

# پروژه پایانی درس یادگیری عمیق

کورس تقی پور پاسدار

۱۲ بهمن ۱۴۰۳

## فهرست مطالب

۳	<b>Preprocess</b>	۱
۳	. . . . . Import Libraries	۱.۱
۳	. . . . . Preprocess Flickr8k Dataset	۲.۱
۳	. . . . . Download and Unzip Dataset	۱.۲.۱
۳	. . . . . Read and Save Captions	۲.۲.۱
۴	. . . . . Extract Translated Captions and Merge	۳.۲.۱
۴	. . . . . Check Translation Validity	۴.۲.۱
۵	<b>Define Model</b>	۲
۵	<b>Define Dataset</b>	۳
۶	<b>Fine-Tune</b>	۴
۶	. . . . . Freeze Some Layers	۱.۴
۷	<b>Import Evaluation Data</b>	۵
۸	<b>Text-to-Image Retrieval</b>	۶

## ۱ Preprocess

در این بخش، به import کردن کتابخانه‌های لازم و همچنین دانلود دیتاست Flickr8k و import کردن ترجمه‌ای که قبلاً انجام داده‌ایم می‌پردازیم.

### ۱.۱ Import Libraries

```
[ ] import torch
import torch.nn as nn
import clip
import requests
import os
import cv2
import glob
import zipfile
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from statistics import mean
from PIL import Image
from tqdm import tqdm
from torchvision.transforms import ToPILImage
```

## ۲.۱ Preprocess Flickr8k Dataset

### ۱.۲.۱ Download and Unzip Dataset

در ابتدا تابعی برای دانلود و ذخیره دیتاست تعریف می‌کنیم.

```
[ ] flickr_url = "https://flickr-photos-dataset-releases/download/v1/flickr.zip"
os.makedirs('datasets/flickr', exist_ok = True)
download_and_extract(flickr_url, 'datasets/flickr')
```

```
[ ] def download_and_extract(url, extract_to='.'):
    filename = url.split('/')[-1]
    print(f'Downloading {filename} ...')
    response = requests.get(url, stream=True)
    if response.status_code == 200:
        with open(filename, 'wb') as f:
            for chunk in response.iter_content(chunk_size=1024):
                f.write(chunk)
            print(f'Extracting {filename} ...')
            with zipfile.ZipFile(filename, 'r') as zip_ref:
                zip_ref.extractall(extract_to)
            print(f'Extracted to {extract_to}')
            os.remove(filename)
    else:
        print(f'Failed to download {url}')
```

### ۲.۲.۱ Read and Save Captions

سپس به خواندن و استخراج کپشن‌ها از این دیتاست می‌پردازیم.

```
[ ] captions_flickr = []
with open("datasets/flickr/captions.txt", 'r') as file:
    for line in file:
        img_name, caption = line.strip().split(',', 1)
        captions_flickr.append((img_name, caption))
captions_flickr = captions_flickr[1:]

[ ] captions = pd.DataFrame(captions_flickr, columns=['image', 'caption_en'])
captions['id'] = range(len(captions))

captions.head()
```

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

### ۳.۲.۱ Extract Translated Captions and Merge

سپس کپشن‌های فارسی ترجمه شده از قبل را از فایل‌ها خوانده و با کپشن‌های انگلیسی در یک دیتافریم<sup>۱</sup> قرار می‌دهیم.

```
[ ] files_path = '/content/drive/myDrive/'

fa_captions = []
for i in range(1,7):
    data = pd.read_excel(''.join([files_path, f'flickr8k_translated_captions{i}.xlsx']))
    fa_captions.append(data)

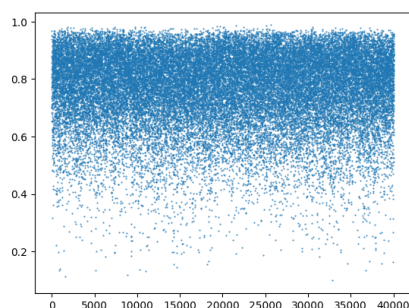
merged_df = pd.concat(fa_captions, axis=0).dropna(subset=['caption_fa'])
merged_df = merged_df.drop_duplicates(subset = ['image', 'caption_en'])
merged_caps = pd.merge(captions, merged_df, on=['image', 'caption_en'], how='inner')
merged_caps.head()
```

	image	caption_en	id	caption_fa
0	1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set o...	0	...کودکی با لباس صورتی در حال بالا رفتن از مجموعه
1	1000268201_693b08cb0e.jpg	A girl going into a wooden building .	1	...دختری که وارد یک ساختمان چوبی می‌شود
2	1000268201_693b08cb0e.jpg	A little girl climbing into a wooden playhouse .	2	...دختر بچه‌ای در حال بالا رفتن از یک خانه بازی چ
3	1000268201_693b08cb0e.jpg	A little girl climbing the stairs to her playh...	3	...دختر بچه‌ای که از پله‌ها به سمت خانه بازی اش ب
4	1000268201_693b08cb0e.jpg	A little girl in a pink dress going into a woo...	4	...دختر بچه‌ای با لباس صورتی که وارد یک کابین چوب

### ۴.۲.۱ Check Translation Validity

سپس با استفاده از یک مدل هوش مصنوعی Sentence Transformer به مقایسه و بررسی کیفیت ترجمه می‌پردازیم.

<sup>1</sup>DataFrame



(د) رسم بصری هر یک از مقادیر تطابق

```
[ ] similarities = []
for i, row in tqdm(merged_caps.iterrows()):
    sentences = [row['caption_en'], row['caption_fa']]
    embeddings = model.encode(sentences)
    original_embed = embeddings[0]
    translated_embedding = embeddings[1]
    similarity = cosine_similarity(original_embed, translated_embedding)
    similarities.append(similarity)

[ ] mean(similarities)
0.741666

[ ] plt.scatter(range(len(similarities)), similarities, s=0.2)
```

(ج) بررسی کیفیت ترجمه داده‌ها و نمایش میانگین تطابق

## ۲ Define Model

حال در اینجا به تعریف مدل می‌پردازیم. برای این کار، از مدل CLIP استفاده کرده‌ایم و به تعریف دقیق‌تر، یک مدل pretrain شده را استفاده می‌کنیم که سعی می‌کنیم با Fine-tune کردن آن، برای تسک مورد نظر آموزش دهیم.

```
[ ] from transformers import CLIPProcessor, CLIPModel

[ ] import os
os.environ['HF_TOKEN'] = # Insert your HF_TOKEN

from huggingface_hub import login
login(token=os.getenv('HF_TOKEN'))

[ ] model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
```

## ۳ Define Dataset

در اینجا، دیتاست خود را تعریف می‌کنیم. منظور از تعریف دیتاست، تعریف کلاس<sup>۲</sup> دیتاست است تا داده‌ها را از آن بگیریم.

<sup>۲</sup>Class

```

1 class ImageCaptionDataset(Dataset):
2     def __init__(self, dataset, path, processor):
3         self.dataset = dataset
4         self.processor = processor
5         self.image_folder_path = path
6
7     def __len__(self):
8         return len(self.dataset)
9
10    def __getitem__(self, idx):
11        item = self.dataset.iloc[idx]
12        caption = item['caption_fa']
13        image = Image.open(''.join([self.image_folder_path, item['image']]))
14        inputs = self.processor(text=caption),
15        images=image,
16        return_tensors="pt",
17        padding=True,
18        truncation=True,
19        max_length=77)
20
21        input_ids = inputs['input_ids'].squeeze(0)
22        attention_mask = inputs['attention_mask'].squeeze(0)
23        max_text_len = 77
24        input_ids_padded = torch.nn.functional.pad(input_ids,
25        (0, max_text_len - input_ids.size(0)),
26        value=processor.tokenizer.pad_token_id)
27        attention_mask_padded = torch.nn.functional.pad(attention_mask, (0, max_text_len - attention_mask.size(0)))
28        pixel_values = inputs['pixel_values'].squeeze(0)
29        return {
30            'input_ids': input_ids_padded,
31            'attention_mask': attention_mask_padded,
32            'pixel_values': pixel_values
33        }

```

سپس داده‌های آموزشی و تست را با استفاده از train test split از کتابخانه sklearn جدا می‌کنیم.

```

[ ] x_train, x_test, y_train, y_test = train_test_split(merged_caps[['caption_fa', 'id']],
                                                    merged_caps[['image', 'id']],
                                                    random_state=42,
                                                    test_size=0.2,
                                                    shuffle=True)

path = '/content/datasets/flickr/Images/'

[ ] train_dataset = ImageCaptionDataset(pd.merge(x_train, y_train, on='id', how='inner'), path, processor)
test_dataset = ImageCaptionDataset(pd.merge(x_test, y_test, on='id', how='inner'), path, processor)

train_dataloader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=2)
test_dataloader = DataLoader(test_dataset, batch_size=32, shuffle=True, num_workers=2)

```

## ۴ Fine-Tune

حال به بخش Fine-Tune مدل می‌رسیم.

### ۱.۴ Freeze Some Layers

با توجه به اینکه این یک شبکه‌ی از پیش آموزش دیده است، پس توانایی برقراری ارتباط بین تصویر و متن را دارد. لازم است که بخش‌های استخراج ویژگی از تصویر و همچنین برقراری ارتباط بین ویژگی‌های تصویر و متن ثابت مانده و تنها استخراج ویژگی از متن تغییر کند. به همین دلیل، به فریز کردن بخش بزرگی از شبکه می‌پردازیم.

```
[ ] for param in model.parameters():
    param.requires_grad = False

    for param in model.text_model.embeddings.token_embedding.parameters():
        param.requires_grad = True

[ ] optimizer = torch.optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=1e-2)
    loss_fn = nn.CrossEntropyLoss()
    device = "cuda" if torch.cuda.is_available() else "cpu"

[ ] device
```

همچنین از بهینه‌ساز Adam و تابع CrossEntropyLoss برای محاسبه خطا استفاده می‌کنیم. سپس به تعریف تابع train می‌پردازیم.

```
10 def train():
11     val_loss = 0
12     for batch in tqdm(train_dataloader):
13         batch = DayTensor(batch) for key, tensor in batch.items():
14             tensor = tensor.to(device)
15         logits_per_image = model.fc_logits_per_image
16         logits_per_text = model.fc_logits_per_text
17         targets = torch.arange(batch["text_id"].size(0)).to(logits_per_image.device)
18         image_loss = loss_fn(logits_per_image, targets)
19         text_loss = loss_fn(logits_per_text, targets)
20         loss = (text_loss + image_loss) / 2
21         val_loss += loss.detach()
22     val_loss_list.append(val_loss.item())
23     print(f"loss {train_loss / len(train_dataloader)}, val loss = {val_loss / len(val_dataloader)}")
24     return train_loss, val_loss, loss
```

```
1 def train_model(model, train_dataloader, val_dataloader, optimizer, criterion, device="cpu"):
2     train_loss_list = []
3     val_loss_list = []
4     model = model.to(device)
5     model.train()
6     for epoch in range(epochs):
7         train_loss = 0
8         for batch in tqdm(train_dataloader):
9             batch = DayTensor(batch) for key, tensor in batch.items():
10                 tensor = tensor.to(device)
11             outputs = model(**batch)
12             logits_per_image = model.fc_logits_per_image
13             logits_per_text = model.fc_logits_per_text
14             targets = torch.arange(batch["text_id"].size(0)).to(logits_per_image.device)
15             image_loss = loss_fn(logits_per_image, targets)
16             text_loss = loss_fn(logits_per_text, targets)
17             loss = (text_loss + image_loss) / 2
18             criterion_loss = criterion(outputs, targets)
19             train_loss += loss.detach()
20             optimizer.zero_grad()
21             loss.backward()
22             train_loss += loss.detach()
23         optimizer.step()
24         train_loss_list.append(train_loss.item())
25         train_dataloader
```

سپس به آموزش مدل می‌پردازیم.

```
[ ] def save_model(model):
    torch.save(model.state_dict(), "/content/drive/MyDrive/model_parameters.pth")

[ ] train_loss1, val_loss1 = train_model(model, train_dataloader, test_dataloader, optimizer, loss_fn, 10, device)
    optimizer = torch.optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=1e-3)
    loss_fn = nn.CrossEntropyLoss()
    device = "cuda" if torch.cuda.is_available() else "cpu"
    train_loss2, val_loss2 = train_model(model, train_dataloader, test_dataloader, optimizer, loss_fn, 10, device)
    save_model(model)
```

## ۵ Import Evaluation Data

حال در اینجا به ویدیوهای تست، استخراج فریم‌ها و encode کردن آنها به آرایه‌های تنسور<sup>۳</sup> می‌پردازیم.

<sup>3</sup>Tensor Array

```
[ ] def extract_frames_from_videos(video_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)
    video_files = glob.glob(os.path.join(video_folder, "*.mp4"))

    for video_file in video_files:
        video_name = os.path.splitext(os.path.basename(video_file))[0]
        video_frames_folder = os.path.join(output_folder, video_name)
        os.makedirs(video_frames_folder, exist_ok=True)
        video_frame_extract_get_fps(video_file, video_frames_folder)
        print(f"Extracted frames from file {video_file}")
```

(ح) تابعی برای استخراج فریم‌های تمام ویدیوهای داخل یک پوشه

```
[ ] def preprocess_videos(video_folder, output_folder, frame_folder):
    frame_list = []
    extract_frames_from_videos(video_folder, output_folder)
    os.makedirs(frame_folder, exist_ok=True)
    for _, folder in tqdm(enumerate(os.listdir(output_folder))):
        frames = encode_frames(os.path.join(output_folder, folder))
        torch.save(frames, f"{frame_folder}/{folder}.pt")
```

(ی) تابعی برای استخراج tensor های تمام ویدیوها. این تابع دربردارنده تابع‌های قبلی است.

```
[ ] def load_frame_tensor(frame_folder):
    frame_array = []
    for frame in sorted(os.listdir(frame_folder)):
        frame_path = os.path.join(frame_folder, frame)
        frame_array.append(torch.load(frame_path))
    return torch.concat(frame_array)
```

(ک) تابعی برای لود کردن tensor ها

```
[ ] def video_frame_extract_get_fps(video_path, frames_dir):
    os.makedirs(frames_dir, exist_ok=True)

    cap = cv2.VideoCapture(video_path)
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    step = 1 if fps < 2 else int(fps / 2)
    frame_count = 0
    frame_step = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frame_filename = os.path.join(frames_dir, f"frame_{frame_count:04d}.jpg")
        if frame_step < step:
            continue
        cv2.imwrite(frame_filename, frame)
        frame_step += 1
        frame_count += 1

    cap.release()
```

(ز) تابع استخراج فریم‌های یک ویدیو

```
[ ] def encode_frames(frames_dir):
    # List to store encoded frames
    encoded_frames = []

    # Iterate over each frame image
    for frame_file in sorted(os.listdir(frames_dir)):
        if frame_file.endswith('.jpg'):
            frame_path = os.path.join(frames_dir, frame_file)
            image = Image.open(frame_path)

            # Process the image to the required format
            inputs = processor(images=image, return_tensors="pt")

            # Get the pixel values
            pixel_values = inputs['pixel_values'].squeeze(0) # remove batch dimension
            encoded_frames.append(pixel_values)

    # Stack the encoded frames into a single tensor
    encoded_frames_tensor = torch.stack(encoded_frames)

    return encoded_frames_tensor
```

(ط) تابعی برای encode کردن ویدیوها و تبدیل آنها به یک Tensor

## ۶ Text-to-Image Retrieval

حال به تابع یافتن فریم ویدیو براساس متن ورودی می‌پردازیم. در این تابع براساس متن ورودی و آرایه‌ای که در قبل از ویدیوها ساختیم، بهترین k فریم را پیدا می‌کنیم. این مقدار قابل تغییر است.



```

1 def text_image_retrieval(model, processor, text, frames_tensor, top_k=5, device="cpu"):
2     model = model.to(device)
3     model.eval()
4     frames_tensor = frames_tensor.to(device)
5     text_inputs = processor(text=[text], return_tensors="pt", padding=True,
6                             truncation=True, max_length=77).to(device)
7     text_features = model.get_text_features(**text_inputs)
8     frame_features = model.get_image_features(pixel_values=frames_tensor)
9
10    text_features = text_features / text_features.norm(dim=-1, keepdim=True)
11    frame_features = frame_features / frame_features.norm(dim=-1, keepdim=True)
12
13    # Similarities
14    similarities = torch.matmul(text_features, frame_features.T)
15
16    top_k_indices = similarities.topk(top_k, dim=1).indices.squeeze(0).tolist()
17
18    for i, idx in enumerate(top_k_indices):
19
20        most_similar_frame = frames_tensor[idx].cpu()
21
22        mean = torch.tensor([0.485, 0.456, 0.406])
23        std = torch.tensor([0.229, 0.224, 0.225])
24
25        # Convert the tensor back to a PIL image
26        denormalized_pixel_values = most_similar_frame * std[:, None, None] + mean[:, None, None]
27        denormalized_pixel_values = torch.clamp(denormalized_pixel_values, 0, 1)
28        image = ToPILImage()(denormalized_pixel_values)
29
30        plt.figure()
31        plt.imshow(image)
32        plt.title(f"Top {i+1} Similar Frame.")
33        plt.axis('off') # Hide axes for better visualization
34    plt.show()

```