



به نام خدا  
دانشگاه تهران  
دانشکده مهندسی برق و کامپیوتر



## درس شبکه‌های عصبی و یادگیری عمیق

### تمرین چهارم

نام و نام خانوادگی	عرفان باقری سولا – محمد قره حسنلو
شماره دانشجویی	810198461 – 810198361
تاریخ ارسال گزارش	1402.3.3

## فهرست

پاسخ 2. تشخیص اندیشه.....	5
1- معماری LSTM و embedding.....	5
2- پیش پردازش دادگان.....	8
3- پیاده سازی طبقه بندی نیت.....	11
4- پیاده سازی مدل Responder.....	25
پاسخ 1 - توصیف عکس.....	26
1. Model with Frozen CNN.....	26
2. Model with Trainable CNN.....	31

- شکل 1: پیش پردازش های انجام شده ..... 10
- شکل 2: embedded کردن جملات ..... 11
- شکل 3: مقادیر accuracy, precision, recall, f1\_score برای مدل اول با hidden\_state=100 ..... 13
- شکل 4: مقادیر accuracy, precision, recall, f1\_score برای مدل اول با hidden\_state=25 ..... 14
- شکل 5: ماتریس درهم ریختگی برای مدل اول با hidden\_state=100 ..... 15
- شکل 6: ماتریس درهم ریختگی برای مدل اول با hidden\_state=25 ..... 15
- شکل 7: مدل اول با hidden\_size=100 ..... 16
- شکل 8: مدل اول با hidden\_size=25 ..... 17
- شکل 9: مدل دوم با hidden\_size=100 ..... 18
- شکل 10: مدل دوم با hidden\_size=25 ..... 18
- شکل 11: مقادیر accuracy, precision, recall, f1\_score برای مدل دوم و کلاس های اصلی با hidden\_state=100 ..... 20
- شکل 12: مقادیر accuracy, precision, recall, f1\_score برای مدل دوم و زیرکلاس ها با hidden\_state=100 ..... 21
- شکل 13: مقادیر accuracy, precision, recall, f1\_score برای مدل دوم و کلاس های اصلی با hidden\_state=25 ..... 22
- شکل 14: مقادیر accuracy, precision, recall, f1\_score برای مدل دوم و زیرکلاس ها با hidden\_state=25 ..... 23
- شکل 15: ماتریس درهم ریختگی برای مدل دوم با hidden\_state=100 ..... 24
- شکل 16: ماتریس درهم ریختگی برای مدل دوم با hidden\_state=25 ..... 24
- شکل 17: مدل responder ..... 25
- شکل 18: نتایج مدل responder برای مثال های خواسته شده ..... 26
- شکل 19: پیاده سازی مدل در Tensorflow ..... 28
- شکل 20: نمودار خطای آموزش و تست ..... 29
- شکل 21: خروجی مدل روی داده تست ..... 29
- شکل 22: خروجی مدل روی داده تست ..... 30
- شکل 23: خروجی مدل روی داده تست ..... 30
- شکل 24: نمودار خطای آموزش و تست ..... 31
- شکل 25: خروجی مدل روی داده تست ..... 32
- شکل 26: خروجی مدل روی داده تست ..... 32

شکل 27: خروجی مدل روی داده تست ..... 33

## جدول‌ها

جدول 1: مقادیر accuracy مدل اول و مقاله ..... 14

جدول 2: مقادیر accuracy مدل دوم و مقاله ..... 25

## پاسخ 2. تشخیص اندیشه

### 1- معماری LSTM و embedding

در این مقاله، هدف طبقه بندی سوالان با توجه به معنی آنها نمیباشد بلکه نوع جوابی که نیاز است به آنها داده شود مورد بررسی قرار میگیر که بعدا بتوان آن را به معماری responder اضافه کرد تا به جواب دقیق تر کمک کند.

در معماری RNN، مشکل feedforward های سنتی این بود که اطلاعات را از ترتیب زمانی استخراج نمیکرد و فقط ورودی فعلا را برای آن در نظر میگرفت. چون در متن ها، ترتیب کلمات به نوعی مهم هستند، ایده استفاده از RNN به میان آمد. در RNN به دلیل استفاده از loop ها داخل شبکه آن، اجازه میدهد که اطلاعات در قدم های مختلف در بلاک ها منتقل شوند و خروجی فقط وابسته به همان ورودی که داخل ان هستیم، نداشته باشد و در واقع یک زنجیره ای تشکیل میدهند که پیام را به بعدی میفرستد. مشکل RNN این است که در هنگام محاسبه گرادیان در هنگام backward با مشکل vanishing مواجه میشود؛ بدین معنا که در طول زمان، هنگام محاسبه گرادیان در شبکه، تاثیر مقادیر ابتدایی و نزدیک به ابتدا کاهش می یابد و در واقع اثر مقادیر خروجی مورد نظر که به تابع هزینه کمک می کنند می تواند انقدر کوچک شود که تنها پس از چند مرحله، دیگر سهم کافی در یادگیری پارامتر وجود ندارد. به این مشکل، vanishing gradient میگویند. این بدین معناست که RNN تنها میتواند short term memory را حفظ کند و مثلا اگر یک جمله طولانی را به RNN بدهیم به خوبی عمل نخواهد کرد و برای جملات کوتاه تر بهتر عمل میکند.

برای حل مشکل RNN ها، LSTM ها معرفی شدند. معماری LSTM به طور خاص برای مشکل حفظ حافظه در شبکه های عصبی طراحی شده است و مزایای زیادی در مقایسه با معماری RNN دارد.

یکی از مزایای مهم معماری LSTM این است که این شبکه ها می توانند به خوبی با داده های طولانی و پیچیده کار کنند. شبکه های RNN نمی توانند اطلاعاتی را که در فاصله زیادی از زمان فعلی قرار دارند، به خوبی حفظ کنند. با استفاده از معماری LSTM، این مشکل بسیار کمتر شده و شبکه می تواند اطلاعاتی را که در زمان های دورتر از حال قرار دارند، به خوبی حفظ کند.

همچنین، معماری LSTM دارای یک دروازه بندی قوی است که به شبکه این امکان را می دهد که به خوبی تصمیم بگیرد که کدام اطلاعات را در حافظه خود ذخیره کند و کدام را حذف کند. این به شبکه این امکان را می دهد که به طور خودکار و بدون نیاز به داده های بیشتر، ویژگی های مهم را در داده ها شناسایی

کند. در کل، معماری LSTM به دلیل قابلیت حفظ حافظه بیشتر، در پردازش داده‌های طولانی و پیچیده، به خصوص در حوزه NLP، با موفقیت بیشتری مواجه شده است.

Word embedding یک تکنیک در حوزه پردازش زبان طبیعی (NLP) است که به کار می‌رود تا کلمات را در فضای عددی نمایش دهد. در این روش، هر کلمه با یک بردار عددی یکتا نشان داده می‌شود که ویژگی‌های مختلف آن کلمه را در خود جای می‌دهد. برای مثال، کلماتی که به معنای مشابهی هستند، در فضای برداری به نزدیکی یکدیگر قرار می‌گیرند.

استفاده از word embedding برای مدل‌سازی زبانی در حوزه NLP بسیار مفید است. با استفاده از این تکنیک، می‌توان اطلاعات بیشتری از معنای کلمات را در داده‌های زبانی به دست آورد و از آن به عنوان ورودی برای الگوریتم‌های یادگیری ماشین استفاده کرد. علاوه بر این، word embedding به ما این امکان را می‌دهد که با استفاده از این بردارها، میزان شباهت بین کلمات را در معنای واقعی‌تر به دست آوریم و در حل مسائلی مانند دسته‌بندی متن، تشخیص احساسات و ترجمه ماشینی بهبود قابل توجهی را به دست آوریم.

روش‌های تولید word embedding به طور کلی به دو دسته تقسیم می‌شوند: روش‌های مبتنی بر شمارش و روش‌های مبتنی بر پیش‌بینی.

در روش‌های مبتنی بر شمارش، ابتدا یک ماتریس شمارش برای کلمات موجود در مجموعه داده ساخته می‌شود. سپس با استفاده از روش‌های خاصی مانند Singular Value Decomposition (SVD) و Principal Component Analysis (PCA)، این ماتریس به یک فضای برداری با ابعاد کم تبدیل می‌شود. در نهایت، بردارهای کلمات به عنوان word embedding استفاده می‌شوند. این روش به عنوان روش‌هایی مانند Latent Semantic Analysis (LSA) شناخته می‌شود.

در روش‌های مبتنی بر پیش‌بینی، به جای شمارش تعداد حضور کلمات، از مدل‌های یادگیری ماشین استفاده می‌شود تا به صورت پیش‌بینی‌ای که کلمات در یک متن به هم مرتبط هستند، word embedding ساخته شود. برای مثال، در روش Word2Vec، یک مدل شبکه عصبی بازگشتی به کلمات متن ورودی داده می‌شود و شبکه سعی می‌کند بهترین بردار برای هر کلمه را به دست آورد. این روش به عنوان روش‌های مبتنی بر پیش‌بینی شناخته می‌شود.

در روش های مبتنی بر گو-وردینگ، یک ماتریس کو-وردینگ برای کلمات ساخته می شود که در آن هر سطر نمایانگر یک کلمه و هر ستون نمایانگر یک ویژگی است. سپس با استفاده از روش های خاصی مانند Non-negative Matrix Factorization (NMF)، این ماتریس به یک فضای برداری با ابعاد کم تبدیل می شود. در نهایت، بردارهای کلمات به عنوان word embedding استفاده می شوند. این روش به عنوان روش هایی مانند GloVe شناخته می شود.

همه ی این روش های تولید word embedding به صورت موفقیت آمیزی مورد استفاده در حوزه پردازش زبان طبیعی قرار گرفته اند و بسته به نوع مسئله و داده ها می توان از یکی از آن ها استفاده کرد. چند روش خاص word embedding عبارتند از:

1- Word2Vec: این روش یک مدل شبکه ی عصبی بازگشتی (RNN) است که برای ساخت word embedding استفاده می شود. این روش در دو حالت CBOW و Skip-gram مورد استفاده قرار می گیرد.

2- GloVe: این روش بر پایه ی شمارش آماری کلمات در متون بنیانگذاری شده است. در این روش، ابتدا یک ماتریس شمارش برای کلمات ساخته می شود و سپس با استفاده از روش Singular Value Decomposition (SVD)، به فضایی با ابعاد کم تبدیل می شود.

3- FastText: این روش نسخه بهبود یافته ی Word2Vec محسوب می شود و برای ساخت word embedding استفاده می شود. در این روش، برای هر کلمه، بردارهای مربوط به زیرکلمات آن نیز در نظر گرفته می شود.

4- ELMo: این روش یک مدل شبکه ی عصبی بازگشتی (RNN) است که برای ساخت word embedding استفاده می شود. در این روش، به جای ساخت یک بردار برای هر کلمه، بردارهای متفاوتی برای هر کلمه در نظر گرفته می شود که نمایانگر رفتار کلمه در متون مختلف است.

در واقع، GloVe برای هر کلمه یک بردار یکتا تولید می کند که نماینده ی معنای کلمه در متن است. بنابراین، ممکن است در متونی که کلمات چند معنای متفاوت دارند، بردارهای تولید شده توسط GloVe نتوانند تمایز قائل شوند و باعث ایجاد ابهام در پردازش متن شوند.

مشکل GloVe در کلماتی که چند معنی متفاوت دارند، این است که این روش برای هر کلمه یک بردار یکتا تولید می کند که نماینده ی معنای کلمه در متن است. به عبارت دیگر، GloVe نمی تواند تمایز قائل شود که آیا در یک متن خاص کلمه با یک معنی خاص استفاده شده است یا با معانی دیگری.



این مشکل در GloVe به دلیل استفاده از یک ماتریس شمارش برای ساختاردهی به داده‌های زبانی و تبدیل آن‌ها به یک فضای برداری رخ می‌دهد. در این روش، برای هر کلمه، تعداد استفاده از آن در متون مختلف شمرده می‌شود و بر اساس این شمارش، ماتریس شمارش برای هر کلمه ساخته می‌شود. سپس با استفاده از روش Singular Value Decomposition (SVD)، این ماتریس به فضایی با ابعاد کم تبدیل می‌شود و بردارهای نهایی برای هر کلمه تولید می‌شوند.

بنابراین، برای کلمات چند معنایی، استفاده از GloVe به تنهایی بهترین گزینه نیست و باید از روش‌های دیگری مانند Sense Embedding یا Polysemy-Aware Embedding استفاده کرد.

## 2- پیش پردازش دادگان

ابتدا چندین روش برای پیش پردازش در داده‌های متنی را معرفی می‌کنیم و سپس روش‌های استفاده شده را توضیح می‌دهیم:

**Normalization:** یکی از مراحل مهم در پیش پردازش متن، normalization است. در این مرحله، متن ورودی به یک فرم استاندارد تبدیل می‌شود. این فرم ممکن است شامل حذف علائم نگارشی، تبدیل حروف به حروف کوچک یا حذف استفاده از کلمات تو رشته‌های خاص باشد. مهمترین روش‌های normalization عبارتند از:

حذف علائم نگارشی (Punctuation Removal): در این روش، علائم نگارشی مانند نقطه، ویرگول، و دو نقطه از متن حذف می‌شوند.

تبدیل حروف به حروف کوچک (Lowercase Conversion): در این روش، همه حروف متن به حروف کوچک تبدیل می‌شوند تا به حروف بزرگ و کوچک تفاوتی نباشد.

تبدیل اختصارات به کلمات کامل (Expansion of Abbreviations): در این روش، اختصارات رایج مانند "Mr." و "Dr." به کلمات کامل تبدیل می‌شوند.

**Tokenization:** در مرحله‌ی tokenization، متن ورودی به یک سری از توکن‌های کوچکتر تقسیم می‌شود. این توکن‌ها معمولاً کلمات هستند، اما می‌توانند شامل عبارات، عدد و علائم نگارشی نیز باشند. توکن‌ها معمولاً به عنوان واحدهای پردازشی برای الگوریتم‌های مختلف در پردازش متن استفاده می‌شوند. مهمترین روش‌های tokenization عبارتند از:

تقسیم بر اساس فاصله (Whitespace Tokenization): در این روش، متن ورودی بر اساس فاصله بین کلمات به توکن‌های جداگانه تقسیم می‌شود.

تقسیم بر اساس علائم نگارشی (Punctuation Tokenization): در این روش، متن ورودی بر اساس علائم نگارشی به توکن‌های جداگانه تقسیم می‌شود. به عنوان مثال، اگر عبارت "Hello, world!" ورودی باشد، توکن‌های تولید شده شامل "Hello"، "،"، "world"، و "!" خواهند بود.

تقسیم بر اساس عبارات منظم (Regular Expression Tokenization): در این روش، متن ورودی بر اساس الگوهای منظم تقسیم می‌شود. به عنوان مثال، الگوی "\w+/" برای جستجوی هر کلمه‌ی متن استفاده می‌شود.

تقسیم بر اساس POS تگ‌ها (Part-of-Speech Tokenization): در این روش، متن ورودی بر اساس تگ‌های POS به توکن‌های جداگانه تقسیم می‌شود. به عنوان مثال، در جمله "He is eating an apple"، "He" به عنوان یک توکن با تگ اسم (noun) و "eating" به عنوان یک توکن با تگ فعل (verb) شناخته می‌شود.

هر یک از این روش‌ها می‌توانند برای پیش پردازش متن استفاده شوند و بسته به نوع متن و الگوریتمی که به آن اعمال می‌شود، یک یا چند روش می‌تواند مناسب باشد. به طور کلی، مهمترین روش‌های پیش پردازش متن عبارتند از:

Stop Word Removal: در این روش، کلمات پرتکرار و بی‌معنی مانند "the" و "and" از متن حذف می‌شوند.

Stemming: در این روش، پسوندهای کلمات مانند "-ing" و "-ed" برای به حذف شدن قرار می‌گیرند. به عنوان مثال، "running" و "ran" به "run" تبدیل می‌شوند.

Lemmatization: در این روش، کلمات به شکل پایه‌ای خود تبدیل می‌شوند. به عنوان مثال، "ran" به "run" تبدیل می‌شود.

Named Entity Recognition (NER): در این روش، اسم‌ها، مکان‌ها، شرکت‌ها و سایر موجودیت‌های موجود در متن شناسایی و برچسب‌گذاری می‌شوند.

Chunking: در این روش، مجموعه‌ای از توکن‌ها به عنوان یکی از واحدهای معنایی مانند عبارت، جمله یا پاراگراف شناسایی می‌شوند.

Sentiment Analysis: در این روش، متن ورودی برای تشخیص احساسات مثبت، منفی یا خنثی آنالیز می‌شود.

Machine Translation: در این روش، متن ورودی از یک زبان به زبان دیگر ترجمه می‌شود.

در این مقاله، از دیتاستی استفاده شده است که اندازه train\_df برابر 5452 و اندازه test\_df برابر 500 میباشد. در هر کدام سه ستون به نام های label\_coarse، label\_fine و text وجود دارد که label\_coarse برابر کلاس اصلی و label\_fine برابر زیرکلاس میباشد. مقادیر یکتا در label\_coarse برابر 6 (NUM، LOC، HUM، DESC، ENTY، ABBR) میباشد و همچنین 47 زیرکلاس وجود دارد.

روش هایی که در این مقاله خواسته شده بود، یکی تبدیل همه حروف به حروف کوچک بود و همچنین خواسته شده بود تا علائم نگارشی به جز علامت سوال حذف شود. در قطعه کد زیر این کار انجام شده است:

```
def clean_string(string):  
    # Remove special characters and punctuation symbols (excluding '?')  
    string = re.sub(r'^\w\s\?$', '', string)  
    # Transform all words to lowercase  
    string = string.lower()  
    return string
```

شکل 1: پیش پردازش های انجام شده

همچنین به شکل زیر پس از پیش پردازش های گفته شده، جملات را embedded میکنیم. برای این امر، ابتدا کلمات جمله را جدا میکنیم. سپس به اندازه max\_length که 50 تعریف کرده ایم، جمله را pad میکنیم و در نهایت هر کلمه را بر اساس مدل glove که تعریف کرده ایم، به شکل بردار های 300 تایی که در glove وجود دارد، در می آوریم. تعداد بردارها در آن 400000 میباشد که یعنی 400000 کلمه به بردارهای 300 تایی در آمده اند. به طور مثال سایز embedded\_train\_questions برابر 50، 5452 (300 می شود).

```
# Define the maximum length of the input sequence
max_length = 50

# Define a function to convert a sentence to a matrix of GloVe embeddings
def sentence_to_embeddings(sentence, max_length):
    # Split the sentence into words
    words = sentence.split()

    # pad the sequence to the maximum length
    words = words[:max_length] + [''] * (max_length - len(words))

    # Convert each word to its corresponding embedding
    embeddings = [glove_model[word] if word in glove_model else np.zeros(glove_model.vector_size) for word in words]

    return embeddings
```

شکل 2: embedded کردن جملات

### 3- پیاده سازی طبقه بندی نیت

در اینجا دو مدل گفته شده در مقاله پیاده سازی شده است. در مدل اول، با دو hidden\_size مختلف (25 و 100) مدل برازش شده است.

در مدل اول با hidden\_size=100، مقادیر خواسته شده در epoch آخر برابر شده است با:

```
loss: 0.0515
accuracy: 0.9967
precision: 0.9906
recall: 0.9894
f1_score: 0.9900
val_loss: 0.5184
val_accuracy: 0.9674
val_precision: 0.9038
val_recall: 0.9004
val_f1_score: 0.9020
```

در مقادیری که در مقاله آمده است، accuracy برابر 99.98 درصد در داده های آموزش شده است که در این پروژه برابر 99.67 درصد شده است. همچنین برای مقادیر validation نیز در مقاله برابر 90.20 درصد شده که در این پروژه برابر 96.74 درصد شده است که نتیجه بهتری حاصل شده است که به نوع خود جالب است.

در مدل اول با hidden\_size=25، مقادیر خواسته شده در epoch آخر برابر شده است با:

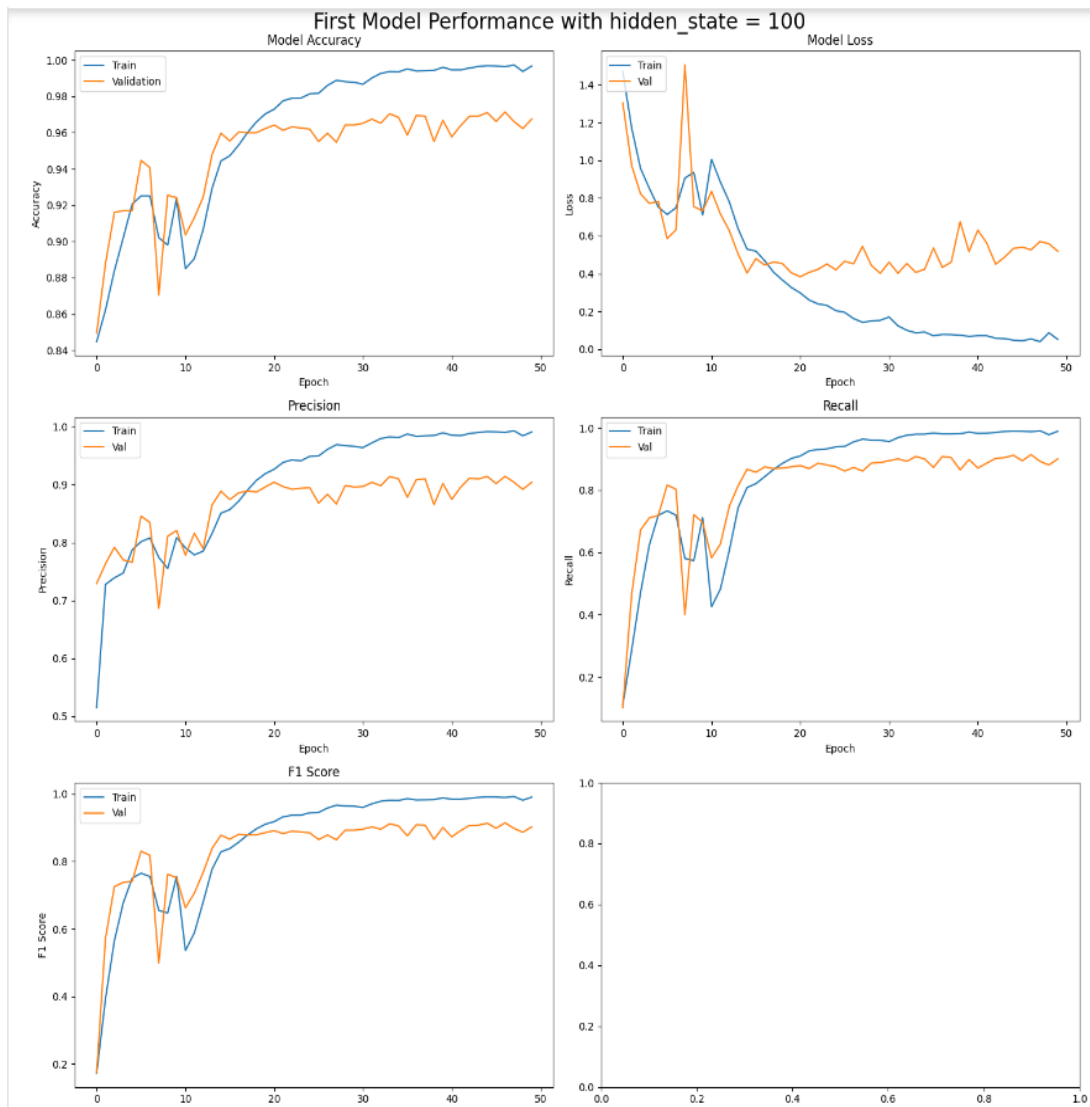
```
loss: 0.1720
accuracy: 0.9898
```

```
precision: 0.9726
recall: 0.9659
f1_score: 0.9691
val_loss: 0.6730
val_accuracy: 0.9515
val_precision: 0.8577
val_recall: 0.8496
val_f1_score: 0.8535
```

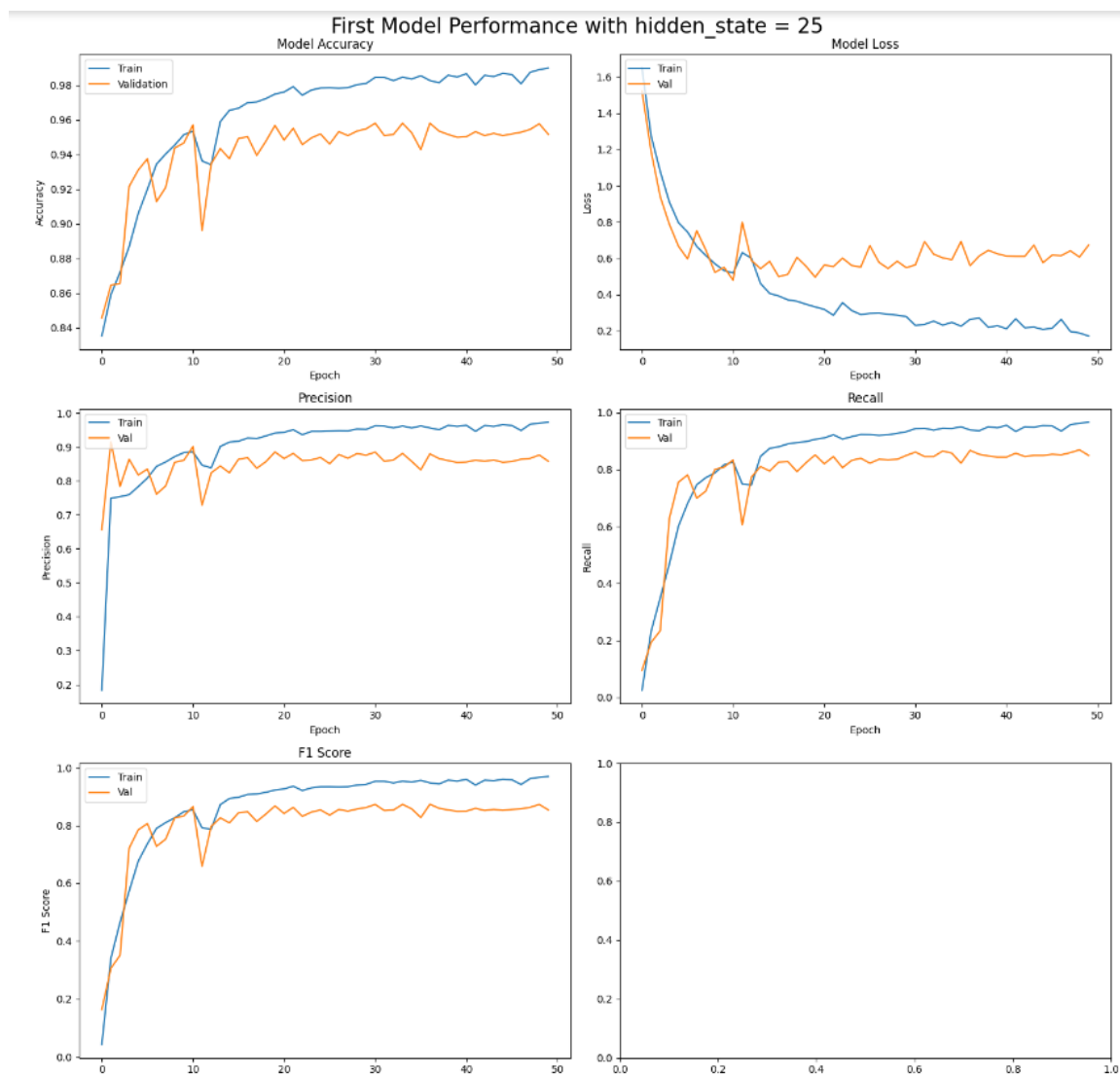
در مقادیری که در مقاله آمده است، accuracy برابر 99.26 درصد در داده های آموزش شده است که در این پروژه برابر 98.98 درصد شده است. همچنین برای مقادیر validation نیز در مقاله برابر 87.80 درصد شده که در این پروژه برابر 95.15 درصد شده است که نتیجه بهتری حاصل شده است که به نوع خود جالب است.

با توجه به نتایج بالا، تعداد hidden\_size با مقدار 100 نسبت به 25 بهتر عمل کرده است، چون پیچیدگی بیشتری برای مدل به ارمغان آورده است و توانسته مقادیر سنجش بهتری را برای طبقه بندی کلاس اصلی به وجود بیاورد. لزوماً مقدار زیاد hidden\_size نیز نمیتواند خوب باشد و میتواند منجر به زمان بالای آموزش و تعداد پارامترهای خیلی زیاد شود که آن نیز موجب overfitting میشود.

در مدل اول کلاس اصلی پیش بینی میشود که مقادیر مختلف سنجش های مختلف در شکل زیر آمده است:



شکل 3: مقادیر accuracy, precision, recall, f1\_score برای مدل اول با hidden\_state=100



شکل 4: مقادیر accuracy, precision, recall, f1\_score برای مدل اول با hidden\_state=25

جدول 1: مقادیر accuracy مدل اول و مقاله

h dimention	Training set (%)	Test set (%)	Training set in paper (%)	Test set in paper (%)
25	99.67	96.74	99.26	87.80
100	98.98	95.15	99.98	90.20

Confusion matrix for main classes classification:

```
[[128  6  3  0  0  1]
 [  6 76  0  4  1  7]
 [  3  0  6  0  0  0]
 [  0  4  0 60  0  1]
 [  6  0  0  0 104  3]
 [  1  5  0  0  0 75]]
```

شکل 5: ماتریس درهم ریختگی برای مدل اول با hidden\_state=100

Confusion matrix for main classes classification:

```
[[130  4  3  0  1  0]
 [ 20 62  2  8  1  1]
 [  3  0  6  0  0  0]
 [  0  2  0 62  0  1]
 [ 13  1  0  0 96  3]
 [  0  5  2  0  1 73]]
```

شکل 6: ماتریس درهم ریختگی برای مدل اول با hidden\_state=25

مدلی که استفاده شده است، مدلی است که بعد از embedded کردن، وارد بلوک های LSTM شده و بعد از آن نیز یک لایه Dropout با مقدار 0.5 گذاشته شده است. در آخر خروجی بلوک آخر از یک لایه softmax عبور داده میشود که خروجی به تعداد کلاس های اصلی میباشد. شکل مدل با hidden\_size=100 در شکل زیر قابل مشاهده است.



```

hidden_size = 100

model1 = Sequential()
model1.add(LSTM(hidden_size,
                 input_shape=(max_length, glove_model.vector_size)))
model1.add(Dropout(0.5))
model1.add(Dense(output_size_main, activation='softmax'))

model1.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100)	160400
dropout (Dropout)	(None, 100)	0
dense (Dense)	(None, 6)	606
Total params: 161,006		
Trainable params: 161,006		
Non-trainable params: 0		

شکل 7: مدل اول با hidden\_size=100

مدل اول با hidden\_state=25 نیز در شکل زیر قابل مشاهده است که تعداد پارامترهای آن تقریباً یک پنجم بوده و در نتیجه پیچیدگی کمتری مهیا کرده است.

```

hidden_size = 25

model2 = Sequential()
model2.add(LSTM(hidden_size,
                 input_shape=(max_length, glove_model.vector_size)))
model2.add(Dropout(0.5))
model2.add(Dense(output_size_main, activation='softmax'))

model2.summary()

```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 25)	32600
dropout_1 (Dropout)	(None, 25)	0
dense_1 (Dense)	(None, 6)	156
Total params: 32,756		
Trainable params: 32,756		
Non-trainable params: 0		

شکل 8: مدل اول با hidden\_size=25

در مدل دوم، یک مقدار pad به آخر سوال اضافه میشود و سپس به مدل داده میشود. همچنین در مدل نیز دو بلوک آخر lstm دارای خروجی خواهد شد که یکی مانده به آخر برای پیش بینی همان کلاس اصلی بوده ولی برای آخر برای پیش بینی زیرکلاس که 47 حالت بود میباشد. برای این کار دو loss باید تعریف شود که هر دو categorical\_crossentropy خواهد بود که برای مسئله کلاس بندی که خروجی بیش از دو تا دارند، مناسب میباشد. ما برای اضافه کردن pad، به طور کلی یک مقدار max\_length در نظر گرفتیم و برای مدل نیز دو خروجی تعریف کردیم که یکی به اندازه تعداد کلاس های اصلی و دیگری به اندازه تعداد زیرکلاس ها میباشد.

در مدل دوم، metric هایی که برای زیرکلاس در نظر گرفتیم، مقادیر بسیار پایین تری نسبت به metric ها برای کلاس اصلی شده است که این امر کاملاً میتوان در مقدار f1\_score متوجه شد. یکی از دلایل آن این است که تعداد مقادیری که میتواند بگیرد، 47 تا بوده و نسبت به 6 که برای کلاس اصلی میباشد تفاوت خیلی زیادی دارد. این باعث میشود که داده بسیار بیشتری برای یادگیری زیرکلاس نیاز شود تا بتواند مقادیر metric را بهبود ببخشد.

در اینجا برای زیرکلاس ها نیز مقادیر metric برای hidden\_size=100 نسبت به hidden\_size=25 عملکرد بهتری داشته است، چون پیچیدگی بالاتری دارد و الگوریتم های پیچیده تر و تعداد پارامترهای بیشتری را یاد میگیرد.

بهتر است مقادیر  $f1\_score$  را بیشتر مدنظر داشت که برآورد بهتری از عملکرد مدل بدهد که بر اساس آن توانسته تا حد خوبی زیرکلاس ها را یاد بگیرد اما به دلیل تعداد زیاد زیرکلاس ها و همچنین stratify داده ها نسبت به کلاس اصلی عملکرد خیلی خوبی از خود نشان نداده است.

مدل با  $hidden\_size=25$  و  $hidden\_size=100$  به شکل زیر میباشد.

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	[(None, 50, 300)]	0	[]
lstm_layer (LSTM)	(None, 50, 100)	160400	['input_layer[0][0]']
dropout_2 (Dropout)	(None, 50, 100)	0	['lstm_layer[0][0]']
tf.__operators__.getitem (SlicingOpLambda)	(None, 100)	0	['dropout_2[0][0]']
tf.__operators__.getitem_1 (SlicingOpLambda)	(None, 100)	0	['dropout_2[0][0]']
main_class (Dense)	(None, 6)	606	['tf.__operators__.getitem[0][0]']
sub_class (Dense)	(None, 47)	4747	['tf.__operators__.getitem_1[0][0]']

=====  
Total params: 165,753  
Trainable params: 165,753  
Non-trainable params: 0

شکل 9: مدل دوم با  $hidden\_size=100$

Model: "model\_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	[(None, 50, 300)]	0	[]
lstm_layer (LSTM)	(None, 50, 25)	32600	['input_layer[0][0]']
dropout_3 (Dropout)	(None, 50, 25)	0	['lstm_layer[0][0]']
tf.__operators__.getitem_2 (SlicingOpLambda)	(None, 25)	0	['dropout_3[0][0]']
tf.__operators__.getitem_3 (SlicingOpLambda)	(None, 25)	0	['dropout_3[0][0]']
main_class (Dense)	(None, 6)	156	['tf.__operators__.getitem_2[0][0]']
sub_class (Dense)	(None, 47)	1222	['tf.__operators__.getitem_3[0][0]']

=====  
Total params: 33,978  
Trainable params: 33,978  
Non-trainable params: 0

شکل 10: مدل دوم با  $hidden\_size=25$

برای hidden\_size=100 در epoch آخر metric های گفته شده به شکل زیر شده است:

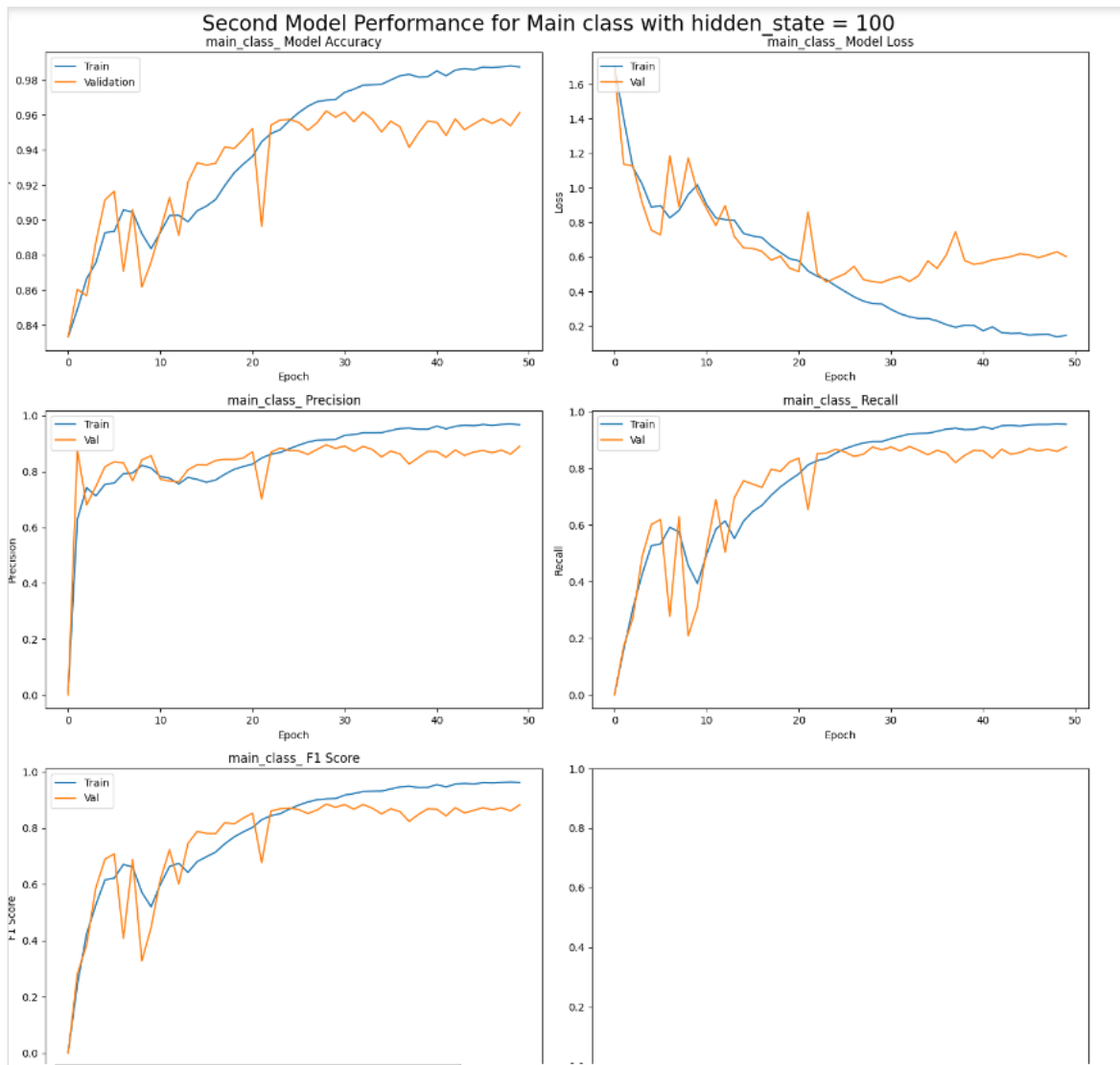
```
loss: 0.9647 - main_class_loss: 0.1463 - sub_class_loss: 0.8184 -  
main_class_accuracy: 0.9874 - main_class_precision: 0.9685 -  
main_class_recall: 0.9551 - main_class_f1_score: 0.9615 -  
sub_class_accuracy: 0.9916 - sub_class_precision: 0.8971 -  
sub_class_recall: 0.6846 - sub_class_f1_score: 0.7724 - val_loss:  
2.2805 - val_main_class_loss: 0.6023 - val_sub_class_loss: 1.6782 -  
val_main_class_accuracy: 0.9613 - val_main_class_precision: 0.8918 -  
val_main_class_recall: 0.8750 - val_main_class_f1_score: 0.8827 -  
val_sub_class_accuracy: 0.9872 - val_sub_class_precision: 0.7641 -  
val_sub_class_recall: 0.5742 - val_sub_class_f1_score: 0.6528
```

برای hidden\_size=100 در epoch آخر metric های گفته شده به شکل زیر شده است:

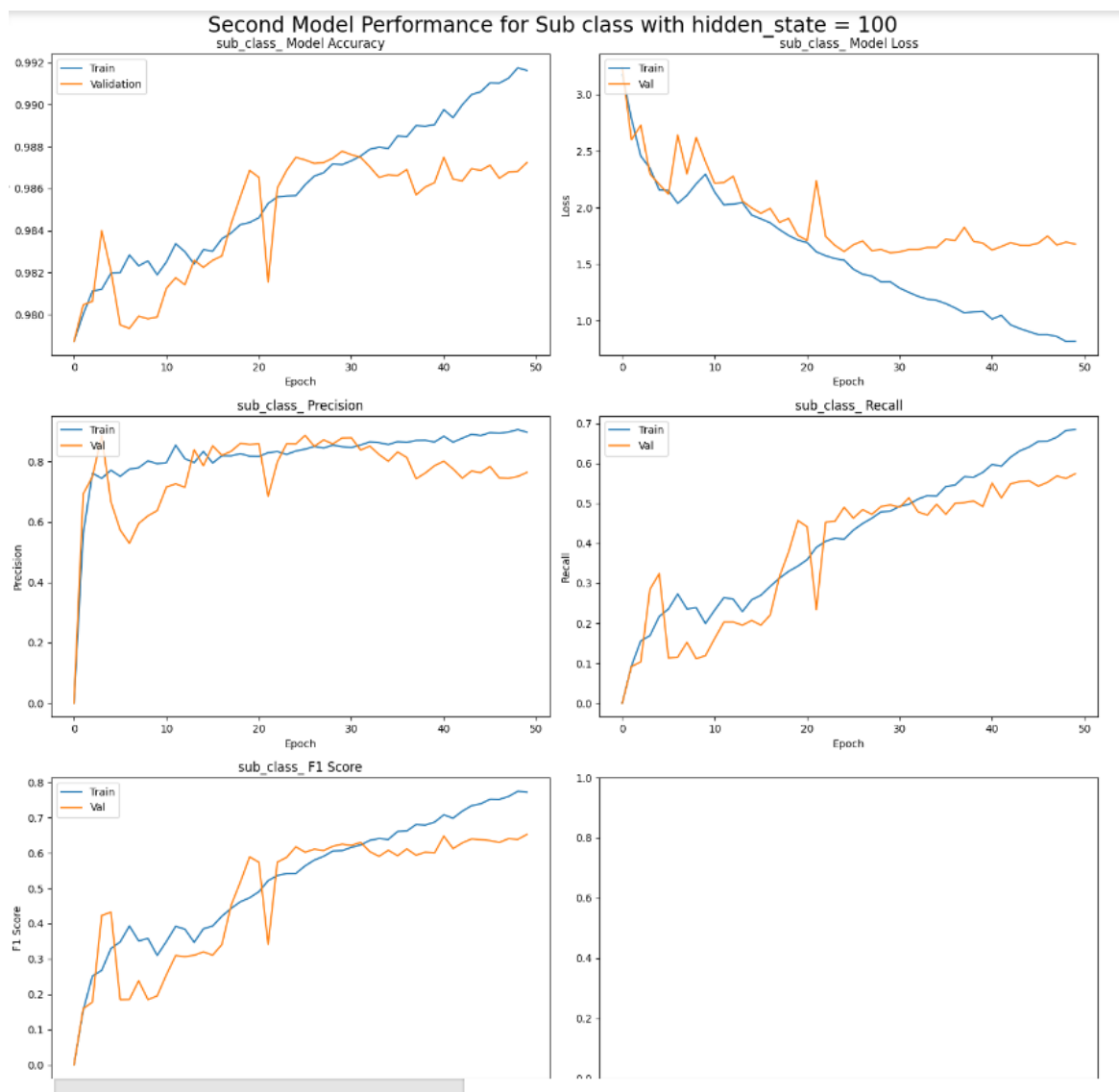
```
loss: 2.0975 - main_class_loss: 0.4643 - sub_class_loss: 1.6333 -  
main_class_accuracy: 0.9618 - main_class_precision: 0.9032 -  
main_class_recall: 0.8631 - main_class_f1_score: 0.8820 -  
sub_class_accuracy: 0.9839 - sub_class_precision: 0.7473 -  
sub_class_recall: 0.3665 - sub_class_f1_score: 0.4846 - val_loss:  
2.5746 - val_main_class_loss: 0.6302 - val_sub_class_loss: 1.9444 -  
val_main_class_accuracy: 0.9427 - val_main_class_precision: 0.8354 -  
val_main_class_recall: 0.8184 - val_main_class_f1_score: 0.8265 -  
val_sub_class_accuracy: 0.9835 - val_sub_class_precision: 0.6877 -  
val_sub_class_recall: 0.3984 - val_sub_class_f1_score: 0.4985
```

metric های خواسته شده برای مدل دوم با هر دو hidden\_size (100 و 25) و برای هر دو

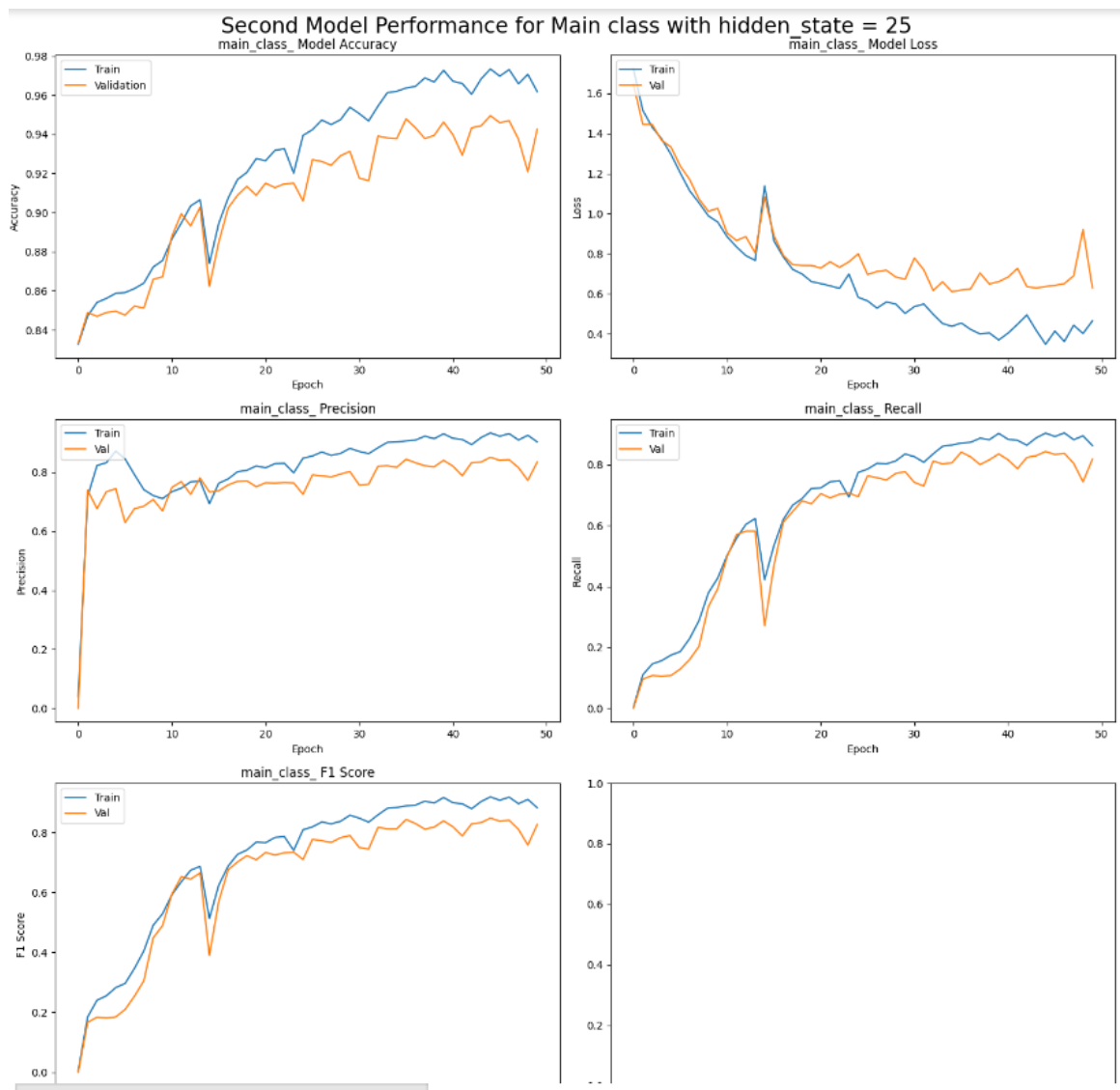
حالت کلاس اصلی و زیرکلاس به شکل زیر است که با مقادیر گفته شده در مقاله آمده است:



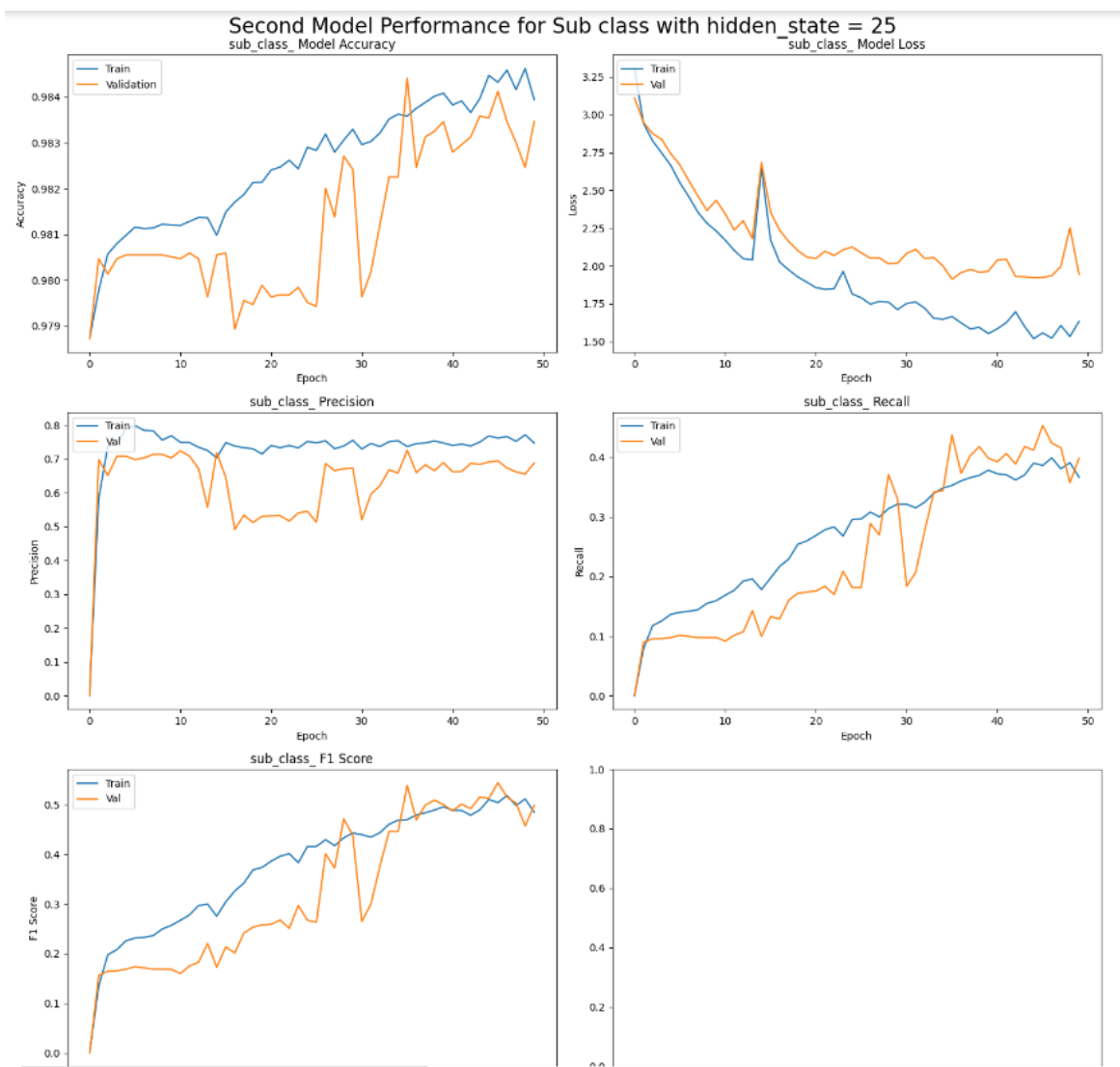
شکل 11: مقادیر accuracy, precision, recall, f1\_score برای مدل دوم و کلاس های اصلی با hidden\_state=100



شکل 12: مقادیر accuracy, precision, recall, f1\_score برای مدل دوم و زیرکلاس ها با hidden\_state=100



شکل 13: مقادیر accuracy, precision, recall, f1\_score برای مدل دوم و کلاس های اصلی با hidden\_state=25



شکل 14: مقادیر accuracy, precision, recall, f1\_score برای مدل دوم و زیرکلاس ها با hidden\_state=25



Confusion matrix for main classes classification:

```
[[131  1  6  0  0  0]
 [ 12 75  0  4  1  2]
 [  3  0  6  0  0  0]
 [  1  3  0 61  0  0]
 [  9  2  1  0 97  4]
 [  4  6  0  0  0 71]]
```

Confusion matrix for sub classes classification:

```
[[2 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 1 7 ... 0 0 0]
 ...
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

شکل 15: ماتریس درهم ریختگی برای مدل دوم با hidden\_state=100

Confusion matrix for main classes classification:

```
[[119 16  0  0  1  2]
 [ 14 66  5  4  1  4]
 [  2  0  7  0  0  0]
 [  0  4  0 60  0  1]
 [  8  0  0  2 96  7]
 [  5  5  0  0  8 63]]
```

Confusion matrix for sub classes classification:

```
[[2 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 7 0 ... 0 0 0]
 ...
 [0 2 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

شکل 16: ماتریس درهم ریختگی برای مدل دوم با hidden\_state=25

جدول 2: مقادیر accuracy مدل دوم و مقاله

h Dimension	Paper				Project			
	Training set		Test set		Training set		Test set	
	Main class (%)	Sub class (%)	Main class (%)	Sub class (%)	Main class (%)	Sub class (%)	Main class (%)	Sub class (%)
25	99.24	96.86	86.20	74.40	96.18	98.39	94.27	98.35
100	99.82	99.67	91.20	82.20	98.74	99.16	96.13	98.72

نکته: مقدار accuracy برخلاف دیگر metric ها به طرز عجیبی بالاتر بوده و به صورت هم  $metrics=['accuracy']$  و هم  $metrics=[accuracy()]$  که به صورت دستی پیاده سازی شده است، امتحان شده است و هر دو مقدار بسیار بالایی را نشان میدادند که دلیل آن را متوجه نشدم اما بقیه metric ها متادیر معقولی را نشان میدادند.

#### 4- پیاده سازی مدل Responder

مدلی که طراحی کردیم به شکل زیر است و نمونه های جواب در ادامه آمده است. در این مدل از یک BiDirectional Layer استفاده کردیم که از وزن های مدل دوم با 100 تا hidden\_size استفاده میکند. در آخر از softmax استفاده کرده و همچنین از mse برای loss استفاده کردیم.

```
hidden_size = 100

input_shape = (max_length, glove_model.vector_size)
input_layer = Input(shape=input_shape, name='input_layer')

# Feed the remaining LSTM hidden states as input to the BLSTM
blstm_layer = Bidirectional(LSTM(units=hidden_size, return_sequences=True, name='blstm_layer'), weights=model3.layers[1].get_weights())(input_layer)
blstm_layer = Dropout(0.5)(blstm_layer)

dense_layer = Dense(units=glove_model.vector_size, name='dense_layer', activation='softmax')(blstm_layer)

model_response = Model(inputs=input_layer, outputs=dense_layer)

model_response.summary()
```


Model: "model\_2"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	[(None, 50, 300)]	0
bidirectional_1 (Bidirectional)	(None, 50, 200)	320800
dropout_2 (Dropout)	(None, 50, 200)	0
dense_layer (Dense)	(None, 50, 300)	60300

Total params: 381,100  
 Trainable params: 381,100  
 Non-trainable params: 0

شکل 17: مدل responder

جواب های به دست آمده دقیق نیستند و نمیتوان به آنها استناد کرد اما میتوان جنس آنها را دید که تقریباً درست میباشند.

✓ 2s  generate\_answer('How many people speak French?', model\_response)

1/1 [=====] - 0s 21ms/step  
'10'

✓ 2s [42] generate\_answer('What day is today?', model\_response)

1/1 [=====] - 0s 22ms/step  
'week'

✓ 2s [43] generate\_answer('Who will win the war?', model\_response)

1/1 [=====] - 0s 21ms/step  
'last'

✓ 2s [44] generate\_answer('Who is Italian first minister?', model\_response)

1/1 [=====] - 0s 23ms/step  
'40'

✓ 2s [45] generate\_answer('When World War II ended?', model\_response)

1/1 [=====] - 0s 32ms/step  
'80'

✓ 2s [46] generate\_answer('When Gandhi was assassinated?', model\_response)

1/1 [=====] - 0s 19ms/step  
'1823'

شکل 18: نتایج مدل responder برای مثال های خواسته شده

## پاسخ 1 – توصیف عکس

### 1. Model with Frozen CNN

ابتدا دیتاست مربوطه را در محیط گوگل کولب دانلود می کنیم. سپس توابعی برای لود کردن و Preprocess کردن دیتاست پیاده سازی می کنیم. از آنجایی که تقریباً 8000 عکس داریم و ورودی معمول

شبکه Resnet18 به صورت  $224 \times 224$  پیکسل می باشد، لود کردن کل تصاویر دیتاست روی رم سیستم به صورت Float32، حدود 4.8 گیگابایت فضا نیاز خواهد داشت و با توجه به داشتن رم کافی در محیط کولب، کل دیتاست را در همان ابتدا لود کرده و پیش پردازش می کنیم تا فرآیند آموزش را در سریع ترین حالت ممکن جلو ببریم و عملیات IO تبدیل به یک bottleneck برای آموزش نشوند.

برای پیش پردازش کپشن ها، از راهکار های مختلفی مثل lowercase کردن حروفات، حذف علائم نگارشی، حذف whitespace های اضافی و حذف حروفاتی که فقط یک حرف دارند استفاده می کنیم. همچنین به انتهای تمام جملات توکن endseq را اضافه می کنیم که پایان جمله را اعلام می کند.

چون در مدل این مقاله، شبکه LSTM فقط یکبار فیچر های تصویر را دریافت می کند و شبکه خطی بالایی نیز اطلاعی از این فیچر ها ندارد، طبق آزمایشات ما استفاده از توکن startseq در ابتدای جمله باعث می شود که تولید کلمات با مفهوم جمله یک گام به تعویق بیفتند و این باعث افت عملکرد مدل می شود. همچنین نیاز خاصی نیز به این توکن نداریم چرا که در ادامه از Post Padding برای یکسان سازی طول جملات استفاده می کنیم به طوری که جمله همواره در ابتدای Sequence ظاهر می شود. پس در نتیجه فقط به توکن endseq نیاز داریم.

به دلیل مشابه، کلمات یک حرفی مانند a که اکثرا در ابتدای جملات ظاهر می شوند (a man is ...) نیز می توانند عملکرد مدل را تحت تاثیر قرار دهند. به همین دلیل این کلمات نیز در فرایند پیش پردازش از جملات حذف شده اند.

در این قسمت مدل Resnet18 را لود کرده و آن را Freeze می کنیم. لایه خطی پیش فرض در انتهای شبکه Resnet را لود نمی کنیم چرا که Activation Function این لایه Softmax می باشد که به علت استفاده از عبارات exponential معمولاً تمایل دارد در خروجی نوروں های فعال تر را تقویت و نوروں هایی فعالیت کمتری دارند را تضعیف کند. این اتفاق در این مسئله مطلوب نیست چرا که نمی خواهیم مدل فقط روی یک ویژگی تمرکز کند بلکه باید متوجه حضور چندین ویژگی همزمان در تصویر شود. در ادامه نیز طبق آزمایشی که انجام دادیم، عدم استفاده از این لایه علاوه بر کاهش پارامتر های مدل، دقت را روی داده های ارزیابی کاهش نمی دهد.

طبق این توضیحات ترجیح دادیم لایه خطی انتهای Resnet را لود نکنیم و فقط از یک لایه خطی به سائز 300 روی فیچر مپ انتهای Resnet استفاده کنیم که ویژگی ها را به فضای embedding ببرد. همچنین استفاده از تابع LeakyRelu برای این لایه بهترین نتیجه را حاصل کرد.

مدل پیاده سازی شده از قرار زیر می باشد:

```
def build_model(trainable_cnn):

    resNet = ResNet18(
        include_top=False, weights="imagenet", input_shape=(image_dim, image_dim, 3))

    resNet.trainable = trainable_cnn

    input1 = keras.layers.Input(shape=(image_dim, image_dim, 3))
    x = resNet(input1, training=False)
    x = keras.layers.GlobalAveragePooling2D()(x)
    x = keras.layers.Dense(300, activation=keras.layers.LeakyReLU(alpha=0.05))(x)
    features = keras.layers.Reshape((1, 300))(x)

    input2 = keras.layers.Input(shape=(max_length-1,))
    embeddings = keras.layers.Embedding(vocab_size, 300)(input2)

    x = keras.layers.Concatenate(axis=1)([features, embeddings])
    x = keras.layers.LSTM(256, return_sequences=True, dropout=0.2)(x)
    output = keras.layers.Dense(vocab_size, activation='softmax')(x)

    model = keras.models.Model(inputs=[input1, input2], outputs=output)

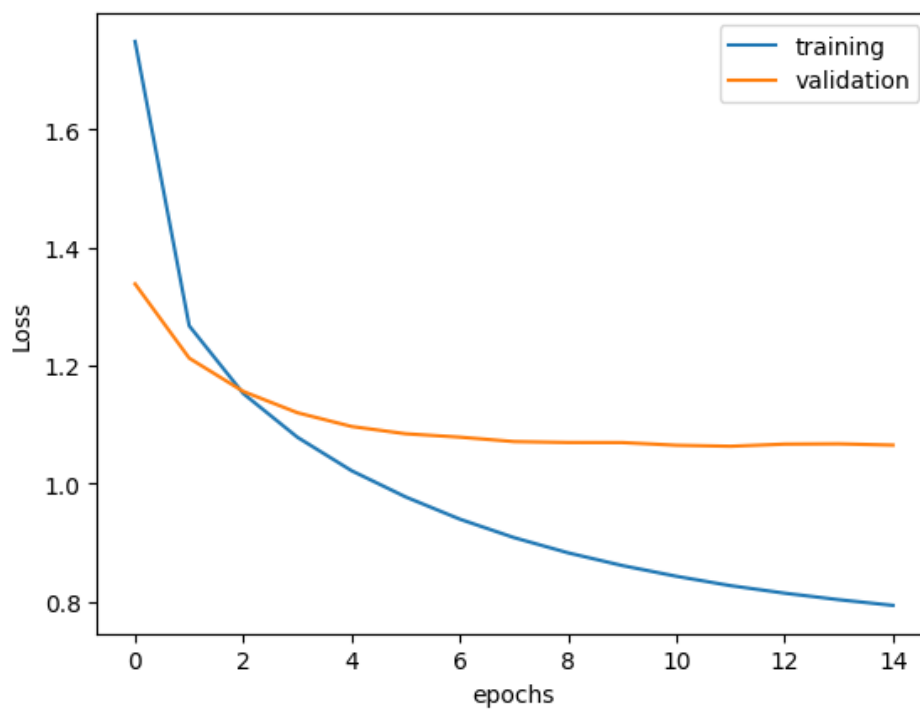
    return model
```

### شکل 19: پیاده سازی مدل در Tensorflow

برای خروجی لایه LSTM نیز از dropout با احتمال 0.2 استفاده می کنیم که در آزمایشات ما روی خروجی صحیح مدل روی داده های تست تاثیر بسیاری خوبی داشت.

آموزش این مدل ها، به خصوص مدل دوم که نیاز به آموزش مدل CNN نیز دارد بسیار طول می کشد در نتیجه برای اینکه فرآیند آموزش مطمئن تر و پایدارتری داشته باشیم از Learning Rate Scheduler نیز استفاده می کنیم که در هر epoch به صورت نمایی با ضریب 0.85 نرخ یادگیری را کاهش می دهد.

سپس دیتاست را به دو بخش Train و Test تقسیم می کنیم و Data Generator مربوط به آنها را می سازیم. در ادامه مدل را با استفاده از این دیتاست آموزش می دهیم که نمودار خطای آموزش و تست و همچنین خروجی مدل روی سه عدد از دادگان تست در ادامه قابل مشاهده است.



شکل 20: نمودار خطای آموزش و تست



two men in red and white uniforms are playing soccer endseq

شکل 21: خروجی مدل روی داده تست



two boys are playing in water fountain endseq

شکل 22: خروجی مدل روی داده تست



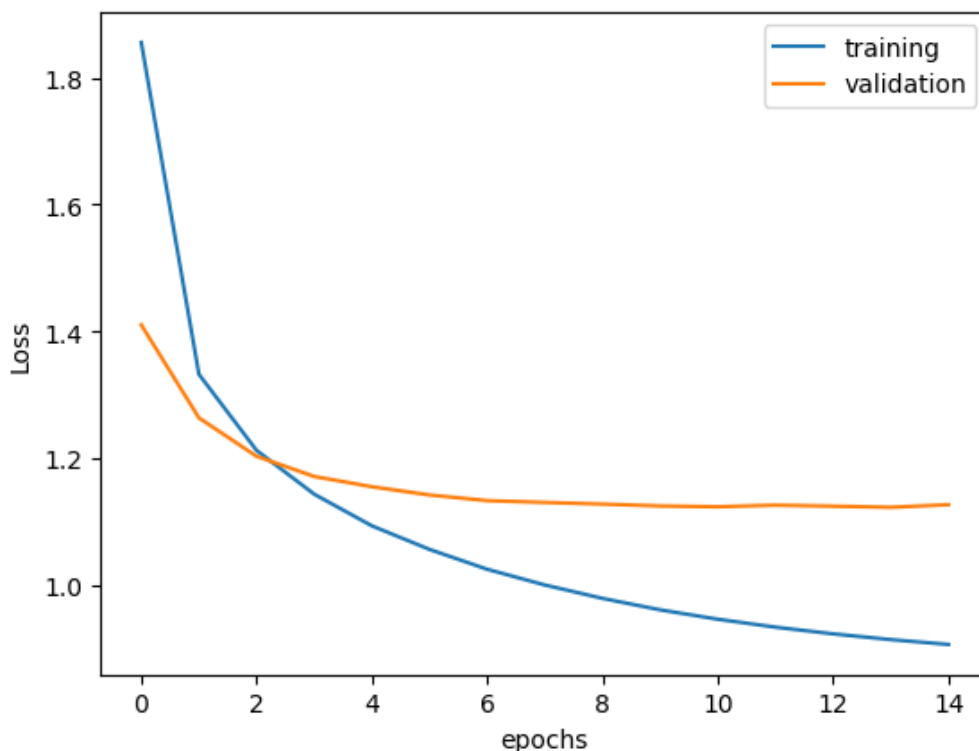
dog is running through the water endseq

شکل 23: خروجی مدل روی داده تست

## Model with Trainable CNN 2

در اینجا نیز همانند توضیحات بخش قبل عمل می کنیم با این تفاوت که به مدل Resnet اجازه آموزش می دهیم. البته باید توجه کنیم که در هنگام استفاده از Resnet در مدل، همچنان پارامتر training را False قرار می دهیم تا لایه های Batch Norm در حالت inference اجرا شوند و توزیع هایی که روی imagenet یادگرفته اند از بین نرود.

در این حالت مدل نمی تواند به خوبی تسک مورد نظر را یاد بگیرد چرا که آموزش شبکه CNN نیاز به داده های بیشتری دارد. همچنین شبکه CNN بدون توجه به این که قبلا آموزش داده شده است به همراه شبکه آموزش داده نشده در انتهای خود از همان ابتدا آموزش می بیند و گرادیان های بزرگی که در ابتدای آموزش ایجاد می شوند، وزن های یادگرفته شده توسط Resnet را به صورت نامطلوب تغییر می دهند. در این تسک بهتر بود پس از آموزش مدل بخش قبل، همان را Unfreeze کرده و Fine Tune می کردیم. ولی به هر حال طبق خواسته سوال این مدل را ایجاد کرده و از ابتدا آموزش می دهیم.



شکل 24: نمودار خطای آموزش و تست

با اینکه در اینجا مدل پارامتر های بیشتری برای یادگیری داشت، خطای مدل روی داده های آموزش بیشتر از مدل بخش قبل است.





man in black shirt and black pants is standing on the sidewalk endseq

شکل 25: خروجی مدل روی داده تست



man in black shirt and black pants is standing on the sidewalk endseq

شکل 26: خروجی مدل روی داده تست



man in black shirt and black pants is standing on the sidewalk endseq

شکل 27: خروجی مدل روی داده تست

مدل روی داده های تست بسیار بد عمل می کند به صورتی که برای همه آنها یک کپشن تولید می کند!