Agentic Ai Lab

Nirob Paul (2023000168)

Section – F

Group – 1

```
# ==============================
# Install dependencies
# ==============================
!pip install -q git+https://github.com/huggingface/transformers.git@main
!pip install -q datasets accelerate torchvision

# ==============================
# Imports
# ==============================
import torch
from torch.utils.data import Dataset, DataLoader
from datasets import load_dataset
from transformers import BlipProcessor, BlipForConditionalGeneration
from tqdm import tqdm

# ==============================
# Device
# ==============================
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Using device:", device)

# ==============================
# Load Dataset
# ==============================
dataset = load_dataset("ybelkada/football-dataset", split="train")

# ==============================
```

```
# Load Model + Processor
# ==================================
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained(
    "Salesforce/blip-image-captioning-base"
).to(device)


# ==================================
# Custom Dataset Class
# ==================================
class ImageCaptioningDataset(Dataset):
    def __init__(self, dataset, processor):
        self.dataset = dataset
        self.processor = processor

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        item = self.dataset[idx]
        image = item["image"]
        text = item["text"]


        encoding = self.processor(
            images=image,
            text=text,
```

```
            padding="max_length",
            truncation=True,
            max_length=64,
            return_tensors="pt"
        )

        # remove batch dim
        encoding = {k: v.squeeze(0) for k, v in encoding.items()}
        return encoding

# ===================================
# Dataloader
# ===================================
train_dataset = ImageCaptioningDataset(dataset, processor)
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)

# ===================================
# Optimizer
# ===================================
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)

# ===================================
# Training Loop
# ===================================
epochs = 3

model.train()
```

```
for epoch in range(epochs):
    print(f"\nEpoch {epoch+1}/{epochs}")
    loop = tqdm(train_loader)

    for batch in loop:
        batch = {k: v.to(device) for k, v in batch.items()}

        outputs = model(
            pixel_values=batch["pixel_values"],
            input_ids=batch["input_ids"],
            attention_mask=batch["attention_mask"],
            labels=batch["input_ids"]
        )

        loss = outputs.loss
        loss.backward()

        optimizer.step()
        optimizer.zero_grad()

        loop.set_description(f"Epoch {epoch+1}")
        loop.set_postfix(loss=loss.item())

print("\n✅ Training complete!")

# ================================
# Inference Test
```

```
# ====================================
model.eval()

example = dataset[0]
image = example["image"]

inputs = processor(images=image, return_tensors="pt").to(device)
pixel_values = inputs.pixel_values

with torch.no_grad():
    generated_ids = model.generate(pixel_values=pixel_values, max_length=50)

generated_caption = processor.batch_decode(
    generated_ids, skip_special_tokens=True
)[0]

print("\n🖼 Generated Caption:")
print(generated_caption)
```