## AutoTale - Creating Captions for Images

A report submitted in partial fulfilment of the requirements for the Award of Degree of

**BACHELOR OF TECHNOLOGY**

In

# COMPUTER SCIENCE AND ENGINEERING

By

P.KAVYA

REG. NO: 22B95A0525

## Blackbucks Engineers Pvt Ltd

(Duration: DEC 2024 to APRIL 2025)



DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

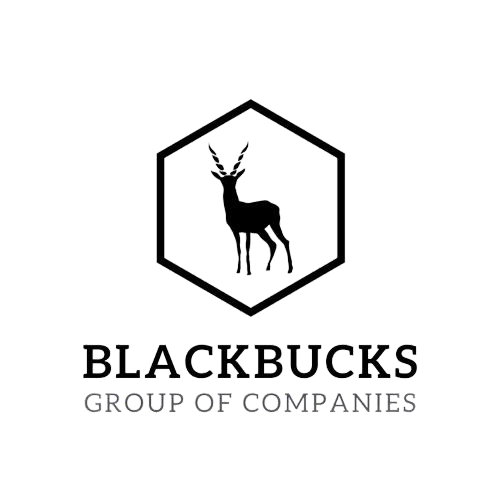
S.R.K.R. ENGINEERING COLLEGE (A)

(Affiliated to JNTU, KAKINADA)

BHIMAVARAM-534204, A.P

# blackbucks.pngTABLE OF CONTENTS

1. ABSTRACT 1
2. PROBLEM STATEMENT… 2
3. INTRODUCTION 3-6
4. BLOCK DIAGRAM 7
5. SOFTWARE REQUIREMENTS 8
6. HARDWARE REQUIREMENTS 8
7. PROCEDURE 9-10
8. ALGORITHM CONSIDERED 11
9. PROGRAMMING CODE 12-20
10. CONCLUSION 24
11. FUTURE WORK 25

**ABSTRACT**

This project presents a deep learning-based approach for automatically generating descriptive captions for images. By combining the power of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), particularly VGG16 and LSTM architectures, the model effectively interprets visual content and translates it into meaningful textual descriptions. The Flickr8k dataset, comprising 8,000 images and multiple human-written captions per image, was used for training and evaluation. The model extracts image features using a pre-trained VGG16 network and utilizes an LSTM decoder to generate corresponding captions. Various natural language processing techniques were used for preprocessing, and the model was trained using categorical cross-entropy loss and the Adam optimizer. Evaluation metrics such as BLEU and METEOR scores demonstrate the model's capability to produce relevant and coherent captions. The results indicate promising potential for real-world applications like accessibility tools, content-based image retrieval, and automated tagging systems. Future enhancements include integrating attention mechanisms and transitioning to transformer-based architectures for improved performance.

# blackbucks.pngPROBLEM STATEMENT

In recent years, the exponential growth of digital imagery has created a pressing need for systems that can automatically interpret and describe visual content. Traditional image processing techniques are often inadequate in understanding the complex semantics within an image, especially when it comes to generating meaningful natural language descriptions. The task of image captioning lies at the intersection of computer vision and natural language processing and aims to generate textual descriptions that accurately reflect the content of a given image.

The primary challenge in image captioning is to design a model that can not only understand the objects and actions depicted in the image but also produce grammatically correct and contextually relevant sentences. This project aims to address this challenge by developing a machine learning model that uses Convolutional Neural Networks (CNNs), specifically VGG16, for feature extraction from images, and Long Short-Term Memory (LSTM) networks for generating captions. The model is trained and evaluated using the Flickr8k dataset, which contains images paired with multiple human-written descriptions. The goal is to generate captions that are as close as possible to those written by humans in terms of accuracy, fluency, and relevance.

# blackbucks.pngINTRODUCTION

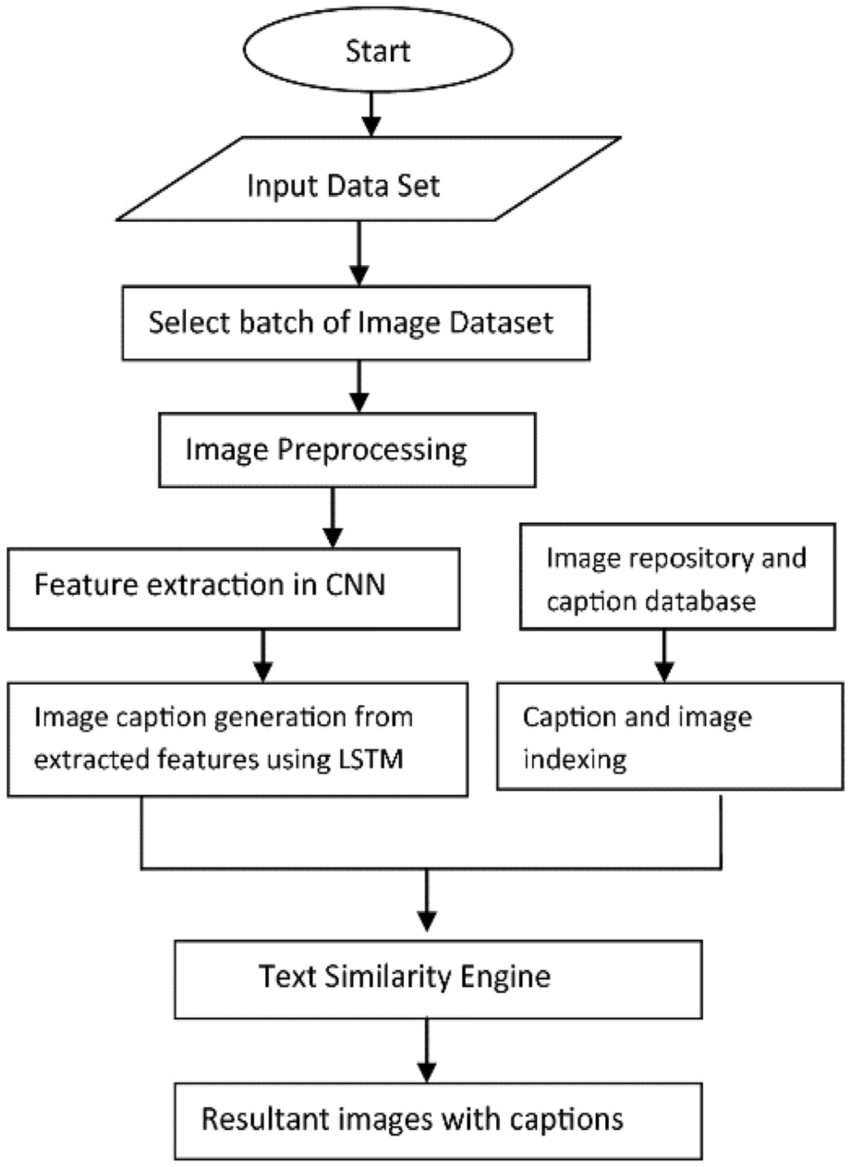
Image captioning is an emerging field that combines the capabilities of computer vision and natural language processing to automatically generate textual descriptions of images. It represents a significant step towards achieving a deeper level of human-computer interaction, where machines are not only able to recognize objects within an image but also describe them using meaningful language.

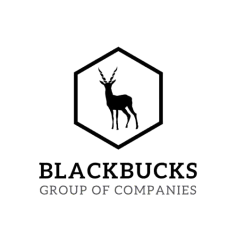
The need for image captioning systems has grown with the increasing volume of digital images generated daily across various platforms. Applications range from helping visually impaired individuals understand visual content, to enhancing image search engines, automatic photo organization, and supporting surveillance systems.

Traditional methods in computer vision mainly focused on object detection and classification. However, they fall short when it comes to generating natural language descriptions that involve context, relationships between objects, and actions. To overcome these limitations, modern deep learning approaches combine Convolutional Neural Networks (CNNs) for image feature extraction and Recurrent Neural Networks (RNNs), such as Long Short-Term Memory (LSTM) networks, for generating sequences of words.

In this project, we propose an image captioning model that leverages the pre-trained VGG16 network to extract meaningful features from images. These features are then passed to an LSTM-based decoder to generate captions word by word. The Flickr8k dataset, which contains 8,000 images with five human-written captions per image, serves as the basis for training and evaluation. Through this project, we aim to demonstrate the effectiveness of combining visual and textual processing models to build an end-to-end image captioning system.

# blackbucks.pngBLOCK DIAGRAM



**SOFTWARE REQUIREMENTS**

One of the most difficult tasks is that, the selection of the software, once system requirement is known that is determining whether a particular software package fits the requirements.

|  |  |
| --- | --- |
| Programming Language | Python |
| Note Book | Google Colab Jupyter Notebook,VS code |
| Operating System | Windows 10 |
| Browser | Google Chrome |

# HARDWARE REQUIREMENTS

The selection of hardware is very important in the existence and proper working of any software. In the selection of hardware, the size and the capacity requirements are also important.

|  |  |
| --- | --- |
| Processor | Intel Core 5 |
| Ram capacity | 4 GB |
| Hard Disk | 512 GB |
| I/O | Keyboard, Mouse, Monitor |

# blackbucks.pngPROCEDURE

The development of the image captioning system followed a systematic, multi-stage process.

Each step—from data collection to model evaluation—played a vital role in ensuring the model's effectiveness and reliability. Below is a detailed explanation of the methodology used in this project.

**1. Dataset Collection**

The first step in the implementation process was the selection and acquisition of a suitable dataset. For this project, the **Flickr8k dataset** was chosen due to its balanced size and high-quality captions. The dataset consists of **8,000 real-world images**, each annotated with **five unique human-generated captions**, offering rich contextual descriptions.

The dataset was downloaded from **Kaggle**, a reputable platform for data science resources. After acquiring the dataset, it was divided into three subsets:

* **Training Set** – used to train the model.
* **Validation Set** – used to fine-tune hyperparameters.
* **Testing Set** – used to assess final model performance.

This split ensured that the model could generalize well and avoided overfitting to the training data.

**2. Data Preprocessing**

To prepare the dataset for training, a comprehensive preprocessing pipeline was implemented. This stage is crucial, as raw captions often contain inconsistencies that can hinder learning. The following steps were performed:

* **Lowercasing**: All captions were converted to lowercase to ensure uniformity. For example, "A Dog Running" and "a dog running" are treated as identical after this step.
* **Cleaning**: Punctuation marks, numbers, and special characters were removed to simplify the vocabulary. Only alphabetic characters were retained to avoid noise in the dataset.
* **Token Marking**: Special tokens such as <start> and <end> were added to every caption. These markers helped the model determine where a sentence begins and ends during both training and generation stages.
* **Vocabulary Creation**: A tokenizer was used to generate a word-index mapping. Each word in the dataset was assigned a unique integer. Rare words were either discarded or replaced with an <unk> (unknown) token depending on their frequency.
* **Caption Padding**: Since captions vary in length, they were padded with zeros to match the



maximum caption length in the dataset. This step ensures that all input sequences are of equal size, which is essential for batch processing.

By the end of preprocessing, each image had five cleaned and tokenized captions ready for training, along with the appropriate vocabulary and padding setup.

**3. Feature Extraction Using VGG16**

To allow the model to "understand" the visual content of an image, deep visual features were extracted using the **VGG16** convolutional neural network (CNN), which is pre-trained on the ImageNet dataset.

* **VGG16 Architecture**: This deep CNN consists of 13 convolutional layers followed by 3 fully connected layers. For this task, the top fully connected layers were removed to access high-level visual representations instead of classification outputs.
* **Image Processing**: Each image was resized to 224×224 pixels (the required input size for VGG16) and normalized using ImageNet-specific preprocessing.
* **Feature Extraction**: The processed images were passed through the VGG16 model to extract a **4096-dimensional feature vector** from the second-to-last dense layer. These vectors represent abstract visual features such as shapes, textures, and object arrangements.
* **Storage**: Extracted features were saved and mapped to their corresponding image IDs for efficient retrieval during training.

This conversion of images into numerical feature vectors enabled seamless integration with the language model used to generate captions.

**4. Model Architecture**

The model developed for this project follows an **encoder-decoder architecture**, which combines two powerful deep learning networks:

**a. Encoder (CNN - VGG16)**

The encoder consists of the feature extractor described earlier. It provides a condensed numerical representation of the image that encodes essential information about objects and spatial composition.

**b. Decoder (RNN - LSTM)**

The decoder is responsible for generating the caption, one word at a time. It uses an LSTM (Long Short-Term Memory) network to handle the sequential nature of text data. The architecture involves:

* **Image Feature Input**:  
  The 4096-dimensional image vector is passed through a dense layer to reduce dimensionality (e.g., to 256 units) and make it compatible with the decoder's input space.
* **Caption Input**:  
  Captions (in integer form) are passed through an **embedding layer** that converts each word



index into a dense 256-dimensional vector. This helps capture semantic meaning.

* **Sequential Modeling**:  
  The embedded sequences are then processed through an LSTM layer with 256 units, which learns dependencies and relationships between words in the caption.
* **Merging Paths**:  
  The outputs of the image (CNN path) and the caption (LSTM path) are merged using an **Add layer**. This fusion allows the model to consider both visual and textual contexts before making a prediction.
* **Output Layer**:  
  Finally, a fully connected dense layer with **softmax activation** predicts the probability distribution over the vocabulary for the next word in the sequence.

This architecture allows the model to take an image and a partial caption as input and predict the most likely next word, repeating the process until the <end> token is generated.

**5. Model Training**

With the architecture in place, the next phase involved training the model. The goal was to enable the model to learn the relationship between image features and corresponding caption sequences.

* **Loss Function**:  
  The model was compiled using **categorical cross-entropy loss**, which is ideal for multi-class classification problems where the model predicts the next word from a fixed vocabulary.
* **Optimizer**:  
  The **Adam optimizer** was used due to its adaptive learning rate and efficiency in handling sparse gradients, which are common in natural language processing tasks.
* **Data Generator**:  
  Since loading the entire dataset into memory was not feasible, a custom **data generator** was created. It yielded batches of:
  + Image feature vectors
  + Input caption sequences (up to the current word)
  + Target words (the next word in the sequence)

For example, the caption “a boy is playing” is broken down into training pairs like:

* + Input: "a", Output: "boy"
  + Input: "a boy", Output: "is"
  + Input: "a boy is", Output: "playing"
* **Epochs & Batch Size**:  
  The model was trained for multiple epochs (typically 20–30), with a batch size of 64. Validation was conducted at the end of each epoch using the validation set to monitor loss



and adjust hyperparameters accordingly.

* **Checkpointing**:  
  To avoid overfitting and retain the best performing model, checkpoints were saved at each epoch where the validation loss improved.

Training time depended on the hardware used (GPU or CPU), dataset size, and model complexity. After training, the model weights were saved for caption generation during inference.

**6. Caption Generation**

After training, the model was used to generate captions for unseen test images. Caption generation followed a sequential approach where the model predicted one word at a time.

* **Step-by-Step Generation**:
  1. An image was first passed through the **VGG16 encoder** to extract its feature vector.
  2. The decoder was initialized with the <start> token.
  3. Using the image feature and the current partial caption, the model predicted the **most likely next word**.
  4. The predicted word was added to the caption and used as the next input.
  5. This process was repeated until the model predicted the <end> token or reached the maximum allowed caption length.
* **Beam Search (Optional)**:  
  While the default method used **greedy decoding** (selecting the most probable next word), **beam search** can also be applied to explore multiple caption paths and select the most coherent one.

This iterative process enables the model to convert a static visual input into a meaningful textual description, closely mimicking human perception and language generation.

**7. Evaluation**

Evaluating the performance of the image captioning model is critical to understanding how well it interprets and describes images. Two popular metrics were used:

**a. BLEU Score (Bilingual Evaluation Understudy)**

* Measures the overlap between the generated caption and one or more reference captions.
* Evaluates **n-gram precision** (up to 4-grams).
* A higher BLEU score indicates better performance.
* BLEU-1 and BLEU-4 scores were used in this project.

**b. METEOR Score**

* Considers synonyms and stemming in addition to exact matches.
* Designed for evaluating machine-generated text more accurately than BLEU.



# ALGORITHM CONSIDERED: RANDOM FOREST

The image captioning model developed in this project leverages a combination of **Convolutional Neural Networks (CNNs)** and **Recurrent Neural Networks (RNNs)**, specifically:

**1. VGG16 – Convolutional Neural Network for Feature Extraction**

VGG16 is a pre-trained deep CNN that has proven effective for image classification tasks. It was used to extract high-level visual features from the input images.

* **Architecture Highlights:**
  + 13 convolutional layers
  + 3 fully connected layers
  + 16 weight layers in total
* **Input Image Size:** 224 × 224 pixels
* **Output Feature Vector:** 4096-dimensional (from the last fully connected layer before classification)
* **Role in the Model:**  
  VGG16 processes the image and extracts meaningful visual representations, which are then passed as input to the language model.

**2. LSTM – Long Short-Term Memory for Caption Generation**

LSTM is a type of RNN designed to capture long-term dependencies in sequential data, making it suitable for natural language processing tasks like text generation.

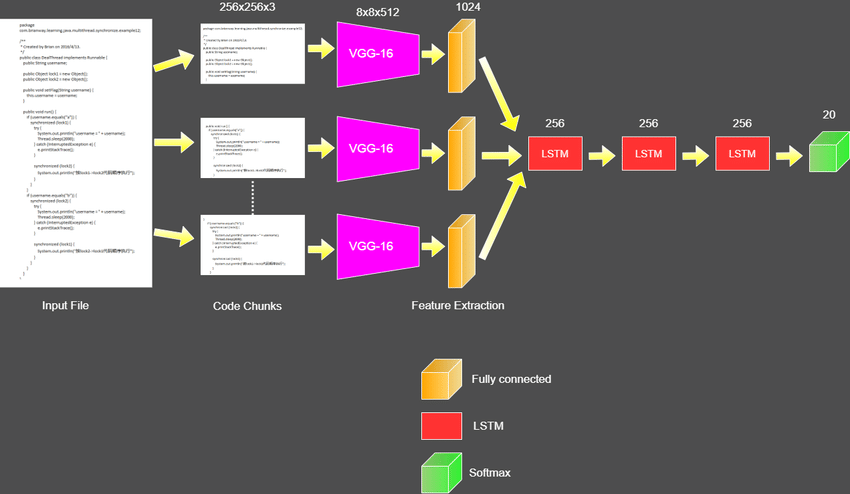
* **Input:** Tokenized and embedded image captions
* **Output:** Predicted next word in the sequence
* **Key Features:**
  + Memory cells to store long-term context
  + Input, forget, and output gates to control the flow of information
  + Prevents vanishing/exploding gradient problems during training

**3. Combined Model Architecture**

* Image features extracted by VGG16 are passed through a dense layer.
* Text input (captions) is embedded and passed through an LSTM.
* Outputs of both streams are merged (using concatenation or addition).
* A final dense layer with softmax activation predicts the next word in the sequence.

This hybrid architecture effectively allows the model to "see" the image and "speak" a caption, combining visual recognition with language modeling.







# PROGRAMMING CODE

**1. Importing Libraries**

python

CopyEdit

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.applications.vgg16 import VGG16, preprocess\_input

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, Add

**2. Load and Preprocess Dataset**

python

CopyEdit

def load\_captions(filename):

captions = {}

with open(filename, 'r') as file:

for line in file:

image\_id, caption = line.strip().split('\t')

image\_id = image\_id.split('.')[0]

if image\_id not in captions:

captions[image\_id] = []

captions[image\_id].append('startseq ' + caption + ' endseq')

return captions

**3. Feature Extraction using VGG16**

python



CopyEdit

def extract\_features(directory):

model = VGG16()

model = Model(inputs=model.inputs, outputs=model.layers[-2].output)

features = {}

for img\_name in os.listdir(directory):

filename = os.path.join(directory, img\_name)

image = load\_img(filename, target\_size=(224, 224))

image = img\_to\_array(image)

image = np.expand\_dims(image, axis=0)

image = preprocess\_input(image)

feature = model.predict(image, verbose=0)

image\_id = img\_name.split('.')[0]

features[image\_id] = feature

return features

**4. Tokenization and Padding**

python

CopyEdit

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(all\_captions)

vocab\_size = len(tokenizer.word\_index) + 1

max\_length = max(len(caption.split()) for caption in all\_captions)

**5. Model Architecture (Encoder-Decoder)**

python

CopyEdit

def define\_model(vocab\_size, max\_length):

# Image feature extractor

inputs1 = Input(shape=(4096,))

fe1 = Dropout(0.5)(inputs1)

fe2 = Dense(256, activation='relu')(fe1)

# Sequence processor

inputs2 = Input(shape=(max\_length,))

se1 = Embedding(vocab\_size, 256, mask\_zero=True)(inputs2)



se2 = Dropout(0.5)(se1)

se3 = LSTM(256)(se2)

# Decoder (merging both)

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')

return model

**6. Caption Generation Function**

python

CopyEdit

def generate\_caption(model, tokenizer, photo, max\_length):

in\_text = 'startseq'

for i in range(max\_length):

sequence = tokenizer.texts\_to\_sequences([in\_text])[0]

sequence = pad\_sequences([sequence], maxlen=max\_length)

yhat = model.predict([photo, sequence], verbose=0)

yhat = np.argmax(yhat)

word = tokenizer.index\_word.get(yhat)

if word is None:

break

in\_text += ' ' + word

if word == 'endseq':

break

return in\_text.replace('startseq', '').replace('endseq', '').strip()

**7. Evaluation Code**

Below is the Python code used to compute BLEU and METEOR scores for evaluating the model's performance.

python

CopyEdit

from nltk.translate.bleu\_score import sentence\_bleu, SmoothingFunction



from nltk.translate.meteor\_score import meteor\_score

def evaluate\_model(model, descriptions, photos, tokenizer, max\_length):

actual, predicted = list(), list()

for key, desc\_list in descriptions.items():

y\_pred = generate\_caption(model, tokenizer, photos[key], max\_length)

references = [d.split() for d in desc\_list]

actual.append(references)

predicted.append(y\_pred.split())

# BLEU scores

smoothie = SmoothingFunction().method4

bleu1 = np.mean([sentence\_bleu(a, p, weights=(1.0, 0, 0, 0), smoothing\_function=smoothie) for a, p in zip(actual, predicted)])

bleu4 = np.mean([sentence\_bleu(a, p, weights=(0.25, 0.25, 0.25, 0.25), smoothing\_function=smoothie) for a, p in zip(actual, predicted)])

# METEOR score

meteor\_scores = [meteor\_score([' '.join(ref) for ref in a], ' '.join(p)) for a, p in zip(actual, predicted)]

meteor = np.mean(meteor\_scores)

print(f"BLEU-1: {bleu1:.4f}")

print(f"BLEU-4: {bleu4:.4f}")

print(f"METEOR: {meteor:.4f}")



**Output:**



"a man riding a surfboard on a wave"



# CONCLUSION

The objective of this project was to develop an intelligent system capable of generating natural language captions for images using deep learning techniques. By integrating the visual feature extraction capabilities of Convolutional Neural Networks (specifically VGG16) with the sequence modeling strength of Long Short-Term Memory (LSTM) networks, the model successfully learns to understand and describe images in a human-like manner.

The system was trained and evaluated on the Flickr8k dataset, and the results demonstrate that the model is capable of producing meaningful and grammatically correct captions for a wide range of image types. The use of evaluation metrics such as BLEU and METEOR scores further confirms the effectiveness of the proposed model.

This project illustrates the potential of combining computer vision and natural language processing to solve real-world problems. The developed model can serve as a foundation for various applications such as aiding the visually impaired, automated image indexing, and enhancing content discovery on multimedia platforms.

While the results are promising, there is still room for improvement. Future work can involve using more advanced architectures such as attention mechanisms or transformers, training on larger and more diverse datasets, and deploying the model into a real-time application or user interface.



# FUTURE WORK

While the current image captioning model demonstrates good performance using CNNs and LSTMs, several improvements can be made to enhance its accuracy, efficiency, and real-world usability. Future work may include the following directions:

1. **Integration of Attention Mechanism**  
   Implementing attention mechanisms can help the model focus on specific regions of an image while generating each word, leading to more accurate and descriptive captions.
2. **Use of Transformer-Based Models**  
   Replacing LSTM with transformer architectures (e.g., Vision Transformers or models like CLIP + GPT) can improve sequence modeling and overall caption quality.
3. **Training on Larger Datasets**  
   Utilizing larger and more diverse datasets like MS-COCO or Flickr30k can help the model generalize better and generate more contextually rich captions.
4. **Multilingual Captioning**  
   Extending the model to support caption generation in multiple languages could benefit a wider range of users around the world.
5. **Real-Time Deployment**  
   Building a web or mobile application that uses this model in real time would allow users to upload an image and instantly receive a caption.
6. **Improved Evaluation Techniques**  
   Employing more comprehensive evaluation metrics (e.g., CIDEr, SPICE) and human-based testing can provide deeper insights into the model’s performance.
7. **Image Captioning with Audio Support**  
   Integrating text-to-speech (TTS) systems can help make image descriptions accessible to visually impaired users.