OCR Artwork to Text Extraction Application

Objective: To deploy a Streamlit-based OCR application using the DeepSeek-VL2 model for text extraction from images, initially developed on Google Colab Pro with an A100 GPU, and later attempted on Hugging Face Spaces with Streamlit.

Project Overview

The goal of this project was to create an OCR (Optical Character Recognition) application capable of extracting text from images with high fidelity, preserving formatting such as spaces, bullets, and numbers, exclusively in English. The application leverages the DeepSeek-VL2 model, a vision-language model designed for tasks like image-based text extraction. The development was initially performed on Google Colab Pro with an NVIDIA A100 GPU due to the computational requirements of the model. However, for persistent deployment, the application was migrated to Hugging Face Spaces using Streamlit, where it encountered deployment issues due to the lack of GPU support.

Requirements

Hardware Requirements

- **Development Environment**: Google Colab Pro with NVIDIA A100-SXM4-40GB GPU
- **Deployment Target**: Hugging Face Spaces (Should GPU A100 Support).

Software Requirements

The following dependencies were used in the project, as specified in the requirements.txt file and additional installations:

Initial requirements.txt (for Hugging Face Spaces)

```
--extra-index-url https://download.pytorch.org/whl/cu117
torch==2.0.1+cu117
streamlit==1.18.0
pillow==9.4.0
transformers==4.38.2
xformers==0.0.21
mdtex2html==1.3.0
timm==1.0.15
accelerate==1.5.2
sentencepiece==0.1.96
attrdict==2.0.1
einops==0.8.1
```

```
deepseek-vl2 @ git+https://github.com/deepseek-ai/DeepSeek-VL2
```

Additional Installations (Performed on Colab)

To ensure compatibility and functionality on Google Colab Pro with the A100 GPU, the following installations were executed:

- **Install PyTorch with CUDA 11.8**:

  ```
  !pip install torch torchvision torchaudio --index-url
  https://download.pytorch.org/whl/cu118
  ```

  Installed torch==2.6.0+cu118, torchvision==0.21.0+cu118, and torchaudio==2.6.0+cu118 to leverage the A100 GPU.

- **Install Specific Versions**:

  ```
  !pip install xformers==0.0.21
  !pip install git+https://github.com/huggingface/transformers
  !pip install gradio attrdict numpy==1.26.4 mdtex2html==0.15.2
  ```

- **Install DeepSeek-VL2 Requirements**:

  ```
  !pip install -r https://raw.githubusercontent.com/deepseek-ai/DeepSeek-
  VL2/main/requirements.txt
  ```

Development Steps on Google Colab Pro

Step 1: Verify GPU Availability

```
import torch
print("CUDA Available:", torch.cuda.is_available())
print("GPU Name:", torch.cuda.get_device_name(0) if torch.cuda.is_available()
else "No GPU")
```

Output :-

```
CUDA Available: True
GPU Name: NVIDIA A100-SXM4-40GB
```

This confirmed the A100 GPU was available, necessary due to the CPU's inadequacy for DeepSeek-VL2's computational demands.

Step 2: Install Dependencies

Encountered an interruption during xformers installation, resolved by reinstalling with:

```
!pip install xformers --no-cache-dir
```

(Installed xformers==0.0.29.post3)

Step 3: Develop the Application

The application code (app.py) was written and tested on Colab pro :

```
import os
import gradio as gr
import torch
from PIL import Image
from deepseek_vl2.serve.inference import load_model, deepseek_generate,
convert_conversation_to_prompts
from deepseek_vl2.models.conversation import SeparatorStyle
from deepseek_vl2.serve.app_modules.utils import configure_logger,
strip_stop_words, pil_to_base64
from google.colab import files

logger = configure_logger()

MODELS = ["deepseek-ai/deepseek-vl2-tiny"]
DEPLOY_MODELS = {}
IMAGE_TOKEN = "<image>"

def fetch_model(model_name: str, dtype=torch.bfloat16):
    global DEPLOY_MODELS
    if model_name not in DEPLOY_MODELS:
        print(f"Loading {model_name}...")
        model_info = load_model(model_name, dtype=dtype)
        tokenizer, model, vl_chat_processor = model_info
        device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
        model = model.to(device)
        DEPLOY_MODELS[model_name] = (tokenizer, model, vl_chat_processor)
        print(f"Loaded {model_name} on {device}")
    return DEPLOY_MODELS[model_name]

def generate_prompt_with_history(text, images, history, vl_chat_processor,
tokenizer, max_length=2048):
    conversation = vl_chat_processor.new_chat_template()
    if history:
        conversation.messages = history
    if images:
        text = f"{IMAGE_TOKEN}\n{text}"
        text = (text, images)
    conversation.append_message(conversation.roles[0], text)
    conversation.append_message(conversation.roles[1], "")
    return conversation

def to_gradio_chatbot(conv):
    ret = []
    for i, (role, msg) in enumerate(conv.messages[conv.offset:]):
        if i % 2 == 0:
            if isinstance(msg, tuple):
                msg, images = msg
                for image in images:
                    img_b64 = pil_to_base64(image, "user upload", 
max_size=800, min_size=400)
                    msg = msg.replace(IMAGE_TOKEN, img_b64, 1)
            ret.append([msg, None])
        else:
            ret[-1][-1] = msg
    return ret

def predict(text, images, chatbot, history, model_name="deepseek-
ai/deepseek-vl2-tiny"):
    print("Starting predict function...")
    tokenizer, vl_gpt, vl_chat_processor = fetch_model(model_name)
    if not text:
        print("Empty text input detected.")
        return chatbot, history, "Empty context."

    print("Processing images...")
    pil_images = [Image.open(img).convert("RGB") for img in images] if 
images else []
    conversation = generate_prompt_with_history(
        text, pil_images, history, vl_chat_processor, tokenizer
    )
    all_conv, _ = convert_conversation_to_prompts(conversation)
    stop_words = conversation.stop_str
    gradio_chatbot_output = to_gradio_chatbot(conversation)

    full_response = ""
    print("Generating response...")
    try:
        with torch.no_grad():
            for x in deepseek_generate(
                conversations=all_conv,
                vl_gpt=vl_gpt,
                vl_chat_processor=vl_chat_processor,
                tokenizer=tokenizer,
                stop_words=stop_words,
                max_length=2048,
                temperature=0.1,
                top_p=0.9,
                repetition_penalty=1.1
            ):
                full_response += x
                response = strip_stop_words(full_response, stop_words)
                conversation.update_last_message(response)
                gradio_chatbot_output[-1][1] = response
                print(f"Yielding partial response: {response[:50]}...")
                yield gradio_chatbot_output, conversation.messages,
"Generating..."

        print("Generation complete.")
        torch.cuda.empty_cache()
        yield gradio_chatbot_output, conversation.messages, "Success"
    except Exception as e:
        print(f"Error in generation: {str(e)}")
        yield gradio_chatbot_output, conversation.messages, f"Error:
{str(e)}"

# Gradio interface for OCR
def upload_and_process(image):
    if image is None:
        return "Please upload an image.", []
    prompt = "Extract all text from this image exactly as it appears,
ensuring the output is in English only. Preserve spaces, bullets, numbers,
and all formatting. Do not translate, generate, or include text in any
other language."
    chatbot = []
    history = []
    print("Starting upload_and_process...")
    for chatbot_output, history_output, status in predict(prompt, [image],
chatbot, history):
        print(f"Status: {status}")
        if status == "Success":
            return chatbot_output[-1][1], history_output
    return "Processing failed.", []

# Launch Gradio app
with gr.Blocks() as demo:
    gr.Markdown("### OCR Artwork to Text Extraction Application")
    image_input = gr.Image(type="filepath", label="Upload Image")
    output_text = gr.Textbox(label="Extracted Text")
    history_state = gr.State([])
    submit_btn = gr.Button("Extract Text")
    submit_btn.click(upload_and_process, inputs=image_input,
outputs=[output_text, history_state])

demo.launch(share=True, debug=True)  # Added debug=True for more Gradio
logs
```

Step 4: Test the Application

Uploaded sample images (e.g., job postings) and verified text extraction.

The application ran successfully on Colab, generating a temporary URL (valid for 72 hours).

Deployment Attempt on Hugging Face Spaces

Step 1: Prepare Files

- Created a Hugging Face Space with the following files:
  - **requirements.txt**: As listed above.
  - **app.py**: Copied from Colab

Step 2: Initial Deployment

- Pushed the files to Space and attempted to build.
- Encountered an error:

  ```
  RuntimeError: Found no NVIDIA driver on your system
  ```

- **Cause**: Hugging Face Spaces (free tier) provides only CPU resources, while deepseek_vl2/serve/inference.py hardcodes .cuda() calls, assuming a GPU.

Step 3: Attempted Fix

Modified fetch_model in app.py to use CPU if no GPU is available:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = model.to(device).eval()
```

- It took a huge compilation time because of CPU running.

Challenges Encountered

1. **GPU Dependency**:
   - DeepSeek-VL2 requires significant computational resources, making CPU-only deployment impractical without optimization.
   - Hugging Face Spaces (free tier) lacks GPU support, unlike Colab Pro's A100.
2. **Temporary URL on Colab**:
   - Colab's URL expires after 72 hours, unsuitable for persistent deployment.
3. **Memory and Performance**:
   - Even with the patched code, the model's size and inference time may exceed Hugging Face Spaces' free-tier limits.

Proposed Solutions

1. **Use Hugging Face Spaces with Paid Tier**:
   - Upgrade to a paid plan with GPU support to match Colab's environment.
2. **Alternative Platforms for deploy**:
   - Deploy on a cloud service like AWS, Google Cloud, or Heroku with GPU instances.
3. **Local Deployment**:
   - Host on a local machine with GPU for persistent access.

Conclusion

The DeepSeek-VL2 OCR application was successfully developed and tested on Google Colab Pro with an A100 GPU, leveraging its computational power. However, deployment on Hugging Face Spaces failed due to the absence of NVIDIA GPU support in the free tier. While patching the code to use CPU was attempted, further optimization or a GPU-enabled environment is required for successful deployment. Future efforts will focus on securing GPU resources or optimizing the model for CPU compatibility.