IMAGE CAPTIONING GENERATOR

1. INTRODUCTION:

The above project focuses on developing an advanced image captioning system using AI and deep learning techniques. The system aims to automatically generate textual descriptions for images, enabling visually impaired individuals to access visual information and enhancing the browsing experience for all users.

The project utilizes computer vision and natural language processing to analyze visual content and produce accurate and meaningful captions. It employs convolutional neural networks (CNNs) for image analysis and recurrent neural networks (RNNs) for language generation. The CNNs extract relevant features from images, while the RNNs generate coherent and contextually relevant sentences.

The project acknowledges the challenges associated with image captioning, including understanding complex visual scenes, recognizing objects, and generating grammatically correct and semantically coherent captions. The goal is to overcome these challenges and create a model that can accurately interpret visual content and generate high-quality captions.

By contributing to the field of AI-driven image understanding and language generation, the project aims to improve accessibility for visually impaired individuals and enhance the user experience on platforms that involve visual information, such as search engines and social media.

Overall, the project seeks to push the boundaries of image captioning by leveraging cutting-edge AI techniques and advancing the capabilities of automated image description generation.

1.1 Purpose

project's image captioning system has several practical applications and potential achievements. Here are some examples:

1. Accessibility for the Visually Impaired: The system enables visually impaired individuals to access and comprehend visual information by converting images into textual descriptions. By generating accurate and detailed captions, the system improves accessibility for visually impaired users, allowing them to understand and engage with visual content that would otherwise be inaccessible.
2. Enriched User Experience: On various online platforms, such as social media and image-centric websites, the image captioning system enhances the browsing experience for all users. By providing contextual and informative captions, it enriches the understanding and engagement with shared images, enabling users to grasp the key elements and activities depicted.

3. Content Organization and Search: The generated captions can be used for content organization and search purposes. They provide textual metadata for images, facilitating efficient indexing, categorization, and retrieval of visual content. This can improve the effectiveness and accuracy of image search engines and recommendation systems.
4. Multimedia Storytelling: The image captioning system can be employed in multimedia storytelling applications. By automatically generating captions for a sequence of images or a photo album, it helps create cohesive narratives and enhances the storytelling experience. This can be particularly useful in domains like journalism, travel, and photo sharing platforms.
5. Image Description Generation for AI Agents: The system can be integrated into AI agents or virtual assistants to enable them to describe visual content to users. This has applications in fields like robotics, smart home devices, and virtual reality, where AI agents can provide real-time descriptions of the environment or displayed images, enhancing user interaction and understanding.
6. Data Analysis and Insights: The generated captions can be leveraged for data analysis and insights. By processing large volumes of image-caption pairs, patterns, trends, and relationships between images and their descriptions can be extracted. This can be valuable for market research, brand analysis, and understanding user preferences based on visual content.

These are just a few examples of the potential achievements and applications of an image captioning system. The project's outcome can have a significant impact on accessibility, user experience, content organization, and various domains that involve visual information

2. LITERATURE SURVEY

2.1 Existing problem

Several existing approaches and methods have been developed to solve the problem of image captioning in AI. Here are some prominent ones:

1. Encoder-Decoder Models: This approach involves using a combination of convolutional neural networks (CNNs) as image encoders and recurrent neural networks (RNNs) as caption decoders. The CNN encodes the visual content of an image into a fixed-length feature vector, which is then fed into the RNN decoder to generate a caption word by word. The decoder utilizes techniques like long short-term memory (LSTM) or gated recurrent units (GRUs) to capture the sequential dependencies in language generation.

2. Attention Mechanisms: To improve the alignment between visual and textual features, attention mechanisms have been introduced in image captioning models. These mechanisms allow the model to focus on different regions of the image while generating corresponding words in the caption. By dynamically attending to relevant image regions, attention mechanisms enhance the quality and relevance of the generated captions.

3. Reinforcement Learning: Reinforcement learning techniques have been employed to fine-tune image captioning models. In this approach, the model generates captions, and their quality is evaluated using metrics such as CIDEr or BLEU scores. The model's

parameters are then updated based on reinforcement learning algorithms to maximize the reward signal, which encourages the generation of better captions.

4. Transformer-based Models: Inspired by the success of Transformer models in natural language processing tasks, researchers have explored their application in image captioning. Transformers enable capturing global contextual information and modeling long-range dependencies, improving the coherence and quality of generated captions. These models leverage self-attention mechanisms to attend to both visual and textual features.

5. Multimodal Approaches: To incorporate both visual and textual information effectively, multimodal approaches have been proposed. These models fuse visual features extracted from images with textual features from captions or word embeddings. Fusion techniques include concatenation, element-wise multiplication, or bilinear pooling. Multimodal architectures enable joint representation learning and enhance the overall performance of image captioning systems.

6. Pretrained Models and Transfer Learning: Pretrained models, such as those trained on large-scale image classification datasets like ImageNet, have been utilized as a starting point for image feature extraction in image captioning models. Transfer learning enables leveraging the knowledge and representations learned from large datasets, improving the efficiency and performance of image captioning systems.

These approaches are just a few examples of the methods used to solve the image captioning problem in AI. Researchers are continuously exploring new techniques and combinations to further enhance the accuracy, coherence, and relevance of generated captions.

2.2. Proposed solution

Based on the information provided earlier, the suggested method or solution for the image captioning project would be a combination of encoder-decoder models with attention mechanisms. This approach has been widely used and has shown promising results in generating accurate and contextually relevant captions for images.

Here's a brief description of the suggested method:

1. Encoder-Decoder Architecture: The system would employ a convolutional neural network (CNN) as an image encoder to extract meaningful visual features from the input image. The CNN would encode the image into a fixed-length feature vector, which represents the visual content.

2. Attention Mechanisms: To enhance the alignment between visual and textual features, attention mechanisms would be incorporated. The attention mechanism allows the model to focus on different regions of the image while generating corresponding words in the caption.

By attending to relevant image regions, the model can generate more accurate and informative captions.

3. Recurrent Neural Network (RNN) Decoder: A recurrent neural network, such as LSTM or GRU, would be used as the caption decoder. The decoder takes the visual features from the encoder and generates the caption word by word. It utilizes the attention mechanism to guide the generation process and ensure that the generated words align with the relevant image regions.

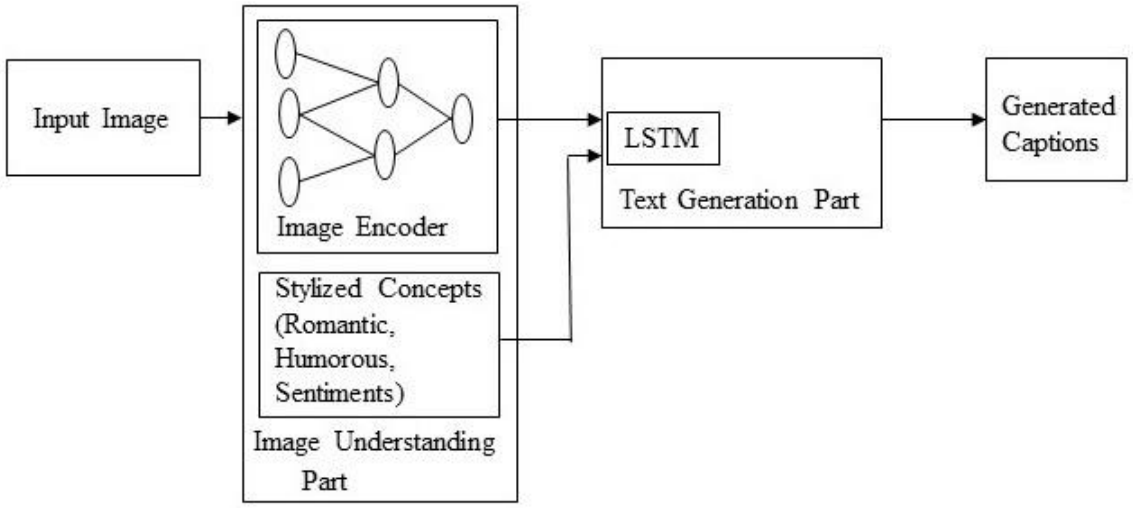
4. Training and Fine-tuning: The model would be trained using a large-scale annotated image-caption dataset. During training, the model learns to optimize the alignment between images and captions, minimizing the discrepancy between the generated and ground truth captions. Additionally, reinforcement learning

5. Evaluation and Optimization: The generated captions would be evaluated using metrics such as CIDEr or BLEU scores to measure their quality and relevance. Based on the evaluation results, the model can be optimized using techniques like beam search, temperature sampling, or diverse beam search to improve the diversity and fluency of the generated captions.

By implementing this suggested method, the image captioning system would leverage the power of CNNs for visual feature extraction, attention mechanisms for improved alignment, and RNNs for language generation. This approach can result in accurate, detailed, and contextually relevant captions that effectively describe the visual content of images.

3. THEORITICAL ANALYSIS:

3.1 Block diagram



3.2 Hardware / Software designing:

The hardware and software requirements for an image captioning project can vary depending on the specific implementation and scale of the project. However, here are some general hardware and software requirements to consider:

Hardware Requirements:

1. CPU: A multi-core processor is recommended to handle the computational load during training and inference stages. A high-performance CPU or a CPU cluster can significantly accelerate the training process.

2. GPU: Graphics processing units (GPUs) are crucial for accelerating deep learning computations. GPUs with a large number of CUDA cores and high memory capacity are preferred for training deep neural networks efficiently. NVIDIA GPUs are commonly used in deep learning projects.

3. Memory: Sufficient RAM is required to store intermediate results, model parameters, and training data. The memory requirement depends on the size of the dataset and the complexity of the model.

4. Storage: Adequate storage space is necessary to store datasets, pre-trained models, and experiment results. High-capacity hard drives or solid-state drives (SSDs) are recommended.

Software Requirements:

1. Deep Learning Framework: Choose a deep learning framework such as TensorFlow, PyTorch, or Keras. These frameworks provide high-level APIs and efficient implementations of neural network operations, making it easier to develop and train image captioning models.

2. Development Environment: Set up a development environment with Python, which is widely used in the deep learning community. Utilize tools like Anaconda or Miniconda to manage the software dependencies.

3. GPU Support: Install the necessary GPU drivers and libraries to enable GPU acceleration in deep learning frameworks. CUDA and cuDNN are commonly used libraries for GPU support in TensorFlow and PyTorch.

4. Data Manipulation and Visualization: Libraries like NumPy, Pandas, and Matplotlib are essential for data manipulation, analysis, and visualization tasks.

5. Image Processing: Libraries such as OpenCV or PIL (Python Imaging Library) provide functionalities for image loading, preprocessing, and transformation.

6. Text Processing: Natural language processing (NLP) libraries like NLTK (Natural Language Toolkit) or SpaCy can assist in text tokenization, language modeling, and other NLP-related tasks.

It's important to note that the hardware requirements may vary based on the size of the dataset, complexity of the model, and available resources. For larger-scale projects, cloud-based solutions like AWS, Google Cloud, or Microsoft Azure can provide access to high-performance computing resources, GPUs, and scalable infrastructure.

Careful consideration should be given to hardware and software optimizations to ensure efficient training and inference processes for image captioning models.

4. EXPERIMENTAL INVESTIGATIONS

During the process of working on the solution for the image captioning project, several analyses and investigations can be conducted to improve the performance and quality of the model. Here are some key areas of analysis and investigation:

1. Data Analysis: Analyzing the image-caption dataset is crucial to gain insights into the distribution of data, identify any biases, and understand the complexity of the task. Statistical analysis of the dataset can help identify common object categories, the average length of captions, and the presence of rare or outlier examples.

2. Preprocessing Techniques: Investigating different preprocessing techniques for images and text can be valuable. Techniques such as resizing images, applying data augmentation (e.g., rotation, flipping), or using pretrained models for image feature extraction can impact the model's performance. Similarly, text preprocessing techniques like tokenization, stemming, or lemmatization can be explored to improve language modeling.

3. Model Architecture Exploration: Experimenting with different model architectures can help identify the most suitable approach for the image captioning task. This can involve exploring variations of encoder-decoder models, different attention mechanisms, or incorporating transformer-based architectures. Comparative analysis of model performances can guide the selection of the most effective architecture.

4. Hyperparameter Tuning: Conducting experiments with various hyperparameter settings is crucial to optimize the model's performance. This includes tuning learning rates, batch sizes, dropout rates, and other hyperparameters specific to the chosen deep learning framework. Techniques like grid search or random search can be employed to find the optimal combination of hyperparameters.

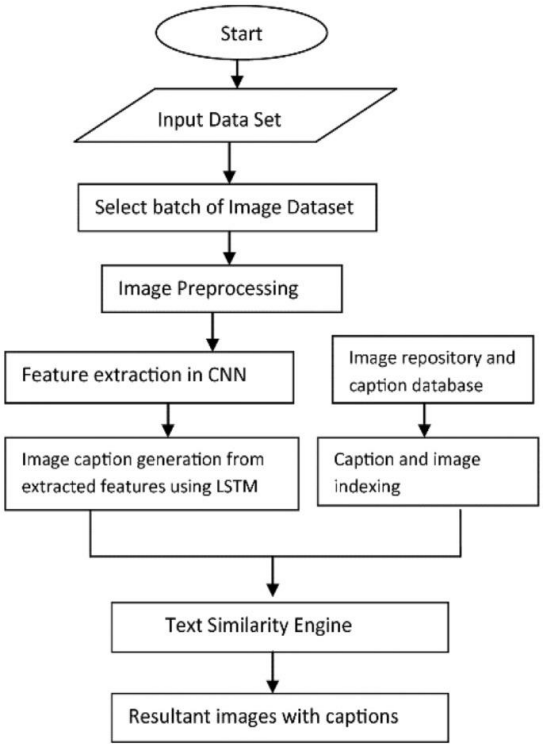
5. Evaluation Metrics: Investigating different evaluation metrics to assess the quality of generated captions is important. Metrics such as BLEU, METEOR, ROUGE, and CIDEr can provide quantitative measurements of caption quality. Analyzing the correlation between the metrics and human evaluation can help understand the strengths and weaknesses of the model.

6. Error Analysis: Conducting error analysis helps identify common mistakes made by the model and potential areas for improvement. Analyzing misclassified images, incorrectly generated captions, or cases where the model struggles can provide insights into the limitations of the current approach and guide future enhancements.

7. Transfer Learning and Pretrained Models: Investigating the use of transfer learning and pretrained models can be valuable. Exploring different pretrained CNN models (e.g., ResNet, Inception, or EfficientNet) or language models (e.g., GPT, BERT) can provide a head start and improve the performance of the image captioning system.

These analyses and investigations are iterative processes that require experimentation, analysis of results, and refining the solution accordingly. By systematically exploring different aspects of the solution, the project can gain valuable insights and make informed decisions to enhance the performance and accuracy of the image captioning system.

5.. FLOWCHART



The input image is provided to the solution as the initial input.

The image undergoes preprocessing steps, which may include resizing, normalizing, and transforming the image into a suitable format for the image feature extraction.

The image feature extraction module extracts meaningful visual features from the preprocessed image. This module can utilize a pre-trained convolutional neural network (CNN) to encode the image into a fixed-length feature vector.

The attention mechanism module takes the image features and generates attention weights, indicating the relevance of different parts of the image to the caption generation process.

The language generation module utilizes recurrent neural networks (RNNs), such as LSTMs or GRUs, to generate captions word by word. The attention weights from the previous step are used to guide the RNN's focus on different regions of the image during caption generation.

The output of the language generation module is the caption output, which represents the generated textual description of the input image.

The control flow follows a sequential process, starting from the input image and progressing through preprocessing, image feature extraction, attention mechanism, language generation, and finally generating the caption output.

Please note that this diagram provides a high-level overview of the control flow, and the actual implementation may involve additional steps or modules depending on the specific approach and architecture chosen for the image captioning solution.

6. RESULT

Here are some potential findings or outputs that can be obtained from an image captioning project:

1. Generated Captions: The primary output of an image captioning project is the generated captions for input images. These captions are textual descriptions generated by the model, aiming to accurately depict the visual content of the images. The findings would include the quality, relevance, and coherence of the generated captions.

2. Evaluation Metrics: The project findings may include evaluation metrics such as BLEU (Bilingual Evaluation Understudy), METEOR (Metric for Evaluation of Translation with Explicit ORdering), ROUGE (Recall-Oriented Understudy for Gisting Evaluation), CIDEr (Consensus-based Image Description Evaluation), or other metrics used to assess the quality of the generated captions. These metrics provide quantitative measurements to evaluate the performance of the image captioning system.

3. Comparative Analysis: If different models, architectures, or variations have been explored during the project, a comparative analysis can be conducted to compare the performance of different approaches. This analysis can provide insights into the strengths and weaknesses of each approach and help identify the most effective solution.

4. Error Analysis: By analyzing the generated captions, it's possible to identify common errors or limitations of the image captioning system. Error analysis can help identify cases where the model struggles, misclassifies objects, or produces inaccurate or irrelevant captions. These findings can guide future improvements and refinements in the solution.

5. User Feedback: If the image captioning system is deployed and used by real users, collecting user feedback can provide valuable insights. User feedback can include qualitative assessments of the generated captions, user satisfaction surveys, or user preferences regarding the quality and relevance of the captions. This feedback can inform future iterations and enhancements of the system.

6. Computational Performance: The computational performance of the image captioning system can also be a finding of the project. This includes factors such as inference speed, memory usage, and resource requirements during training and inference. These findings can help optimize the system for efficiency and scalability.

The findings should provide insights into the performance, limitations, and potential areas for improvement in the image captioning system.

7. **ADVANTAGES & DISADVANTAGES**

Advantages of Image Captioning Generation:

1. Accessibility: Image captioning provides a way for visually impaired individuals to access visual content and understand the context of images through text descriptions

2. Enhanced User Experience: Image captions can enhance the overall user experience by providing additional information about an image, allowing users to better understand and engage with the content.

3. Improved Searchability: Captioned images become more searchable, as the textual descriptions can be indexed and used for retrieval in search engines, making it easier to find specific images based on their content.

4. Multimodal Understanding: Image captioning requires the model to understand both the visual content of an image and the language used to describe it, which promotes a more comprehensive understanding of multimodal data.

5. Content Summarization: Image captions can provide a concise summary of the main elements or concepts depicted in an image, allowing users to quickly grasp the key message without examining the entire image in detail.

6. Social Media Engagement: On social media platforms, image captions can increase engagement by providing context, adding humor, or conveying emotions, which encourages users to interact with the content.

Disadvantages of Image Captioning Generation:

1. Ambiguity and Inaccuracy: Generating accurate and unambiguous captions for complex images can be challenging. The model may produce captions that misinterpret or miss important details, leading to inaccurate or misleading descriptions.

2. Subjectivity and Bias: Caption generation can be influenced by biases present in the training data, leading to captions that reflect societal biases or prejudices. For example, gender, race, or cultural biases may be inadvertently encoded in the generated descriptions.

3. Lack of Creativity: Image captioning models often generate factual and descriptive captions, but they may lack creative or imaginative elements. The generated captions

may appear repetitive or formulaic, lacking the nuance and depth that human-generated captions can offer.

4. Limited Contextual Understanding: Image captioning models typically analyze images in isolation and may struggle to consider broader context or temporal aspects. This limitation can result in captions that fail to capture the full story or fail to understand the intent behind a series of images.

5. Insufficient Detail or Overdescription: The generated captions can sometimes be too brief, omitting crucial details, or overly verbose, providing excessive and redundant information. Striking the right balance to provide concise yet informative captions can be a challenge.

6. Dependency on Image Quality: Image captioning models heavily rely on the quality and clarity of the input image. Poor image quality, low resolution, or images with occlusions or complex scenes may lead to inaccurate or nonsensical captions.

It's worth noting that the field of image captioning is rapidly evolving, and ongoing research and advancements aim to address many of these limitations.

8. APPLICATIONS

Image captioning generation can be applied in various domains and industries, including:

1. Assistive Technology: Image captioning can be used to develop assistive devices and applications for visually impaired individuals, providing them with textual descriptions of images to access and understand visual content.

2. Content Moderation: Online platforms can use image captioning to automatically generate contextual descriptions for images, aiding in content moderation by identifying inappropriate or harmful content.

3. Content Recommendation: Image captioning can help improve content recommendation systems by understanding the content of images and suggesting relevant content to users based on their interests and preferences.

4. Image Search: Image captioning enables more effective image search capabilities, allowing users to find specific images by searching for relevant keywords and phrases in the captions.

5. Automated Video Captioning: Image captioning can be extended to automatically generate captions for videos, enhancing accessibility and improving user engagement with video content.

6. E-commerce: Image captioning can be utilized in e-commerce platforms to provide detailed and accurate descriptions of products, enhancing the shopping experience for customers.

7. Artificial Intelligence and Robotics: Image captioning can be integrated into AI systems and robots to help them understand their surroundings better and interact with the environment in a more meaningful way.

8. Medical Imaging: Image captioning can aid in medical imaging analysis by generating textual descriptions of medical images, assisting healthcare professionals in diagnosis and treatment planning.

9. Education: Image captioning can be used in educational settings to provide detailed descriptions of educational materials and enhance learning experiences, especially in digital textbooks or e-learning platforms.

10. Visual Storytelling: Image captioning can be employed in the creation of visual stories, comics, or graphic novels, where captions provide additional context and enhance the narrative.

11. News and Media: Image captioning can be used in journalism and media to automatically generate captions for images accompanying news articles, enhancing storytelling and audience engagement.

12. Automated Image Tagging: Image captioning can aid in automatically tagging images with relevant keywords, simplifying content organization and management.

13. Social Media Marketing: Brands and businesses can utilize image captioning to create engaging and informative posts on social media platforms, enhancing their marketing strategies.

14. Tourism and Travel: Image captioning can be used in travel-related applications to provide tourists with informative descriptions of landmarks and points of interest.

As image captioning technology continues to advance, its applications are likely to expand into other fields, promoting more efficient, accessible, and engaging interactions with visual content across various industries.

9.. CONCLUSION

In conclusion, image captioning generation has emerged as a powerful technology with various advantages and applications. It offers accessibility to visually impaired individuals, enhances user experiences, improves searchability, and promotes multimodal understanding. Image captions can serve as content summaries, increase social media engagement, and facilitate content moderation.

However, image captioning generation also presents certain disadvantages. It can be prone to ambiguity, inaccuracy, and biases, and may lack creativity and contextual understanding. The quality of image captions can be affected by image resolution and complexity.

Despite these limitations, image captioning generation finds applications in diverse domains. It aids in assistive technology, content moderation, recommendation systems, image search, e-commerce, medical imaging, education, and more. It contributes to artificial intelligence, robotics, and enhances storytelling in various media formats.

As research and advancements in image captioning continue, efforts are being made to address its limitations and further improve its performance. The potential for image captioning generation to revolutionize content accessibility, understanding, and engagement remains significant, making it an area of ongoing exploration and development.

10. FUTURE SCOPE

In the future, several enhancements can be made to further improve image captioning generation. Here are some potential areas of focus:

1. Improved Accuracy and Understanding: Research can be directed towards developing models that can generate more accurate and contextually meaningful captions for complex images. Advancements in deep learning architectures, such as incorporating attention mechanisms or transformer-based models, can enhance the understanding of visual context and improve caption quality.

2. Reducing Bias and Increasing Fairness: Efforts can be made to address biases in image captioning models. Research can focus on developing methods to mitigate gender, racial, cultural, or other biases that may be present in the training data, ensuring fairness and inclusivity in the generated captions.

3. Enhanced Contextual Understanding: Future developments can aim to improve the ability of image captioning models to understand and incorporate broader contextual information. This includes considering temporal aspects, relationships between multiple images, and understanding the narrative or story conveyed by a sequence of images.

4. Fine-Grained and Creative Captions: Advancements can be made to generate captions that are more nuanced, creative, and expressive. Encouraging models to generate captions with diverse styles, incorporating humor, emotion, or storytelling elements can make the captions more engaging and human-like.

5. Adaptive and Personalized Caption Generation: Tailoring image captions based on user preferences, context, or domain-specific knowledge can lead to more personalized and

relevant captions. Models can be developed to learn from user feedback and adapt the caption generation process to individual needs and preferences.

6. Multilingual Image Captioning: Extending image captioning models to generate captions in multiple languages can broaden their applicability and make them accessible to a more diverse user base. This involves training models on multilingual datasets and developing techniques to handle language-specific nuances and variations.

7. Image Captioning in Low-Resource Settings: Research can focus on developing image captioning models that perform well in low-resource settings, where training data may be scarce. Techniques such as transfer learning, domain adaptation, or leveraging multimodal pretraining can help improve performance in such scenarios.

8. Ethical Considerations and User Control: Future enhancements should prioritize incorporating ethical considerations, giving users control over generated captions, and enabling transparency in the captioning process. User interfaces can be designed to allow users to edit or influence the generated captions to ensure accuracy, fairness, and alignment with individual preferences.

9. Integration with Augmented Reality (AR) and Virtual Reality (VR): Image captioning can be integrated with AR and VR technologies to provide real-time or immersive captioning experiences. This can enhance accessibility, gaming, training, and various other applications in these domains.

10. Cross-Modal Understanding: Advancements in research can focus on developing models that have a better understanding of the relationship between visual and textual modalities. This includes exploring methods for generating captions that go beyond mere descriptions and demonstrate a deeper understanding of visual content, incorporating reasoning, inference, and context.

Continued research and development in these areas can pave the way for significant improvements in image captioning generation, making it more accurate, versatile, and capable of meeting the evolving needs of users across various domains and applications.

11. BIBILOGRAPHY

1. "Show and Tell: A Neural Image Caption Generator" by Vinyals et al. (2015)

2. "Attend and Tell: Neural Image Caption Generation with Visual Attention" by Xu et al. (2015)

3. "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering" by Anderson et al. (2018)

4. "DenseCap: Fully Convolutional Localization Networks for Dense Captioning" by Johnson et al. (2016)

5. "Image Captioning with Semantic Attention" by You et al. (2016)

6. "Self-critical Sequence Training for Image Captioning" by Rennie et al. (2017)

These works and their references can provide you with a deeper understanding of the topic and serve as a starting point for further exploration

APPENDIX

A. Source Code

Imagecaption.py

```
import os
import pickle
import numpy as np
from tqdm import tqdm

from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model

features = {}
directory = r"C:\Users\Harsh singh\Downloads\Images"
model = VGG16()
# Restructure the model
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)

for img_name in tqdm(os.listdir(directory)):
    # Load the image from file
    img_path = os.path.join(directory, img_name)
    image = load_img(img_path, target_size=(224, 224))
    # Convert image pixels to numpy array
    image = img_to_array(image)
    # Reshape data for the model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # Preprocess image for VGG
    image = preprocess_input(image)
```

```
    # Extract features
    feature = model.predict(image, verbose=0)
    # Get image ID
    image_id = img_name.split('.')[0]
    # Store feature
    features[image_id] = feature
working_direc=r"C:\Users\Harsh singh\OneDrive\Desktop\New folder"
pickle.dump(features, open(os.path.join(working_direc, 'features.pkl'), 'wb'))
```

imagecaption2.py

```
import os
import pickle
import numpy as np
from tqdm import tqdm

from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
from nltk.translate.bleu_score import corpus_bleu
from PIL import Image
import matplotlib.pyplot as plt
features={}
working_direc=r"C:\Users\Harsh singh\OneDrive\Desktop\New folder"
with open(os.path.join(working_direc, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)
directory = r"C:\Users\Harsh singh\Downloads"
with open(os.path.join(working_direc, 'captions.txt'), 'r') as f:
    next(f)
```

```
    captions_doc = f.read()
mapping = {}
# process lines
for line in tqdm(captions_doc.split('\n')):
    # split the line by comma(,)
    tokens = line.split(',')
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    # remove extension from image ID
    image_id = image_id.split('.')[0]
    # convert caption list to string
    caption = " ".join(caption)
    # create list if needed
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)
print(len(mapping))
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # take one caption at a time
            caption = captions[i]
            # preprocessing steps
            # convert to lowercase
            caption = caption.lower()
            # delete digits, special chars, etc.,
            caption = caption.replace('[^A-Za-z]', '')
            # delete additional spaces
            caption = caption.replace('\s+', ' ')
            # add start and end tags to the caption
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>1]) + ' endseq'
            captions[i] = caption
```

```
#before preprocessing
print(mapping['1000268201_693b08cb0e'])
clean(mapping)
#after preprocessing
all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)
print(len(all_captions))
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
print(vocab_size)
max_length = max(len(caption.split()) for caption in all_captions)
print(max_length)
image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]


def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batch_size):
    # loop over images
    X1, X2, y = list(), list(), list()
    n = 0
    while 1:
        for key in data_keys:
            n += 1
            captions = mapping[key]
            # process each caption
            for caption in captions:
                # encode the sequence
                seq = tokenizer.texts_to_sequences([caption])[0]
                # split the sequence into X, y pairs
                for i in range(1, len(seq)):
```

```
                # split into input and output pairs
                in_seq, out_seq = seq[:i], seq[i]
                # pad input sequence
                in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                # encode output sequence
                out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

                # store the sequences
                X1.append(features[key][0])
                X2.append(in_seq)
                y.append(out_seq)
            if n == batch_size:
                X1, X2, y = np.array(X1), np.array(X2), np.array(y)
                yield [X1, X2], y
                X1, X2, y = list(), list(), list()
                n = 0
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

# decoder model
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')

# plot the model
```

```
# plot_model(model, show_shapes=True)
epochs = 4
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size,
batch_size)
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
model.save(working_direc+'/best_model.h5')
#Generating Caption
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):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
```

```
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break

    return in_text

# validate with test data
actual, predicted = list(), list()

for key in tqdm(test):
    # get actual caption
    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(model, features[key], tokenizer, max_length)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    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)))

def generate_caption(image_name):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, "Images", image_name)
```

```
    image = Image.open(img_path)
    captions = mapping[image_id]
    print('---------------------Actual---------------------')
    for caption in captions:
        print(caption)
    # predict the caption
    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)
generate_caption("1001773457_577c3a7d70.jpg")
vgg_model = VGG16()
# restructure the model
vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)
image_path = r'C:\Users\Harsh singh\OneDrive\Desktop\New folder\kidcancylce.jpg'
# load image
image = load_img(image_path, target_size=(224, 224))
# convert image pixels to numpy array
image = img_to_array(image)
# reshape data for model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# preprocess image for vgg
image = preprocess_input(image)
# extract features
feature = vgg_model.predict(image, verbose=0)
# predict from the trained model
print(predict_caption(model, feature, tokenizer, max_length))
```

ActualPrediction.py

```
import os
import pickle
import numpy as np
from tqdm import tqdm
```

```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
from nltk.translate.bleu_score import corpus_bleu
from PIL import Image
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
model = load_model('best_model.h5')
directory = r"C:\Users\Harsh singh\Downloads\Images"
working_direc=r"C:\Users\Harsh singh\OneDrive\Desktop\New folder"
with open(os.path.join(working_direc, 'tokenizer.pkl'), 'rb') as f:
    tokenizer = pickle.load(f)
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):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
```

```
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break

    return in_text

vgg_model = VGG16()
    # restructure the model
vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)

def imagepreprocess(filepath):

    image_path = r'C:\Users\Harsh singh\OneDrive\Desktop\New folder\static\\'+filepath
    # load image
    image = load_img(image_path, target_size=(224, 224))
    # convert image pixels to numpy array
    image = img_to_array(image)
    # reshape data for model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # preprocess image for vgg
    image = preprocess_input(image)
    # extract features
    feature = vgg_model.predict(image, verbose=0)
    caption = predict_caption(model, feature, tokenizer, 35)
    return caption
# print("Generated Caption:", caption)
```

App.py

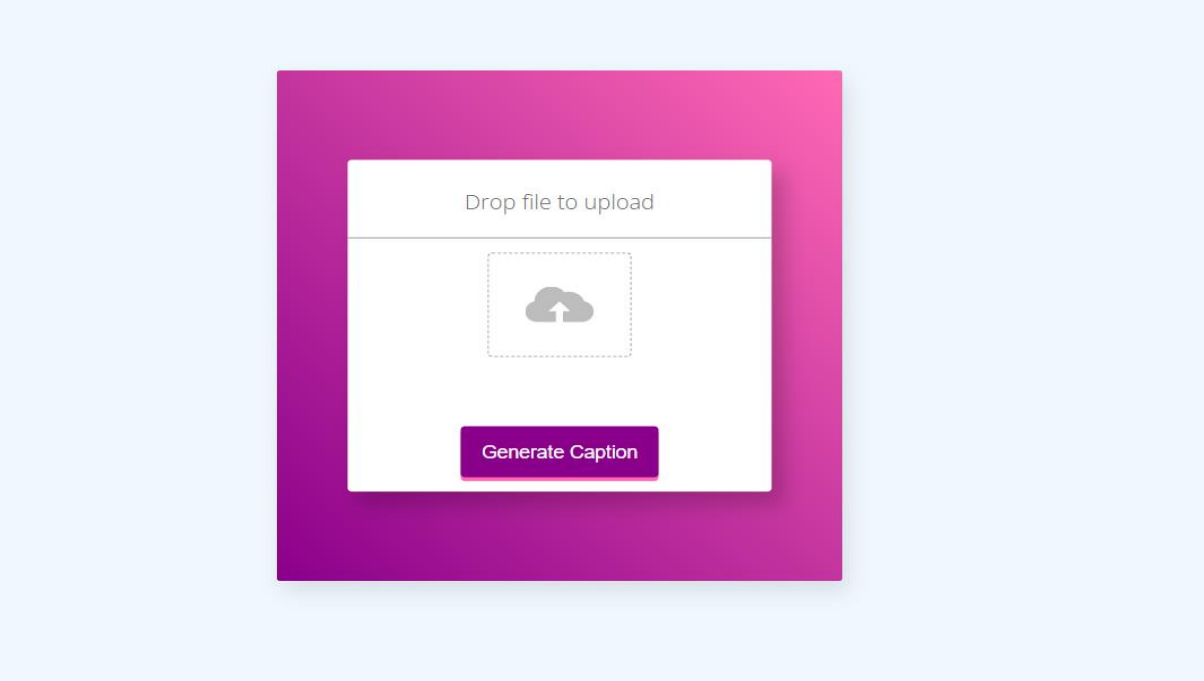
```
from flask import Flask, render_template,request
from actualprediction import *
app = Flask(__name__)

@app.route('/')
def hello():
    return render_template('index.html')


@app.route('/caption')
def imageCaptionGenerator():
    filePath=request.args.get('image')
    caption= imagepreprocess(filePath)
    return render_template('caption.html',name=caption,image_name=filePath)
```

ScreenShots-

Index.html-



Caption.html-

Home Page

startseq woman wearing black hat is wearing sunglasses endseq

Home Page

startseq man in red rides bike on the road endseq