به نام خدا

دانشگاه تهران - دانشگاه صنعتی خواجه نصیرالدین طوسی تهران

دانشکده مهندسی برق و کامپیوتر

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی برق - گروه مهندسی کنترل

توصیف عکس

نام و نام خانوادگی	محمدجواد احمدی
شمارهٔ دانشجویی	۴۰۱۰۰۰۸۶

فهرست مطالب

فهرست تصاویر

پرسش ۱. توصیف عکس

۱ پاسخ پرسش اول

توضیح پوشهٔ کدهای توصیف عکس

کدهای مربوط به این قسمت، علاوه بر پوشهٔ محلی کدها در این لینک گوگل‌کولب آورده شده است. مدل‌های دخیره شده هم از طریق این لینک در دسترس هستند.

۱.۱ پاسخ قسمت ۱

در گام اول و برای راحتی، مجموعه‌داده را روی گوگل‌درایو بارگذاری می‌کنیم. هم‌چنین، فراخوانی این فایل بدون نیاز به Mountکردن و استفاده از gdown در محیط گوگل‌کولب ممکن است با خطا مواجه شود که با استفاده از دستورات زیر مشکل حل می‌شود (منبع) . در ادامه هم دستوراتی برای خروج داده‌ها از حالت فشرده و رفتن به یک پوشهٔ مخصوص مجموعه‌داده نوشته‌ایم و در انتها فایل فشرده را حذف کرده‌ایم.

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1DArAhv1ieTdmvmu0Up8_2u5j4iVgmqn7

!mkdir Dataset
!unzip /content/archive.zip -d Dataset/
```

در ادامه دستوراتی را برای بارگذاری داده‌ها از یک فایل متنی می‌نویسیم. ابتدا مسیر فایل متنی برا تعیین کرده و سپس با استفاده از تابع read_csv، داده‌های موجود در فایل متنی به صورت یک دیتافریم خوانده می‌شوند. در ادامه، تعداد تصاویر دارای شرح، تعداد تصاویر یکتا و تعداد شرح‌های یکتا به صورت متنی چاپ می‌شوند. در انتها نیز، با استفاده از تابع head، پنج سطر اول داده‌های در دیتافریم نمایش داده می‌شوند. دستورات و نتیجهٔ خروجی به شرح زیر هستند:

```
import pandas as pd

# Set the path to the captions file
captions_path = 'captions.txt'

# Load the captions file into a pandas DataFrame
captions_df = pd.read_csv(captions_path)

# Count the number of images with captions
num_images = len(captions_df)

# Print the total number of images with captions
```

```
print(f"Total number of images with captions: {num_images}")

# Print the number of unique images
unique_images = len(captions_df['image'].unique())
print(f"Number of unique images with captions: {unique_images}")

# Print the number of unique captions
unique_captions = len(captions_df['caption'].unique())
print(f"Number of unique captions: {unique_captions}")

# Display the first 5 rows of the DataFrame
print("\nSample captions:")
captions_df.head()
----------------------------------------------------------------
Total number of images with captions: 40455
Number of unique images with captions: 8091
Number of unique captions: 40201

Sample captions:
image caption
0 1000268201_693b08cb0e.jpg A child in a pink dress is climbing up a set o...
1 1000268201_693b08cb0e.jpg A girl going into a wooden building .
2 1000268201_693b08cb0e.jpg A little girl climbing into a wooden playhouse .
3 1000268201_693b08cb0e.jpg A little girl climbing the stairs to her playh...
4 1000268201_693b08cb0e.jpg A little girl in a pink dress going into a woo...
```

در ادامه، دستوراتی شامل یک کلاس به نام Vocabulary می‌نویسیم که وظیفهٔ ساخت و استفاده از یک مجموعه‌واژگان را بر عهده دارد. به طور کلی، این کلاس دارای متدهایی برای ساخت واژگان، توکن‌سازی متن، عددی‌سازی متن و تشکیل دیکشنری واژگان است. در بخش ساخت واژگان، کلاس Vocabulary مقادیر اولیه را برای توکن‌های ویژه (Special Tokens) تعریف می‌کند. به عنوان مثال <PAD> برای پد کردن دنباله‌ها، <SOS> برای نشان دادن شروع جمله، <EOS> برای نشان دادن پایان جمله و <UNK> برای توکن‌های ناشناخته. سپس با استفاده از تابع tokenizer که با استفاده از یک توکن‌ساز با عبارت منظم رشته را به توکن‌ها تقسیم می‌کند و آن‌ها را به حروف کوچک تبدیل می‌کند، و واژگان را براساس آستانه فراوانی می‌سازد. ورودی frequency_threshold مقدار آستانهٔ فراوانی است که توکن‌ها باید برای اضافه‌شدن به واژگان داشته باشند. درواقع، آستانهٔ فراوانی مشخص می‌کند که توکن‌هایی که تعداد تکرار آن‌ها بیشتر از این آستانه است، در واژگان قرار بگیرند. در هنگام ساخت واژگان، همچنین دیکشنری itos و stoi بروزرسانی می‌شوند تا اندیس‌ها و توکن‌ها با هم متناظر شوند. متد numericalize با توجه به متن ورودی توکن‌سازی را انجام داده و توکن‌ها را به اندیس‌های متناظر در دیکشنری واژگان تبدیل می‌کند. اگر توکن در واژگان موجود نباشد، توکن <UNK> (نامشخص) جایگزین می‌شود. در ادامه، یک نمونه از کلاس Vocabulary با آستانهٔ فراوانی ۱ ساخته می‌شود و با استفاده از یک لیست از جملات، واژگان ساخته می‌شود و دیکشنری واژگان چاپ می‌شود. سپس یک متن جدید عددی‌سازی‌شده و با استفاده از واژگان، نمایش داده می‌شود. در این نتیجه، دیکشنری واژگان شامل تمام توکن‌های موجود در جملات است و به هر توکن یک اندیس اختصاص داده شده است. همچنین متن نمونه به صورت عددی نمایش داده شده است، به این صورت که هر توکن در متن با مقدار متناظر در دیکشنری واژگان جایگزین شده است. دستورات و نتیجهٔ اجرای آن به شرح برنامهٔ ۱ است:

Program 1: Creating a Vocabulary

```python
# Import the Vocabulary class
from nltk.tokenize import RegexpTokenizer

class Vocabulary:
  def __init__(self, frequency_threshold):
    # Special tokens and their corresponding indices
    self.itos = {0: "<PAD>", 1: "<SOS>", 2: "<EOS>", 3: "<UNK>"}
    self.stoi = {"<PAD>": 0, "<SOS>": 1, "<EOS>": 2, "<UNK>": 3}
    self.frequency_threshold = frequency_threshold


  def __len__(self):
    # Return the total number of tokens in the vocabulary
    return len(self.itos)

  def tokenizer(self, text):
    # Tokenize the text using a regular expression tokenizer
    tokenizer = RegexpTokenizer(r'\w+')
    return [token.lower() for token in tokenizer.tokenize(text)]

  def build_vocab(self, caption_list):
    frequencies = {}
    idx = 4

    # Iterate over each caption in the list
    for caption in caption_list:
      # Tokenize the caption
      for token in self.tokenizer(caption):
        # Update the token frequencies
        if token not in frequencies:
          frequencies[token] = 1
        else:
          frequencies[token] += 1

        # Check if the token frequency reaches the threshold
        if frequencies[token] == self.frequency_threshold:
          # Add the token to the vocabulary with a new index
          self.stoi[token] = idx
          self.itos[idx] = token
          idx += 1

  def numericalize(self, text):
    # Tokenize the text
    tokenized_text = self.tokenizer(text)
```

```
45
46      # Convert tokens to their corresponding indices in the vocabulary
47      return [self.stoi[token] if token in self.stoi else self.stoi["<UNK>"] for token in
        tokenized_text]
48
49  # Create an instance of the Vocabulary class with a frequency threshold of 1
50  v = Vocabulary(frequency_threshold=1)
51
52  # Build the vocabulary using a list of captions
53  v.build_vocab(["I am Mohammad Javad Ahmadi, a student of Dr. Keller's Deep Learning course."])
54
55  # Print the vocabulary dictionary
56  print(f"Vocabulary dictionary: {v.stoi}")
57
58  # Numericalize a new text using the vocabulary
59  numericalized_text = v.numericalize("I am Mohammad Javad Ahmadi, a student of Dr. Keller's Deep
        Learning course.")
60
61  # Print the numericalized text
62  print(f"Numericalized text: {numericalized_text}")
63  ----------------------------------------------------
64  Vocabulary dictionary: {'<PAD>': 0, '<SOS>': 1, '<EOS>': 2, '<UNK>': 3, 'i': 4, 'am': 5, '
        mohammad': 6, 'javad': 7, 'ahmadi': 8, 'a': 9, 'student': 10, 'of': 11, 'dr': 12, 'keller':
        13, 's': 14, 'deep': 15, 'learning': 16, 'course': 17}
65  Numericalized text: [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
```

در ادامه، یک کلاس با نام `FlickrDataset` تعریف می‌کنیم که از کلاس `torch.utils.data.Dataset` ارث‌بری می‌کند و برای مدیریت مجموعه داده‌ها در فرمت دیتاست فلیکر استفاده می‌شود. ابتدا کتابخانه‌های لازم فراخوانی می‌شوند. سپس، کلاس `Dataset` از کتابخانهٔ `torch.utils.data` وارد می‌شود که یک کلاس پایه برای ایجاد دیتاست‌های شخصی‌سازی‌شده در پای‌تورچ است. هم‌چنین برای کار با تصویر، کلاس `Image` از کتابخانهٔ `PIL` وارد می‌شود. برای کار با تنسورها و سایر موارد و ابزارهای لازم هم تورچ را وارد می‌کنیم. در ادامه، یک کلاس به نام `FlickrDataset` تعریف می‌کنیم که از کلاس `Dataset` ارث‌بری می‌کند. متد `init` سازندهٔ این کلاس است که در آن مقادیر اولیه متغیرها و ورودی‌ها تنظیم می‌شود. سپس، فایل کپشن را با استفاده از تابع `read_csv` از کتابخانه `pandas` می‌خوانیم و به صورت یک جدول داده‌ها (DataFrame) در متغیر `df` ذخیره می‌کنیم. بر اساس نوع داده، تقسیم‌بندی داده‌ها را انجام می‌دهیم. اگر نوع داده `train` باشد، از ابتدای دیتافریم `df` تا ۹۰ درصد آن را به عنوان دادهٔ آموزش انتخاب می‌کنیم. اگر نوع داده `test` باشد، ۱۰ درصد باقی‌مانده دیتافریم `df` را به‌نوان دادهٔ ارزیابی انتخاب می‌کنیم. در تعاریف دقت داریم که بخشی را برای محاسبهٔ اندیس شروع داده‌های ارزیابی براساس ۱۰ درصد باقی‌مانده دیتافریم در نظر بگیریم و داده‌های ارزیابی را از اندیس شروع تا انتهای دیتافریم انتخاب کنیم. همچنین، اندیس‌های ردیف‌ها را دوباره از ابتدا شماره‌گذاری می‌کنیم. مسیر ریشه (root directory) دیتاست را در متغیر `self.root_dir` ذخیره می‌کنیم. تبدیلات و ترانسفورمیشن‌هایی که روی تصاویر باید اعمال شوند را در متغیر `self.transform` ذخیره می‌کنیم. این ورودی اختیاری است و در صورتی که ارائه نشود، مقدار `None` را دارد.: نام تصاویر را از ستون `image` در دیتافریم `self.df` استخراج کرده و در متغیر `self.imgs` ذخیره می‌کنیم. متن‌های کپشن را هم از ستون `caption` در دیتافریم `self.df` استخراج کرده و در متغیر `self.captions` ذخیره می‌کنیم. یک نمونه از کلاس `Vocabulary` را با استفاده از مقدار `frequency_threshold`

به‌عنوان آستانهٔ تکرار استفاده کرده و آن را در متغیر `self.vocab` ذخیره می‌کنیم. با استفاده از نمونهٔ `self.vocab`، لغت‌نامه را بر اساس متن‌های کپشن موجود در `self.captions` ایجاد می‌کنیم و آن را ساختاردهی می‌کنیم. در ادامه، متد `len` را برای بازگرداندن طول دیتاست که برابر با تعداد نمونه‌ها در `self.df` است، تعریف می‌کنیم. متد `getitem` را هم برای بازگرداندن نمونه‌های دیتاست به صورت متناظر با اندیس تعریف می‌کنیم. متن کپشن مربوط به اندیس مورد نظر از متغیر `self.captions` را استخراج کرده و در متغیر `caption` ذخیره می‌کنیم. نام تصویر مربوط به اندیس مورد نظر از متغیر `self.imgs` را هم استخراج کرده و در متغیر `image_name` ذخیره می‌کنیم. ترکیب مسیر ریشه (`self.root_dir`) و نام تصویر `image_name` را به عنوان مسیر کامل تصویر در نظر گرفته و در متغیر `image_path` ذخیره می‌کنیم. سپس، با استفاده از تابع `Image.open` از کتابخانه PIL، تصویر را از مسیر `image_path` باز می‌کنیم و آن را به حالت RGB تبدیل می‌کنیم. تصویر در متغیر `img` ذخیره می‌شود. هم‌چنین اگر تبدیلاتی را مدنظر داشته باشیم، اِعمال می‌کنیم. بردارسازی کپشن را با توجه به نماد شروعِ SOS آغاز می‌کنیم و آن را در متغیر `vectorized_caption` ذخیره می‌کنیم. متن کپشن را به عدد تبدیل کرده و به متغیر `vectorized_caption` اضافه می‌کنیم. تابع `numericalize` از لغت‌نامهٔ `self.vocab` استفاده می‌کند. نماد پایانِ EOS را هم به بردارسازی کپشن اضافه می‌کنیم. تصویر `img` و کپشن بردارسازی‌شده یا `vectorized_caption` را به‌نوان تنسورها با استفاده از تابع `torch.tensor` بازمی‌گردانیم. در نهایت، این کد یک دیتاست شخصی‌سازی‌شده را برای استفاده در آموزش و ارزیابی مدل‌های یادگیری عمیق ایجاد می‌کند. این دیتاست شامل تصاویر و کپشن‌های متنی برای تصاویر است. همچنین، دیتاست شامل توابعی است که لغت‌نامه را براساس کپشن‌ها ساخته و تصاویر و کپشن‌ها را به صورت مناسب برای استفاده در مدل‌های عمیق پیش‌پردازش می‌کند. دستورات در برنامهٔ ۲ آورده شده اند.

Program 2: Creating a Custom Dataset

```python
import pandas as pd
from torch.utils.data import Dataset
from PIL import Image
import torch

class FlickrDataset(Dataset):
  def __init__(self, root_dir, caption_file, transform=None, frequency_threshold=5, data_type='
    train'):
    # Read the caption file into a DataFrame
    df = pd.read_csv(caption_file)

    # Split the dataset into train and test based on the data_type
    if data_type == 'train':
      # Select the first 90% of the DataFrame for training
      self.df = df.iloc[:int(0.9 * len(df))]
    elif data_type == 'test':
      # Select the remaining 10% of the DataFrame for testing
      test_start_index = int(0.9 * len(df))
      self.df = df.iloc[test_start_index:].reset_index(drop=True)

    else:
      # If data_type is neither 'train' nor 'test', do nothing
      pass

```

```
    self.root_dir = root_dir
    self.transform = transform

    # Store the image names and captions
    self.imgs = self.df['image']
    self.captions = self.df['caption']

    # Initialize the vocabulary and build the vocabulary
    self.vocab = Vocabulary(frequency_threshold)
    self.vocab.build_vocab(self.captions.tolist())

  def __len__(self):
    # Return the length of the dataset
    return len(self.df)

  def __getitem__(self, index):
    # Get the caption and image information for the given index
    caption = self.captions[index]
    image_name = self.imgs[index]
    image_path = self.root_dir + '/' + image_name
    img = Image.open(image_path).convert("RGB")

    if self.transform is not None:
      # Apply the specified transformations to the image
      img = self.transform(img)

    # Convert the caption to a vectorized form
    vectorized_caption = [self.vocab.stoi["<SOS>"]]
    vectorized_caption += self.vocab.numericalize(caption)
    vectorized_caption.append(self.vocab.stoi["<EOS>"])

    # Return the image and its vectorized caption as tensors
    return img, torch.tensor(vectorized_caption)
```

سپس، دستوراتی برای تعریف توابع و کلاس‌های مربوط به نمایش و پیش‌پردازش تصاویر و ایجاد نمونه‌های دیتاست می‌نویسیم. ابتدا کتابخانهٔ `matplotlib` را برای نمایش تصاویر وارد می‌کنیم. نام‌پای را هم برای برای کار با آرایه‌ها و محاسبات عددی وارد می‌کنیم. تابع `show_image` را برای نمایش تصویر با استفاده از یک تانسور ورودی استفاده می‌کنیم. تنسور تصویر را به یک آرایهٔ نام‌پای تبدیل کرده و کارهایی از قبیل تغییر ابعاد آن به شکل (ارتفاع، عرض، کانال) انجام می‌دهیم. با استفاده از تابع `imshow` در `matplotlib` تصویر را نمایش می‌دهیم. بخشی را برای نمایش کپشن و عنوان تصویر در نظر می‌گیریم و یک توقف هم برای به‌روزرسانی. تبدیل‌هایی که روی تصاویر باید اعمال شوند را تعریف می‌کنیم. برای این منظور از تابع `Compose` که ترکیبی از تبدیل‌هاست استفاده می‌کنیم. ابعاد تصویر را به ۲۲۴ در ۲۲۴ تبدیل می‌کنیم و تصاویر را به تنسور تلدیل می‌کنیم. سپس با استفاده از کلاس مخصوصی که قبلاً تعریف کردیم، نمونه‌های آموزش و ارزیابی را معین می‌کنیم. دستورات در برنامهٔ ۴ آورده شده اند.

Program 3: Make & Preprocess Dataset and Visualizing Data Samples

```python
import matplotlib.pyplot as plt
import numpy as np

def show_image(image_tensor, title=None):
    """
    Display an image represented as a tensor.

    Args:
        image_tensor (torch.Tensor): The input image tensor.
        title (str, optional): The title of the image. Defaults to None.
    """
    # Convert the image tensor to a NumPy array and change the dimensions
    image_np = image_tensor.numpy().transpose((1, 2, 0))

    # Display the image using matplotlib
    plt.imshow(image_np)

    # Set the title of the image if provided
    if title is not None:
        plt.title(title)

    # Pause a bit to allow the plot to be updated
    plt.pause(0.01)

# Import the necessary libraries
import torchvision.transforms as T

# Define the transform to be applied to the images
transforms = T.Compose([
    T.Resize((224, 224)),  # Resize the images to (224, 224)
    T.ToTensor()  # Convert the images to tensors
])

# Create instances of the FlickrDataset class for training and testing
train_dataset = FlickrDataset(
    root_dir="/content/Dataset/Images",  # Path to the root directory of the images
    caption_file="captions.txt",  # Path to the captions file
    transform=transforms,  # Apply the defined transforms to the images
    data_type='train'  # Specify the data type as 'train'
)

test_dataset = FlickrDataset(
    root_dir="/content/Dataset/Images",  # Path to the root directory of the images
    caption_file="captions.txt",  # Path to the captions file
```

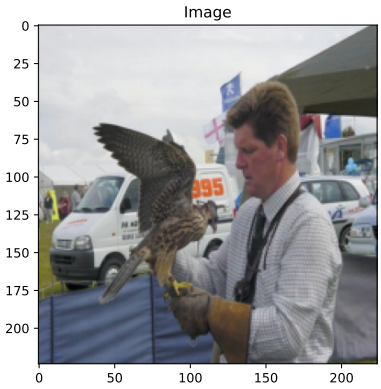
```
    transform=transforms,  # Apply the defined transforms to the images
    data_type='test'  # Specify the data type as 'test'
)
```

در ادامه دستوراتی را برای به نمایش گذاشتن تصویر و نمایش متن متناظر با آن تصویر می‌نویسیم. توضیح هر بخش کد ابتدا تصویر و شرح‌های متناظر با تصویر صدم در داده‌های آموزش از دیتاست train_dataset را بازیابی می‌کنیم. کد تصویر و کپشن‌ها را به متغیرهای image و captions اختصاص می‌دهیم. در ادامه تصویر و عنوان آن را نمایش داده و کپشن‌های متناظر با تصویر را به صورت توکنی چاپ می‌کنیم و سپس کد توکن‌های موجود را به کلمات متناظر تبدیل می‌کنیم. دستورات مربوطه به این قسمت و نتایج مربوط به دو نمونه تصویر خروجی نشان‌داده‌شده در ؟؟ در شکل ۱ آورده شده است.

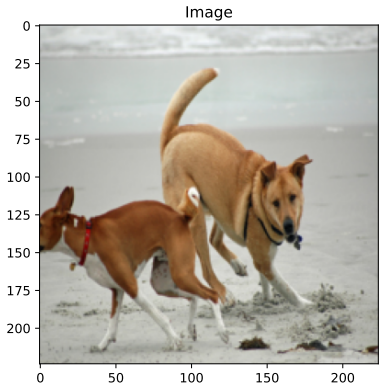
Program 4: Visualizing Data Samples

```
import matplotlib.backends.backend_pdf as pdf_backend

# Retrieve the image and captions for the 100th data point from the training dataset
image, captions = train_dataset[100]

# Display the image using the show_image function
show_image(image, "Image")

# Print the tokenized captions
print("Tokenized Captions:", captions)

# Convert the tokenized captions to their corresponding words
words = [train_dataset.vocab.itos[token] for token in captions.tolist()]

# Print the sentence
print("Sentence:")
print(words)

# Save the image as a PDF
pdf = pdf_backend.PdfPages('image100tr.pdf')
plt.savefig(pdf, format='pdf')
pdf.close()
-----------------------------------------
Train:
Tokenized Captions: tensor([   1,   48,  318, 1895,   10,   29,   15,   20,   49,   96,   12,
    9,
          50,    2])
Sentence:
['<SOS>', 'two', 'different', 'breeds', 'of', 'brown', 'and', 'white', 'dogs', 'play', 'on', 'the
    ', 'beach', '<EOS>']

Test:
Tokenized Captions: tensor([  1,  12,  15,   8,  40,   3,  33,  48,   4,   3,   6,  16, 504,
    2])
```

```
Sentence:
['<SOS>', 'the', 'man', 'is', 'wearing', '<UNK>', 'and', 'holding', 'a', '<UNK>', 'in', 'his', '
    hands', '<EOS>']
```



(ب) صدمین دادهٔ ارزیابی



(آ) صدمین دادهٔ آموزش

شکل ۱: نمایش نمونه‌دادهٔ آموزش و ارزیابی.

در ادامه، یک کلاس به نام Apppadd پیاده‌سازی می‌کنیم که به عنوان تابع تجمیع در دیتالودر استفاده می‌شود تا پدینگ (Padding) را برای جملات در هر دسته‌بندی اعمال کند. متد init این کلاس، دو پارامتر pad_idx و batch_first را دریافت می‌کند. pad_idx نشان‌دهنده شاخص توکن پدینگ است و batch_first مشخص می‌کند که آیا بعد دسته‌ها باید در اولین بُعد باشد یا نه. در ادامه پدینگ لازم روی جملات در دسته اِعمال می‌شود. ابتدا تصاویر را از دسته استخراج می‌کنیم و با استفاده از unsqueeze بُعد بچ را به آن‌ها اضافه می‌کنیم. سپس تمام جملات را از دسته استخراج می‌کنیم و با استفاده از pad_sequence، جملات را با پدینگ توکن مشخص‌شده (pad_idx) پد می‌کنیم. در نهایت، تصاویر و جملات پدشده را به عنوان خروجی بازمی‌گردانیم. با استفاده از دستورات آورده‌شده در برنامهٔ ۵ می‌توانیم تصاویر و جملات را با پدینگ مناسب برای آموزش مدل به صورت دسته‌بندی‌شده (دسته‌ای) دریافت کنیم.

Program 5: Implement Padding for Sentences in Each Batch

```
class Apppadd:
    """
    Collate function to apply padding to captions with DataLoader.
    """

    def __init__(self, pad_idx, batch_first=False):
        """
        Initialize the CapsCollate class.

        Args:
            pad_idx (int): The index of the padding token.
            batch_first (bool): Whether to return the batch dimension as the first dimension.
        """
```

```
        self.pad_idx = pad_idx
        self.batch_first = batch_first

    def __call__(self, batch):
        """
        Apply padding to the captions in the batch.

        Args:
            batch (list): List of tuples containing image and caption pairs.

        Returns:
            torch.Tensor: Batch of images.
            torch.Tensor: Batch of padded captions.
        """
        # Extract images from the batch and unsqueeze them
        imgs = [item[0].unsqueeze(0) for item in batch]
        imgs = torch.cat(imgs, dim=0)

        # Extract captions from the batch
        captions = [item[1] for item in batch]

        # Pad the captions using pad_sequence
        captions = pad_sequence(captions, batch_first=self.batch_first, padding_value=self.
    pad_idx)

        return imgs, captions
```

در ادامه دو متغیر BATCH_SIZE و NUM_WORKERS را تعریف می‌کنیم که به ترتیب برای تعیین اندازهٔ دسته و تعداد فرآیندهای WORKER در دیتالودر استفاده می‌شوند. به عنوان توضیحی از برخی متغیرهای داخل دیتالودر؛ از num_workers برای تعیین تعداد پردازش‌های موازی که برای بارگذاری داده‌ها به کار گرفته می‌شود استفاده می‌شود. به این صورت که، زمانی که داده‌ها برای آموزش شبکه باید از دیسک بارگذاری شوند، تعداد ورکرهایی که برای بارگذاری داده‌ها باید به کار گرفته می‌شود با توجه به مواردی قابل تعیین است: تعداد ورکرها باید کمتر از تعداد کل هسته‌های پردازنده سیستم باشد تا برنامه به صورت صحیح و با عملکرد بهینه اجرا شود. تعداد ورکرها باید به اندازهٔ کافی بزرگ باشد تا بتوانند با توجه به مواردی مانند حجم داده و سرعت بارگذاری آن‌ها، داده‌ها را به صورت موازی بارگذاری و تحویل دهند. بنابراین، با افزایش این پارامتر و متغیر، تعداد ورکرها بیشتر و در نتیجه توانایی بارگذاری موازی داده‌ها به صورت بیشتری افزایش می‌یابد. البته این مقدار نباید بیش از تعداد هسته‌های پردازنده و میزان منابع دیگر مانند حافظه‌ی سیستمی را در نظر گرفته شود. سپس، متغیر pad_idx تعریف شده است که مقدار آن برابر با اندیس توکن پدینگ (PAD) در واژگان دادگان آموزشی (train_dataset.vocab) است. این توکن به عنوان نشانه‌ای برای پدینگ کپشن‌ها در مدل استفاده می‌شود. سپس، با استفاده از تابع دیتالودر، یک شئ دیتالودر برای بارگیری مجموعه دادهٔ آموزشی ایجاد می‌شود. پارامترهای مشخص‌شده عبارتند از: dataset که مشخص می‌کند که کدام مجموعه داده باید بارگیری شود، در اینجا مجموعه دادهٔ آموزشی (train_dataset) استفاده شده است. batch_size که سایز دسته را مشخص می‌کند. num_workers که تعداد فرآیندهای ورکر برای بارگیری داده را تعیین می‌کند. shuffle که مشخص می‌کند آیا داده‌ها برای هر دوره به صورت تصادفی مخلوط شوند یا خیر. collate_fn تابع collate برای پدینگ کپشن‌ها را مشخص می‌کند. در اینجا

از کلاس Apppadd استفاده شده است. در نهایت، با استفاده از این شئ دیتالودر می‌توان داده‌های آموزشی را به‌صورت دسته‌ای با سایز مشخص، با پدینگ کپشن، تصادفی‌شده و با تعداد فرآیندهای ورکر مناسب بارگیری کرد. دستورات مربوطه به شرح زیر است.

```
import multiprocessing

NUM_WORKERS = multiprocessing.cpu_count()
print("Maximum number of workers:", NUM_WORKERS)

BATCH_SIZE = 4
NUM_WORKERS = 2

# Token to represent the padding
pad_idx = train_dataset.vocab.stoi["<PAD>"]

# Create a DataLoader object to load the training dataset
data_loader = DataLoader(
    dataset=train_dataset,              # Specify the dataset to load
    batch_size=BATCH_SIZE,              # Set the batch size
    num_workers=NUM_WORKERS,            # Set the number of worker processes for data loading
    shuffle=True,                       # Shuffle the data for each epoch
    collate_fn=Apppadd(pad_idx=pad_idx, batch_first=True)   # Specify the collate function for
    padding captions
)
```

در ادامه دستوراتی برای نمایش تصاویر و کپشن‌های یک دستهٔ تصادفی می‌نویسیم. ابتدا با استفاده از تابع iter، یک iterator از دیتالودر به نام dataiter ایجاد می‌کنیم. این iterator قادر به بازیابی دسته‌های متوالی از داده‌هاست. با استفاده از تابع next، دسته‌ای از داده‌ها از dataiter بازیابی می‌شود و در متغیر batch قرار می‌گیرد. سپس، با استفاده از عملگر unpacking، دسته از تصاویر و کپشن‌ها را به‌ترتیب در متغیرهای مربوط به خود (متغیرهای images و captions) قرار می‌دهیم. متغیر effective_batch_size برابر با کوچکترین مقدار بین BATCH_SIZE و تعداد تصاویر موجود در images تعیین می‌شود. سپس، با یک حلقهٔ for روی اندیس‌های تا effective_batch_size، اطلاعات هر تصویر در دسته را نمایش می‌دهیم. برای هر تصویر تصویر و توضیحات مربوطه را از متغیرهای images و captions استخراج می‌کنیم. برچسب کپشن را از کپشن عددی‌شده با استفاده از train_dataset.vocab.itos استخراج می‌کنیم. اندیس توکن EOS را پیدا کرده و کپشن را برش می‌دهیم تا کپشن تکمیلی حذف شود. لیست کلمات برچسب کپشن را به یک رشته تبدیل می‌کنیم و یک شکل جدید برای نمایش تصویر ایجاد می‌کنیم. سپس، تصویر را همراه با برچسب کپشن در شکل نمایش می‌دهیم و عرض تصویر را محاسبه کرده و ارتفاع جعبه توضیح را تعیین می‌کنیم. یک جعبهٔ رنگی با برچسب کپشن را اضافه می‌کنیم. شکل را به عنوان یک فایل پی‌دی‌اف ذخیره می‌کنیم و آن را نمایش می‌دهیم. دستورات مربوطه به شرح زیر و نمونه‌ای از اجرا در شکل ۲ نمایش داده شده است.

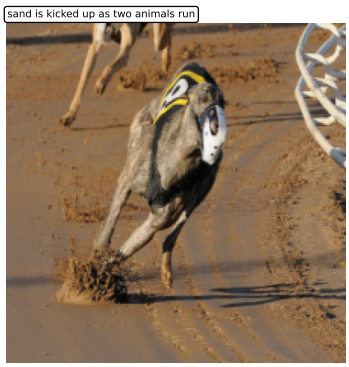
```
import matplotlib.pyplot as plt
import numpy as np

# Generating the iterator from the dataloader
dataiter = iter(data_loader)
```

```
6
7  # Getting the next batch
8  batch = next(dataiter)
9
10 # Unpacking the batch
11 images, captions = batch
12
13 # Determine the effective batch size
14 effective_batch_size = min(BATCH_SIZE, len(images))
15
16 # Showing information of each image in the batch
17 for i in range(effective_batch_size):
18     img, cap = images[i], captions[i]
19
20     # Extracting the caption label from the numericalized caption
21     caption_label = [train_dataset.vocab.itos[token] for token in cap.tolist()]
22
23     # Finding the index of '<EOS>' token to truncate the caption
24     eos_index = caption_label.index('<EOS>')
25     caption_label = caption_label[1:eos_index]
26
27     # Joining the caption label words into a single string
28     caption_label = ' '.join(caption_label)
29
30     # Create a new figure
31     fig = plt.figure()
32
33     # Displaying the image with the caption label
34     plt.imshow(np.transpose(img.numpy(), (1, 2, 0)))
35     plt.axis('off')
36
37     # Calculating the width of the image
38     img_width = img.shape[1]
39
40     # Calculating the height of the caption box
41     caption_height = int(img_width / 50)
42
43     # Adding a colored box with the caption label
44     plt.text(0, -10, caption_label, bbox=dict(facecolor='white', edgecolor='black', boxstyle='
       round'),
45              fontsize=8, color='black', ha='left', va='top')
46
47     # Save the plot as a PDF file
48     plt.savefig(f'captionedimage{i}.pdf', format='pdf')
49
```

```
    # Display the plot
    plt.show()
```

(آ) دادهٔ اول

(ب) دادهٔ دوم



(ج) دادهٔ سوم

(د) دادهٔ چهارم

شکل ۲: نمونه‌ای از تصویر/کپشن یک دسته از داده‌ها.

حال به سراغ پیاده‌سازی مدل می‌رویم و قبل از هرچه تکلیف داده‌ها را با توجه به تمامی کلاس‌ها و توابعی که تاکنون نوشته‌ایم مشخص می‌کنیم. برای این منظور دستوراتی می‌نویسیم که در آن متغیر `pad_idx` نشان می‌دهد که در نوشتار عددی کپشن‌ها، علامت `PAD` با چه عددی نشان داده شود. متغیر `transforms` را هم برای ایجاد یک ترکیب از تبدیلات که بر تصاویر باید اعمال شوند تعریف می‌کنیم. این تحویل‌ها شامل تغییر اندازهٔ پیکسلب تصاویر، بریدن تصادفی تصاویر، تبدیل تصاویر به تنسور، و نرمال‌سازی تصاویر با توجه به اعداد موردنیاز برای رزنت پیش‌آموزش‌دیده هستند. سپس، یک نمونه از کلاس `FlickrDataset` با استفاده از پارامترهای مناسب ایجاد می‌کنیم. این نمونه برای آموزش استفاده می‌شود و ترکیب تبدیل‌های مختلف روی تصاویر اعمال می‌شود. در ادامه، یک نمونه از کلاس دیتالودر برای مجموعه آموزش ایجاد می‌شود. این نمونه از دیتالودر تعداد تصاویر در هر دسته را برابر با موارد از پیش مشخص‌شده تعیین می‌کند. همچنین، داده‌ها در هر دورهٔ آموزش قبل از اعمال دسته‌بندی، مخلوط می‌شوند تا ترتیب داده‌ها تاثیری بر فرآیند آموزش نداشته باشد. همچنین، تابع `collate_fn` برای اعمال پرکردن (`padding`) به کپشن‌ها استفاده می‌شود. دستورات به شرح زیر است:

```
BATCH_SIZE = 256
NUM_WORKERS = 2
pad_idx = train_dataset.vocab.stoi["<PAD>"]

# Define the transformations to be applied, including resizing, random cropping,
# converting to tensor, and normalization using ResNet statistics
transforms = T.Compose([
    T.Resize(256),
    T.RandomCrop(224),
    T.ToTensor(),
    T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

# Create an instance of the FlickrDataset for training
train_dataset = FlickrDataset(
    root_dir="/content/Dataset/Images",
    caption_file="captions.txt",
    transform=transforms,
    data_type='train'
)

# Create a DataLoader for the training dataset
train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    shuffle=True,
    collate_fn=Apppadd(pad_idx=pad_idx, batch_first=True)
)
```

در قسمت تعریف مدل یک سری کلاس‌ها و مدل‌ها را تعریف می‌کنیم که در ترکیب با یکدیگر برای تولید کپشن برای تصاویر استفاده می‌شوند. سه کلاس `ResNet`، `LSTM` و `CNNtoRNN` تعریف شده‌اند که در ترکیب با هم برای عملیات توصیف تصاویر استفاده می‌شوند. هر یک از این کلاس‌ها شامل متدهایی هستند که عملکرد و ویژگی‌های مربوط به آن کلاس را تعیین می‌کنند.

- کلاس `ResNet` یک مدل از شبکه عصبی ResNet را تعریف می‌کند. این مدل دارای یک لایه `Embedding` است که از تصاویر ویژگی‌های آن‌ها را استخراج می‌کند. لایهٔ Embedding خروجی لایهٔ تماماً‌متصل آخر مدل ResNet-18 را به یک فضای تعبیه‌شده با اندازهٔ مشخص تبدیل می‌کند. در این کلاس، می‌توانیم تصمیم بگیریم که آیا بخش‌های مختلف مدل ResNet قابل آموزش باشند یا خیر. در متد اولیهٔ این کلاس، مدل ResNet-18 پیش‌آموزش‌دیده بارگذاری می‌شود. سپس پارامتر `train_resnet` که نشان می‌دهد آیا باید مدل قابل آموزش باشد یا خیر، استفاده می‌شود تا تصمیم گیری شود که آیا پارامترهای مدل قابل آموزش باشند یا خیر. سپس، لایه‌های مورد نیاز از مدل تا لایهٔ تماماً‌متصل آخر با استفاده از `list(resnet.children())[:-1]` استخراج می‌شوند و به عنوان لایهٔ `resnet` تعریف می‌شوند. همچنین، لایهٔ `embed` به‌عنوان لایهٔ تبدیل بردار ویژگی‌ها به بردار تعبیه‌شده استفاده می‌شود. در نهایت، لایه‌های `relu` (تابع فعال‌سازی ReLU) و `dropout` (لایه Dropout) هم در کار وارد می‌شوند.

- کلاس LSTM یک مدل شبکه عصبی بازگشتی LSTM را تعریف می‌کند. این مدل هم دارای یک لایهٔ Embedding است که کلمات کپشن را به یک فضای تعبیه‌شده تبدیل می‌کند. سپس، با استفاده از لایهٔ LSTM، بردارهای ویژگی تصویر و بردارهای تعبیه‌شدهٔ کلمات کپشن با یکدیگر ترکیب می‌شوند و خروجی را تولید می‌کنند. این خروجی را می‌توان برای پیش‌بینی کلمات کپشن بعدی استفاده کرد. در متد اولیهٔاین کلاس، لایه تعبیهٔ کلمات (embedding) با اندازهٔ embed_size تعریف می‌شود. همچنین، لایهٔ LSTM با اندازهٔ ورودی embed_size و اندازهٔ وضعیت مخفی hidden_size تعریف می‌شود. از لایهٔ خطی نیز برای پیش‌بینی کلمهٔ بعدی استفاده می‌شود. در انتها، لایهٔ Dropout نیز تعریف می‌شود.
- کلاس CNNtoRNN یک مدل ترکیبی از ResNet و LSTM را تعریف می‌کند. این مدل شامل دو بخش اصلی است: بخش انکودر و بخش دیکودر. بخش انکودر شامل مدل رزنت می‌شود که وظیفهٔ استخراج ویژگی‌های تصویر را دارد. بخش دیکودر مدل بازگشتی را شامل می‌شود که با استفاده از ویژگی‌های استخراج‌شده توسط انکودر و کلمات کپشن، خروجی‌های پیش‌بینی را تولید می‌کند. در متد اولیهٔ این کلاس، یک نمونه از مدل رزنت به عنوان انکودر و یک نمونه از مدل بازگشتی LSTM به عنوان دیکودر ایجاد می‌شود. در متد forward این کلاس، تصاویر از مدل انکودر گذر داده شده و ویژگی‌های تصویر استخراج می‌شوند. سپس ویژگی‌ها و کپشن‌ها از مدل دیکودر گذر داده شده و خروجی‌ها محاسبه می‌شوند.

در مجموع، توابع init هر کلاس وظیفهٔ مقداردهی اولیه پارامترها و تعریف لایه‌ها و ماژول‌های مورد نیاز مدل را دارند. توابع forward هر کلاس وظیفهٔ انجام محاسبات لازم برای انتقال به جلوی مدل را دارند. در کلاس رزنت، ویژگی‌های استخراج‌شده توسط مدل به فضای تعبیهٔ خروجی مدل تبدیل می‌شوند. در کلاس بازگشتی هم، بردارهای تعبیهٔ کلمات و ویژگی‌های تصویر با یکدیگر ترکیب کرده و خروجی را تولید می‌کنند. در کلاس CNNtoRNN، تصاویر از طریق بخش انکودر یا مدل رزنت عبور می‌کنند و ویژگی‌های استخراج‌شده به بخش دیکودر یا مدل بازگشتی منتقل می‌شوند تا خروجی‌های پیش‌بینی را تولید کنند. در ادامه، کلاس بازگشتی دارای تابع generate_caption است که با استفاده از تصاویر ویژگی و نمونه‌های مخفی آغازین، شروع به تولید کپشن‌های متنی می‌کند. تا زمانی که علامت EOS که نشان‌دهندهٔ پایان کپشن است تولید نشود یا تعداد کلمات کپشن تولیدشده به حداکثر تعداد کلمات مجاز نرسد، کلمات پیش‌بینی‌شده اضافه و به عنوان ورودی به مدل داده می‌شود. در نهایت، نمایش رشته‌ای کلمات کپشن تولیدشده به عنوان خروجی تابع ارائه می‌شود. با ارائهٔ این توضیحات مدل در برنامهٔ ۶ تعریف شده است.

Program 6: Model

```python
#ResNet Model
class ResNet(nn.Module):
    def __init__(self, embed_size, train_resnet=False):
        """
        Initialize the ResNet model with a specified embedding size.

        Args:
            embed_size (int): Size of the embedding output.
            train_resnet (bool): Whether to train the ResNet backbone or not.
        """
        super(ResNet, self).__init__()

        # Load the pretrained ResNet-18 model
```

```
        resnet = models.resnet18(pretrained=True)

        # Set the requires_grad flag of the ResNet parameters
        # to control whether they are trainable or not
        for param in resnet.parameters():
            param.requires_grad_(train_resnet)

        # Extract the modules of the ResNet model up to the last fully connected layer
        modules = list(resnet.children())[:-1]

        # Create the ResNet backbone with the extracted modules
        self.resnet = nn.Sequential(*modules)

        # Create the embedding layer
        self.embed = nn.Linear(resnet.fc.in_features, embed_size)

        # Activation function and dropout layer
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)

    def forward(self, images):
        """
        Forward pass of the ResNet model.

        Args:
            images (tensor): Input images tensor.

        Returns:
            features (tensor): Embedded features tensor.
        """
        features = self.dropout(self.relu(self.resnet(images)))
        features = features.view(features.size(0), -1)
        features = self.embed(features)
        return features

# LSTM Network
class LSTM(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1, drop_prob=0.5):
        """
        Initialize the LSTM model with specified sizes and parameters.

        Args:
            embed_size (int): Size of the word embedding.
            hidden_size (int): Size of the hidden state of the LSTM.
            vocab_size (int): Size of the vocabulary.
```

```
            num_layers (int): Number of layers in the LSTM (default: 1).
            drop_prob (float): Dropout probability (default: 0.5).
        """
        super().__init__()

        # Word embedding layer
        self.embedding = nn.Embedding(vocab_size, embed_size)

        # LSTM layer
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers=num_layers, batch_first=True)

        # Linear layer for prediction
        self.linear = nn.Linear(hidden_size, vocab_size)

        # Dropout layer
        self.dropout = nn.Dropout(drop_prob)

    def forward(self, features, captions):
        """
        Forward pass of the LSTM model.

        Args:
            features (tensor): Image features tensor.
            captions (tensor): Captions tensor.

        Returns:
            x (tensor): Output tensor.
        """
        # Vectorize the caption by passing it through the embedding layer
        embeds = self.dropout(self.embedding(captions[:, :-1]))

        # Concatenate the features and captions
        x = torch.cat((features.unsqueeze(1), embeds), dim=1)

        # Pass through the LSTM layer
        x, _ = self.lstm(x)

        # Pass through the linear layer for prediction
        x = self.linear(x)

        return x

    def generate_caption(self, inputs, hidden=None, max_len=20, vocab=None):
        """
        Generate captions given the image features.
```

```
104
105         Args:
106             inputs (tensor): Input tensor of image features.
107             hidden (tuple): Hidden state of the LSTM (default: None).
108             max_len (int): Maximum length of the generated caption (default: 20).
109             vocab (Vocab): Vocabulary object (default: None).
110
111         Returns:
112             caption (list): Generated caption as a list of words.
113         """
114         # Inference part
115         # Given the image features, generate the captions
116
117         batch_size = inputs.size(0)
118
119         captions = []
120
121         for i in range(max_len):
122             output, hidden = self.lstm(inputs, hidden)
123             output = self.linear(output)
124             output = output.view(batch_size, -1)
125
126             # Select the word with the highest value
127             predicted_word_idx = output.argmax(dim=1)
128
129             # Save the generated word
130             captions.append(predicted_word_idx.item())
131
132             # End if <EOS> is detected
133             if vocab.itos[predicted_word_idx.item()] == "<EOS>":
134                 break
135
136             # Send the generated word as the next caption
137             inputs = self.embedding(predicted_word_idx.unsqueeze(0))
138
139         # Convert the vocabulary indices to words and return the sentence
140         return [vocab.itos[idx] for idx in captions]
141
142 # Utilizing the powerful fusion of ResNet and LSTM for image captioning
143 class CNNtoRNN(nn.Module):
144     def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1, drop_prob=0.5,
    train_resnet=False):
145         """
146         Initialize the CNNtoRNN model with specified sizes and parameters.
147
```

```
        Args:
            embed_size (int): Size of the word embedding.
            hidden_size (int): Size of the hidden state of the LSTM.
            vocab_size (int): Size of the vocabulary.
            num_layers (int): Number of layers in the LSTM (default: 1).
            drop_prob (float): Dropout probability (default: 0.5).
            train_resnet (bool): Whether to train the ResNet backbone or not (default: False).
        """
        super().__init__()

        # Encoder (ResNet)
        self.encoder = ResNet(embed_size, train_resnet)

        # Decoder (LSTM)
        self.decoder = LSTM(embed_size, hidden_size, vocab_size, num_layers, drop_prob)

    def forward(self, images, captions):
        """
        Forward pass of the CNNtoRNN model.

        Args:
            images (tensor): Input images tensor.
            captions (tensor): Captions tensor.

        Returns:
            outputs (tensor): Output tensor.
        """
        # Pass the images through the encoder (ResNet) to get features
        features = self.encoder(images)

        # Pass the features and captions through the decoder (LSTM) to get outputs
        outputs = self.decoder(features, captions)

        return outputs
```

در ادامه دستوراتی را برای تعریف یک حلقهٔ آموزشی برای آموزش مدل شبکه عصبی می‌نویسیم و سپس نموداری از تغییرات خطا را در طول آموزش رسم می‌کند. ابتدا تعداد دوره‌های آموزشی را تعریف می‌کنیم. سپس، یک لیست خالی ایجاد می‌کنیم تا خطاهای آموزش ذخیره شوند. یک حلقه به‌ازای تعداد دوره‌های آموزشی معین ایجاد می‌کنیم. متغیر `running_loss` را ایجاد و آن را با مقدار صفر مقداردهی اولیه می‌کنیم. این متغیر برای محاسبه میانگین خطا در هر دوره استفاده می‌شود. سپس، یک حلقهٔ داخلی روی داده‌های آموزش ایجاد می‌کنیم و تصاویر و کپشن‌ها را به دستگاه مشخص‌شده (مثلاً GPU) منتقل می‌کنیم. پس از آن، مدل را در حالت آموزش قرار می‌دهیم و گرادیان‌های موجود در بهینه‌ساز را صفر می‌کنیم. این کار قبل از محاسبهٔ گرادیان برای دستهٔ فعلی ضروری است. با اعمال مدل روی تصاویر و کپشن‌ها، خروجی‌های پیش‌بینی را تولید می‌کنیم. خطای دسته‌های فعلی را با استفاده از تابع خطا محاسبه می‌کنیم. outputs و captions ابعاد مختلفی دارند، بنابراین آن‌ها را به ابعاد یکسان تغییر شکل می‌دهیم. در ادامه، گرادیان را برای همه پارامترها با استفاده از الگوریتم پس‌انتشار خطا محاسبه می‌کنیم.

بهینه‌ساز، گام‌های بهینه‌سازی را برای به‌روزرسانی پارامترها براساس محاسبات انجام‌شده در مرحله قبل انجام می‌دهد. خطا برای دستهٔ فعلی به running_loss اضافه می‌شود تا در نهایت میانگین خطا برای هر دوره محاسبه شود. بنابراین، میانگین خطا برای هر دوره محاسبه می‌شود، با این توضیح که idx نشان‌دهندهٔ شمارهٔ دستهٔ فعلی در داده‌های آموزش است. در نهایت مقادیر شمارهٔ دوره، خطا و... چاپ می‌شوند. میانگین خطای آموزش را به لیست train_loss اضافه می‌کنیم تا بعداً برای رسم نمودار استفاده شود. مدل را پس از هر دوره در فایلی با نام Model.pth ذخیره می‌کنیم و نمودارهای مورد نیاز را نمایش می‌دهیم. مجموعهٔ این دستورات در زیر آورده شده‌اند:

```
# Set the number of training epochs
num_epochs = 100

# Create an empty list to store the training loss
train_loss = []

# Training loop
for epoch in range(num_epochs):
    running_loss = 0

    # Iterate over the training data loader
    for idx, (image, captions) in enumerate(iter(train_loader)):
        # Move the image and captions to the specified device
        image, captions = image.to(device), captions.to(device)

        # Set the model to train mode
        model.train()

        # Zero the gradients in the optimizer
        optimizer.zero_grad()

        # Feed forward
        outputs = model(image, captions)

        # Calculate the batch loss
        loss = criterion(outputs.view(-1, vocab_size), captions.view(-1))

        # Backward pass
        loss.backward()

        # Update the parameters in the optimizer
        optimizer.step()

        # Accumulate the running loss
        running_loss += loss.item()

    # Calculate the average loss for the epoch
    average_loss = running_loss / (idx + 1)
```

```
    # Print the epoch number and the average loss
    print(f'Epoch: {epoch+1} - Train Loss: {average_loss}')

    # Append the average loss to the train_loss list
    train_loss.append(average_loss)

    # Save the model after each epoch
    torch.save(model.cpu().state_dict(), 'Model.pth')

    # Move the model back to the specified device
    model.cuda()

import matplotlib.pyplot as plt

# Set the figure size and dpi for better quality
plt.figure(figsize=(8, 6), dpi=80)

# Plot the training loss
plt.plot(train_loss, label='Training Loss')

# Set the plot title and axis labels
plt.title("Training Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")

# Customize the grid and ticks
plt.grid(True, linestyle='--', linewidth=0.5)
plt.xticks(range(len(train_loss)))
plt.yticks()

# Add a legend
plt.legend()

# Save the plot as a PDF file
plt.savefig("loss_plot.pdf", format='pdf', bbox_inches='tight')

# Show the plot
plt.show()
```

حال برای قسمت ارزیابی مطابق توضیحات داده‌شده در قسمت آموزش، دیتالودر قسمت تست را هم به همان ترتیب تشکیل می‌دهیم و سپس مدل را روی تعدادی از دادگان تست می‌کنیم و نتیجه را به‌صورتی مناسب نمایش می‌دهیم. دستورات را به صورتی نوشته‌ایم که یک تصویر تصادفی از مجموعهٔ ارزیابی انتخاب کند و کپشن‌های پیش‌بینی‌شده و واقعی را روی تصویر و در جعبه‌هایی با رنگ‌های متفاوت نشان دهد. دستورات به شرح زیر است:

```
import matplotlib.pyplot as plt
```

```
2
3  def show_image_with_captions(image, predicted_caption, real_caption):
4      """Display an image with predicted and real captions."""
5
6      # Denormalize the image tensor
7      image[0] = image[0] * 0.229
8      image[1] = image[1] * 0.224
9      image[2] = image[2] * 0.225
10     image[0] += 0.485
11     image[1] += 0.456
12     image[2] += 0.406
13
14     # Convert the image tensor to a numpy array and transpose the dimensions
15     image = image.numpy().transpose((1, 2, 0))
16
17     # Display the image
18     plt.imshow(image)
19
20     # Add predicted caption box
21     plt.text(
22         0, -20, predicted_caption, color='white', backgroundcolor='blue',
23         fontsize=12, verticalalignment='top', bbox=dict(facecolor='blue', alpha=0.8, edgecolor='
       white', pad=5)
24     )
25
26     # Add real caption box
27     plt.text(
28         0, -2, real_caption, color='black', backgroundcolor='green',
29         fontsize=12, verticalalignment='top', bbox=dict(facecolor='green', alpha=0.8, edgecolor='
       white', pad=5)
30     )
31
32     plt.axis('off')
33     plt.tight_layout()
34
35 # Define the test data transformations
36 test_transforms = T.Compose([
37     T.Resize((224, 224)),  # Resize the images to the specified size
38     T.ToTensor(),  # Convert the images to tensors
39     T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))  # Normalize the image tensors
40 ])
41
42 # Create the test dataset
43 test_dataset = FlickrDataset(
44     root_dir="Images",  # Root directory of the dataset
```

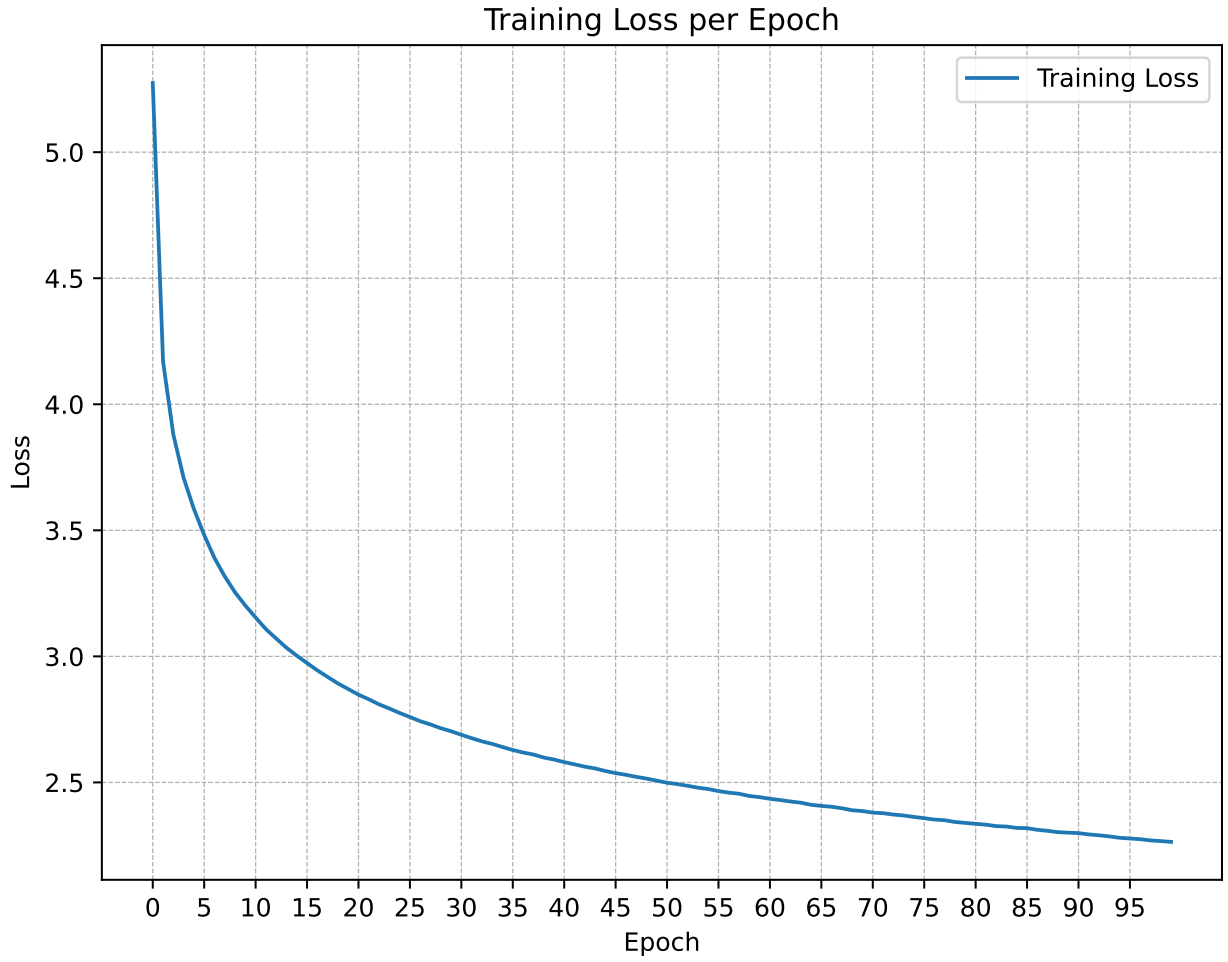
```
    caption_file="captions.txt",  # File containing the captions
    transform=test_transforms,  # Apply the specified transformations to the images
    frequency_threshold=1,  # Frequency threshold for filtering captions
    data_type='test'  # Specify the type of data (in this case, 'test')
)

# Create the test data loader
test_loader = DataLoader(
    dataset=test_dataset,  # Use the created test dataset
    batch_size=BATCH_SIZE,  # Number of samples per batch
    num_workers=NUM_WORKERS,  # Number of worker threads for data loading
    shuffle=True,  # Shuffle the data for each epoch
    collate_fn=Apppadd(pad_idx=pad_idx, batch_first=True)  # Function to collate and preprocess
    the data
)

# Get a batch of images and captions from the test loader
images, captions = next(iter(test_loader))

# Iterate over a single image and its captions
for i in range(1):
    # Set the model to evaluation mode
    model.eval()

    # Get a single test image and create a clone
    test_image = torch.clone(images)[i].unsqueeze(0)

    # Disable gradient calculation during inference
    with torch.no_grad():
        # Encode the test image using the model's encoder
        features = model.encoder(test_image[0:1].to(device))

        # Generate captions for the test image using the model's decoder
        predicted_captions = model.decoder.generate_caption(features.unsqueeze(0), vocab=
    train_dataset.vocab)

        # Get the ground truth caption for the test image
        cap = torch.clone(captions)[i]
        caption_label = [test_dataset.vocab.itos[token] for token in cap.tolist()]

        # Find the index of the end-of-sequence token '<EOS>' in the caption
        eos_index = caption_label.index('<EOS>')

        # Extract the caption tokens from the start to the '<EOS>' token
        caption_label = caption_label[1:eos_index]
```

```
88
89          # Convert the caption tokens to a string
90          caption_label = ' '.join(caption_label)
91
92          # Create the predicted caption string with proper formatting
93          predicted_caption = "Predicted Caption: " + ' '.join(predicted_captions[1:len(
    predicted_captions)-1])
94
95          # Create the actual caption string with proper formatting
96          real_caption = "Actual Caption: " + caption_label
97
98          # Display the image with the predicted and real captions
99          show_image_with_captions(test_image[0], predicted_caption, real_caption)
100
101 # Save the plot as a PDF file
102 plt.savefig("testimagecaptions.pdf", format='pdf', bbox_inches='tight')
```

با اجرای این فرآیند و دستورات نمودار خطا به صورتی است که در شکل ۳ نمایش داده شده و نتایج سه نمونهٔ تصادفی از مجموعهٔ ارزیابی به صورتی است که در شکل ۴ تا شکل ۸ نمایش داده شده است. همان‌طور که از نتایج مشخص است مدل هم‌چنان جا برای آموزش دارد اما به‌دلیل کمبود زمان (تمرین‌ها را تک‌نفره انجام می‌دهم) از ادامهٔ آموزش صرف‌نظر شده است. نتایج کپشن‌های پیش‌بینی‌شده روی داده‌های ارزیابی هم نشان می‌دهد که آموزش مدل قابل قبول بوده است و کپشن‌ها نزدیکی بالایی با آن‌چه در تصویر جریان دارد دارند. هرچند بعضی‌جاها لغزش‌های بسیار کوچکی هم دیده می‌شود.

```
Epoch: 1 - Train Loss: 5.2742267722016445
Epoch: 2 - Train Loss: 4.169356649572199
Epoch: 3 - Train Loss: 3.8797206595227434
Epoch: 4 - Train Loss: 3.7080819373364213
Epoch: 5 - Train Loss: 3.584198301488703
Epoch: 6 - Train Loss: 3.481250404477953
Epoch: 7 - Train Loss: 3.3901297129117527
Epoch: 8 - Train Loss: 3.3170354816463443
Epoch: 9 - Train Loss: 3.254032086659145
Epoch: 10 - Train Loss: 3.2015588150157797
.
.
.
Epoch: 90 - Train Loss: 2.3006849122214152
Epoch: 91 - Train Loss: 2.2989422974886593
Epoch: 92 - Train Loss: 2.293771370307549
Epoch: 93 - Train Loss: 2.2902511366597422
Epoch: 94 - Train Loss: 2.2857611779566414
Epoch: 95 - Train Loss: 2.2802960872650146
Epoch: 96 - Train Loss: 2.277562533225213
Epoch: 97 - Train Loss: 2.274853866417091
Epoch: 98 - Train Loss: 2.269936553248159
```

```
Epoch: 99 - Train Loss: 2.267233173330347
Epoch: 100 - Train Loss: 2.26415508443659
```



شکل ۳: نمودار خطا در حالت رزنت فریز.

برای نمایش خطای تست، از آن‌جا که دادن داده‌های تست در هنگام آموزش به نوعی تقلب محسوب می‌شود، دستورات زیر را نوشتیم تا بخشی از داده‌های آموزش را به‌عنوان دادهٔ اعتبارسنجی درنظر بگیریم و بتوانیم خطای تست را گزارش و رسم کنیم.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as T
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Set the batch size, number of workers, and pad index
BATCH_SIZE = 256
```

شکل ۴: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت فریز – نمونهٔ اول.

شکل ۵: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت فریز – نمونهٔ دوم.

```
11 NUM_WORKERS = 10
12 pad_idx = train_dataset.vocab.stoi["<PAD>"]
```

شکل ۶: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت فریز – نمونهٔ سوم.

شکل ۷: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت فریز – نمونهٔ چهارم.

```
13
14 # Define the transformations to be applied, including resizing, random cropping,
15 # converting to tensor, and normalization using ResNet statistics
16 transforms = T.Compose([
```

شکل ۸: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت فریز - نمونهٔ پنجم.

```
    T.Resize(256),
    T.RandomCrop(224),
    T.ToTensor(),
    T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

# Create an instance of the FlickrDataset for training
train_dataset = FlickrDataset(
    root_dir="/content/Dataset/Images",
    caption_file="captions.txt",
    transform=transforms,
    data_type='train'
)

# Split the dataset into training and validation sets
train_dataset, val_dataset = train_test_split(train_dataset, test_size=0.1, random_state=42)

# Create a DataLoader for the training dataset
train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    shuffle=True,
```

```
    collate_fn=Apppadd(pad_idx=pad_idx, batch_first=True)
)

# Create a DataLoader for the validation dataset
val_loader = DataLoader(
    dataset=val_dataset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    shuffle=False,
    collate_fn=Apppadd(pad_idx=pad_idx, batch_first=True)
)

# Set the number of training epochs
num_epochs = 20

# Create empty lists to store the training and validation loss
train_loss = []
val_loss = []

# Training loop
for epoch in range(num_epochs):
    running_train_loss = 0
    running_val_loss = 0

    # Training
    model.train()
    for idx, (image, captions) in enumerate(train_loader):
        image, captions = image.to(device), captions.to(device)
        optimizer.zero_grad()
        outputs = model(image, captions)
        loss = criterion(outputs.view(-1, vocab_size), captions.view(-1))
        loss.backward()
        optimizer.step()
        running_train_loss += loss.item()
    average_train_loss = running_train_loss / len(train_loader)
    train_loss.append(average_train_loss)

    # Validation
    model.eval()
    with torch.no_grad():
        for idx, (image, captions) in enumerate(val_loader):
            image, captions = image.to(device), captions.to(device)
            outputs = model(image, captions)
            loss = criterion(outputs.view(-1, vocab_size), captions.view(-1))
            running_val_loss += loss.item()
```

```
    average_val_loss = running_val_loss / len(val_loader)
    val_loss.append(average_val_loss)

    print(f'Epoch: {epoch+1} - Train Loss: {average_train_loss:.4f} - Val Loss: {average_val_loss:.4f}')

    # Save the model after each epoch
    torch.save(model.cpu().state_dict(), 'ModelwithVal.pth')
    model.cuda()

# Plotting the training and validation loss
plt.figure(figsize=(8, 6), dpi=80)
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title("Training and Validation Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True, linestyle='--', linewidth=0.5)
plt.xticks(range(num_epochs))
plt.legend()
plt.savefig("loss_plot.pdf", format='pdf', bbox_inches='tight')
plt.show()
```

نتایج به صورتی است که در شکل ۱۶ آمده است. این‌باز برای عدم تکرار فرآیند قبل و صرفاً برای نمایش نتایج تنها برای ۲۰ دوره آموزش دادیم. نتیجه روی سه دادهٔ تست تصادفی هم در مورد ارزیابی قرار گرفته که در دفترچه‌کد گوگل‌کولب قابل مشاهده است و این‌جا برای جلوگیری از ازدحام بیش‌تر نیاورده‌ایم.

```
Epoch: 1 - Train Loss: 5.3937 - Val Loss: 4.4301
Epoch: 2 - Train Loss: 4.2150 - Val Loss: 4.0397
Epoch: 3 - Train Loss: 3.9220 - Val Loss: 3.8049
Epoch: 4 - Train Loss: 3.7434 - Val Loss: 3.6553
Epoch: 5 - Train Loss: 3.6204 - Val Loss: 3.5436
Epoch: 6 - Train Loss: 3.5186 - Val Loss: 3.4460
Epoch: 7 - Train Loss: 3.4264 - Val Loss: 3.3641
Epoch: 8 - Train Loss: 3.3506 - Val Loss: 3.2957
Epoch: 9 - Train Loss: 3.2884 - Val Loss: 3.2414
Epoch: 10 - Train Loss: 3.2355 - Val Loss: 3.1959
Epoch: 11 - Train Loss: 3.1880 - Val Loss: 3.1525
Epoch: 12 - Train Loss: 3.1436 - Val Loss: 3.1122
Epoch: 13 - Train Loss: 3.1050 - Val Loss: 3.0762
Epoch: 14 - Train Loss: 3.0685 - Val Loss: 3.0463
Epoch: 15 - Train Loss: 3.0353 - Val Loss: 3.0153
Epoch: 16 - Train Loss: 3.0030 - Val Loss: 2.9872
Epoch: 17 - Train Loss: 2.9733 - Val Loss: 2.9627
Epoch: 18 - Train Loss: 2.9463 - Val Loss: 2.9385
Epoch: 19 - Train Loss: 2.9210 - Val Loss: 2.9180
```

```
Epoch: 20 - Train Loss: 2.8963 - Val Loss: 2.8987
```

شکل ۹: نمودار خطای آموزش و تست در حالت رزنت فریز.

۲.۱ پاسخ قسمت ۲

برای پاسخ به این قسمت از تکرار مجدد موارد گفته‌شده در پاسخ قسمت ۱ صرف‌نظر می‌کنیم. تنها قسمتی که نیاز به تغییر دارد قسمت تعریف مدل است. در قسمت قبل قابلیت آموزش لایه‌های رزنت را خاموش نگه داشته بودیم و در این قسمت، مطابق با خواستهٔ سوال آن را فعال می‌کنیم. دستور تغییریافته و مجموعهٔ دستورات تغییریافتهٔ مربوط به این بخش در زیر آورده شده است (`train_resnet=True`):

```
#ResNet Model
class ResNet(nn.Module):
    def __init__(self, embed_size, train_resnet=False):
        """
        Initialize the ResNet model with a specified embedding size.
```

```
        Args:
            embed_size (int): Size of the embedding output.
            train_resnet (bool): Whether to train the ResNet backbone or not.
        """
        super(ResNet, self).__init__()

        # Load the pretrained ResNet-18 model
        resnet = models.resnet18(pretrained=True)

        # Set the requires_grad flag of the ResNet parameters
        # to control whether they are trainable or not
        for param in resnet.parameters():
            param.requires_grad_(train_resnet)

        # Extract the modules of the ResNet model up to the last fully connected layer
        modules = list(resnet.children())[:-1]

        # Create the ResNet backbone with the extracted modules
        self.resnet = nn.Sequential(*modules)

        # Create the embedding layer
        self.embed = nn.Linear(resnet.fc.in_features, embed_size)

        # Activation function and dropout layer
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)

    def forward(self, images):
        """
        Forward pass of the ResNet model.

        Args:
            images (tensor): Input images tensor.

        Returns:
            features (tensor): Embedded features tensor.
        """
        features = self.dropout(self.relu(self.resnet(images)))
        features = features.view(features.size(0), -1)
        features = self.embed(features)
        return features

# LSTM Network
class LSTM(nn.Module):
```

```
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1, drop_prob=0.5):
        """
        Initialize the LSTM model with specified sizes and parameters.

        Args:
            embed_size (int): Size of the word embedding.
            hidden_size (int): Size of the hidden state of the LSTM.
            vocab_size (int): Size of the vocabulary.
            num_layers (int): Number of layers in the LSTM (default: 1).
            drop_prob (float): Dropout probability (default: 0.5).
        """
        super().__init__()

        # Word embedding layer
        self.embedding = nn.Embedding(vocab_size, embed_size)

        # LSTM layer
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers=num_layers, batch_first=True)

        # Linear layer for prediction
        self.linear = nn.Linear(hidden_size, vocab_size)

        # Dropout layer
        self.dropout = nn.Dropout(drop_prob)

    def forward(self, features, captions):
        """
        Forward pass of the LSTM model.

        Args:
            features (tensor): Image features tensor.
            captions (tensor): Captions tensor.

        Returns:
            x (tensor): Output tensor.
        """
        # Vectorize the caption by passing it through the embedding layer
        embeds = self.dropout(self.embedding(captions[:, :-1]))

        # Concatenate the features and captions
        x = torch.cat((features.unsqueeze(1), embeds), dim=1)

        # Pass through the LSTM layer
        x, _ = self.lstm(x)

```

```
        # Pass through the linear layer for prediction
        x = self.linear(x)

        return x

    def generate_caption(self, inputs, hidden=None, max_len=20, vocab=None):
        """
        Generate captions given the image features.

        Args:
            inputs (tensor): Input tensor of image features.
            hidden (tuple): Hidden state of the LSTM (default: None).
            max_len (int): Maximum length of the generated caption (default: 20).
            vocab (Vocab): Vocabulary object (default: None).

        Returns:
            caption (list): Generated caption as a list of words.
        """
        # Inference part
        # Given the image features, generate the captions

        batch_size = inputs.size(0)

        captions = []

        for i in range(max_len):
            output, hidden = self.lstm(inputs, hidden)
            output = self.linear(output)
            output = output.view(batch_size, -1)

            # Select the word with the highest value
            predicted_word_idx = output.argmax(dim=1)

            # Save the generated word
            captions.append(predicted_word_idx.item())

            # End if <EOS> is detected
            if vocab.itos[predicted_word_idx.item()] == "<EOS>":
                break

            # Send the generated word as the next caption
            inputs = self.embedding(predicted_word_idx.unsqueeze(0))

        # Convert the vocabulary indices to words and return the sentence
        return [vocab.itos[idx] for idx in captions]
```

```python

# Utilizing the powerful fusion of ResNet and LSTM for image captioning
class CNNtoRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1, drop_prob=0.5, train_resnet=False):
        """
        Initialize the CNNtoRNN model with specified sizes and parameters.

        Args:
            embed_size (int): Size of the word embedding.
            hidden_size (int): Size of the hidden state of the LSTM.
            vocab_size (int): Size of the vocabulary.
            num_layers (int): Number of layers in the LSTM (default: 1).
            drop_prob (float): Dropout probability (default: 0.5).
            train_resnet (bool): Whether to train the ResNet backbone or not (default: False).
        """
        super().__init__()

        # Encoder (ResNet)
        self.encoder = ResNet(embed_size, train_resnet)

        # Decoder (LSTM)
        self.decoder = LSTM(embed_size, hidden_size, vocab_size, num_layers, drop_prob)

    def forward(self, images, captions):
        """
        Forward pass of the CNNtoRNN model.

        Args:
            images (tensor): Input images tensor.
            captions (tensor): Captions tensor.

        Returns:
            outputs (tensor): Output tensor.
        """
        # Pass the images through the encoder (ResNet) to get features
        features = self.encoder(images)

        # Pass the features and captions through the decoder (LSTM) to get outputs
        outputs = self.decoder(features, captions)

        return outputs
.
.
.
```

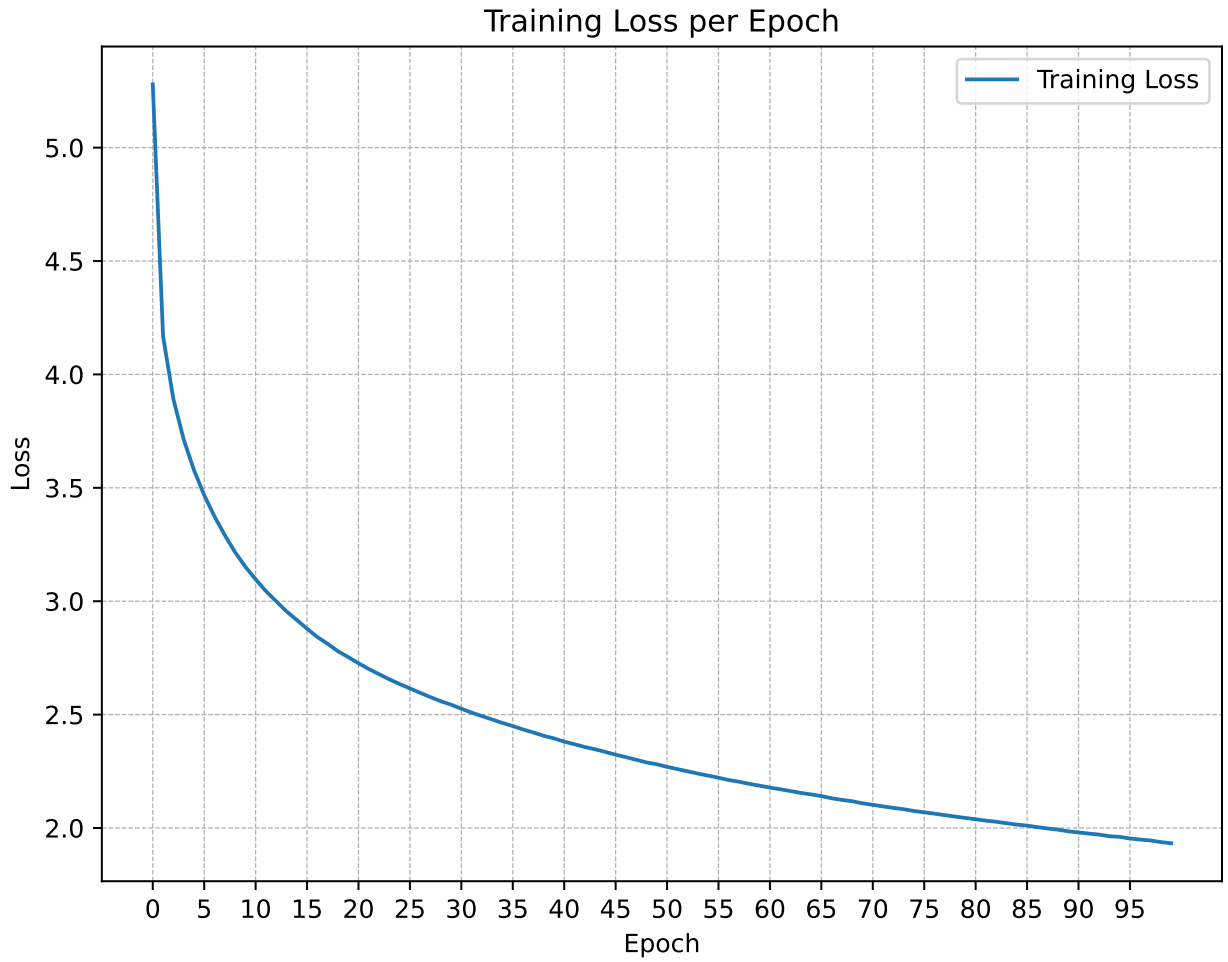
```
# Initialize the model, loss function, and optimizer
NoFreeze_model = CNNtoRNN(embed_size, hidden_size, vocab_size, num_layers, train_resnet=True).to(
    device)
criterion = nn.CrossEntropyLoss(ignore_index=train_dataset.vocab.stoi["<PAD>"])
optimizer = optim.Adam(NoFreeze_model.parameters(), lr=learning_rate)
```

با اجرای این فرآیند و دستورات نمودار خطا به صورتی است که در شکل ۱۰ نمایش داده شده و نتایج سه نمونهٔ تصادفی از مجموعهٔ ارزیابی به صورتی است که در شکل ۱۱ تا شکل ۱۵ نمایش داده شده است. همان‌طور که از نتایج مشخص است مدل هم‌چنان جا برای آموزش دارد اما به‌دلیل کمبود زمان (تمرین‌ها را تک‌نفره انجام می‌دهم) از ادامهٔ آموزش صرف‌نظر شده است. می‌توان از `torch.cuda.amp.GradScaler()` هم برای افزایش سرعت و بهره‌وری در فرآیند آموزش استفاده کرد. به هر حال، نتایج کپشن‌های پیش‌بینی‌شده روی داده‌های ارزیابی هم نشان می‌دهد که آموزش مدل قابل قبول بوده است و کپشن‌ها نزدیکی بالایی با آن‌چه در تصویر جریان دارد دارند. هرچند بعضی‌جاها لغزش‌های بسیار کوچکی هم دیده می‌شود.

```
Epoch: 1 - Train Loss: 5.2784986095828605
Epoch: 2 - Train Loss: 4.1686154712330215
Epoch: 3 - Train Loss: 3.892264472854721
Epoch: 4 - Train Loss: 3.71403486245162
Epoch: 5 - Train Loss: 3.5789838320725447
Epoch: 6 - Train Loss: 3.4679061566199456
Epoch: 7 - Train Loss: 3.3738862034324164
Epoch: 8 - Train Loss: 3.2915008068084717
Epoch: 9 - Train Loss: 3.2161747125478892
Epoch: 10 - Train Loss: 3.152081139437802
.
.
.
Epoch: 90 - Train Loss: 1.985385377090294
Epoch: 91 - Train Loss: 1.9803814221095373
Epoch: 92 - Train Loss: 1.9754978493377044
Epoch: 93 - Train Loss: 1.9709438619080124
Epoch: 94 - Train Loss: 1.9638588253434721
Epoch: 95 - Train Loss: 1.961058676659644
Epoch: 96 - Train Loss: 1.9537817581550225
Epoch: 97 - Train Loss: 1.9489550223717322
Epoch: 98 - Train Loss: 1.9450463039891703
Epoch: 99 - Train Loss: 1.937939615516396
Epoch: 100 - Train Loss: 1.9327541773135846
```

برای نمایش خطای تست، از آن‌جا که دادن داده‌های تست در هنگام آموزش به نوعی تقلب محسوب می‌شود، دستورات زیر را نوشتیم تا بخشی از داده‌های آموزش را به‌عنوان دادهٔ اعتبارسنجی درنظر بگیریم و بتوانیم خطای تست را گزارش و رسم کنیم.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as T
from torch.utils.data import DataLoader
```



شکل ۱۰: نمودار خطا در حالت رزنت بدون فریز.

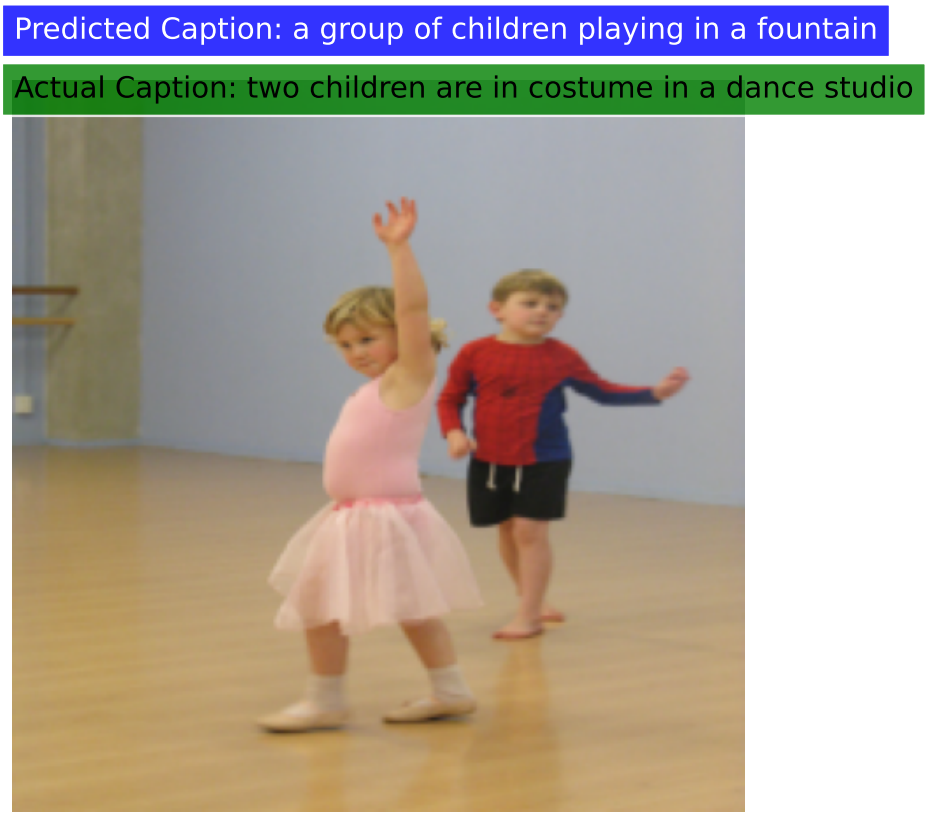
```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Set the batch size, number of workers, and pad index
BATCH_SIZE = 256
NUM_WORKERS = 10
pad_idx = train_dataset.vocab.stoi["<PAD>"]

# Define the transformations to be applied, including resizing, random cropping,
# converting to tensor, and normalization using ResNet statistics
transforms = T.Compose([
    T.Resize(256),
    T.RandomCrop(224),
    T.ToTensor(),
    T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])
```

شکل ۱۱: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت بدون فریز – نمونهٔ اول.

شکل ۱۲: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت بدون فریز – نمونهٔ دوم.

```
22
23 # Create an instance of the FlickrDataset for training
24 train_dataset = FlickrDataset(
25     root_dir="/content/Dataset/Images",
26     caption_file="captions.txt",
27     transform=transforms,
28     data_type='train'
```

شکل ۱۳: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت بدون فریز – نمونهٔ سوم.



شکل ۱۴: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت بدون فریز – نمونهٔ چهارم.

```
)

# Split the dataset into training and validation sets
```

شکل ۱۵: نتایج پیش‌بینی روی داده‌های ارزیابی در حالت رزنت بدون فریز - نمونهٔ پنجم.

```
train_dataset, val_dataset = train_test_split(train_dataset, test_size=0.1, random_state=42)

# Create a DataLoader for the training dataset
train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    shuffle=True,
    collate_fn=Apppadd(pad_idx=pad_idx, batch_first=True)
)

# Create a DataLoader for the validation dataset
val_loader = DataLoader(
    dataset=val_dataset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    shuffle=False,
    collate_fn=Apppadd(pad_idx=pad_idx, batch_first=True)
)

# Set the number of training epochs
```

```
num_epochs = 20

# Create empty lists to store the training and validation loss
train_loss = []
val_loss = []

# Training loop
for epoch in range(num_epochs):
    running_train_loss = 0
    running_val_loss = 0

    # Training
    model.train()
    for idx, (image, captions) in enumerate(train_loader):
        image, captions = image.to(device), captions.to(device)
        optimizer.zero_grad()
        outputs = model(image, captions)
        loss = criterion(outputs.view(-1, vocab_size), captions.view(-1))
        loss.backward()
        optimizer.step()
        running_train_loss += loss.item()
    average_train_loss = running_train_loss / len(train_loader)
    train_loss.append(average_train_loss)

    # Validation
    model.eval()
    with torch.no_grad():
        for idx, (image, captions) in enumerate(val_loader):
            image, captions = image.to(device), captions.to(device)
            outputs = model(image, captions)
            loss = criterion(outputs.view(-1, vocab_size), captions.view(-1))
            running_val_loss += loss.item()
    average_val_loss = running_val_loss / len(val_loader)
    val_loss.append(average_val_loss)

    print(f'Epoch: {epoch+1} - Train Loss: {average_train_loss:.4f} - Val Loss: {average_val_loss
    :.4f}')

    # Save the model after each epoch
    torch.save(model.cpu().state_dict(), 'ModelwithVal.pth')
    model.cuda()

# Plotting the training and validation loss
plt.figure(figsize=(8, 6), dpi=80)
plt.plot(train_loss, label='Training Loss')
```

```
plt.plot(val_loss, label='Validation Loss')
plt.title("Training and Validation Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True, linestyle='--', linewidth=0.5)
plt.xticks(range(num_epochs))
plt.legend()
plt.savefig("loss_plot.pdf", format='pdf', bbox_inches='tight')
plt.show()
```

نتایج به صورتی است که در شکل ۱۶ آمده است. این‌باز برای عدم تکرار فرآیند قبل و صرفاً برای نمایش نتایج تنها برای ۲۰ دوره آموزش دادیم. نتیجه روی سه دادهٔ تست تصادفی هم در مورد ارزیابی قرار گرفته که در دفترچه‌کد گوگل‌کولب قابل مشاهده است و این‌جا برای جلوگیری از ازدحام بیش‌تر نیاورده‌ایم.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as T
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Set the batch size, number of workers, and pad index
BATCH_SIZE = 256
NUM_WORKERS = 10
pad_idx = train_dataset.vocab.stoi["<PAD>"]

# Define the transformations to be applied, including resizing, random cropping,
# converting to tensor, and normalization using ResNet statistics
transforms = T.Compose([
    T.Resize(256),
    T.RandomCrop(224),
    T.ToTensor(),
    T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

# Create an instance of the FlickrDataset for training
train_dataset = FlickrDataset(
    root_dir="/content/Dataset/Images",
    caption_file="captions.txt",
    transform=transforms,
    data_type='train'
)

# Split the dataset into training and validation sets
train_dataset, val_dataset = train_test_split(train_dataset, test_size=0.1, random_state=42)
```

```
33
34 # Create a DataLoader for the training dataset
35 train_loader = DataLoader(
36     dataset=train_dataset,
37     batch_size=BATCH_SIZE,
38     num_workers=NUM_WORKERS,
39     shuffle=True,
40     collate_fn=Apppadd(pad_idx=pad_idx, batch_first=True)
41 )
42
43 # Create a DataLoader for the validation dataset
44 val_loader = DataLoader(
45     dataset=val_dataset,
46     batch_size=BATCH_SIZE,
47     num_workers=NUM_WORKERS,
48     shuffle=False,
49     collate_fn=Apppadd(pad_idx=pad_idx, batch_first=True)
50 )
51
52 # Set the number of training epochs
53 num_epochs = 20
54
55 # Create empty lists to store the training and validation loss
56 train_loss = []
57 val_loss = []
58
59 # Training loop
60 for epoch in range(num_epochs):
61     running_train_loss = 0
62     running_val_loss = 0
63
64     # Training
65     model.train()
66     for idx, (image, captions) in enumerate(train_loader):
67         image, captions = image.to(device), captions.to(device)
68         optimizer.zero_grad()
69         outputs = model(image, captions)
70         loss = criterion(outputs.view(-1, vocab_size), captions.view(-1))
71         loss.backward()
72         optimizer.step()
73         running_train_loss += loss.item()
74     average_train_loss = running_train_loss / len(train_loader)
75     train_loss.append(average_train_loss)
76
77     # Validation
```

```
    model.eval()
    with torch.no_grad():
        for idx, (image, captions) in enumerate(val_loader):
            image, captions = image.to(device), captions.to(device)
            outputs = model(image, captions)
            loss = criterion(outputs.view(-1, vocab_size), captions.view(-1))
            running_val_loss += loss.item()
    average_val_loss = running_val_loss / len(val_loader)
    val_loss.append(average_val_loss)

    print(f'Epoch: {epoch+1} - Train Loss: {average_train_loss:.4f} - Val Loss: {average_val_loss
    :.4f}')

    # Save the model after each epoch
    torch.save(model.cpu().state_dict(), 'ModelwithValNoFreeze.pth')
    model.cuda()

# Plotting the training and validation loss
plt.figure(figsize=(8, 6), dpi=80)
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Test Loss')
plt.title("Training and Testing Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True, linestyle='--', linewidth=0.5)
plt.xticks(range(num_epochs))
plt.legend()
plt.savefig("lossplotwithval2.pdf", format='pdf', bbox_inches='tight')
plt.show()
```

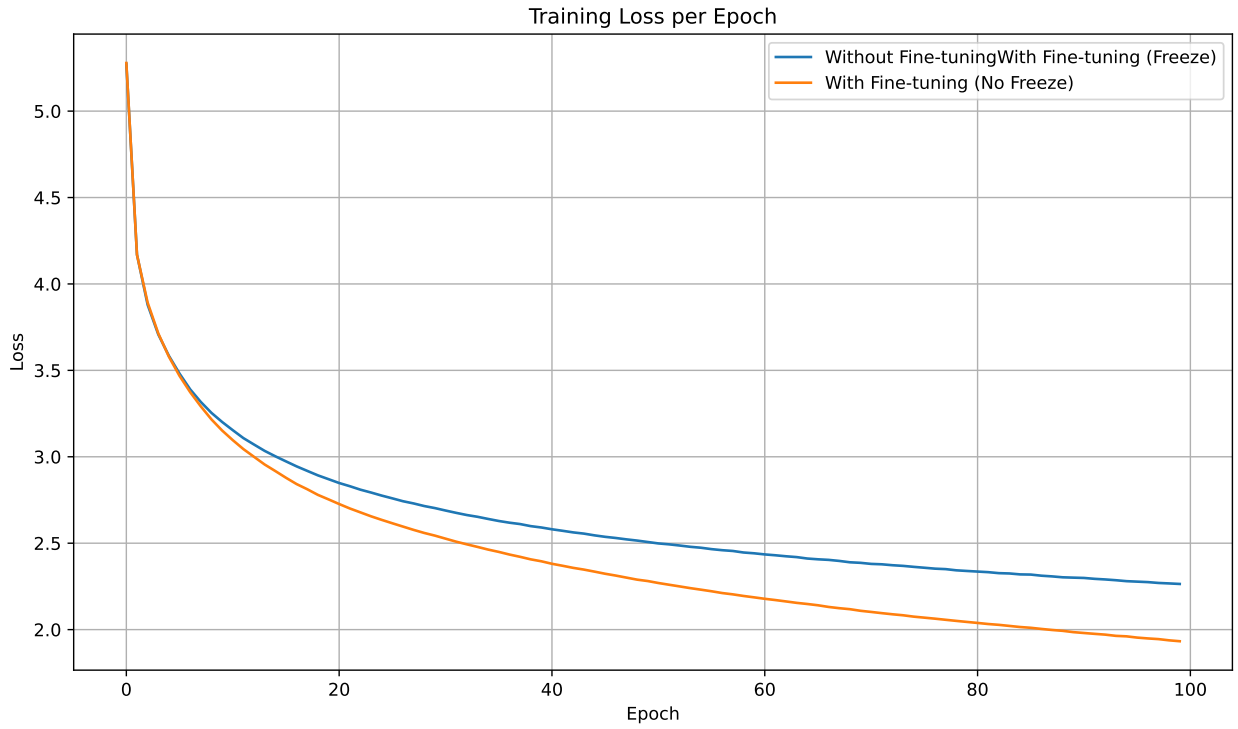
در ادامه از دستورات زیر برای مقایسهٔ قسمت‌های اول و دوم و نمایش نمودار خطا برای هر دوی آن‌ها استفاده می‌کنیم و نتایج به‌صورتی است که در شکل ۱۷ نشان داده شده است.

```
# Import the necessary libraries
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(10, 6))

# Set the plot title and axis labels
plt.title("Training Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")

# Plot the training loss curves
plt.plot(train_loss, label='Without Fine-tuning (Freeze)')
```

شکل ۱۶: نمودار خطای آموزش و تست در حالت رزنت بدون فریز.

```
plt.plot(NoFreeze_train_loss, label='With Fine-tuning (No Freeze)')

# Add a legend to the plot
plt.legend()

# Add grid lines to the plot
plt.grid(True)

# Adjust the layout for a more compact and professional appearance
plt.tight_layout()

# Add a legend
plt.legend()

# Save the plot as a PDF file
plt.savefig("comparelossplot2.pdf", format='pdf', bbox_inches='tight')
```

```

# Display the plot
plt.show()
```



شکل ۱۷: نمودار خطا در حالت رزنت فریز و بدون فریز.

به‌عنوان یک حرکت اضافی همان حالت قبلی (پاسخ قسمت ۲) را با استفاده از LSTM دوطرفه هم آزمودیم (bidirectional=True):

```
#ResNet Model
class ResNet(nn.Module):
    def __init__(self, embed_size, train_resnet=False):
        """
        Initialize the ResNet model with a specified embedding size.

        Args:
            embed_size (int): Size of the embedding output.
            train_resnet (bool): Whether to train the ResNet backbone or not.
        """
        super(ResNet, self).__init__()

        # Load the pretrained ResNet-18 model
        resnet = models.resnet18(pretrained=True)

        # Set the requires_grad flag of the ResNet parameters
        # to control whether they are trainable or not
```

```
        for param in resnet.parameters():
            param.requires_grad_(train_resnet)

        # Extract the modules of the ResNet model up to the last fully connected layer
        modules = list(resnet.children())[:-1]

        # Create the ResNet backbone with the extracted modules
        self.resnet = nn.Sequential(*modules)

        # Create the embedding layer
        self.embed = nn.Linear(resnet.fc.in_features, embed_size)

        # Activation function and dropout layer
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)

    def forward(self, images):
        """
        Forward pass of the ResNet model.

        Args:
            images (tensor): Input images tensor.

        Returns:
            features (tensor): Embedded features tensor.
        """
        features = self.dropout(self.relu(self.resnet(images)))
        features = features.view(features.size(0), -1)
        features = self.embed(features)
        return features

# LSTM Network
class LSTM(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1, drop_prob=0.5):
        """
        Initialize the LSTM model with specified sizes and parameters.

        Args:
            embed_size (int): Size of the word embedding.
            hidden_size (int): Size of the hidden state of the LSTM.
            vocab_size (int): Size of the vocabulary.
            num_layers (int): Number of layers in the LSTM (default: 1).
            drop_prob (float): Dropout probability (default: 0.5).
        """
        super().__init__()
```

```
63
64          # Word embedding layer
65          self.embedding = nn.Embedding(vocab_size, embed_size)
66
67          # LSTM layer
68          self.lstm = nn.LSTM(embed_size, hidden_size, num_layers=num_layers, batch_first=True,
    bidirectional = True)
69
70          # Linear layer for prediction
71          self.linear = nn.Linear(hidden_size*2, vocab_size)
72
73          # Dropout layer
74          self.dropout = nn.Dropout(drop_prob)
75
76      def forward(self, features, captions):
77          """
78          Forward pass of the LSTM model.
79
80          Args:
81              features (tensor): Image features tensor.
82              captions (tensor): Captions tensor.
83
84          Returns:
85              x (tensor): Output tensor.
86          """
87          # Vectorize the caption by passing it through the embedding layer
88          embeds = self.dropout(self.embedding(captions[:, :-1]))
89
90          # Concatenate the features and captions
91          x = torch.cat((features.unsqueeze(1), embeds), dim=1)
92
93          # Pass through the LSTM layer
94          x, _ = self.lstm(x)
95
96          # Pass through the linear layer for prediction
97          x = self.linear(x)
98
99          return x
100
101     def generate_caption(self, inputs, hidden=None, max_len=20, vocab=None):
102         """
103         Generate captions given the image features.
104
105         Args:
106             inputs (tensor): Input tensor of image features.
```

```
            hidden (tuple): Hidden state of the LSTM (default: None).
            max_len (int): Maximum length of the generated caption (default: 20).
            vocab (Vocab): Vocabulary object (default: None).

        Returns:
            caption (list): Generated caption as a list of words.
        """
        # Inference part
        # Given the image features, generate the captions

        batch_size = inputs.size(0)

        captions = []

        for i in range(max_len):
            output, hidden = self.lstm(inputs, hidden)
            output = self.linear(output)
            output = output.view(batch_size, -1)

            # Select the word with the highest value
            predicted_word_idx = output.argmax(dim=1)

            # Save the generated word
            captions.append(predicted_word_idx.item())

            # End if <EOS> is detected
            if vocab.itos[predicted_word_idx.item()] == "<EOS>":
                break

            # Send the generated word as the next caption
            inputs = self.embedding(predicted_word_idx.unsqueeze(0))

        # Convert the vocabulary indices to words and return the sentence
        return [vocab.itos[idx] for idx in captions]

# Utilizing the powerful fusion of ResNet and LSTM for image captioning
class CNNtoRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1, drop_prob=0.5,
    train_resnet=False):
        """
        Initialize the CNNtoRNN model with specified sizes and parameters.

        Args:
            embed_size (int): Size of the word embedding.
            hidden_size (int): Size of the hidden state of the LSTM.
```

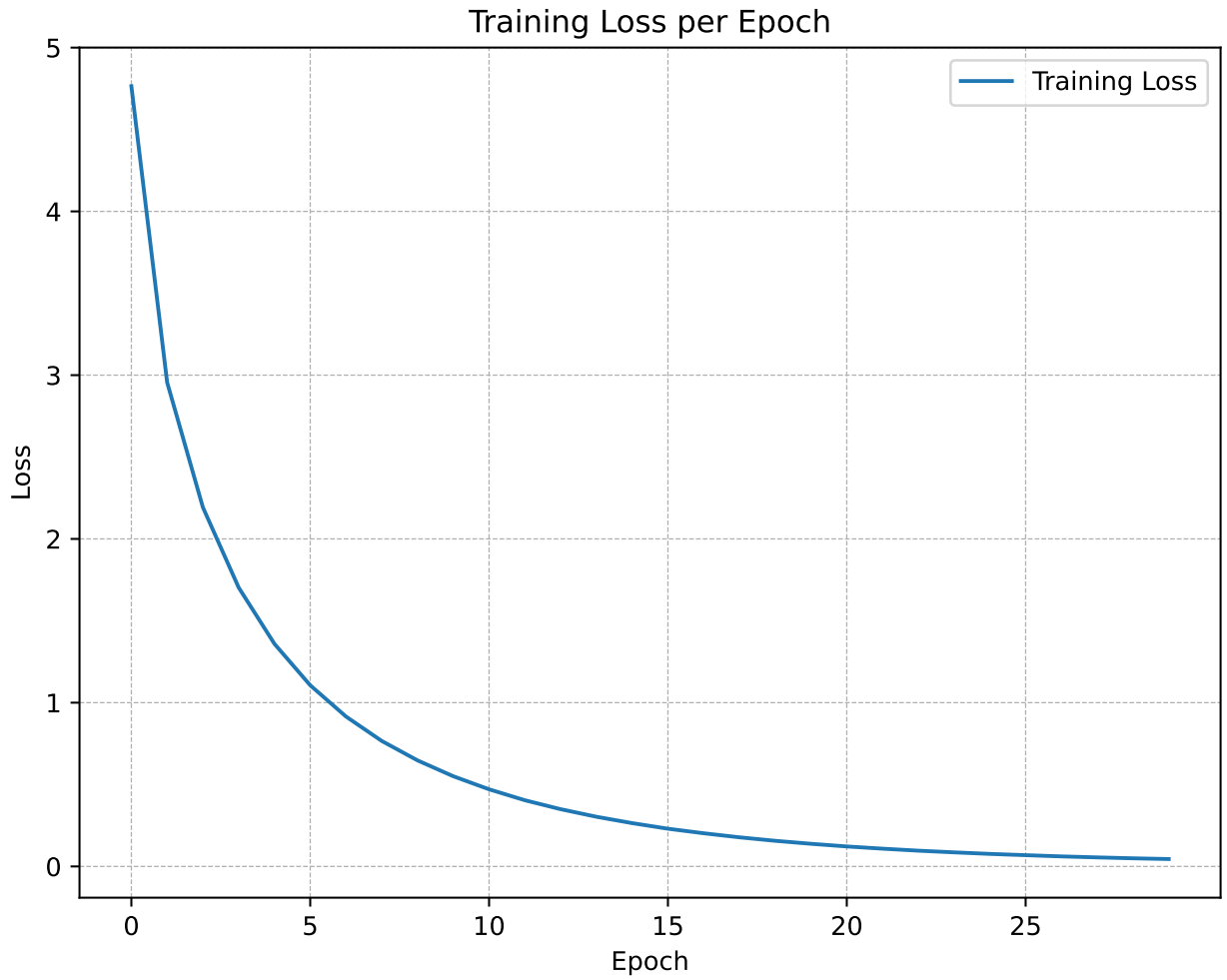
```
            vocab_size (int): Size of the vocabulary.
            num_layers (int): Number of layers in the LSTM (default: 1).
            drop_prob (float): Dropout probability (default: 0.5).
            train_resnet (bool): Whether to train the ResNet backbone or not (default: False).
        """
        super().__init__()

        # Encoder (ResNet)
        self.encoder = ResNet(embed_size, train_resnet)

        # Decoder (LSTM)
        self.decoder = LSTM(embed_size, hidden_size, vocab_size, num_layers, drop_prob)

    def forward(self, images, captions):
        """
        Forward pass of the CNNtoRNN model.

        Args:
            images (tensor): Input images tensor.
            captions (tensor): Captions tensor.

        Returns:
            outputs (tensor): Output tensor.
        """
        # Pass the images through the encoder (ResNet) to get features
        features = self.encoder(images)

        # Pass the features and captions through the decoder (LSTM) to get outputs
        outputs = self.decoder(features, captions)

        return outputs
.
.
.
# Initialize the model, loss function, and optimizer
NoFreeze_model_bi = CNNtoRNN(embed_size, hidden_size, vocab_size, num_layers, train_resnet=True).
    to(device)
criterion = nn.CrossEntropyLoss(ignore_index=train_dataset.vocab.stoi["<PAD>"])
optimizer = optim.Adam(NoFreeze_model_bi.parameters(), lr=learning_rate)
```

با اجرای این فرآیند و دستورات نمودار خطا به صورتی است که در شکل ۱۸ نمایش داده شده و همان‌طور که از نتایج مشخص است هرچند مدل هم‌چنان جا برای آموزش دارد؛ اما نتایج در تعداد دوره‌های کم‌تر به مراتب بهتر از حالات قبلی شده است. در این پیاده‌سازی از مقیاس‌کردن گرادیان‌ها یا `torch.cuda.amp.GradScaler()` هم استفاده شده است تا سرعت و بهره‌وری فرآیند آموزش بالاتر رود. نتایج کپشن‌های پیش‌بینی‌شده روی داده‌های ارزیابی اما نشان می‌دهد که هرچند نتایج آموزش مدل قابل قبول بوده است؛ اما نتیجهٔ ارزیابی قابل قبول نبوده و به نوعی اورفیت رخ داده است.

```
Epoch: 1 - Train Loss: 4.764396175637946
Epoch: 2 - Train Loss: 2.955980390935511
Epoch: 3 - Train Loss: 2.192415646739773
Epoch: 4 - Train Loss: 1.7028564723221573
Epoch: 5 - Train Loss: 1.3591378785513497
Epoch: 6 - Train Loss: 1.106173901707976
Epoch: 7 - Train Loss: 0.9153213192532946
Epoch: 8 - Train Loss: 0.7661450617796891
Epoch: 9 - Train Loss: 0.6474878558745751
Epoch: 10 - Train Loss: 0.5507056039946896
.
.
.
Epoch: 20 - Train Loss: 0.1382495445700792
Epoch: 21 - Train Loss: 0.12208282926699499
Epoch: 22 - Train Loss: 0.10844990913267735
Epoch: 23 - Train Loss: 0.09631544533934626
Epoch: 24 - Train Loss: 0.08601176170827625
Epoch: 25 - Train Loss: 0.07654622593126097
Epoch: 26 - Train Loss: 0.06880965792424196
Epoch: 27 - Train Loss: 0.06132213951095001
Epoch: 28 - Train Loss: 0.05519910268746056
Epoch: 29 - Train Loss: 0.04966954329422304
Epoch: 30 - Train Loss: 0.045182802746971173
```

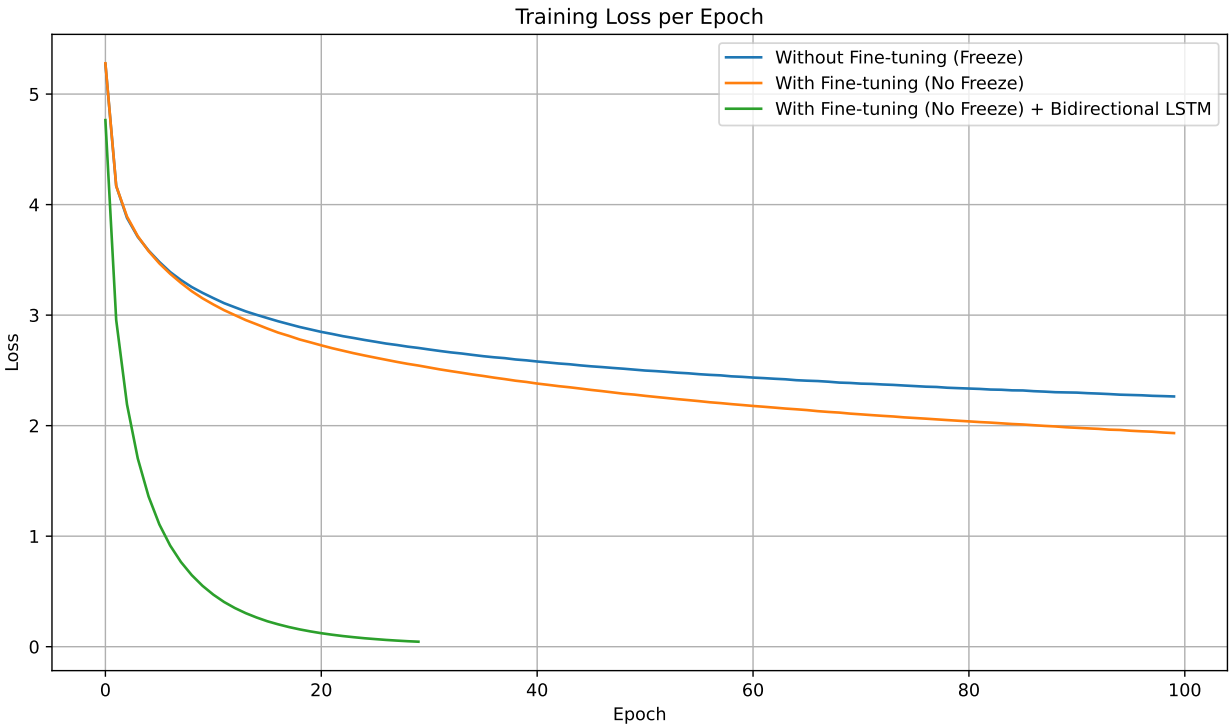
در ادامه از دستورات زیر برای مقایسهٔ قسمت‌های اول و دوم و این قسمت و نمایش نمودار خطا برای هر سه‌ی این‌ها استفاده می‌کنیم و نتایج به‌صورتی است که در شکل ۱۹ نشان داده شده است.

```python
# Import the necessary libraries
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(10, 6))

# Set the plot title and axis labels
plt.title("Training Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")

# Plot the training loss curves
plt.plot(train_loss, label='Without Fine-tuning (Freeze)')
plt.plot(NoFreeze_train_loss, label='With Fine-tuning (No Freeze)')
plt.plot(NoFreeze_train_loss_bi, label='With Fine-tuning (No Freeze) + Bidirectional LSTM')

# Add a legend to the plot
plt.legend()
```



شکل ۱۸: نمودار خطا در حالت رزنت بدون فریز (Bidirectional LSTM).

```
19
20 # Add grid lines to the plot
21 plt.grid(True)
22
23 # Adjust the layout for a more compact and professional appearance
24 plt.tight_layout()
25
26 # Add a legend
27 plt.legend()
28
29 # Save the plot as a PDF file
30 plt.savefig("comparelossplot3.pdf", format='pdf', bbox_inches='tight')
31
32 # Display the plot
33 plt.show()
```



شکل ۱۹: نمودار خطا در حالت رزنت فریز، بدون فریز و بدون فریز (Bidirectional LSTM).