



درس شبکه‌های عصبی و یادگیری عمیق
تمرین امتیازی

مبینا مهرآذر - 810100216

محمد رضا محمد هاشمی - 810100206

فهرست

1.....	پرسش 1. تنظیم دقیق مدل‌های زبانی بزرگ برای گفتگو در زبان فارسی.....
1.....	1-1. دادگان و انتخاب مدل.....
1.....	2-1. روش‌های SoftPrompts.....
1.....	3-1. روش‌های مبتنی بر LoRA.....
1.....	4-1. تغییر وزن برخی از لایه‌ها.....
1.....	5-1. جمع‌بندی و تحلیل مقایسه‌ای.....
2.....	پرسش 2 - تولید کپشن برای تصاویر ((Image Captioning)).....
2.....	1-2. مقدمه
2.....	2-2. آماده‌سازی دیتابست
2.....	3-2. پیاده‌سازی CNN-RNN
2.....	4-2. پیاده‌سازی Attention based CNN-RNN
2.....	5-2. پیاده‌سازی CNN-Transformer
2.....	6-2. بخش امتیازی

پرسش 1. تنظیم دقیق مدل‌های زبانی بزرگ برای گفتگو در زبان فارسی

1-1. دادگان و انتخاب مدل

پک نمونه از دادگان این دیتا ست را در زیر مشاهده می‌کنید:

'role': 'های تراکتور، تراکتور از مزرعه، تراکتور با تریلر200c\u200cها را اضافه کنید: بازی200c\u200cفاصله: 'content': 'user'}, {'content': 'های اضافه شده آمده است200c\u200cدر اینجا عبارات جدا شده با فاصله: 'user'}, {'content': 'ن1.\n\nن1. حال اجازه دهید توضیح مفصلی درباره تراکتور با تریلر. ن3.\n\nن3. تراکتور از مزرعه. ن2.\n\nن2. های تراکتور200c\u200cبازی ای200c\u200cهاي تراکتور به دسته200c\u200cهاي تراکتور: بازی200c\u200cبازی. ن1.\n\nن1. هر عبارت جدا شده ارائه دهنده تراکتورها به عنوان بخش اصلی یا200c\u200cهاي ويدئوي يا آنلاین اشاره دارد که در آن200c\u200cاز بازی توانند اهداف و مقاصد مختلفی داشته200c\u200cها می200c\u200cها ناپذیر بازی وجود دارند. این بازی200c\u200cجدايی های تراکتور به طبق200c\u200cسازی کشاورزی يا هدایت تراکتور. بازی200c\u200cباشند، از جمله مسابقه، شبیه سازی200c\u200cدانند، تا بزرگسالانی که از شبیه200c\u200cوسیعی از افراد، از کودکان که تراکتورها را جذاب می200c\u200cبرند، جذاب است. برخی از200c\u200cآلات سنگین لذت می200c\u200cهاي کار با ماشین200c\u200cچالش ساز کشاورزی، جنون تراکتور، آزمایش تراکتور و بسیاری200c\u200cهاي محبوب تراکتور شامل شبیه200c\u200cبازی های هوشمند،200c\u200cهاي مختلفی مانند کامپیوتر، گوشی200c\u200cها در پلتفرم200c\u200cديگر است. این بازی های مبتنی بر مرورگر ارائه200c\u200cهاي که بازی200c\u200cسايت200c\u200cهاي بازی و حتی وب200c\u200cکنسول تراکتور از مزرعه: این عبارت احتمالاً به یک تراکتور واقعی اشاره. ن2.\n\nن2. دهنده، در دسترس هستند200c\u200cامي آلات چندمنظوره200c\u200cشود. تراکتورها ماشین200c\u200cدارد که در یک مزرعه يا زمین کشاورزی استفاده می200c\u200cها وسایل نقلیه تخصصی هستند که برای200c\u200cقدرتمندی هستند که در مزارع مدرن ضروری هستند. آن ورزی، کاشت و برداشت200c\u200cاند، مانند شخم زدن، خاک200c\u200cانجام وظایف مختلفی طراحی شده های مختلف قدرت200c\u200ctوانند با دامنه200c\u200cشان، تراکتورها می200c\u200cمحصولات. بسته به انداره توانند با انواع مختلفی از ملحقات برای انجام وظایف خاص تجهیز شوند. 200c\u200cخروجی عرضه شوند و می200c\u200cها، تجهیزات کشت، تریلرها برای حمل کالا و200c\u200cزن200c\u200cبرخی از ملحقات رایج تراکتور شامل شخم های جلو برای بلند کردن و جابجایی اشیاء سنگین است. بسیاری از مزارع حداقل یک تراکتور200c\u200cبارکننده دارند و معمولاً به عنوان یک ابزار ضروری برای عملیات کشاورزی کارآمد و مؤثر در نظر گرفته تراکتور با تریلر: تراکتور با تریلر به تراکتوری اشاره دارد که به یک تریلر متصل است و. ن3.\n\nن3. تواند برای مقاصد مختلف استفاده شود. تریلرها برای حمل کالا، تجهیزات و مواد از یک مکان به200c\u200cامي مکان دیگر در مزرعه يا بین مزارع ضروری هستند. ترکیب تراکتور و تریلر امکان استفاده مؤثر از انرژی را فراهم دهد که بارهای بزرگ را در یک سفر حمل کنند. تریلرهای200c\u200cکند و به کشاورزان این امکان را می200c\u200cامي

بارها، تریلرهای بسته و تریلرهای تخصصی ۲۰۰\۲۰۰ توانند انواع مختلفی داشته باشند، مانند تخت ۲۰۰\۲۰۰ می‌برای وظایف خاص مانند حمل دام، آب یا لجن. کشاورزان در کارهای روزمره خود از تراکتورها با تریلرهای استفاده ۲۰۰\۲۰۰ جویی کنند و هزینه ۲۰۰\۲۰۰ وری را افزایش دهند، زمان را صرفه ۲۰۰\۲۰۰ کنند تا بهره ۲۰۰\۲۰۰ می‌کار را کاهش دهند

همانطور که قابل مشاهده است دیتا پوینت‌ها از سه قسمت اصلی تشکیل شده اند:

1. پیام سیستم : یک پیام کلی که نحوه پاسخ دهی مدل را مشخص می‌کند.
2. پیام کاربر : نمونه‌ای از ورودی کاربر است.
3. پیام دستیار : نمونه‌ای از پاسخ GPT-4 به آن سوال است.

از برخی از علل ایجاد دادگان به این صورت به موارد زیر می‌توان اشاره کرد :

1. شبیه سازی نقش مدل در مکالمات و کمک به درک مدل از نقشش در مکالمات
2. وجود پیام‌های سیستم جهت کنترل دستورالعمل‌ها
3. کمک به مدل در تفکیک نقش کاربر و مدل در مکالمات

تفاوت نسخه base و instruct :

نسخه instruct از مدل LLaMA 3.2 به طور ویژه برای دنبال کردن دستورات کاربر تنظیم و بهینه شده است. در این نسخه، با استفاده از داده‌های آموزشی مبتنی بر دستورالعمل، پاسخ‌ها به شکل دقیق‌تر و کاربردی‌تری ارائه می‌شوند.

در مقابل، نسخه base مدل به عنوان مدل پایه، بدون تنظیمات اضافی و خاص برای پیروی از دستورات آموزش دیده است.

به همین دلیل، instruct در تولید پاسخ‌های منسجم، هدفمند و سازگار با نیازهای کاربر عملکرد بهتری دارد. این تفاوت باعث می‌شود که نسخه instruct برای کاربردهای تعاملی و ارائه راهنمایی‌های دقیق‌تر توصیه شود.

به دلیل فرمت دادگان و کارایی بالاتر مدل instruct ما تصمیم به استفاده از مدل llama3.2 3b instruct گرفتیم.

آماده سازی دادگان:

در این مرحله ابتدا فایل json مجموعه داده را دانلود کرده و آن را به کمک قطعه کد زیر پردازش می‌کنیم و آن را به یک دیتابست تبدیل می‌کنیم :

```
def read_dataset(ds_path):  
    dataset = []
```

```

with open(ds_path, 'r') as f:
    for line in f:
        line = json.loads(line)
        filtered_conversations = [
            message for message in line
            if message['role'] in ['user', 'assistant']
        ]
        conversation = {
            "conversations": filtered_conversations
        }
        dataset.append(conversation)
return dataset

```

و پس از آن دادگان را به کمک توکنایزر مدل لاما به توکن های اماده ورود به مدل توکنایز میکنیم.

```

tokenizer.apply_chat_template(convos,
tokenize=False, add_generation_prompt=False) for
convos in convos

```

تست مدل:

برای پرامپت (یک داستان کوتاه در مورد دانشجوی دانشگاه تهران تعریف کن) بر روی مدل آموزش داده نشده پاسخ زیر دریافت شد:

یک داستان کوتاه در مورد دانشجوی دانشگاه تهران تعریف کنیم. این دانشجو با استفاده از یک کامپیوتر و یک تبلت (یعنی یک نسل از لوازم الکترونیکی متصل به اینترنت) در یک محیط کاربری که شامل یک سرور، یک پایپ لینک (رام)، یک کارد استریچ (پایپ لینک برای صوتی و تصویری)، یک پورت ایرز (پورت برای ایرز (موسیک))، یک پورت اورج (پورت برای اورج (داغ))، یک پورت اورج-لایت (پورت برای نور خورشید) و یک مایکروفون (موسیک) برای صدا و یک پورت کیبی (پورت برای کیبورد) در یک محیط کاربری در یک phòng حال حاضر در یک Room_1 در University_of_Tehran، یک فیلتر برای آفسور ()

2-1. روشهای SoftPrompts

تکنیکهای soft prompt به افزودن توکن‌های قابل یادگیری به ورودی مدل بدون تغییر وزن‌های اصلی اشاره دارند.

در روش Prompt Tuning، تنها پارامترهای مربوط به prompt به روزرسانی می‌شوند و وزن‌های مدل ثابت نگه داشته می‌شوند.

Prefix Tuning به اضافه کردن پیشوندهای قابل یادگیری در چندین لایه مدل می‌پردازد که این امر باعث تغییر همزمان در چندین سطح می‌شود.

این رویکرد به مدل اجازه می‌دهد تا بدون تغییر گستره پارامترها به وظایف جدید سازگار شود.

P-Tuning نیز نوعی روش ترکیبی است که با بهره‌گیری از مفاهیم Prompt و Prefix Tuning، به بهینه‌سازی دقیق‌تری می‌پردازد.

این تکنیک‌ها موجب کاهش هزینه‌های محاسباتی و افزایش کارایی در تنظیم مدل‌های بزرگ می‌شوند.

با استفاده از این روش‌ها، می‌توان به سرعت مدل را برای وظایف خاص سفارشی کرد بدون اینکه نیاز به آموزش مجدد کل پارامترها باشد.

در نهایت، این تکنیک‌ها امکان تنظیم دقیق‌تر و انعطاف‌پذیرتر مدل‌ها را فراهم کرده و به خصوص در محیط‌های محدود به منابع بسیار مفید هستند.

در نهایت در ابتدا هدف به استفاده از روش prefix tuning داشتیم ولی به دلیل مشکل رم در ترین مدل برای این روش در نهایت به روش prompt tuning رو آوردیم.

جهت این مسئله به کمک قطعه کد زیر مدل و توکنایزر را در مدل 4 بیت لود کردیم :

```
def load_model(model_name="meta-llama/Llama-3.2-3B-Instruct"):  
    bnb_config = BitsAndBytesConfig(  
        load_in_4bit=True,  
        bnb_4bit_quant_type="nf4",  
        bnb_4bit_compute_dtype=torch.float16,  
        bnb_4bit_use_double_quant=True,
```

```

        )

tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    device_map="auto",
    torch_dtype=torch.float16
)

return model, tokenizer

```

سپس مدل لود شده را به کمک تابع زیر مدل قابل آموزش توسط prompt tuning به وسیله کتابخانه peft دریافت میکنیم :

```

def setup_prompt_tuning(model):
    peft_config = PromptTuningConfig(
        task_type=TaskType.CAUSAL_LM,
        num_virtual_tokens=20,
        prompt_tuning_init="TEXT",
        prompt_tuning_init_text="Answer the following question:",
        tokenizer_name_or_path="meta-llama/Llama-3.2-3B-Instruct",
    )
    return get_peft_model(model, peft_config)

```

و پس از آن به کمک تابع زیر مدل را بر روی مجموعه داده آموزش دادیم :

```

def train_model(model, tokenizer, train_dataset, eval_dataset=None):
    def tokenize_function(examples):
        return tokenizer(

```

```
        examples["text"],
        truncation=True,
        max_length=512,
        padding="max_length",
        add_special_tokens=True
    )

    tokenized_train = train_dataset.map(tokenize_function,
batched=True)

    if eval_dataset:

        tokenized_eval = eval_dataset.map(tokenize_function,
batched=True)

    training_args = TrainingArguments(
        output_dir='./prompt_tuning_results',
        per_device_train_batch_size=4,
        per_device_eval_batch_size=4,
        num_train_epochs=5,
        learning_rate=3e-5,
        logging_steps=20,
        evaluation_strategy="steps" if eval_dataset else "no",
        eval_steps=20,
        save_strategy="no",
        optim="paged_adamw_32bit",
        gradient_accumulation_steps=2,
        fp16=True,
        report_to="none"
    )
```

```

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_eval if eval_dataset else None,
    data_collator=lambda data: {
        "input_ids": torch.stack([torch.tensor(d["input_ids"])
for d in data]),
        "attention_mask":
torch.stack([torch.tensor(d["attention_mask"]) for d in data]),
        "labels": torch.stack([torch.tensor(d["input_ids"]) for
d in data])
    }
)

trainer.train()
return trainer

```

و در نهایت به کمک تابع زیر لاس های فرایند ترین را پلات میکردیم :

```

def plot_loss(trainer):
    history = trainer.state.log_history
    train_loss = [x['loss'] for x in history if 'loss' in x]
    eval_loss = [x['eval_loss'] for x in history if 'eval_loss' in
x]

    plt.figure(figsize=(10, 6))
    plt.plot(train_loss, label='Training Loss')
    if eval_loss:

```

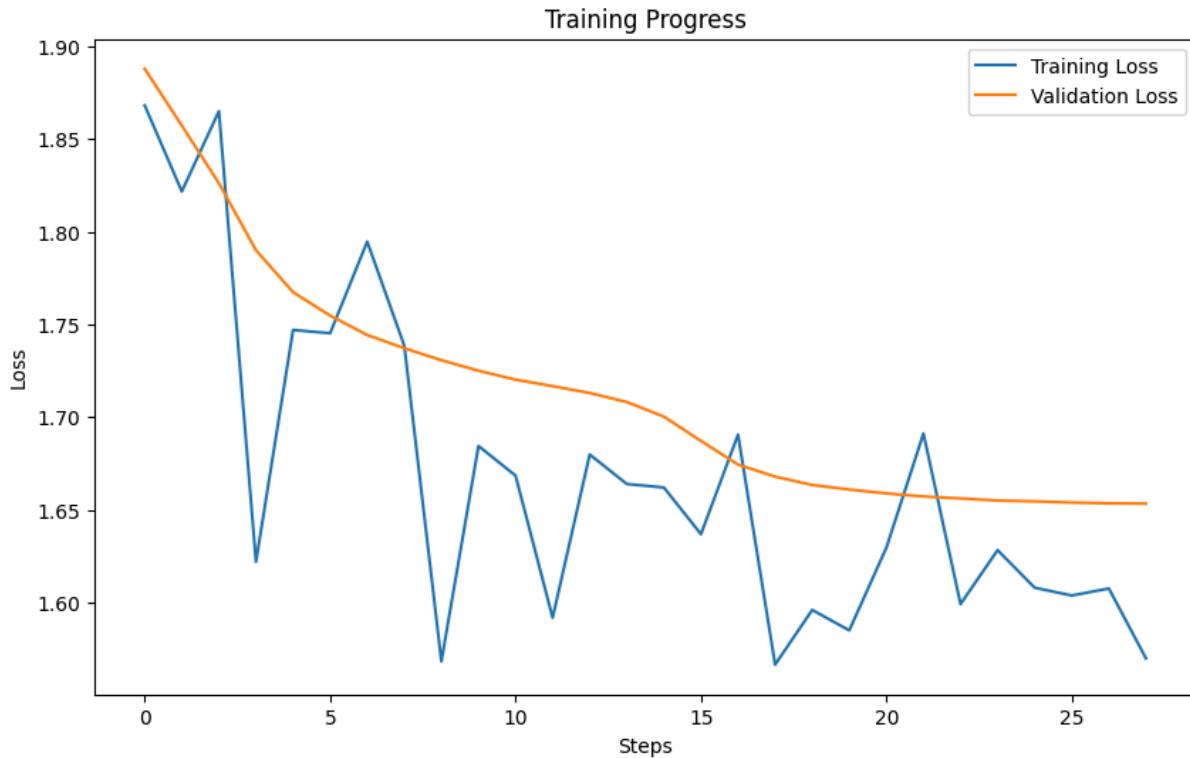
```
plt.plot(eval_loss, label='Validation Loss')
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.title('Training Progress')
plt.legend()
plt.show()
```

در زیر نتیجه ترین را مشاهده میکنید :

[560/560 1:41:20, Epoch 4/5]

Step	Training Loss	Validation Loss
20	1.868200	1.887984
40	1.821800	1.857313
60	1.865100	1.826233
80	1.621900	1.790056
100	1.747000	1.767395
120	1.745300	1.754773
140	1.794700	1.744300
160	1.738500	1.737175
180	1.568200	1.730713
200	1.684400	1.725024
220	1.668400	1.720229
240	1.591700	1.716709
260	1.679800	1.713015
280	1.663900	1.708140
300	1.662000	1.700234
320	1.636800	1.687146
340	1.690500	1.674387
360	1.566400	1.667853
380	1.595900	1.663365
400	1.584900	1.660907
420	1.629500	1.658822
440	1.691100	1.657132
460	1.599100	1.656113
480	1.628200	1.654963
500	1.607900	1.654533
520	1.603700	1.653875
540	1.607500	1.653464
560	1.569800	1.653312

و همچنین پلات لاس های ترین به صورت زیر است :



و نتیجه پرامپت تست قبلی پس از آموزش به صورت زیر میشود :

یک داسنان کوتاه در مورد دانشجوی دانشگاه تهران تعریف کنه

داسنان کوتاه (توسعه‌ی عقل و ذهن) به عنوان یک فرایند بافتی و بوم‌شناختی، در حال حاضر یک پدیده اجتماعی-مادی است که با توجه به سطوح مختلف آن در جامعه، از سطوح آگاهی‌دار تا سطوح آگاهانه و به طور کلی، با توجه به این که در یک جامعه‌ای با تاریخ Culture پرنگ، به عنوان یک میراث فرهنگی و عرفانی و نیز یک رویداد اجتماعی-مادی، به صورت متنوع و منحصر به فرد، به طور منظم و همگام با فرایندی متعارف در جوامع مختلف و بر اساس سطوح مختلف آگاهی، و با society

همانطور که مشاهده میشود اولین مسئله بهبود ارتباط پاسخ به سوال است و حد اقل جملاتی نسبی و تا حد اندکی مرتبط ساخته شد گرچه همچنان کلمات انگلیسی و عدم انسجام جملات موجود است. و کوتاهی پاسخ به دلیل کوتاهی max_length است.

3-1. روش‌های مبتنی بر LoRA

رویکرد LoRA (Low-Rank Adaptation) روشی برای تنظیم دقیق مدل‌های بزرگ به صورت کارآمد است.

در این روش، به جای به روزرسانی تمامی وزن‌های مدل، تنها پارامترهای افزوده شده با ساختار رده پایین آموزش داده می‌شوند.

مدل اصلی ثابت نگه داشته می‌شود و تغییرات لازم از طریق ماتریس‌های کم‌رده اعمال می‌گردد.

این امر موجب کاهش چشمگیر هزینه‌های محاسبات و حافظه مصرفی می‌شود.

به کمک LoRA، فرآیند تنظیم دقیق با سرعت بالاتر و منابع کمتر قابل انجام است.

این تکنیک امکان سفارشی‌سازی مدل‌های بزرگ در محیط‌های محدود به منابع را فراهم می‌کند.

کاربرد LoRA به عنوان روشی بهینه و اقتصادی در بهبود عملکرد مدل‌های بزرگ شناخته شده است.

در نهایت، LoRA به توسعه‌دهندگان اجازه می‌دهد تا بدون نیاز به تغییرات گسترده در مدل، به نتایج مطلوب دست یابند.

در مدل‌های زبانی مبتنی بر ترنسفورمر، تکنیک LoRA عمدهاً بر روی بخش‌های مرتبط با مکانیزم خودتوجهی (Self-Attention) اعمال می‌شود.

به‌طور مشخص، ماتریس‌های وزن مربوط به پرسش (Query)، کلید (Key) و مقدار (Value) از مهم‌ترین اجزایی هستند که LoRA روی آن‌ها به کار گرفته می‌شود.

همچنین می‌توان لایه‌های خروجی توجه را نیز هدف قرار داد تا تغییرات لازم با هزینه‌ی محاسباتی کمتر اعمال شود.

در برخی موارد، استفاده از LoRA بر روی لایه‌های شبکه چندلایه (Feed-Forward) نیز مدنظر قرار می‌گیرد تا تطبیق دقیق‌تر صورت گیرد.

این رویکرد باعث حفظ ساختار اصلی مدل شده و تنها تغییرات ضروری برای بهبود عملکرد اعمال می‌شود.

انتخاب لایه‌های مناسب برای اعمال LoRA به معماری دقیق مدل و هدف از تنظیم دقیق بستگی دارد.

در نهایت، تمرکز عمدۀ بر روی لایه‌های خودتوجهی است، زیرا بخش عمدۀ ای از محاسبات و تاثیر مدل را تشکیل می‌دهد.

در این بخش از مدل لود شده به کمک کتابخانه unsloth به دلیل سرعت بالاتر و رم کمتر مورد نیاز جهت آموزش مدل استفاده شده است و مدل قابل آموزش لورا را نیز به کمک همین کتابخانه به جای peft دریافت کردیم :

```
max_seq_length = 2048
dtype = None
load_in_4bit = True
quantized_model, tokenizer =
FastLanguageModel.from_pretrained(
    model_name="unsloth/Llama-3.2-3B-Instruct",
    max_seq_length=max_seq_length,
    dtype=dtype,
    load_in_4bit=load_in_4bit,
)
```

```
lora_layers_and_quantized_model = FastLanguageModel.get_peft_model(
    quantized_model,
    r=16,
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj",
    "gate_proj", "up_proj", "down_proj"],
    lora_alpha=16,
    lora_dropout=0,           bias="none",
    use_gradient_checkpointing="unsloth",      random_state=3407,
    use_rslora=False,     )
```

سپس پس از دریافت مدل قابل آموزش به روش لورا به کمک کد زیر و کتابخانه trl به اموزش این مدل پرداختیم :

```
trainer = SFTTrainer(  
    model = lora_layers_and_quantized_model,    tokenizer = tokenizer,  
    train_dataset = dataset,  
    dataset_text_field = "text",  
    max_seq_length = max_seq_length,  
    data_collator = DataCollatorForSeq2Seq(  
        tokenizer = tokenizer,  
        padding = True,                  pad_to_multiple_of = 8,      ),  
    dataset_num_proc = 2,  
    packing = False,  
    args = TrainingArguments(  
        report_to="none",  
        per_device_train_batch_size = 4,  
        gradient_accumulation_steps = 4,  
        warmup_steps = 20,  
        max_steps = 300,  
        learning_rate = 1.5e-4,  
        fp16 = not is_bfloat16_supported(),  
        bfloat16 = is_bfloat16_supported(),  
        logging_steps = 10,  
        optim = "adamw_8bit",  
        weight_decay = 0.02,  
        lr_scheduler_type = "linear",  
        seed = 3407,  
        output_dir = "outputs",
```

```

) ,
)

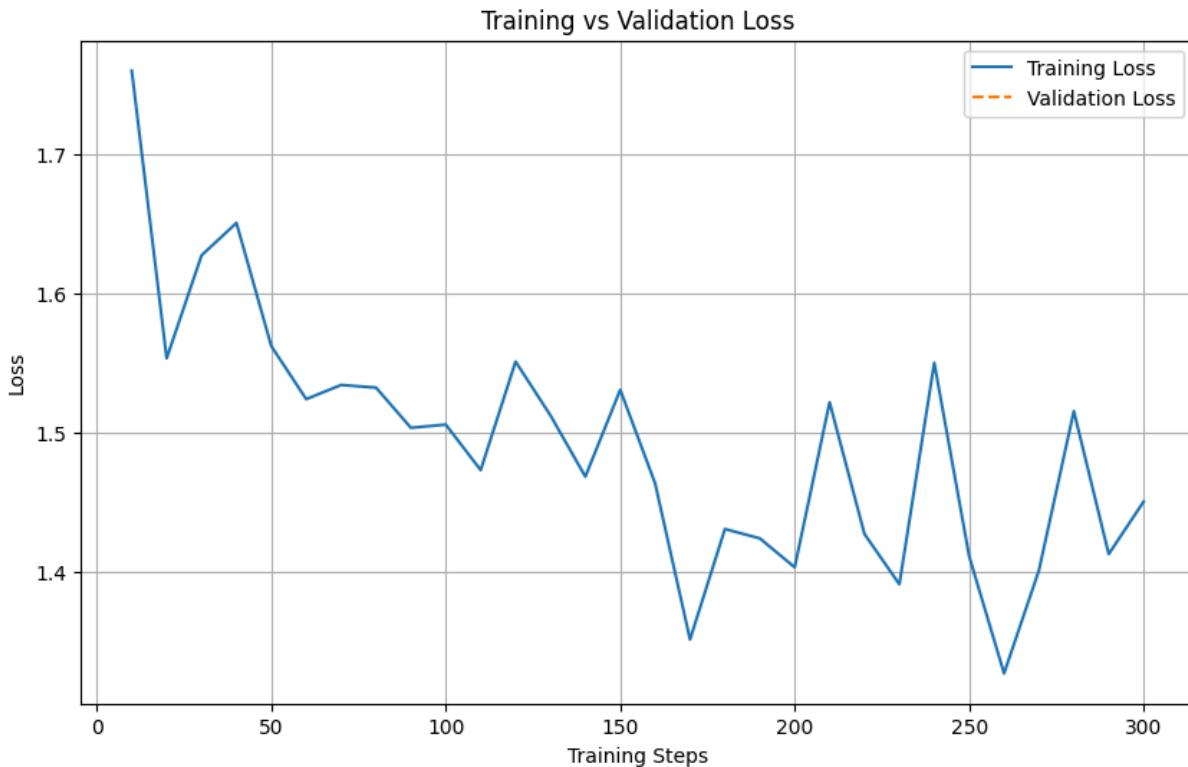
trainer = train_on_responses_only(
    trainer,
    instruction_part = "<|start_header_id|>user<|end_header_id|>\n\n",
    response_part =
"<|start_header_id|>assistant<|end_header_id|>\n\n",
)
trainer_stats = trainer.train()

```

که نتیجه ترین آن به صورت زیر است :

Step	Training Loss
10	1.760000
20	1.553200
30	1.627200
40	1.650500
50	1.561900
60	1.523800
70	1.534000
80	1.532100
90	1.503200
100	1.505500
110	1.472800
120	1.550800
130	1.512100
140	1.468100
150	1.530500
160	1.463200
170	1.351000
180	1.430400
190	1.423700
200	1.402900
210	1.521500
220	1.426800
230	1.390700
240	1.549900
250	1.410900
260	1.326600
270	1.400700
280	1.515200

و نمودار لاس آموزش به صورت زیر در آمد :



و پاسخ تولید شده جهت توسط این مدل آموزش داده بر روی همان پرامپت تست به شرح زیر درامد :

[یک داستان را درباره یک دانشجوی دانشگاه تهران بنویسید.\n\nبه نام آنسان می 200گویند، می 200توانید بگویید "صعب" است و این کسی یک دانشجوی دانشگاه تهران است. آن شخص، پس از آن که در کلاسی که در این دانشگاه می 200خواهد ب Study می 200آید، در یک روز با یک گروهی از همکارانش در یک جلسه در کاخ پارس، صحبت می 200کند و می 200پرسد:\n\n"من در یک گروه از دوستانم در university had کار می 200کنم و در یک جلسه در کاخ پارس داریم، پس چه کار می 200توانیم انجام دهیم تا آن 200ها را با هم closer get karim?"\n\nآن 200ها به آن 200گویند که چه چیز 200ها می 200توانند برای نزدیک 200تر شدن به هم کار کنند.\n\nدر درک این جمله، یک جمله برای انجام این کار ارائه می 200دهند که با یک کار مرتبط باشد، مانند:\n\n"مثلا، می 200توانیم از آن 200ها را در یک نشستی درباره کارهای خود در talk university کنیم و به 200طور کلی بیشتر درک کنیم که چگونه کار می 200کنند.\n\nاین پاسخ درک کرده و گزینه 200های مورد نظر را برای انجام کار برای نزدیک 200تر شدن به هم فراهم

می‌آورد. این عمل به آن شخص که در این داستان نقش دارد، احساس بهتری درباره ش در نهایت آن را با هم closer get karim می‌کند و در نهایت university می‌گردد.

به دلیل ماکسیمم طول توکن‌های بیشتر حین آموزش این مدل متن طولانی تری تولید شده و همچنین با دقت در خروجی این مدل خروجی ارتباط معنایی بالاتری با پرامپت ورودی داشته و مدل بهتر آموزش دیده است.

4-1. تغییر وزن برخی از لایه‌ها

در این بخش دیگر مدل را به صورت کوانتايزد لود نکردیم زیرا امکان فاین تیون کردن این مدل کوانتايزد وجود نداشت به همین دلیل به صورت زیر مدل را لود کردیم:

```
def load_model(model_name="meta-llama/Llama-3.2-3B-Instruct"):
```

```
    tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
    tokenizer.pad_token = tokenizer.eos_token
```

```
    model = AutoModelForCausalLM.from_pretrained(
```

```
        model_name,
```

```
        # quantization_config=bnb_config,
```

```
        device_map="auto",
```

```
        torch_dtype=torch.bfloat16
```

```
)
```

```
    return model, tokenizer
```

همانطور که قابل مشاهده است دیگر مدل کوانتايزد نیست و از دیتا تایپ bf16 استفاده شده است.

ساختمان مدل به شرح زیر است:

```
▶ model.eval()

[19]: LlamaForCausalLM(
      (model): LlamaModel(
        (embed_tokens): Embedding(128256, 3072)
        (layers): ModuleList(
          (0-27): 28 x LlamaDecoderLayer(
            (self_attn): LlamaAttention(
              (q_proj): Linear(in_features=3072, out_features=3072, bias=False)
              (k_proj): Linear(in_features=3072, out_features=1024, bias=False)
              (v_proj): Linear(in_features=3072, out_features=1024, bias=False)
              (o_proj): Linear(in_features=3072, out_features=3072, bias=False)
            )
            (mlp): LlamaMLP(
              (gate_proj): Linear(in_features=3072, out_features=8192, bias=False)
              (up_proj): Linear(in_features=3072, out_features=8192, bias=False)
              (down_proj): Linear(in_features=8192, out_features=3072, bias=False)
              (act_fn): SiLU()
            )
            (input_layernorm): LlamaRMSNorm((3072,), eps=1e-05)
            (post_attention_layernorm): LlamaRMSNorm((3072,), eps=1e-05)
          )
        )
        (norm): LlamaRMSNorm((3072,), eps=1e-05)
        (rotary_emb): LlamaRotaryEmbedding()
      )
      (lm_head): Linear(in_features=3072, out_features=128256, bias=False)
)
```

سپس مدل را به کمک کد زیر آموزش میدهیم (در این بخش به دلیل کمبود وقت و منابع بر روی 100 دیتابیونت مدل آموزش داده شد) :

```
def train_model(model, tokenizer, train_dataset,
eval_dataset=None):

    def tokenize_function(examples):
        return tokenizer(
            examples["text"],
            truncation=True,
            max_length=512,
            padding=True
        )

        tokenized_train = train_dataset.map(tokenize_function,
batched=True)

        data_collator = DataCollatorForLanguageModeling(
            tokenizer=tokenizer,
            mlm=False
        )
```

```
training_args = TrainingArguments(  
    output_dir='./full_finetune_results',  
    per_device_train_batch_size=1,  
    num_train_epochs=1,  
    learning_rate=1e-5,  
    weight_decay=0.01,  
    warmup_ratio=0.1,  
    gradient_accumulation_steps=1,  
    gradient_checkpointing=True,           fp16=False,  
evaluation_strategy="no",  
    # eval_steps=20,  
    save_strategy="steps",  
    save_steps=20,  
    logging_steps=20,  
    report_to="none",  
    optim="adafactor",  
    remove_unused_columns=True,  
    bf16=False  
)  
  
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=tokenized_train,  
    eval_dataset=None,  
    data_collator=data_collator,  
)
```

```
trainer.train()

model.save_pretrained("./fully_finetuned_llama")
tokenizer.save_pretrained("./fully_finetuned_llama")

return trainer
```

نتیجه آموزش مدل به شیوه زیر به شرح زیر است:

Step	Training Loss
20	2.411300
40	1.917300
60	1.901500

و به دلیل محدودیت منابع اموزش مدل تکمیل نشده و پلات لاس ها را نداریم.

و پاسخ به پرامپت تست به صورت زیر است :

یک داسنان کوتاه در مورد دانشجوی دانشگاه تهران تعریف کنید.

جهت مقایسه پاسخ‌ها ارتباط معنایی ابتدای جمله و سیر داستانی آن بسیار بهتر از دو روش قبل است اما به دلیل مشکلات موجود در فرایند ترین که در بالا ذکر شد مشاهده میشود که مدل در به خاطر سپاری مطالب دچار مشکل شده و به تکرار یک عبارت رسیده است که با آموزش کامل مدل این مسئله رفع میشود.

5-1. جمع‌بندی و تحلیل مقایسه‌ای

در روش Prompt Tuning، تنها توکن‌های قابل یادگیری به ورودی مدل اضافه می‌شوند و بقیه وزن‌ها بدون تغییر باقی می‌مانند، به همین دلیل این روش از نظر زمان و منابع بسیار بهینه است؛ اما ممکن است در برخی وظایف از نظر دقّت به اندازه سایر روش‌ها عمل نکند. از سوی دیگر، رویکرد LoRA با اضافه کردن ماتریس‌های کم‌رده به لایه‌های مهم مدل مانند مکانیزم خودتوجهی، تنها تغییرات لازم را اعمال می‌کند و به این صورت مدل اصلی بدون تغییر باقی می‌ماند؛ این امر باعث می‌شود که LoRA در عین حفظ دقّت قابل قبول، از منابع و زمان کمتری نسبت به تنظیم دقیق کامل استفاده کند. در مقابل، تنظیم دقیق وزن‌های اصلی مدل (Full Fine Tuning) شامل بهروزرسانی تمامی پارامترهای مدل است که معمولاً بهترین عملکرد را ارائه می‌دهد، اما این دقّت با هزینه‌ای بالا از نظر محاسباتی و زمانی همراه است. بنابراین، اگر منابع محدود باشد یا نیاز به سرعت بالا داشته باشد، LoRA یا Prompt Tuning گزینه‌های مناسبی هستند، در حالی که در مواقعي که هدف به دست آوردن حداکثر دقّت است و محدودیت منابع وجود ندارد، تنظیم دقیق کامل ترجیح داده می‌شود.

پرسش 2 - تولید کپشن برای تصاویر (Image Captioning)

1-2. مقدمه

در این مسئله

مجموعه داده استفاده شده در این مسئله شامل تصاویر به همراه چندین جمله کپشن در مورد تصویر مربوطه می باشد:

A race cars muffler catches on fire .

A race car sparks .

A sports car with flames coming out the exhaust .

The blue and white race car has flames coming out of its tail pipe .

The white racing car has fire bursting from its exhaust pipe .



A boy in an orange shirt and red hat is standing next to a girl with a white shirt and blue pants in the sand next to the ocean .
A woman and boy playing on a beach .
Two children are running towards the ocean on a beach .
Two children on the beach .



A group of girls and two are pulling on some type of yarn item in a girls bag .
The girls are braiding .
Two women pull on another coat .
Women pull on yarn-like strings of another woman 's plaid bag .



2-2. آماده سازی دیتاست

در مسئله image captioning، از special token هایی مانند <unk>, <pad>, <eos> و <sos> استفاده می شود که کاربرد هر کدام به شرح زیر است:

- start of sequence : به مدل می گوید که زمان شروع یک جمله است و معمولاً در ابتدای توالی قرار می گیرد.
- end of sequence : نشان دهنده پایان جمله است و به مدل کمک می کند تا زمانی که توالی به اتمام رسیده است، متوقف شود.

- padding : برای هماهنگ‌سازی طول توالی‌های مختلف به کار می‌رود و در صورتی که طول جمله کوتاه‌تر از حد مشخص باشد، از آن برای پر کردن فضای خالی استفاده می‌شود.
- unknown : برای کلمات ناشناخته یا خارج از دیکشنری استفاده می‌شود و به مدل کمک می‌کند تا با واژگان جدید یا ناآشنا برخورد کند.

این توکن‌ها به مدل‌های image captioning اجازه می‌دهند که توالی‌های متنی را به شکل موثرتری پردازش و تولید کنند.

تبديل کیشن‌ها به طول ثابت در مسائل پردازش زبان طبیعی شامل دلایل زیر است:

اولاً، مدل‌های یادگیری عمیق معمولاً به ورودی‌هایی با اندازه ثابت نیاز دارند تا بتوانند به‌طور مؤثر آموزش بینند و پیش‌بینی کنند. اگر طول کیشن‌ها متغیر باشد، اندازه ورودی به مدل نیز تغییر می‌کند و این موضوع می‌تواند باعث پیچیدگی بیشتر در طراحی مدل شود.

ثانیاً، توالی‌های با طول ثابت کمک می‌کنند تا پردازش بهینه‌تری انجام شود. در شبکه‌های عصبی مانند LSTM یا RNN که برای پردازش داده‌های توالی‌ای استفاده می‌شوند، داشتن توالی‌های هم‌طول کمک می‌کند تا از مشکلاتی مثل بارگذاری غیر موثر حافظه جلوگیری شود.

ما در این مسئله از طول ثابت 15 کلمه استفاده کردایم و اگر کمتر از این مقدار بود، از padding به صورت زیر استفاده شده است:

```
def pad_caption(caption, word_to_index, max_length=20):
    tokens = ["<sos>"] + caption.lower().split() + ["<eos>"]
    token_ids = []
    for word in tokens:
        token_id = word_to_index.get(word, word_to_index["<unk>"])
        token_ids.append(token_id)

    if len(token_ids) < max_length:
        token_ids += [word_to_index["<pad>"]] * (max_length - len(token_ids))
    else:
        token_ids = token_ids[:max_length]
    return torch.tensor(token_ids)

PAD_IDX = word_to_index_dictionary["<pad>"]
```

در این مسئله مجموعه داده را به سه دسته train, test, validation تقسیم کردہایم.

```
def split_dataset(captions_dict, train_ratio=0.8, val_ratio=0.1, test_ratio=0.1, random_seed=42):
    assert abs(train_ratio + val_ratio + test_ratio - 1.0) < 1e-6, "Ratios must sum to 1."

    flattened_captions = []
    for image_id, captions in captions_dict.items():
        for caption in captions:
            flattened_captions.append((image_id, caption))

    random.seed(random_seed)
    random.shuffle(flattened_captions)

    total_samples = len(flattened_captions)
    train_end = int(total_samples * train_ratio)
    val_end = train_end + int(total_samples * val_ratio)

    train_samples = flattened_captions[:train_end]
    val_samples = flattened_captions[train_end:val_end]
    test_samples = flattened_captions[val_end:]

    train_captions = {image_id: [] for image_id, _ in train_samples}
    for image_id, caption in train_samples:
        train_captions[image_id].append(caption)

    val_captions = {image_id: [] for image_id, _ in val_samples}
    for image_id, caption in val_samples:
        val_captions[image_id].append(caption)

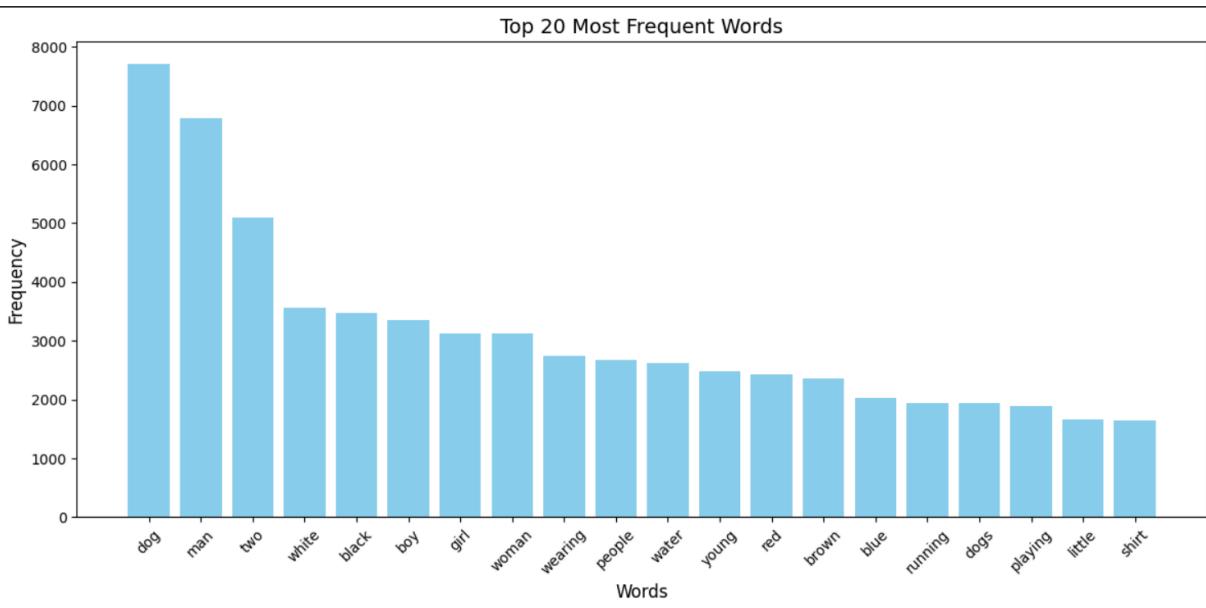
    test_captions = {image_id: [] for image_id, _ in test_samples}
    for image_id, caption in test_samples:
        test_captions[image_id].append(caption)

    return train_captions, val_captions, test_captions
```

توجه کنید که به ازای هر تصویر از این دیتاست، چندین جمله به عنوان کپشن ذخیره شده است که ما در مراحل آموزش به صورت تصادفی یکی از آنها را انتخاب کردہایم.

همینطور اندازه تصاویر هم یکسان نمی باشد که پیشتر به اندازه 224x224 تغییر سایز دادهایم.

برخی از نمودارهای خواسته شده برای بررسی بیشتر مجموعه داده به شرح زیر هستند:



قبل این نمایش کلمات استاپ ورد را حذف کرده ایم تا 20 کلمه پر تکرار، کلمات با ارزش از نظر معنایی باشند و دید خوبی نسبت به ماهیت مجموعه داده بدھند.

در تصویر فوق، تعداد تکرار 20 تا از پر حرف ترین کلمات این مجموعه داده، پس از انجام فرایند پریپراسس زیر هستیم:

```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"[^\w\s]", "", text) # Remove punctuation
    text = re.sub(r"\d+", "", text) # Remove numbers
    words = text.split() # Tokenize text
    words = [word for word in words if word not in stop_words] # Remove stop words
    return " ".join(words).strip()
```

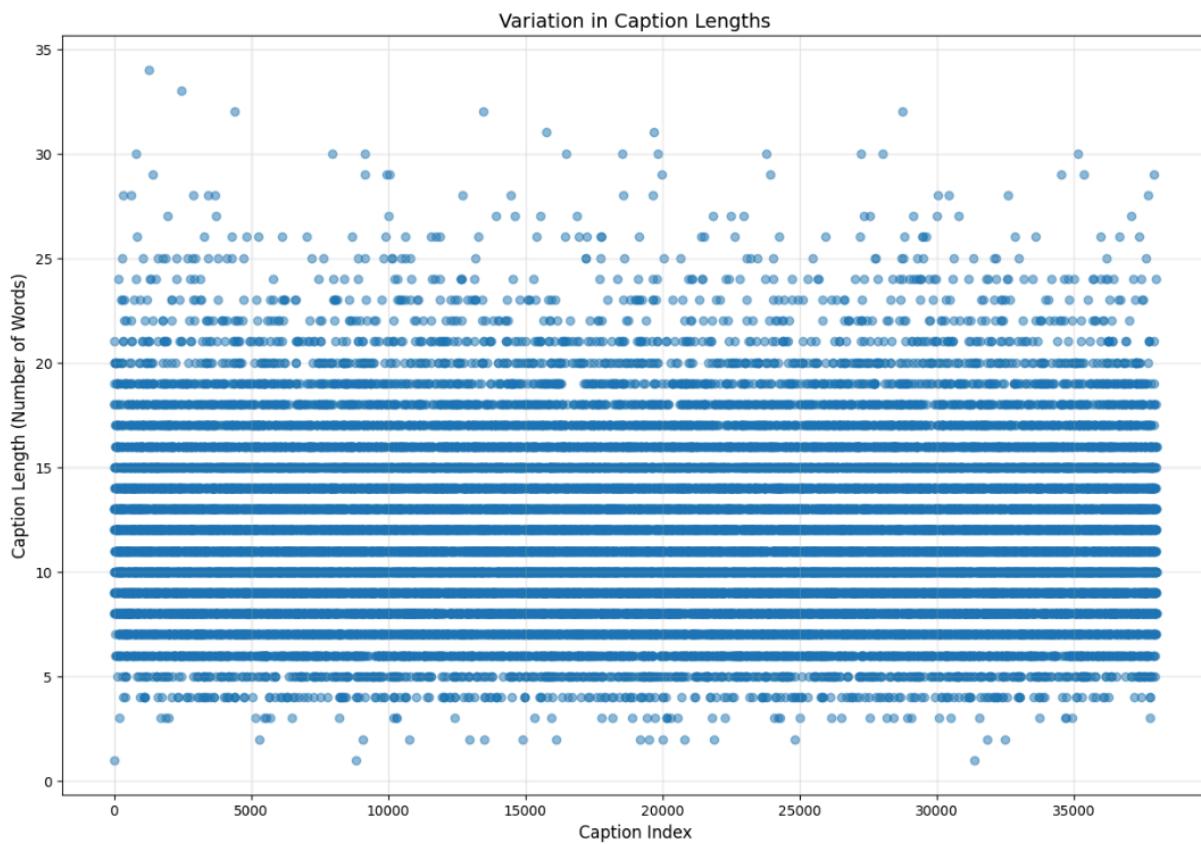
همانطور که در صورت پروژه خواسته شده بود، اعداد و ارقام اضافی، پانکچوئیشن‌ها، و کلمات استاپ ورد را از کپشن‌ها حذف کرده‌ایم.

حذف اعداد و ارقام اضافی، پانکچوئیشن‌ها و کلمات استاپ ورد (stop words) از کپشن‌ها معمولاً برای بهبود عملکرد مدل‌های پردازش زبان طبیعی و کاهش پیچیدگی داده‌ها انجام می‌شود. دلایل اصلی این کار عبارتند از:

حذف کلمات استاپ ورد (مانند "و"، "در"، "از" و ...) که بار معنایی زیادی ندارند و اغلب در تمامی جملات تکرار می‌شوند، به مدل کمک می‌کند تا تمرکز بیشتری روی کلمات مهمتر و معنی‌دار داشته باشد. اعداد و ارقام اضافی معمولاً برای مدل درک مفهومی ندارند، مگر اینکه در متن خاصی اهمیت

داشته باشند. حذف این موارد از مدل جلوگیری می‌کند که آن‌ها بر فرآیند یادگیری تأثیر منفی بگذارند.

پانکچوئیشن‌ها نیز اغلب اطلاعات معنایی خاصی به مدل اضافه نمی‌کنند و می‌توانند باعث افزایش ابعاد و پیچیدگی فضای جستجو شوند. حذف آن‌ها باعث کاهش حجم داده‌ها و بهبود کارایی مدل می‌شود. وقتی که این عناصر اضافی حذف شوند، مدل قادر خواهد بود که بیشتر بر روی کلمات کلیدی و محتوای اصلی متن تمرکز کند و از آن برای یادگیری بهتر استفاده کند.



در تصویر فوق، میزان تکرار انواع طول کپشن در کل مجموعه داده را گزارش دادیم. با توجه به اینکه تکرار تعداد در بازه‌های ۰ تا ۵ و ۲۰ تا ۳۵ بسیار کم بود، ما در طول مسئله max_length را برابر مقدار 20 قرار دادیم.

بر روی تصاویر نمونه‌هایی از transform را اعمال کردہ‌ایم تا علاوه بر یکسان شدن سایز تصاویر، همگی در tensor جای گیرند و با مقادیری که متناسب مجموعه داده هستند، نرمال شده باشند. توجه شود که tranform استفاده شده برای مجموعه داده ترین و تست متفاوت بوده و علت

آن، روابست تر شدن مدل آموزش دیده در فرآیند آموزش و جامعیت پاسخگویی آن میباشد، با این حال محدوده کراپ را به مقدار اندکی قرار دادیم تا محتوای تصویر از دست نرود.

```
● ✓ transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

✓ val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

✓ denormalize = transforms.Compose([
    transforms.Normalize(mean=[-0.485/0.229, -0.456/0.224, -0.406/0.225],
    std=[1/0.229, 1/0.224, 1/0.225])
])
```

لازم به ذکر است که مقدار patience به اندازه 15 ایپاک تنظیم شده است.

3-2. پیاده‌سازی CNN-RNN

1. دلیل استفاده از embedding در بخش Decoder در مسئله :image captioning در بخش Decoder مدل‌های توصیف تصویر، از embedding برای تبدیل کلمات ورودی (که معمولاً به صورت اندیس‌های عددی نمایش داده می‌شوند) به بردارهایی با ابعاد کمتر ولی معنادار استفاده می‌شود. این کار باعث می‌شود که مدل بتواند روابط معنایی بین کلمات را بهتر یاد بگیرد و وابستگی‌های معنایی بین واژه‌ها را در فضای برداری حفظ کند. همچنین، استفاده از embedding باعث کاهش ابعاد ورودی و جلوگیری از مشکل sparsity (پراکندگی داده‌ها) می‌شود.

2. مزیت استفاده از word embedding به جای بردارهای one-hot

- بردارهای one-hot دارای ابعاد بسیار بزرگ هستند (برابر با تعداد کل کلمات موجود در واژگان مدل) و بیشتر عناصر آن‌ها صفر است که باعث هدررفت حافظه و افزایش پیچیدگی محاسباتی می‌شود.

در مقابل، بردارهای word embedding دارای ابعاد کوچکتر و مترادف‌تر هستند و می‌توانند اطلاعات معنایی و نحوی را حفظ کنند. این بردارها امکان نمایش شباهت بین کلمات را فراهم می‌کنند؛ به عنوان مثال، کلماتی مانند "گربه" و "سگ" در فضای one-hot نزدیک به هم قرار می‌گیرند، ولی در نمایش embedding هیچ ارتباطی بین آن‌ها وجود ندارد.

3. تبدیل رمزگذار (Encoder) و رمزگشای (Decoder) به یک مدل End-to-End قابل آموزش: برای ایجاد یک مدل End-to-End که قابلیت آموزش داشته باشد، باید مراحل زیر را انجام داد:

اتصال Encoder به Decoder: خروجی رمزگذار که یک بردار ویژگی (Feature Vector) از تصویر است، باید به عنوان ورودی اولیه به رمزگشای داده شود. معمولاً این کار با استفاده از یک شبکه عصبی بازگشتی (Transformer, RNN, LSTM) انجام می‌شود.

استفاده از تکنیک Attention (در صورت نیاز): برای بهبود عملکرد، می‌توان از مکانیزم توجه (Attention Mechanism) استفاده کرد تا رمزگشای بتواند بخش‌های مهم تصویر را در هر مرحله از تولید توضیح برسی کند.

تعریف تابع هزینه و بهینه‌سازی: مدل باید یک تابع هزینه (مانند Cross-Entropy Loss) برای مقایسه جملات تولید شده با جملات واقعی داشته باشد و با استفاده از الگوریتم‌هایی مانند Adam یا SGD بهینه‌سازی شود.

آموزش End-to-End: رمزگذار و رمزگشای باید به‌طور همزمان و یکپارچه آموزش داده شوند تا مدل بتواند هم ویژگی‌های تصویر و هم روابط کلمات را یاد بگیرد.

به این ترتیب، مدل می‌تواند مستقیماً از ورودی تصویر به خروجی توصیف متن دست پیدا کند و در یک فرآیند End-to-End آموزش ببیند.

مدل‌های استفاده شده در مسئله به شرح زیر هستند:

مدل EncoderCNN از یک شبکه کانولوشنی برای استخراج ویژگی‌های تصویر استفاده می‌کند. این مدل از معماری EfficientNet-B0 به عنوان پیش‌پردازش‌گر تصویر استفاده کرده که از ویژگی‌های پیش‌آموزش دیده‌شده بهره می‌برد. سپس، لایه‌ای به نام fc برای فشرده‌سازی خروجی این شبکه به اندازه مورد نیاز (embed_size) اضافه شده است. این مدل به طور کلی ویژگی‌های تصویر را به فضای برداری منتقل می‌کند که در مرحله بعد توسط Decoder برای تولید متن استفاده می‌شود.

- Click to add a breakpoint .Module):


```

def __init__(self, embed_size):
    super(EncoderCNN, self).__init__()
    self.model = models.efficientnet_b0(pretrained=True)
    self.model.classifier = nn.Identity()
    self.fc = nn.Linear(1280, embed_size)
    self.relu = nn.ReLU()

```

```

def forward(self, images):
    features = self.model(images)
    features = self.fc(features)
    features = self.relu(features)
    return features

```

مدل DecoderRNN وظیفه تولید تصویر را بر عهده دارد. این مدل از یک LSTM (Long Short-Term Memory) به عنوان شبکه بازگشتی برای پردازش دنباله‌های کلمات استفاده می‌کند. ورودی مدل شامل ویژگی‌های استخراج شده از تصویر (توسط Encoder) و همچنین کلمات ورودی (کپشن‌های جزئی) است که به صورت embedding شده وارد می‌شوند. پس از پردازش، مدل با استفاده از لایه Linear و سپس LogSoftmax تولید کلمات بعدی در دنباله را پیش‌بینی می‌کند.

```

class DecoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1):
        super(DecoderRNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)
        self.softmax = nn.LogSoftmax(dim=2)

    def forward(self, features, captions):
        embeddings = self.embedding(captions)
        inputs = torch.cat((features.unsqueeze(1), embeddings), dim=1)
        outputs, _ = self.lstm(inputs)
        outputs = self.fc(outputs)
        return self.softmax(outputs)

```

مدل ImageCaptioningModel یک مدل یکپارچه برای توصیف تصاویر است که از EncoderCNN و DecoderRNN برای پردازش تصویر و تولید کپشن استفاده می‌کند. در این مدل، تصویر ابتدا توسط بخش Encoder پردازش شده و ویژگی‌های آن استخراج می‌شود. سپس این ویژگی‌ها به عنوان ورودی به Decoder داده می‌شود که در نهایت کپشن مربوط به تصویر را تولید می‌کند. این مدل به طور کلی قابلیت یادگیری و تولید توصیف‌هایی دقیق از تصاویر را دارد و به دلیل استفاده از LSTM برای پردازش دنباله‌ها، قادر به تولید جملات معنادار و روان است.

```
class ImageCaptioningModel(nn.Module):
    def __init__(self, vocab_size, embed_size=512, hidden_size=512, num_layers=1, dropout=0.5):
        super(ImageCaptioningModel, self).__init__()
        self.encoder = EncoderCNN(embed_size).to(device)
        self.decoder = DecoderRNN(embed_size, hidden_size, vocab_size, num_layers).to(device) # , dropout

    def forward(self, images, captions):
        features = self.encoder(images)
        outputs = self.decoder(features, captions)
        return outputs
```

هایپر پارامترهای استفاده شده در این مسئله به شرح زیر است:

learning_rate	weight_decay	embed_size	hidden_size	num_layers	dropout
0.001	1e-4	256	512	1	0.5

نتایج مدل روی تصویر validation در چند ایپاک متوالی به شرح زیر است:

Epoch 4, Train Loss: 5.6775

Epoch 4, Validation Loss: 5.7085

Generated Caption: wearing shirt blue standing



Epoch 9, Train Loss: 5.4134

Epoch 9, Validation Loss: 5.5368

No improvement in validation loss for 1 epoch(s).

Generated Caption: boy shirt skateboard



Epoch 24, Train Loss: 5.1074

Epoch 24, Validation Loss: 5.3683

No improvement in validation loss for 1 epoch(s).

Generated Caption: wearing shirt shorts skateboard



Epoch 27, Train Loss: 5.0712

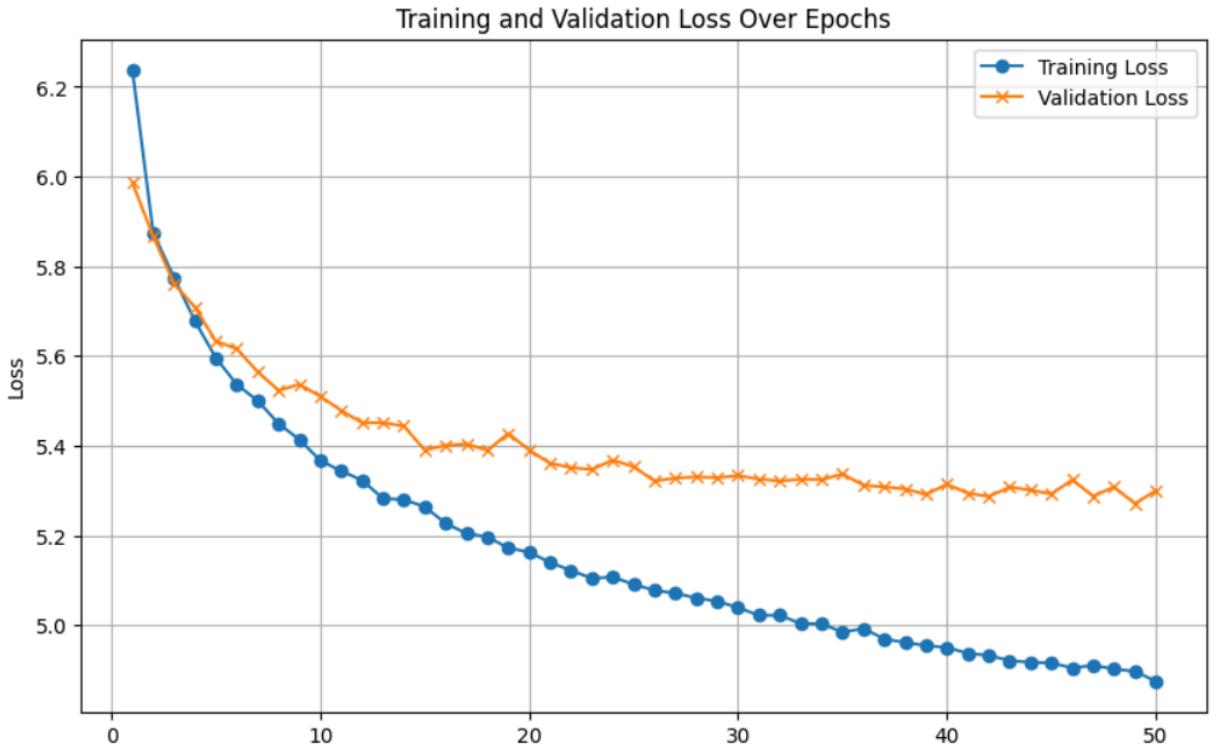
Epoch 27, Validation Loss: 5.3280

No improvement in validation loss for 1 epoch(s).

Generated Caption: girl dress



همینطور نمودار خطا در طول زمان به شرح زیر است:



همانطور که مشاهده می شود این نمودار روند نزولی داشته و در نهایت به علت شروع فرایند overfit دچار early stop شده است.

مقادیر متريک هاي محاسبه شده به شرح زير است:

```

for metric, score in bleu_scores.items():
    print(f'{metric} Score: {score:.4f}')
print(f'ROUGE-L Score: {rouge_l:.4f}')

```

```

BLEU-1 Score: 0.1609
BLEU-2 Score: 0.1102
BLEU-3 Score: 0.0843
BLEU-4 Score: 0.0670
ROUGE-L Score: 0.3351

```

نتایج مدل نشان دهنده عملکرد متوسط آن در تولید کپشن های توصیفی است. امتیاز BLEU-1 (با مقدار 0.1609) نشان می دهد که مدل توانسته است تا حدی کلمات مشابه به کپشن های واقعی تولید کند، ولی هنوز تفاوت های زیادی در انتخاب کلمات دارد. بقیه نتایج به ترتیب

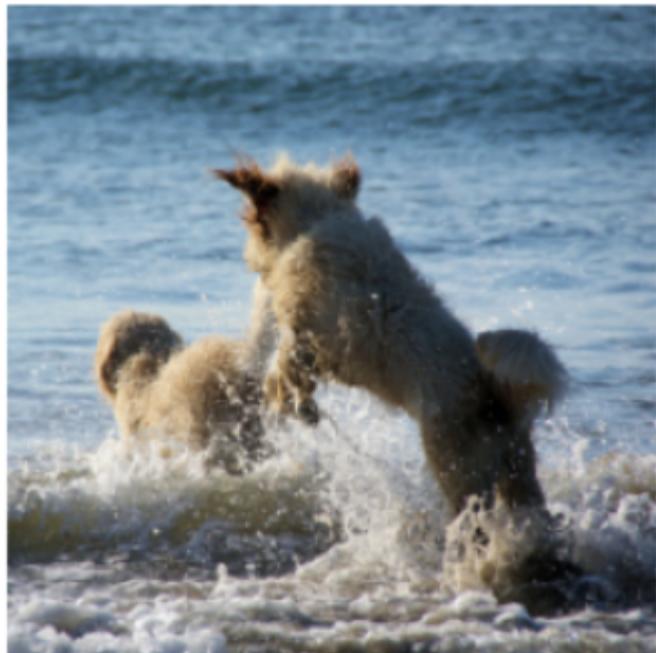
کاهش تدریجی در کیفیت تولید دنباله‌های کلمات نشان می‌دهند که به وضوح نشان‌دهنده مشکلات مدل در تولید جملات پیچیده‌تر و طولانی‌تر است. به‌طور کلی، امتیازهای BLEU به این معناست که مدل در تولید جملات با دقت بالا نیاز به بهبود دارد. از طرف دیگر، امتیاز ROUGE-L 0.3351 که تمرکز بیشتری بر روی شباهت‌های ساختاری و توالی کلمات دارد، نشان می‌دهد که مدل قادر است تا حدودی روابط معنایی و ساختاری در جملات را حفظ کند. با این حال، بهبود این نتایج نیاز به تنظیمات بیشتر و بهینه‌سازی مدل دارد.

در بخش زیر چندین نمونه از کپشن تصاویر تست این مدل مشاهده می‌شود:

Generated Caption: girl shirt hair



Generated Caption: d o g s w a t e r



Generated Caption: p l a y e r j e r s e y p l a y e r g a m e



4-2. پیاده‌سازی Attention based CNN-RNN

مدل Attention-based CNN-RNN در مسئله Image Captioning یک پیشرفت مهم در مقایسه با مدل‌های سنتی CNN-RNN است. این مدل از مکانیزم Attention برای بهبود دقت تولید کپشن‌ها استفاده می‌کند. در این مدل، شبکه CNN وظیفه استخراج ویژگی‌های تصویر را بر عهده دارد، اما برخلاف مدل‌های ساده‌تر، از یک مکانیزم توجه استفاده می‌شود که به مدل این امکان را می‌دهد تا بر روی بخش‌های مختلف تصویر در هر مرحله از تولید جمله تمرکز کند. این به مدل کمک می‌کند که ویژگی‌های مهم تصویر را در هنگام تولید هر کلمه در نظر بگیرد و بنابراین، کپشن‌هایی با کیفیت بالاتر و دقیق‌تر تولید کند.

در مرحله بعد، Decoder RNN که معمولاً یک LSTM است، دنباله‌های کلمات را تولید می‌کند، اما به جای استفاده از ویژگی‌های تصویر به صورت ثابت در طول تولید جمله، از مکانیزم توجه برای تعیین بخش‌های مختلف تصویر که باید در هر مرحله از تولید کلمات به آن‌ها توجه شود، بهره می‌برد. این مکانیزم توجه باعث می‌شود که مدل بتواند به صورت داینامیک بخش‌های مهم تصویر را در هر کلمه از کپشن شناسایی کند و در نتیجه تولید جملات معنی‌دارتر و متناسب‌تر با محتوای تصویر داشته باشد.

استفاده از Attention در مدل‌های CNN-RNN بهبود چشمگیری در دقت تولید کپشن‌ها به همراه دارد، زیرا این مدل قادر به استخراج ویژگی‌های پیچیده‌تر از تصویر و تولید جملات دقیق‌تر است. این رویکرد در مقایسه با مدل‌های سنتی بدون توجه، به طور واضحی نتایج بهتری را ارائه می‌دهد و برای کاربردهای پیچیده‌تر و واقعی‌تر در توصیف تصاویر بسیار مناسب است.

چندین تصویر validation در طول آموزش مدل:

Epoch 6, Train Loss: 4.5254

Epoch 6, Validation Loss: 4.7997

Generated Caption: boy skateboard trick



Epoch 12, Train Loss: 3.8076

Epoch 12, Validation Loss: 4.5747

Generated Caption: boy jumping air



Epoch 16, Train Loss: 3.5622

Epoch 16, Validation Loss: 4.5528

Generated Caption: man wearing black shorts jumping air



Epoch 21, Train Loss: 3.2907

Epoch 21, Validation Loss: 4.4829

Generated Caption: woman wearing pink shirt blue jeans walks away



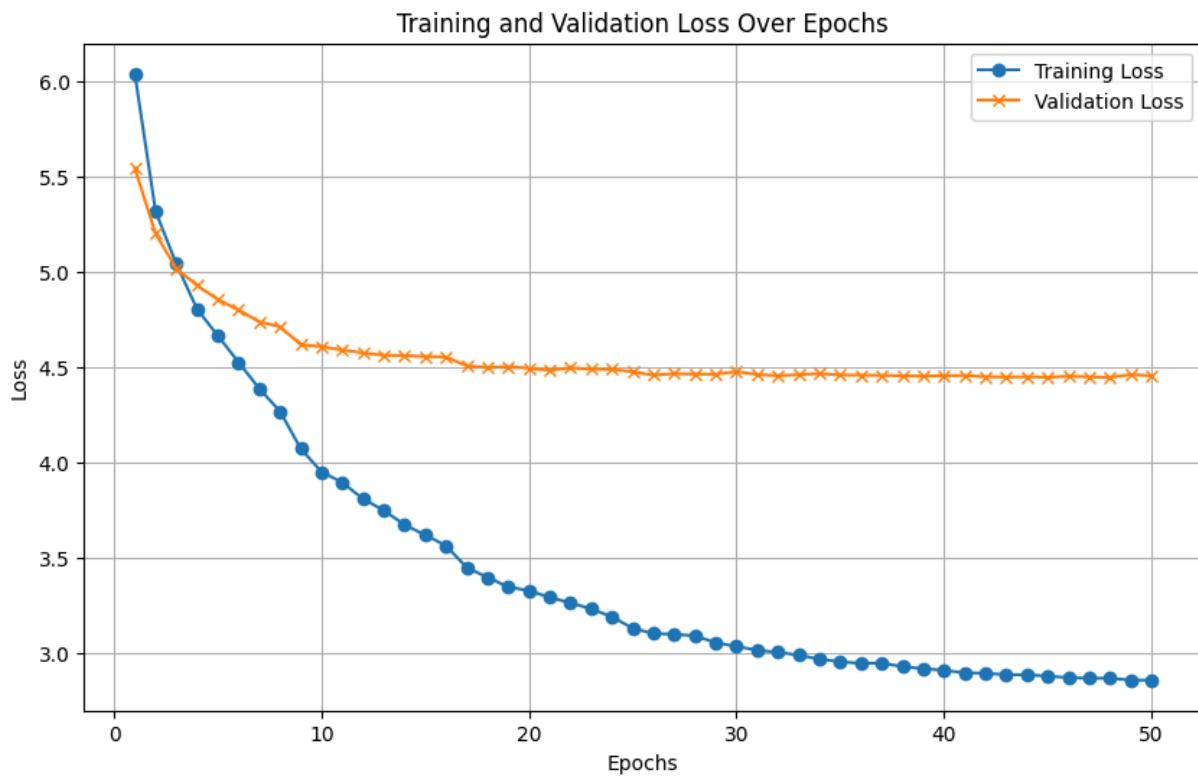
```
Epoch 33, Train Loss: 2.9872
```

```
Epoch 33, Validation Loss: 4.4602  
No improvement in validation loss for 1 epoch(s).
```

Generated Caption: woman wearing pink shirt playing tennis



نمودار خطای مدل دچار اورفیت بیشتری نسبت به مدل قبل شده که علت آن پیچیدگی بیشتر مدل مبتنی بر مکانیزم توجه است:



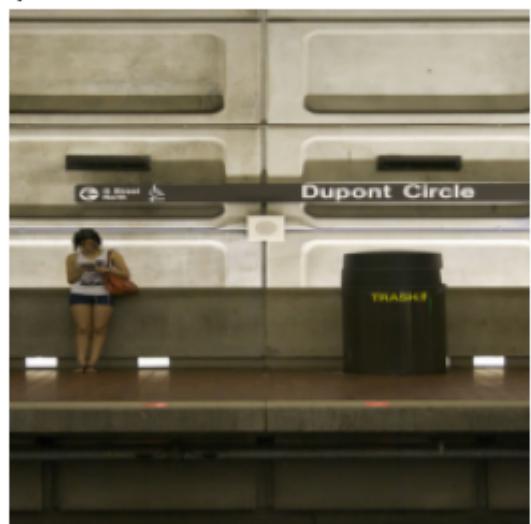
Generated Caption: little girl wearing pink sweater holds stuffed animal



Generated Caption: man climbing rock face



Generated Caption: woman sits bench reading newspaper



Generated Caption: two white dogs splashing around white waves



Generated Caption: football player red white uniform red sooners jersey



Word: <sos>



Word: little



Word: girl



Word: wearing



Word: pink



Word: sweater



Word: holds



Word: stuffed



Word: animal



نقشه حرارتی (Heatmap) در مدل‌های مبتنی بر مکانیزم توجه (Attention) ابزاری است که نمایش بصری از توجه مدل به بخش‌های مختلف ورودی (مثلًا تصویر یا جمله) ارائه می‌دهد. در این نقشه، رنگ‌ها یا شدت‌ها نشان‌دهنده میزان توجه مدل به هر قسمت از ورودی در هنگام تولید پیش‌بینی‌ها هستند. برای مثال، در مدل‌های توصیف تصاویر، نقشه حرارتی می‌تواند بخش‌هایی از تصویر را که مدل بیشتر به آن‌ها توجه کرده است، با رنگ‌های گرم (مانند قرمز) مشخص کند. این ابزار به تحلیل‌گران کمک می‌کند تا بفهمند مدل چگونه ورودی‌ها را پردازش کرده و چه ویژگی‌هایی را برای تولید خروجی در نظر گرفته است.

5-2. امتیازی

معیارهای ارزیابی در توصیف تصاویر:

- امتیاز BLEU Bilingual Evaluation Understudy Score: این معیار با مقایسه n-گرام‌های تولیدشده مدل با n-گرام‌های مرجع، شباهت بین آنها را اندازه‌گیری می‌کند. امتیاز BLEU معمولاً برای ارزیابی کیفیت ترجمه ماشینی استفاده می‌شود، اما در توصیف تصاویر نیز کاربرد دارد.
- امتیاز METEOR Metric for Evaluation of Translation with Explicit ORdering: این معیار با در نظر گرفتن هم‌ریختی‌ها، متراffها و ترتیب کلمات، شباهت بین توضیحات تولیدشده و مرجع را ارزیابی می‌کند.
- امتیاز ROUGE Recall-Oriented Understudy for Gisting Evaluation: این معیار با مقایسه n-گرام‌های تولید شده با n-گرام‌های مرجع، توانایی مدل در بازیابی اطلاعات را می‌سنجد.
- امتیاز CIDEr Consensus-based Image Description Evaluation: این معیار با در نظر گرفتن اهمیت کلمات در مجموعه‌ای از توضیحات مرجع، کیفیت توضیحات تولید شده را ارزیابی می‌کند.
- امتیاز SPICE Semantic Propositional Image Caption Evaluation: این معیار با تجزیه و تحلیل ساختار معنایی جملات، شباهت بین توضیحات تولید شده و مرجع را می‌سنجد.
- امتیاز TER Translation Edit Rate: این معیار با محاسبه تعداد تغییرات لازم برای تبدیل توضیحات تولیدشده به مرجع، کیفیت آنها را ارزیابی می‌کند.
- امتیاز WMD Word Mover's Distance: این معیار با اندازه‌گیری فاصله معنایی بین توضیحات تولیدشده و مرجع، کیفیت آنها را می‌سنجد.

استفاده از ترکیبی از این معیارها می‌تواند ارزیابی جامع‌تری از عملکرد مدل‌های توصیف تصاویر ارائه دهد.