Project Files

- **main.py**: The main Python script.
- **README.md**: Project documentation.
- **requirements.txt**: A list of all required Python libraries.

main.py

This file contains a complete conceptual implementation of an image captioning and segmentation project. It demonstrates the full workflow, from data handling to model definition and conceptual inference.

```
# filename: main.py

# Import all required libraries for the project
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input, Dense, Embedding, LSTM, Reshape, TimeDistributed,
concatenate,
    Conv2D, MaxPooling2D, Conv2DTranspose
)
from tensorflow.keras.applications import VGG16
import numpy as np
import matplotlib.pyplot as plt
import cv2 # OpenCV is used for image processing

# --- Project Configuration ---
# Define key parameters for the models
MAX_CAPTION_LENGTH = 16
VOCAB_SIZE = 5000
EMBEDDING_DIM = 256
LSTM_UNITS = 512

def load_data():
    """
    Conceptual function to simulate loading a real dataset.

    In a real project, this function would load images, captions, and
    segmentation masks from a dataset like MS COCO or Pascal VOC. For
this
    demonstration, we generate dummy data that mimics the real data's
structure.

    Returns:
        tuple: A tuple containing (images, captions, masks) as NumPy
arrays.
```

```
    """
    print("Simulating data loading...")
    # Generate dummy images (a batch of 4, with 224x224x3 dimensions)
    images = np.random.rand(4, 224, 224, 3).astype(np.float32)
    # Generate dummy integer-encoded captions
    captions = np.random.randint(0, VOCAB_SIZE, (4,
MAX_CAPTION_LENGTH))
    # Generate dummy binary segmentation masks
    masks = np.random.randint(0, 2, (4, 224, 224,
1)).astype(np.float32)
    print("Data loading complete.")
    return images, captions, masks

def build_captioning_model():
    """
    Builds a conceptual CNN-LSTM model for image captioning.

    This model has two main parts:
    1. A CNN encoder (VGG16) to extract features from the image.
    2. An LSTM decoder to generate a sequence of words (the caption)
from the features.

    Returns:
        tensorflow.keras.Model: The compiled captioning model.
    """
    print("Building image captioning model...")
    # --- CNN Encoder (Image Feature Extractor) ---
    # Load VGG16 pre-trained on ImageNet, removing the classification
head
    cnn_base = VGG16(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
    cnn_base.trainable = False  # Freeze the pre-trained weights to
speed up training

    # Define the input for the image
    image_input = Input(shape=(224, 224, 3), name='image_input')
    image_features = cnn_base(image_input)
    # Reshape the features to be a 1D sequence for the LSTM
    image_features = Reshape((-1,
image_features.shape[-1]))(image_features)

    # --- LSTM Decoder (Caption Generator) ---
    # Define the input for the captions
    caption_input = Input(shape=(MAX_CAPTION_LENGTH,),
name='caption_input')
    caption_embedding = Embedding(
        input_dim=VOCAB_SIZE,
        output_dim=EMBEDDING_DIM,
```

```
        mask_zero=True
    )(caption_input)

    # Concatenate the image features with the caption embedding to
form the decoder's input
    decoder_input = concatenate([image_features, caption_embedding],
axis=1)

    # Use an LSTM layer to process the sequence
    decoder_lstm = LSTM(LSTM_UNITS,
return_sequences=True)(decoder_input)
    # The TimeDistributed layer applies the Dense layer to each time
step
    decoder_output = TimeDistributed(Dense(VOCAB_SIZE,
activation='softmax'))(decoder_lstm)

    # Create the final model
    model = Model(inputs=[image_input, caption_input],
outputs=decoder_output)

    # Compile the model with an appropriate optimizer and loss
function
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy')
    print("Captioning model built and compiled.")
    return model

def build_segmentation_model():
    """
    Builds a conceptual U-Net-like model for image segmentation.

    This model follows an encoder-decoder structure:
    - Encoder: Downsamples the image to extract features.
    - Decoder: Upsamples the features to create a segmentation mask.

    Returns:
        tensorflow.keras.Model: The compiled segmentation model.
    """
    print("Building image segmentation model...")
    seg_input = Input(shape=(224, 224, 3))

    # Downsampling (Encoder) path
    conv1 = Conv2D(32, 3, activation='relu',
padding='same')(seg_input)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(64, 3, activation='relu', padding='same')(pool1)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
```

```
    # Bottleneck
    bottleneck = Conv2D(128, 3, activation='relu',
padding='same')(pool2)

    # Upsampling (Decoder) path
    up1 = Conv2DTranspose(64, 3, strides=(2, 2), activation='relu',
padding='same')(bottleneck)
    concat1 = concatenate([up1, conv2]) # Skip connection

    up2 = Conv2DTranspose(32, 3, strides=(2, 2), activation='relu',
padding='same')(concat1)

    # Output layer with a single channel (for binary segmentation) and
sigmoid activation
    output = Conv2D(1, 1, activation='sigmoid')(up2)

    model = Model(inputs=seg_input, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy')
    print("Segmentation model built and compiled.")
    return model

def visualize_results(image, segmentation_mask):
    """
    Plots the original image and the image with the segmentation mask
overlaid.
    """
    print("Visualizing results...")
    # Create a figure with two subplots
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))

    # Display the original image
    axes[0].imshow(image)
    axes[0].set_title("Original Image")
    axes[0].axis('off')

    # Display the segmentation mask overlaid on the original image
    axes[1].imshow(image)
    # The alpha parameter makes the mask semi-transparent
    axes[1].imshow(segmentation_mask.squeeze(), cmap='jet', alpha=0.5)
    axes[1].set_title("Image with Segmentation Mask")
    axes[1].axis('off')

    plt.tight_layout()
    plt.show()

def main():
    """
```

```
    The main execution function for the project.

    This function orchestrates the entire process:
    1. Loads dummy data.
    2. Builds both the captioning and segmentation models.
    3. Prints a summary of each model's architecture.
    4. Simulates a conceptual demonstration of the segmentation task.

    Note: A real project would include training loops for both models.
    """
    print("--- Starting Image Captioning and Segmentation Project
---")

    # 1. Load Data
    images, captions, masks = load_data()

    # 2. Build Models
    captioning_model = build_captioning_model()
    segmentation_model = build_segmentation_model()

    # Print model summaries to show their architecture
    captioning_model.summary()
    segmentation_model.summary()

    # 3. Simulate Inference
    # For a real project, you would load a trained model here
    # and use model.predict() on new data.
    print("\nSimulating conceptual inference...")

    # Select a single image and mask for visualization
    sample_image = images[0]
    sample_mask = masks[0]

    # The actual segmentation prediction would look like this:
    # predicted_mask =
segmentation_model.predict(np.expand_dims(sample_image, axis=0))[0]

    # 4. Visualize Results
    visualize_results(sample_image, sample_mask)

    print("--- Project execution finished. ---")
    print("This script is a robust framework. The next step is to load
a real dataset and train the models.")

if __name__ == "__main__":
    main()
```

README.md

🤖 Image Captioning and Segmentation

This repository contains a comprehensive, conceptual implementation of an **Image Captioning and Segmentation** project using Python and TensorFlow. The project showcases the core concepts, model architectures, and workflow required for a real-world application.

📁 Repository Structure

- main.py: The main script that defines the models, handles data, and runs the conceptual demonstration.
- requirements.txt: A list of all required Python libraries to run the code.
- README.md: This file, providing an overview of the project.

📋 Project Overview

The project focuses on two key computer vision tasks:

1. **Image Captioning**: Automatically generating a descriptive natural language caption for a given image. This is achieved using a **CNN-LSTM** model architecture.
2. **Image Segmentation**: Identifying and outlining objects in an image by assigning a class label to each pixel. This uses a **U-Net** inspired model.

⚙️ Tech Stack & Tools

- **Python 3.x**: The core programming language.
- **TensorFlow / Keras**: For building and training the deep learning models.
- **OpenCV**: For image preprocessing and visualization tasks.
- **NumPy**: For efficient numerical operations.
- **Matplotlib**: For generating plots to visualize the results.

🛠️ Setup Instructions

To run this project, follow these steps:

1. **Clone the Repository**:
   ```
   git clone
   https://github.com/your-username/Image-Captioning-and-Segmentation
   .git
   cd Image-Captioning-and-Segmentation
   ```

2. **Create and Activate a Virtual Environment**:
   ```
   python -m venv venv
   source venv/bin/activate  # On Windows: venv\Scripts\activate
   ```

3. **Install Dependencies**:

```
pip install -r requirements.txt
```

🚀 How to Run the Code

Simply execute the main Python script from your terminal:
`python main.py`

The script will perform the following actions:

- Simulate loading a dataset.
- Build and compile both the captioning and segmentation models.
- Display a conceptual diagram of the models.
- Visualize a sample image with an overlaid segmentation mask.

📚 Next Steps & Project Extensions

This project is a solid foundation. To take it to the next level, you would:

- **Data**: Download and process a large-scale dataset like MS COCO.
- **Training**: Implement a comprehensive training loop to train the models on real data.
- **Evaluation**: Calculate evaluation metrics (e.g., BLEU score for captions, IoU for segmentation).
- **Integration**: Build a single inference pipeline that takes an image and outputs both the caption and the segmentation mask.