JSS SCIENCE AND TECHNOLOGY UNIVERSITY, MYSURU

SUBJECT: Deep learning

TOPIC: Image Captioning AI

Prof. Rakshita P
Assistant Professor
Dept. of CS&E, JSS STU Mysore

By:

Pushkar Deshmukh	01JCE21CS078

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, JSS STU MYSORE

CONTENTS

Problem Statement

The exponential growth of multimedia content on the internet has created an urgent need for systems capable of automatically understanding and describing visual information. Among the challenges in this domain, the task of generating textual descriptions for images, commonly referred to as *image captioning*, has emerged as a critical area of research. Image captioning involves translating the contents of an image into meaningful natural language descriptions, effectively bridging the gap between computer vision and natural language processing.

The importance of image captioning extends across multiple applications, including:

- **Enhancing Accessibility**: By enabling visually impaired individuals to understand visual content through audio or textual descriptions, image captioning systems contribute to a more inclusive digital environment.
- **Content Search and Indexing**: Automatically generated captions enhance the searchability of images and improve the organization of multimedia databases, aiding in information retrieval tasks.
- **Social Media Automation**: The ability to automatically tag and describe images on platforms like Instagram, Facebook, and Twitter streamlines user interaction and ensures better content visibility.
- **Human-Computer Interaction**: As a part of smart systems, such descriptions facilitate seamless communication between humans and machines in applications like virtual assistants and robotics.

To address this need, this project focuses on designing and implementing an image captioning system that combines the strengths of two powerful neural network architectures:

1. **Convolutional Neural Networks (CNNs)**: Leveraging the pre-trained VGG16 model through transfer learning, the CNN component extracts high-level visual features from images.
2. **Recurrent Neural Networks (RNNs)**: Specifically, Long Short-Term Memory (LSTM) networks process the extracted features to generate coherent and contextually accurate captions in natural language.

The system will be trained on the widely-used **Flickr8K dataset**, which contains a diverse collection of images paired with corresponding human-generated captions. By integrating advanced deep learning techniques, this project aims to create a robust and effective image captioning model with practical utility across various domains.

Introduction

Artificial Intelligence (AI) has revolutionized the way machines interact with and interpret the world. AI, the simulation of human intelligence in machines, enables computers to perform tasks that typically require human cognition, such as understanding natural language, recognizing objects, and making decisions. The integration of AI in computer vision and natural language processing (NLP) has given rise to advanced capabilities such as image captioning, which bridges visual understanding with linguistic representation.

- **Image Captioning**

  Image captioning is a process of generating textual descriptions for images, blending the fields of computer vision and NLP. The goal is to create a system that can view an image and describe it in a coherent and human-like manner. For example, given a picture of a dog playing in a park, the system should output captions such as *"A dog playing in a grassy park."* This capability has diverse applications.

- **Convolutional Neural Networks (CNNs)**

  CNNs are a class of deep learning models primarily designed for analysing visual data. Inspired by the structure of the human visual cortex, CNNs excel at recognizing patterns in images, such as shapes, textures, and objects. Key components of CNNs include:

  - **Convolutional Layers**: Detect features like edges and textures.
  - **Pooling Layers**: Reduce spatial dimensions while retaining key information.
  - **Fully Connected Layers**: Aggregate the extracted features for classification or other tasks.

  In image captioning systems, CNNs act as feature extractors. Pre-trained CNN models, such as VGG16, are often used to transfer learning by leveraging knowledge from large-scale datasets like ImageNet. This significantly reduces training time and enhances accuracy.

- **Recurrent Neural Networks (RNNs) and LSTMs**

  While CNNs handle visual data, RNNs are designed for sequential data, such as text. Long Short-Term Memory networks (LSTMs), a special type of RNN, address the vanishing gradient problem in traditional RNNs and excel at capturing long-term dependencies in sequences. In image captioning:

  - CNNs provide a feature representation of the image.
  - LSTMs generate descriptive text by learning the sequence of words.

Literature review

1. **Show and Tell: A Neural Image Caption Generator (Vinyals et al., 2015)**
    - Introduced an encoder-decoder framework using CNN for image feature extraction and LSTM for sequence generation.
    - Demonstrated the effectiveness of maximum likelihood estimation in training the model for generating high-quality captions.

2. **Merge and Inject: A Multi-Modality Fusion Approach for Image Captioning (Ma et al., 2018)**
    - Proposed a "merge-and-inject" strategy to fuse image and text features for effective caption generation.
    - Showcased how modular networks can enhance multimodal fusion, improving caption diversity and relevance.

3. **Deep Visual-Semantic Alignments (Karpathy & Fei-Fei, 2015)**
    - Explored region-based image feature extraction to align image regions with sentence fragments.
    - Highlighted the advantage of using region-based approaches over global image feature methods for fine-grained captioning tasks.

4. **Exploring Transfer Learning in Image Captioning (Yao et al., 2017)**
    - Analysed the effectiveness of pre-trained CNN models (e.g., VGG16, ResNet) for transfer learning in captioning tasks.
    - Emphasized that fine-tuning pre-trained models can yield significant performance improvements in domain-specific datasets.

5. **Attention Is All You Need (Vaswani et al., 2017)**
    - Introduced the transformer architecture and its application in vision-language models.
    - Highlighted the elimination of recurrent structures, showcasing improved efficiency in sequence modelling tasks.

6. **Unifying Vision and Language Tasks via Pre-trained Transformers (Li et al., 2020)**
    - Highlighted the role of pre-trained multimodal transformers for tasks like captioning, emphasizing improved generalization.
    - Unified image and text embeddings into a single framework to enhance multimodal learning.

7. **Image Captioning with Deep Bidirectional LSTMs (Donahue et al., 2015)**
    - Demonstrated the effectiveness of bidirectional LSTMs in improving the context understanding of captions.
    - Showed that incorporating temporal dynamics of both past and future improves caption fluency.

8. **Object Detection and Image Captioning with Faster R-CNN (Ren et al., 2016)**
    - Integrated object detection into image captioning pipelines to improve the semantic richness of captions.
    - Highlighted how Faster R-CNN can efficiently localize and classify objects, enhancing image understanding.

9. **Stacked Cross Attention Networks for Image Captioning (Huang et al., 2019)**
    - Introduced cross-attention layers to model deeper visual-textual correlations.
    - Demonstrated that stacked attention networks lead to incremental improvements in caption quality.

10. **Comprehensive Survey of Image Captioning Techniques (Hossain et al., 2019)**
    - Provided a broad overview of methods, challenges, and datasets used in image captioning research.
    - Categorized techniques into template-based, retrieval-based, and neural-network-based methods, showcasing their evolution over time.

Tools Used

- **Programming Language:**
  - **Python:** A versatile programming language known for its simplicity and wide range of libraries, making it ideal for AI, deep learning, and data analysis.
- **Frameworks**
  - **TensorFlow:** An open-source machine learning framework that provides tools to build, train, and deploy deep learning models efficiently.
  - **Keras:** A high-level API running on top of TensorFlow, simplifying the creation and implementation of deep learning models, including CNNs and RNNs.
- **Transfer Learning Model:**
  - **VGG16:** A pre-trained Convolutional Neural Network model that excels at extracting image features. Its 16-layer architecture is widely used in transfer learning.
- **Dataset:**
  - **Flickr8K Dataset:** A dataset containing 8,000 images and corresponding captions, widely used for training and evaluating image captioning systems.
- **Libraries:**
  - **NumPy and Pandas:** Libraries for efficient numerical computation and data manipulation. NumPy is used for handling arrays and matrices, while Pandas manages data preprocessing tasks.
  - **Matplotlib:** A visualization library used to plot training metrics (loss, accuracy) and display sample images with generated captions.
  - **Natural Language Toolkit (NLTK):** A library for natural language processing tasks such as tokenization, stemming, and text cleaning, essential for preparing captions for the model.
- **Environment:**
  - **Jupyter Notebook:** An interactive development environment that allows for writing, debugging, and visualizing Python code with real-time outputs, ideal for deep learning experimentation.

Design Phase

The design process for the image captioning system involves multiple steps, combining computer vision and natural language processing techniques. Below is a detailed explanation of each step:

1. Dataset Preparation

- The Flickr8K dataset is used, consisting of 8,000 images and five captions per image. Images and captions are loaded, and text is tokenized to create a vocabulary of words.
- Steps:
    - Preprocess captions by cleaning text, removing special characters, and normalizing case.
    - Split the dataset into training, validation, and testing subsets.

2. Feature Extraction with CNN

- Features are extracted from images using the VGG16 model, pre-trained on ImageNet. The classification layers of VGG16 are removed to retain only the convolutional base.
- Steps:
    - Feed images into the VGG16 model to generate feature vectors.
    - Store these vectors to avoid redundant computation during training.

3. Caption Preprocessing

- Captions are converted into a format suitable for input to the RNN model. This involves creating sequences of words and numerical representations.
- Steps:
    - Tokenize and pad captions to ensure uniform length.
    - Map words to integer indices using a vocabulary dictionary.

4. Model Architecture

- The model integrates a CNN (encoder) and an RNN (decoder) to generate captions.
- Components:
    - Encoder: The VGG16 model extracts visual features from images.
    - Decoder: An LSTM-based RNN generates textual sequences from the encoded features.
    - Combine image features and text embeddings as inputs to the decoder.

5. Model Training

- The model is trained using the prepared dataset with supervised learning. The loss function and optimizer are chosen to ensure effective learning.
- Steps:
    - Use categorical cross-entropy as the loss function to predict the next word in the sequence.
    - Optimize the model using the Adam optimizer with learning rate adjustments.
    - Apply teacher forcing during training, feeding the actual next word as input to the model.

6. Evaluation

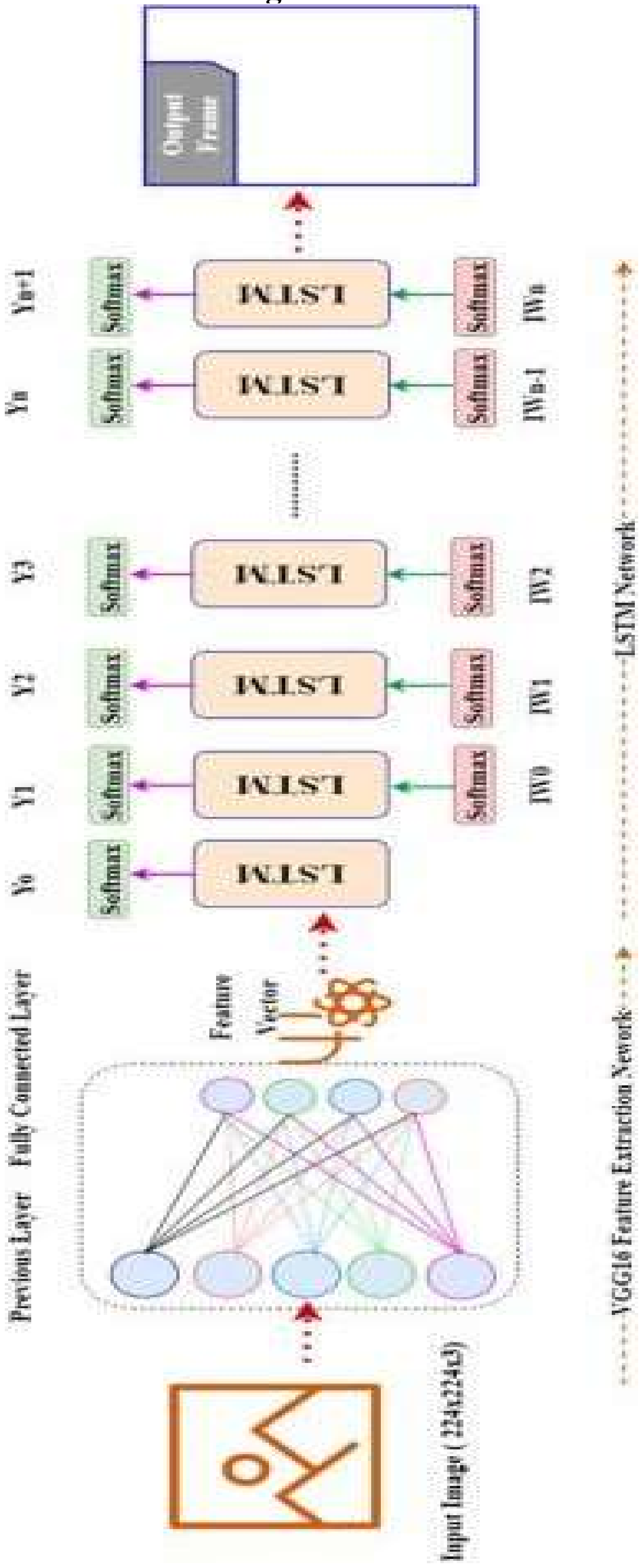
- The model's performance is assessed using standard metrics for image captioning.
- Steps:
    - Evaluate generated captions using BLEU, METEOR, or CIDEr scores to measure their similarity to reference captions.
    - Compare results across training and validation datasets to detect overfitting or underfitting.

7. Deployment

- The trained model is saved and integrated into a simple application for real-time caption generation.
- Steps:
    - Load a trained model and preprocess input images.
    - Generate and display captions for new images using the model's predictions.

Each of these steps ensures a systematic workflow, combining efficient feature extraction and sequence generation to create an accurate and robust image captioning system.

Architecture Diagram



Implementation

1. Data Preparation

a) **Data Collection:**
The dataset includes images and their corresponding captions stored in a text file (captions.txt) and an Images folder under the Data directory.

b) **Feature Extraction:**
The pre-trained VGG16 model was used to extract features from the images.
The last fully connected layer (fc2) of VGG16 was retained to generate 4096-dimensional feature vectors for each image.
Features were stored in a pickle file (features.pkl) for future use.

c) **Caption Preprocessing:**
Captions were cleaned by converting them to lowercase, removing non-alphabetic characters, and stripping extra spaces.
Each caption was wrapped with startseq and endseq tokens to mark its beginning and end.

d) **Tokenizer Creation:**
A tokenizer was created to map words to integer indices.
The tokenizer was fit on the cleaned captions, and the vocabulary size was calculated.
Tokenizer object was saved in a pickle file (tokenizer.pkl).

e) **Train-Test Split:**
The dataset was split into training and testing sets in a 90:10 ratio based on image IDs.

2. Model Architecture

a) **Image Feature Processing:**
   a. Input: 4096-dimensional feature vector from VGG16.
   b. Processing: Dropout layer followed by a dense layer with 256 units and ReLU activation.

b) **Text Processing:**
   a. Input: Integer sequences of tokenized captions.
   b. Processing: Embedding layer (size 256), dropout layer, and LSTM layer (size 256).

c) **Decoder:**
   a. Combination of image and text feature outputs using an addition layer.
   b. Dense layer with 256 units and ReLU activation.
   c. Output layer with softmax activation to predict the next word.

d) **Model Compilation:**
   a. Loss function: Categorical Crossentropy.
   b. Optimizer: Adam.

3. Training Process

a) **Data Generator:**
   a. A generator was created to yield batches of image features, tokenized caption sequences, and their corresponding outputs (next word in the sequence).

b) Training Configuration:

a. Epochs: 50
b. Batch size: 32
c. Steps per epoch: Calculated as the number of training samples divided by the batch size.

c) Training Execution:

a. The model was trained using the data generator, with one epoch per loop iteration.

d) Model Saving:

a. The trained model was saved in a pickle file (best_model.pkl) for inference.

4. Caption Prediction

a) Image Preprocessing:

a. Input images were resized to 224x224 pixels and preprocessed using VGG16's preprocess functions.

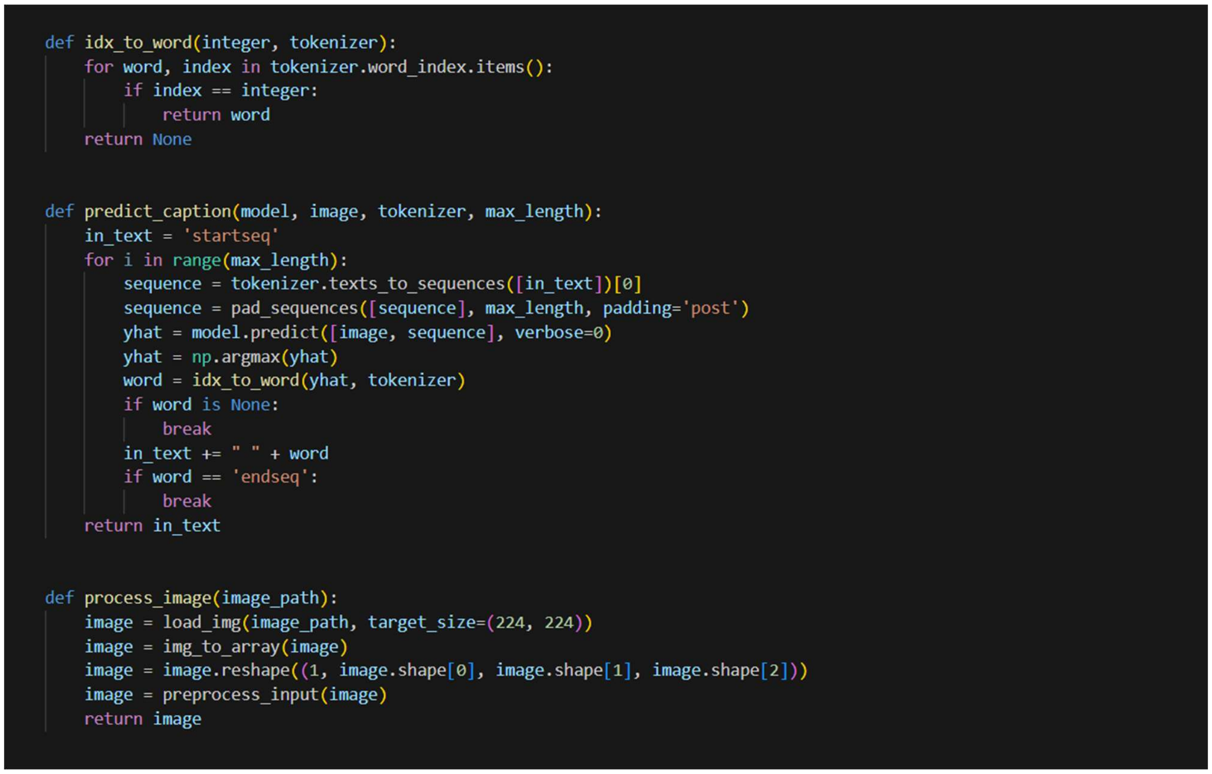
b) Caption Generation:

a. Starting with startseq, the model predicted the next word iteratively until endseq or the maximum sequence length was reached.

5. Evaluation

a) BLEU Score Calculation:

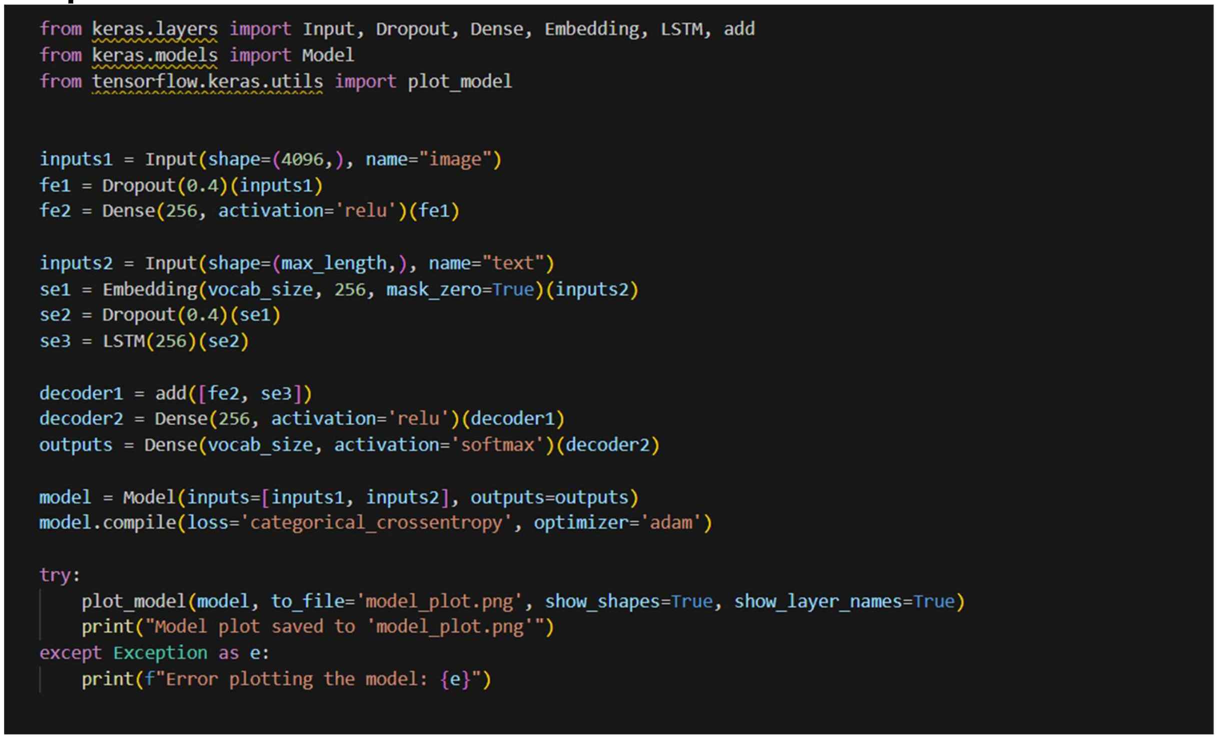
a. BLEU-1 and BLEU-2 metrics were calculated to evaluate the model's performance.
b. Predictions were compared to the actual captions in the test set.



```python
def idx_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None


def predict_caption(model, image, tokenizer, max_length):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], max_length, padding='post')
        yhat = model.predict([image, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = idx_to_word(yhat, tokenizer)
        if word is None:
            break
        in_text += " " + word
        if word == 'endseq':
            break
    return in_text


def process_image(image_path):
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)
    return image
```

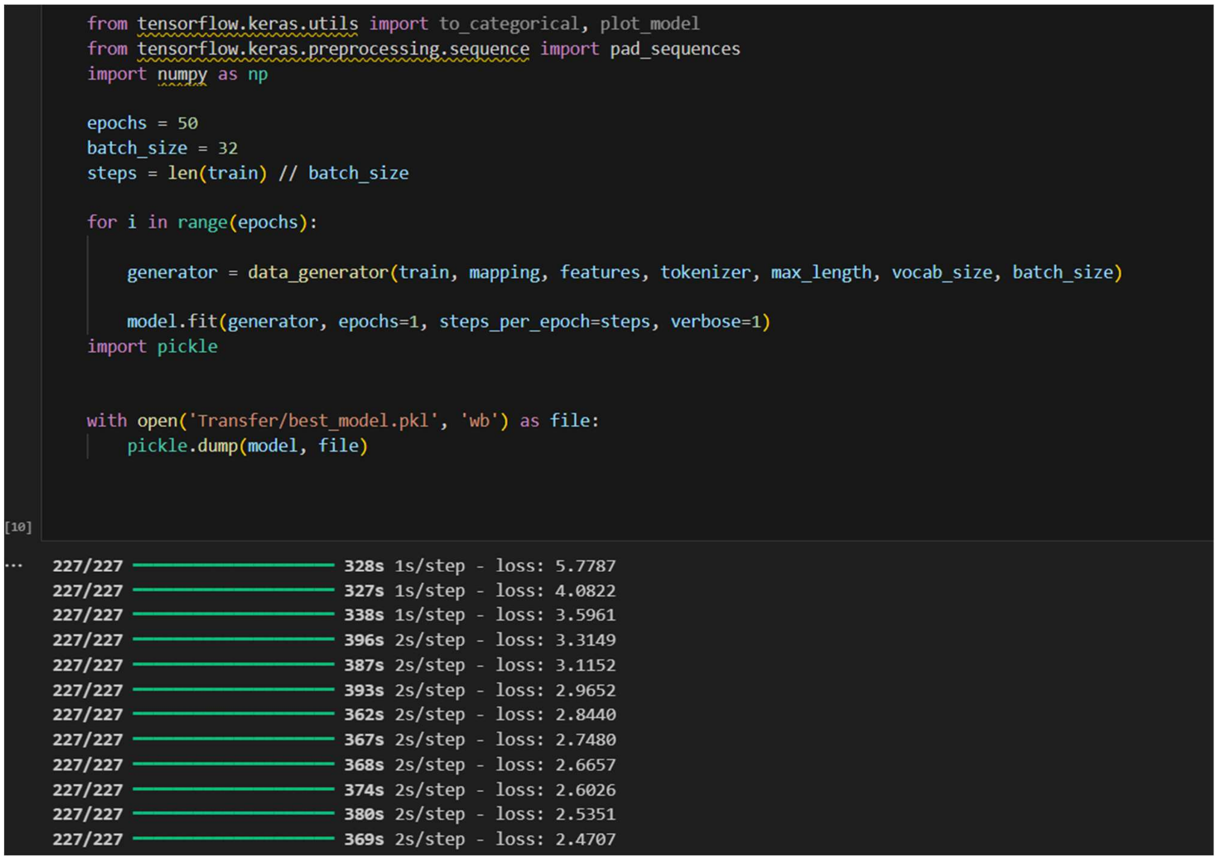
Fig1. Image Preprocessing

Implementation



```python
from keras.layers import Input, Dropout, Dense, Embedding, LSTM, add
from keras.models import Model
from tensorflow.keras.utils import plot_model


inputs1 = Input(shape=(4096,), name="image")
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)

inputs2 = Input(shape=(max_length,), name="text")
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')

try:
    plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
    print("Model plot saved to 'model_plot.png'")
except Exception as e:
    print(f"Error plotting the model: {e}")
```

Fig 2. Model Structure

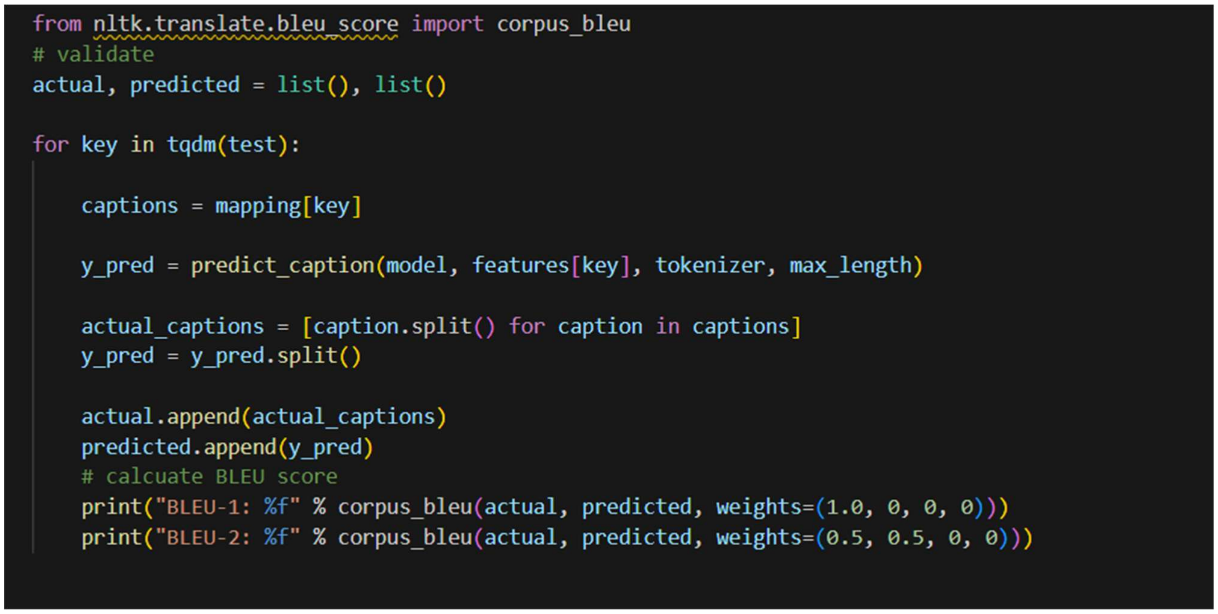


```python
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

epochs = 50
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):

    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, batch_size)

    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
import pickle


with open('Transfer/best_model.pkl', 'wb') as file:
    pickle.dump(model, file)
```

```
[10]
...  227/227 ━━━━━━━━━━━━━━━━━━━━ 328s 1s/step - loss: 5.7787
     227/227 ━━━━━━━━━━━━━━━━━━━━ 327s 1s/step - loss: 4.0822
     227/227 ━━━━━━━━━━━━━━━━━━━━ 338s 1s/step - loss: 3.5961
     227/227 ━━━━━━━━━━━━━━━━━━━━ 396s 2s/step - loss: 3.3149
     227/227 ━━━━━━━━━━━━━━━━━━━━ 387s 2s/step - loss: 3.1152
     227/227 ━━━━━━━━━━━━━━━━━━━━ 393s 2s/step - loss: 2.9652
     227/227 ━━━━━━━━━━━━━━━━━━━━ 362s 2s/step - loss: 2.8440
     227/227 ━━━━━━━━━━━━━━━━━━━━ 367s 2s/step - loss: 2.7480
     227/227 ━━━━━━━━━━━━━━━━━━━━ 368s 2s/step - loss: 2.6657
     227/227 ━━━━━━━━━━━━━━━━━━━━ 374s 2s/step - loss: 2.6026
     227/227 ━━━━━━━━━━━━━━━━━━━━ 380s 2s/step - loss: 2.5351
     227/227 ━━━━━━━━━━━━━━━━━━━━ 369s 2s/step - loss: 2.4707
```

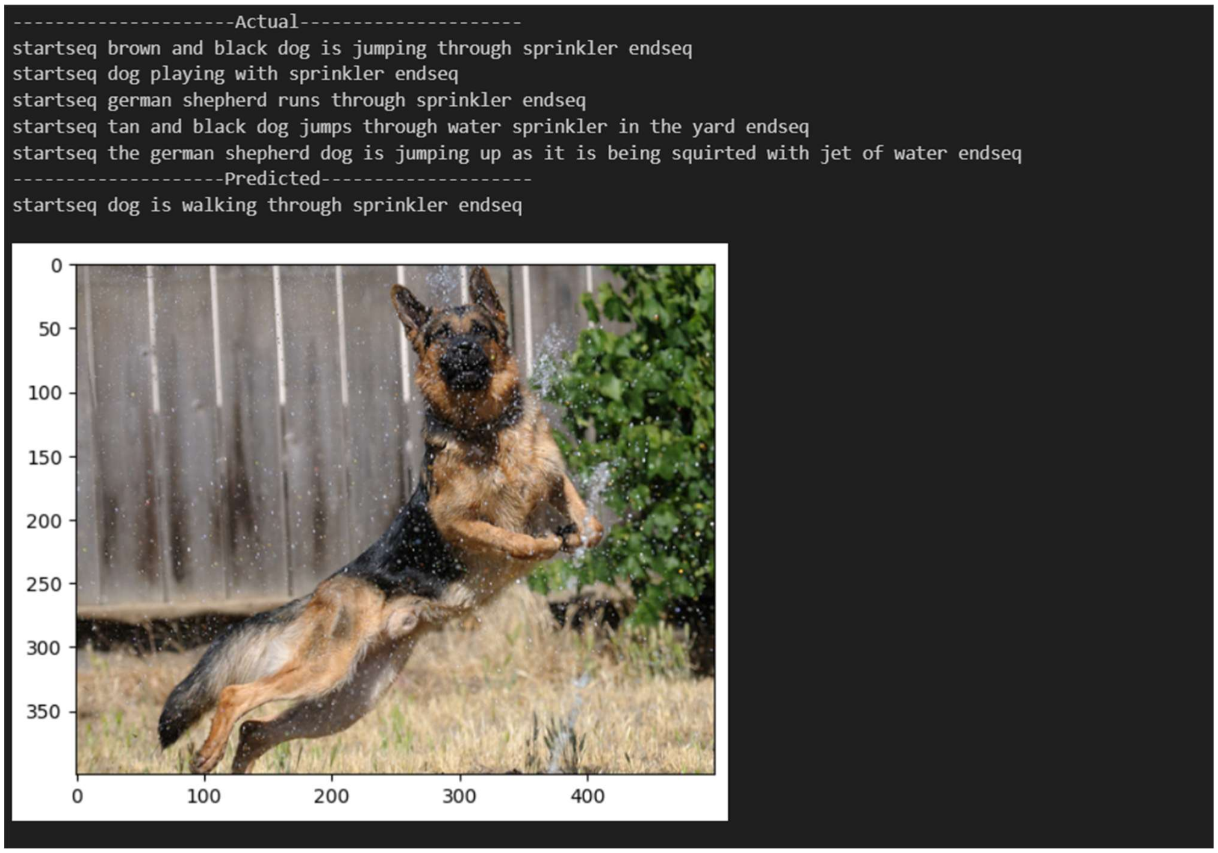
Fig 3. Training Process

Implementation



```
from nltk.translate.bleu_score import corpus_bleu
# validate
actual, predicted = list(), list()

for key in tqdm(test):

    captions = mapping[key]

    y_pred = predict_caption(model, features[key], tokenizer, max_length)

    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()

    actual.append(actual_captions)
    predicted.append(y_pred)
    # calcuate BLEU score
    print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```

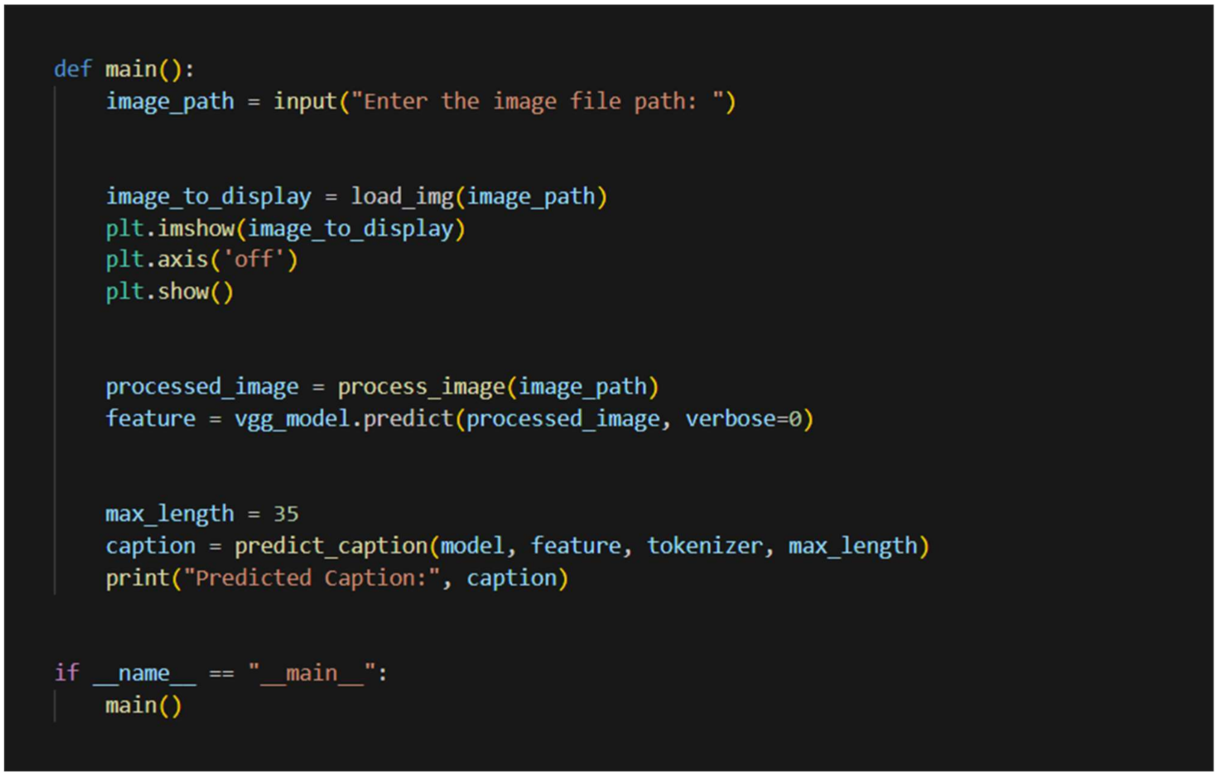
Fig 4. Validation



```
---------------------Actual---------------------
startseq brown and black dog is jumping through sprinkler endseq
startseq dog playing with sprinkler endseq
startseq german shepherd runs through sprinkler endseq
startseq tan and black dog jumps through water sprinkler in the yard endseq
startseq the german shepherd dog is jumping up as it is being squirted with jet of water endseq
--------------------Predicted--------------------
startseq dog is walking through sprinkler endseq
```

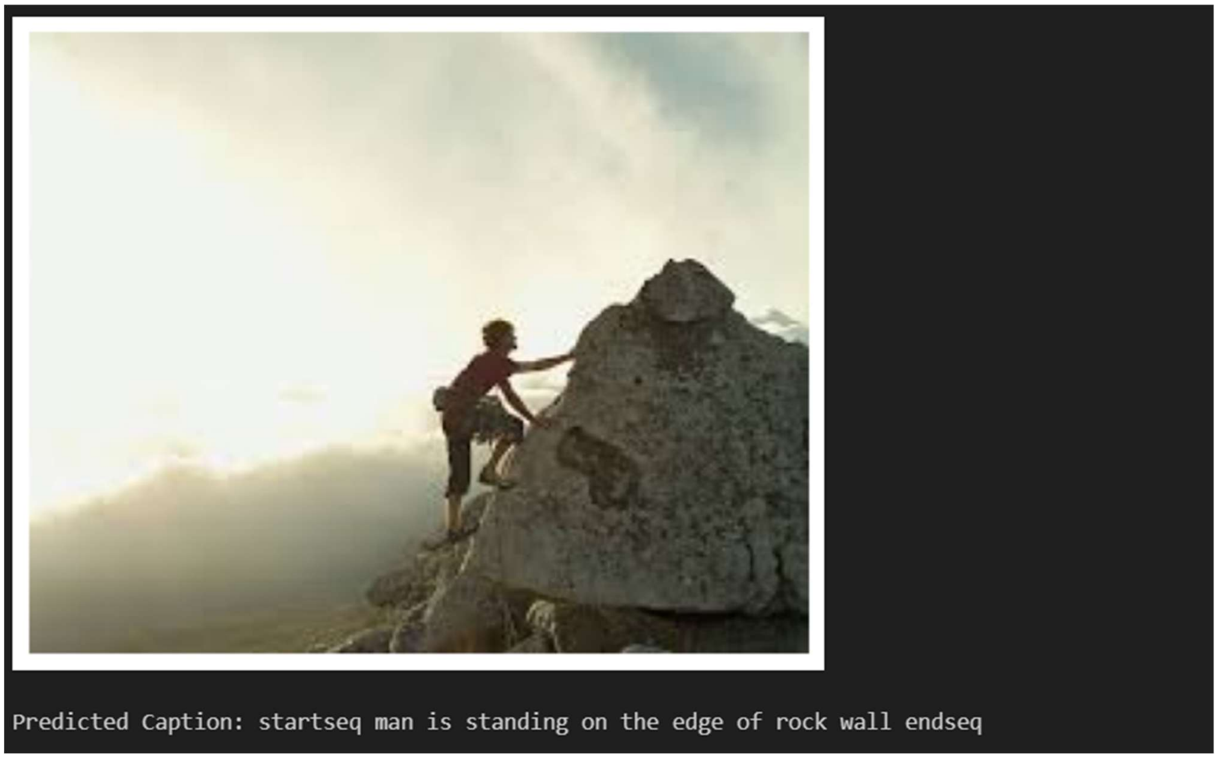
Fig 5. Test Output

Output



```
def main():
    image_path = input("Enter the image file path: ")


    image_to_display = load_img(image_path)
    plt.imshow(image_to_display)
    plt.axis('off')
    plt.show()


    processed_image = process_image(image_path)
    feature = vgg_model.predict(processed_image, verbose=0)


    max_length = 35
    caption = predict_caption(model, feature, tokenizer, max_length)
    print("Predicted Caption:", caption)


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

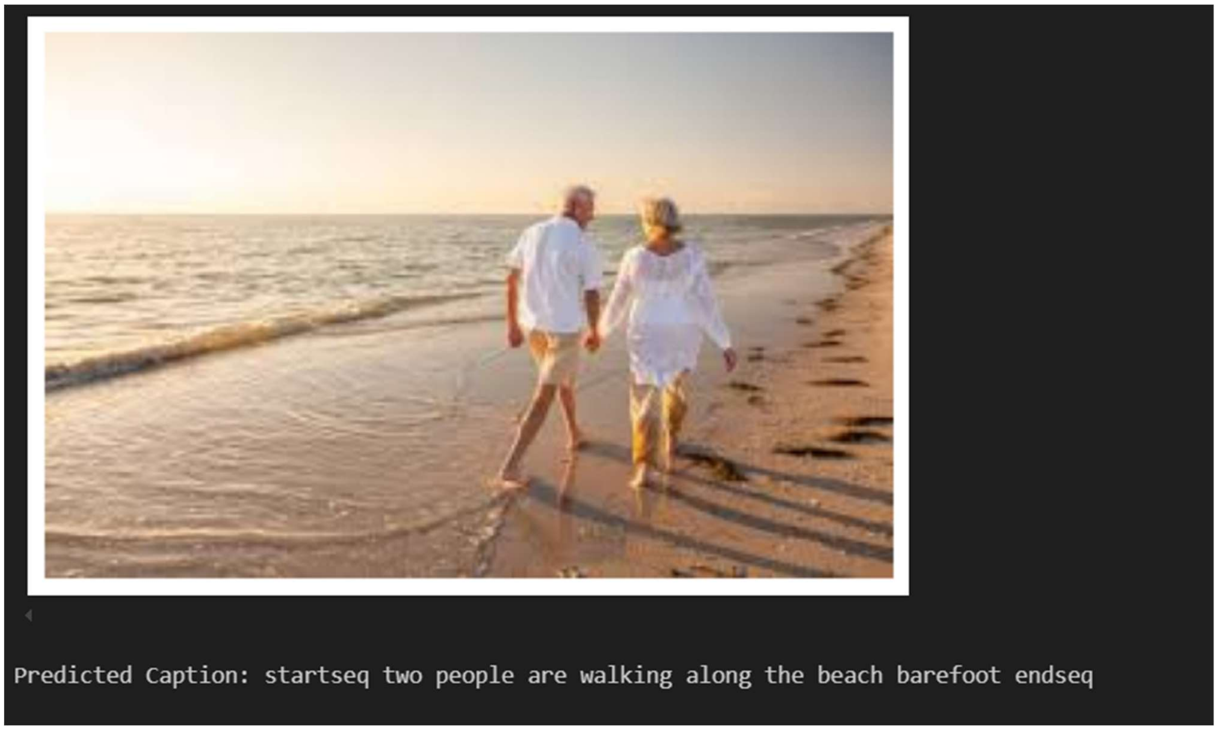
Fig 1. Implementing model



```
Predicted Caption: startseq man is standing on the edge of rock wall endseq
```

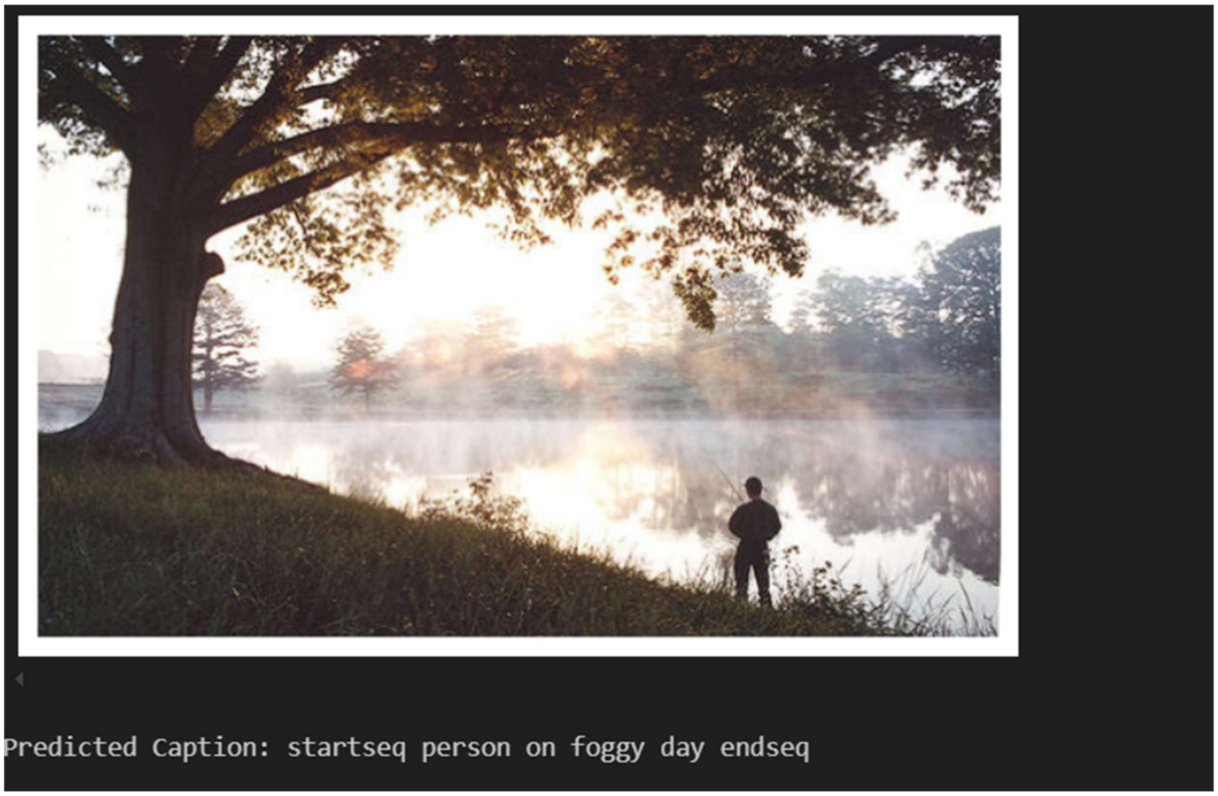
Fig 2. Test Image 1 output

Output



```
Predicted Caption: startseq two people are walking along the beach barefoot endseq
```

Fig 3. Test Image 2 Output



```
Predicted Caption: startseq person on foggy day endseq
```

Fig 4. Test Image 3 Output

References

1. *Vinyals et al. (2015). Show and Tell: A Neural Image Caption Generator. https://arxiv.org/pdf/1411.4555*
2. *Merge and Inject: A Multi-Modality Fusion Approach for Image Captioning (Ma et al., 2018). https://www.sciencedirect.com/science/article/abs/pii/S0925231218313213*
3. *Karpathy & Fei-Fei (2015). Deep Visual-Semantic Alignments. https://arxiv.org/pdf/1412.2306*
4. *Yao et al. (2017). Exploring Transfer Learning in Image Captioning. https://arxiv.org/pdf/1708.02043*
5. *Vaswani et al. (2017). Attention Is All You Need. https://arxiv.org/pdf/1706.03762*
6. *Li et al. (2020). Unifying Vision and Language Tasks via Pre-trained Transformers. https://arxiv.org/pdf/2005.05298*
7. *Donahue et al. (2015). Image Captioning with Deep Bidirectional LSTMs. https://arxiv.org/pdf/1504.06692*
8. *Ren et al. (2016). Faster R-CNN: Towards Real-Time Object Detection. https://arxiv.org/pdf/1506.01497*
9. *Huang et al. (2019). Stacked Cross Attention Networks for Image Captioning. https://arxiv.org/pdf/1803.08024*
10. *Hossain et al. (2019). Comprehensive Survey of Image Captioning Techniques. https://arxiv.org/pdf/1810.04020*
11. *Flickr8K dataset. https://www.kaggle.com/datasets/adityajn105/flickr8k*
12. *VGG 16. https://keras.io/api/applications/vgg/*