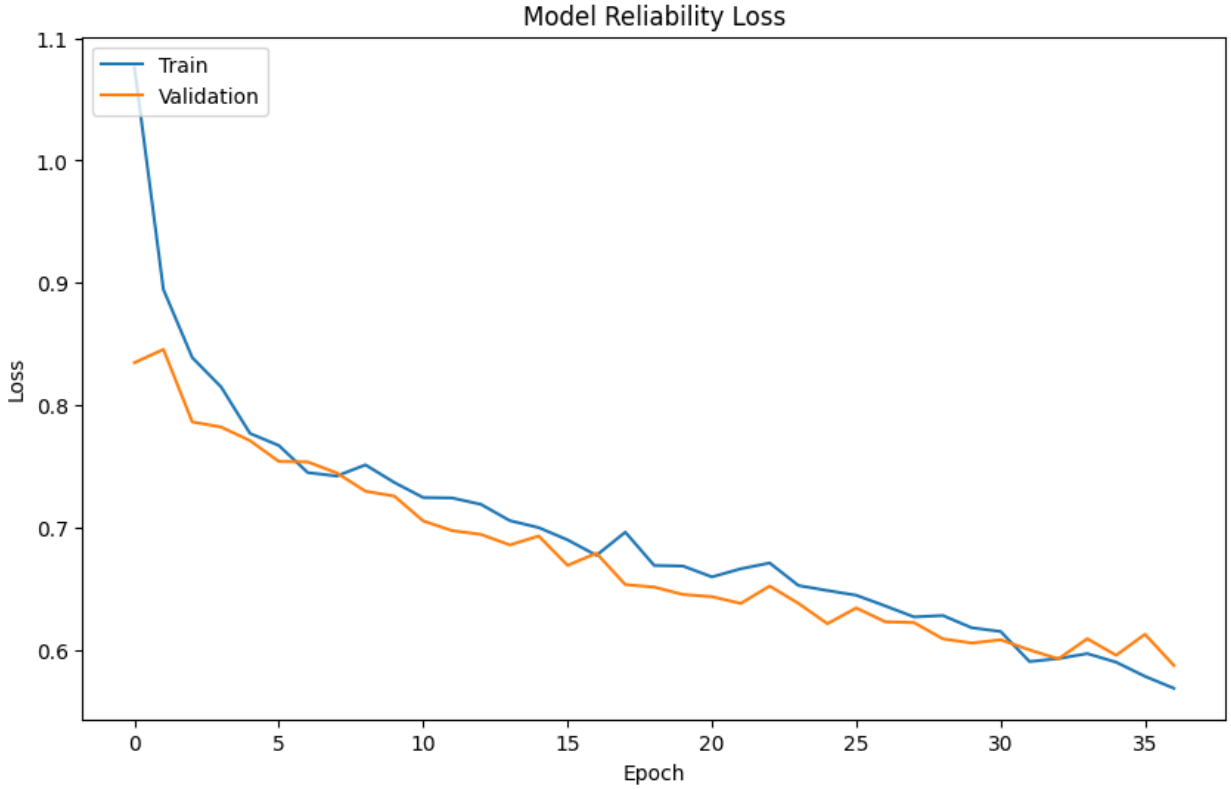
همانطور که دیده می‌شود زیان برای داده‌ی آموزشی و اعتبار سنجی به طور پیوسته کاهش یافته تا اینکه از ایپاک ۲۵ تا ۳۵ مرتب تابع هزینه برای مجموعه داده‌ی اعتبار سنجی افزایش و کاهش داشته، درصوتری که برای آموزشی کاهش می‌یافته است. بنابریان تصمیم بر توقف آموزش گرفته شده است.

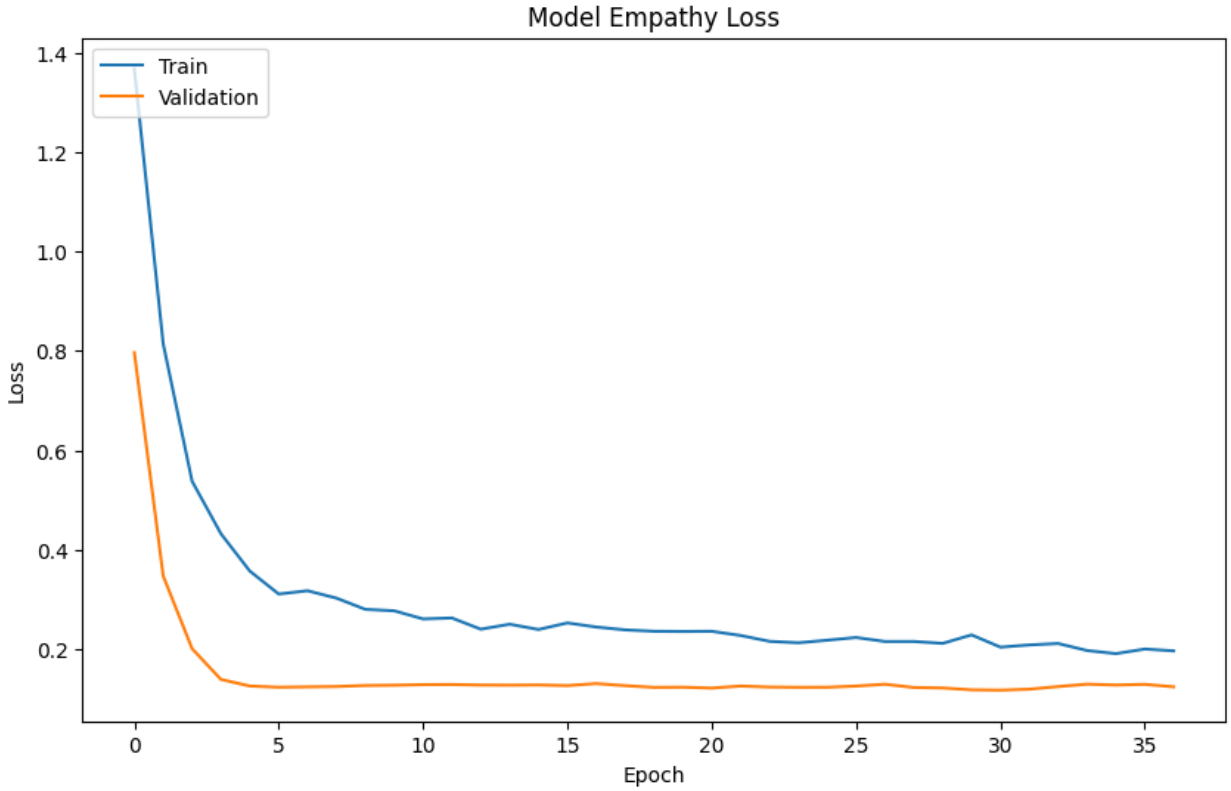
سایز نمودارها:

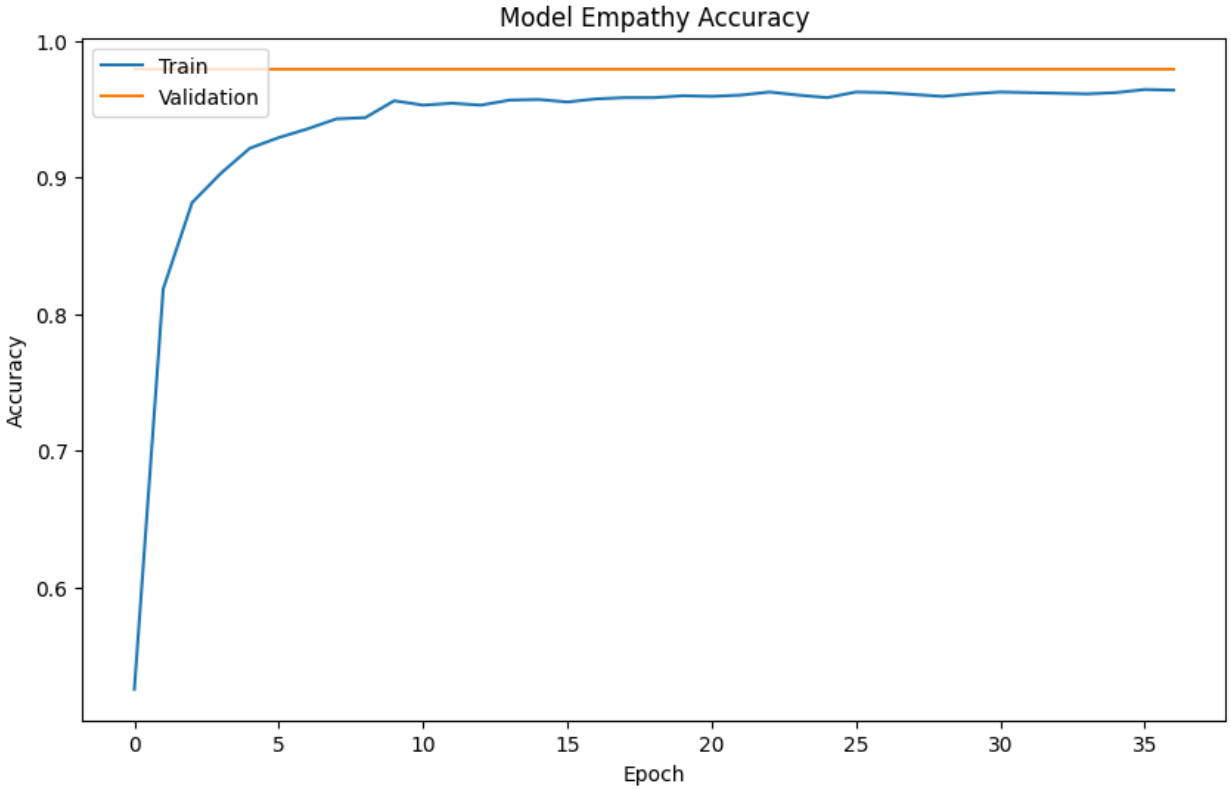

Model Tangibles Accuracy
Train
Validation
Accuracy
Epoch
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30
35

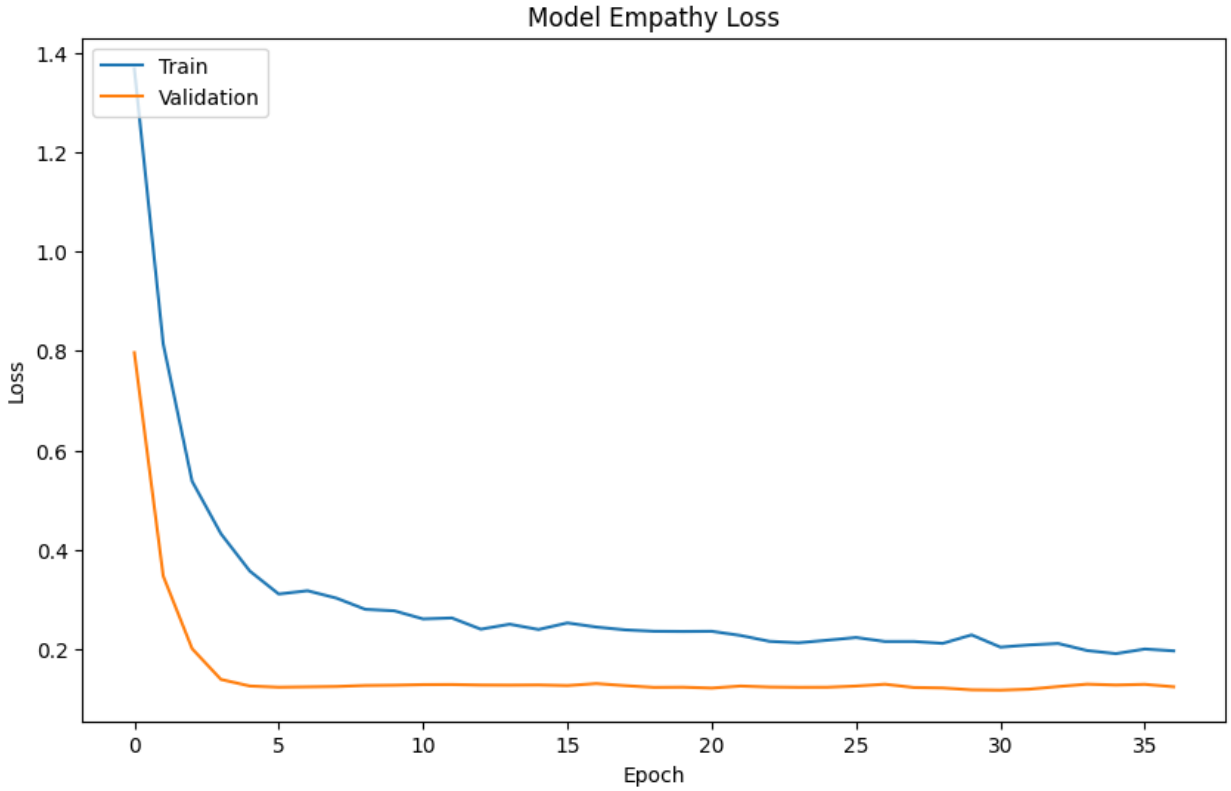

Model Tangibles Loss
Train
Validation
Loss
Epoch
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
0
5
10
15
20
25
30
35

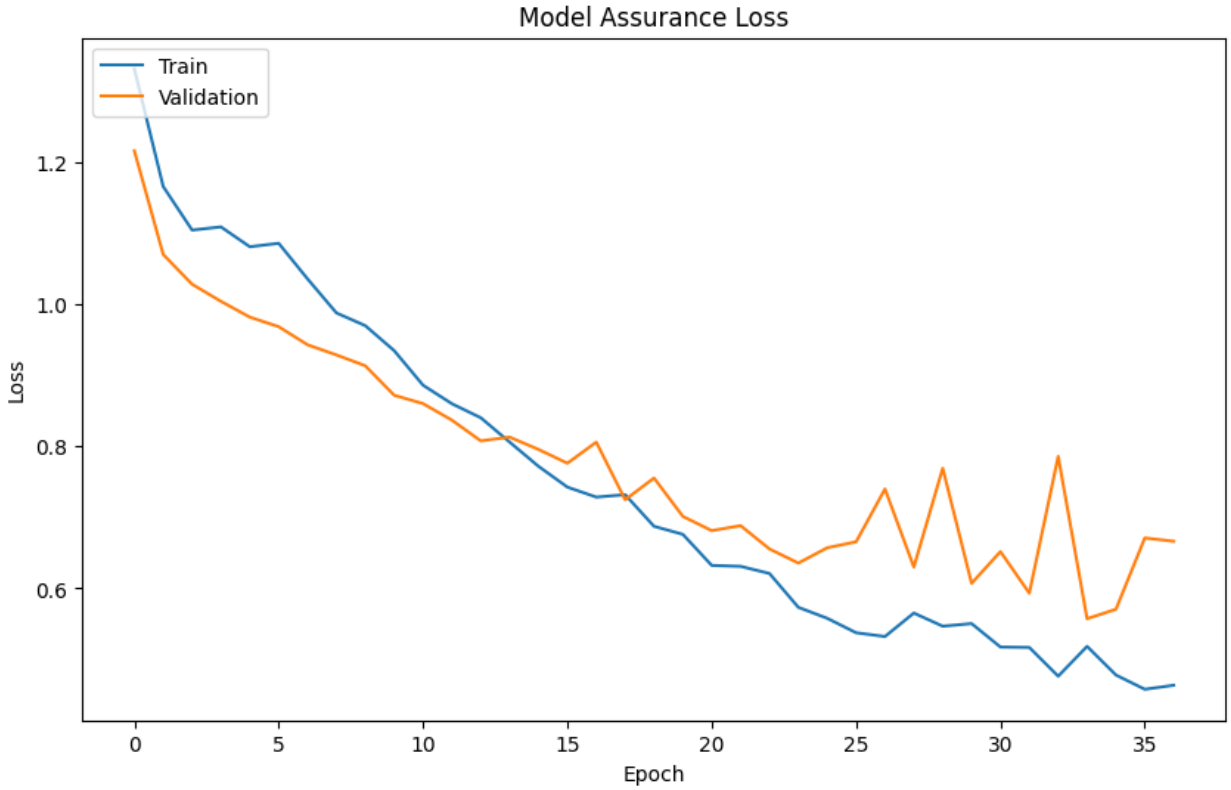

Model Reliability Accuracy
Train
Validation
Accuracy
0.66
0.68
0.70
0.72
0.74
0.76
0.78
0
5
10
15
20
25
30
35
Epoch

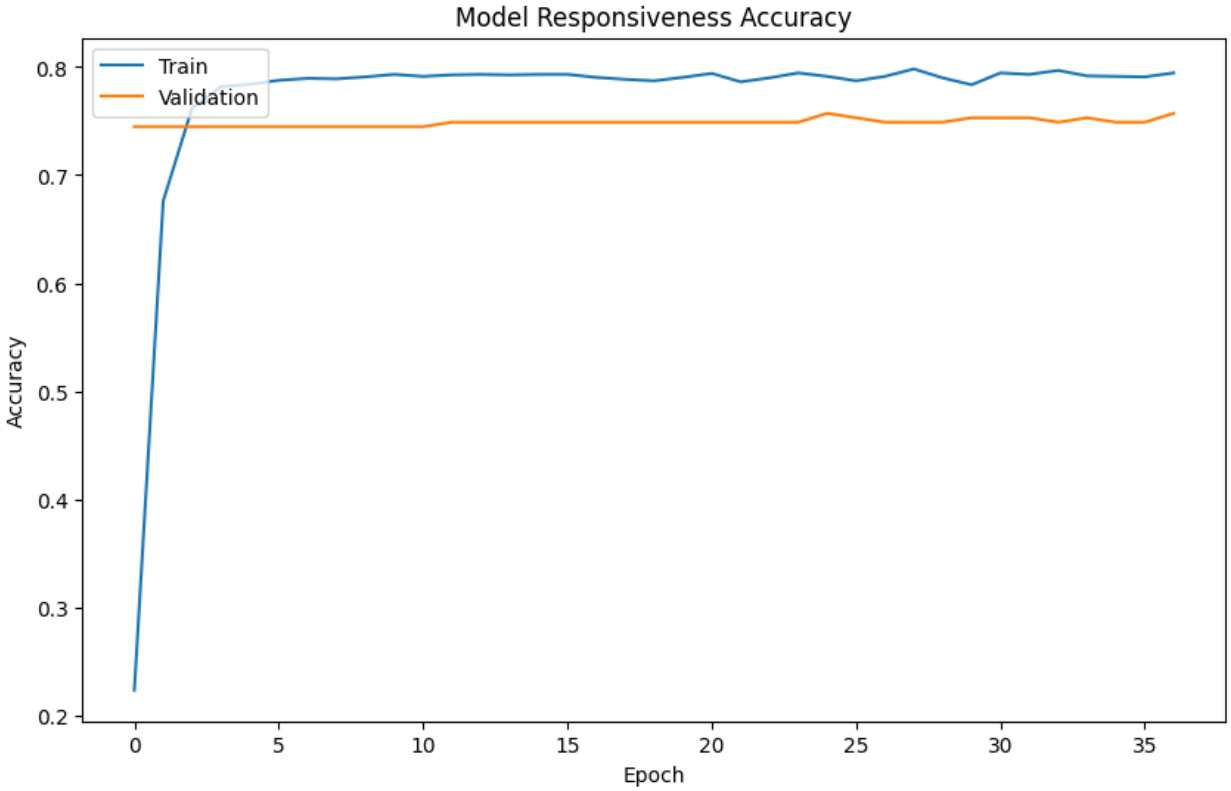

Model Reliability Loss
Train
Validation
Loss
0.6
0.7
0.8
0.9
1.0
1.1
0
5
10
15
20
25
30
35
Epoch

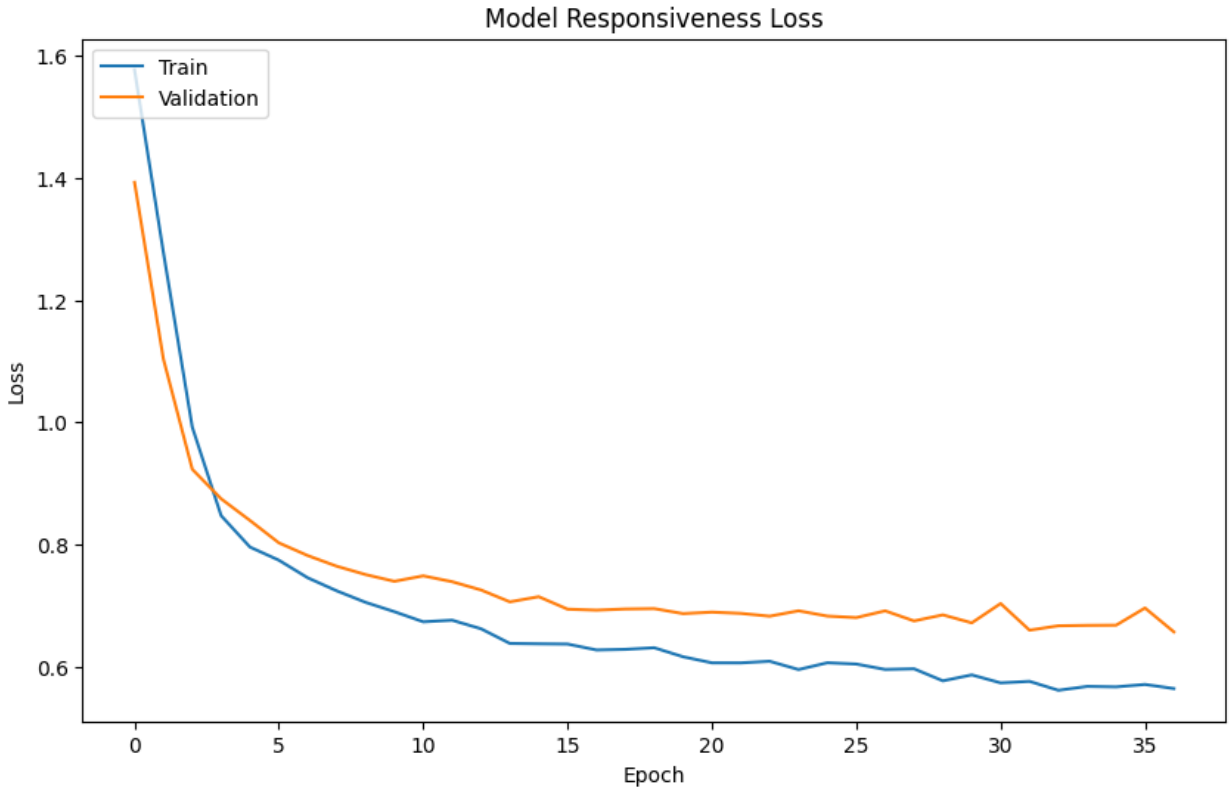

Model Empathy Loss
Train
Validation
Loss
Epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
35


Model Empathy Accuracy
Train
Validation
Accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0
5
10
15
20
25
30
35


Model Empathy Loss
Train
Validation
Loss
Epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
35


Model Assurance Loss
Train
Validation
Loss
Epoch
1.2
1.0
0.8
0.6
0
5
10
15
20
25
30
35


Model Responsiveness Accuracy
Train
Validation
Accuracy
Epoch
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30
35


Model Responsiveness Loss
Train
Validation
Loss
Epoch
0.6
0.8
1.0
1.2
1.4
1.6
0
5
10
15
20
25
30
35

همانطور که دیده می‌شود برای هر ویژگی تا انتهای آموزش زیان در یک ثابتی در مجموعه‌ی اعتبارسنجی متوقف شده و تغییر نداشته است و میزان دقت نیز در انتهای آموزش در مقدار ثابتی محدود شده است.