Project Report

on

LSTM-VGG16: A MODEL FOR IMAGE CAPTIONING USING DEEP LEARNING APPROACHES

Submitted for Internship/ Industrial training requirements

of

BTECH & MTECH

in

MATHEMATICS AND DATA SCIENCE

Submitted by:

Ayush Rupapapra

Scholar No. 214104026

Under the guidance of

Dr. Pushpendra Kumar

Department of Mathematics, Bioinformatics and Computer Applications

Maulana Azad

National Institute of Technology, Bhopal-462003 (India)

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY, BHOPAL (M.P)-462003

CERTIFICATE

This is to certify that Ayush Rupapara (Sch no- 214104026), a student of Dual Degree (B.Tech+M.Tech) of batch 2021-2026 has completed the project titled "LSTM-VGG16: A Model For Image Captioning Using Deep Learning Approaches" being submitted in the partial fulfilment of the requirement of the completion of Dual Degree in Mathematics and Data Science to Maulana Azad National Institute of technology Bhopal under my supervision.

Ayush Rupapara
(214104026)

Date:

Dr. Pushpendra Kumar
Assistant Professor
(Supervisor)

Abstract:

Every day, we encounter large number of images from various sources such as the internet, news articles, document diagrams and advertisements. These sources contain images that viewers would have to interpret themselves. Most images do not have a description, but the human can largely understand them without their detailed captions. However, machine needs to interpret some form of image captions if humans need automatic image captions from it. As long as machines do not think, talk, and behave like humans, natural language descriptions will remain a challenge to be solved. In this project, Image Captioning using deep learning is the process of generation of textual description of an image. Though image captioning is a complicated and difficult task, a lot of researchers have achieved significant improvements. The primary objective of this project was to develop an end-to-end solution for generating descriptive captions for images. We aimed to leverage deep learning techniques, including Long Short-Term Memory (LSTM) networks and the VGG16 model, to create a robust image captioning system. The project aimed to contribute to the field of computer vision and natural language processing by improving the state-of-the-art in image captioning.

1. Introduction

In a world increasingly dominated by visual content, the ability to provide context and understanding to images is becoming ever more vital. The "Image Caption Generator using LSTM and VGG16" project addresses this need by pioneering an innovative solution to automatically generate descriptive captions for images.

The primary objective of this project is to develop an end-to-end image captioning system that bridges the gap between computer vision and natural language processing. Enhanced Human-Computer Interaction: Our project aims to enable more intuitive and meaningful interactions between humans and computers by providing automated, human-like descriptions for images. This feature has the potential to revolutionize applications in fields such as visual search, accessibility for visually impaired individuals, and content tagging.

Image captioning, as a problem, holds substantial significance in the realms of artificial intelligence and human-computer interaction. It bridges the visual and linguistic modalities, enabling machines to comprehend and describe the visual world. Key aspects of its significance include:

Improved Information Accessibility: Image captioning makes visual content accessible to individuals who rely on text-based information. This includes the visually impaired, who can benefit from voice-assisted descriptions of images.

Enhanced Visual Search: Search engines, e-commerce platforms, and content management systems can significantly benefit from the ability to search and retrieve images based on text queries. This makes image captioning indispensable in applications like visual search and recommendation systems.

Content Tagging and Metadata: Image captioning simplifies content management and organization by providing automated tagging and metadata generation. This can streamline content indexing and retrieval.

Image captioning is a complex and interdisciplinary task that requires a deep understanding of both computer vision and natural language processing. At its core, it involves training a model to generate descriptive text that explains the content of an image. This is achieved by learning the relationships between visual features extracted from the image and the corresponding textual descriptions.

The problem can be divided into several stages, including image feature extraction, language modeling, and sequence generation. Models like the VGG16 convolutional neural network extract high-level image features, while LSTM networks generate coherent and contextually relevant captions. The combination of these techniques is central to our project's approach.

Fig .1: sample image for image Captioning

2. LITERATURE SURVEY

Evaluating the trained model is quiet difficult task in image captioning for this purpose various evaluation matrices are created. Most common evaluation mechanisms found in literature are BLEU, ROUGE-L, CIDEr, METEOR, and SPICE. It is found that BLEU score is most popular method of evaluation used by almost all studies. BLEU stands for bilingual evaluation understudy. It is an evaluation mechanism widely use in text generation. It is a mechanism for comparing the machine generated text with one or more manually written text. So basically, it summarizes that how close a generated text is to an expected text. BLEU score is majorly prevalent in automated machine translation but it can be also used in image captioning, text summarization, speech recognition etc.

Image captioning is a very recent and growing research problem nowadays. Day by day various new methods are being introduced to achieve satisfactory results in this field. However, there are still lots of attention required to achieve results as good as a human. Image captioning is defined as the process of generating captions or textual descriptions for images based on the contents of the image. It is a machine learning task that involves both natural language processing (for text generation) and computer vision (for understanding image contents).

To develop an Image captioning model, there are many datasets on which this model can be trained on for acquiring some best results. In literature most common used data sets are MSCOCO and flicker 8k and 30k.Moreover for a text description of specific task like in medical or traffic movement description their own dedicated datasets are created.

3. PROPOSED APPROACH

For Image caption generation, first extraction of features of an image must be done. We are using a pre-trained model to interpret the content of the photos There are many models to choose from. In this case, we will use the Oxford Visual Geometry Group, or VGG, model that won the ImageNet competition in 2014. This is called VGG16 model, because it contains 16 Convolutional layers.

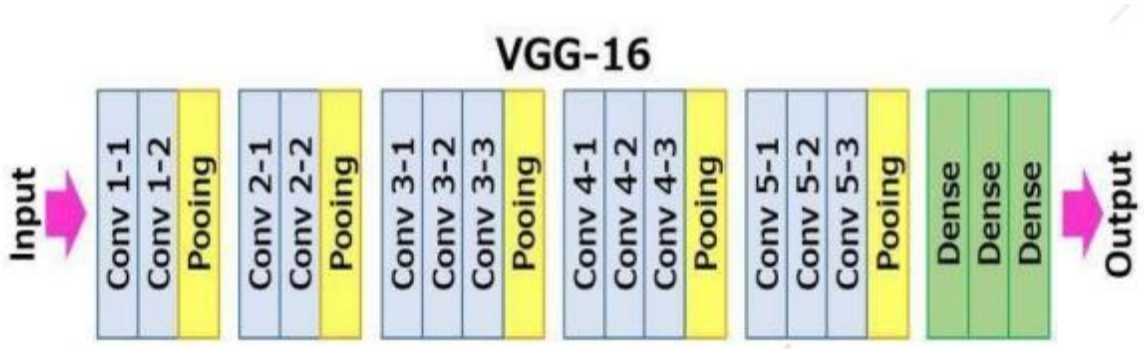


Fig .2: VGG -16 Model

For caption generation, we are using a type of RNN model i.e., LSTM (Long-Short Term Memory) model. LSTM networks are an extension of recurrent neural networks (RNNs) mainly introduced to handle situations where RNNs fail. LSTM's have a nature of remembering information for a long periods of time, which is also their default behavior. LSTM can retain important information over time using memory cells.

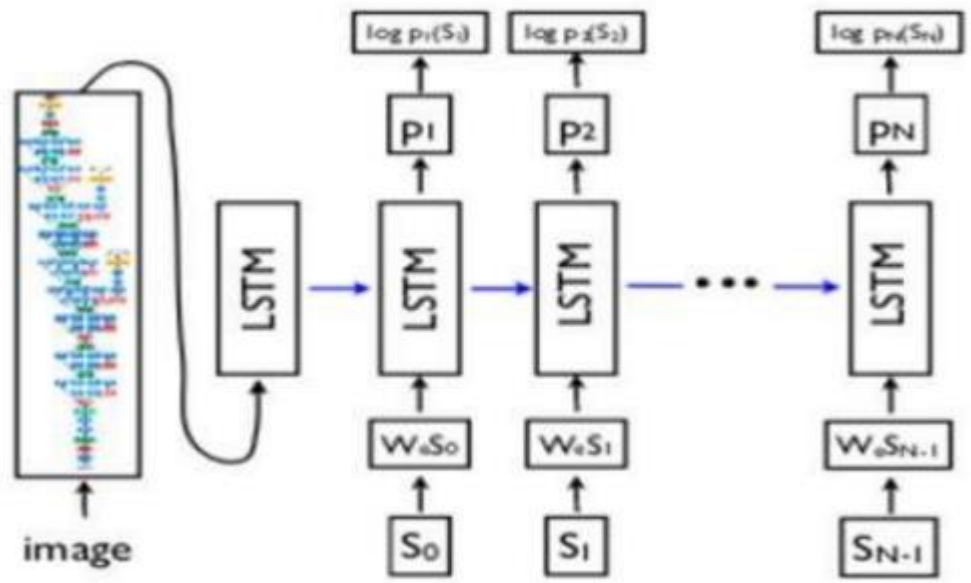


Fig.3: LSTM Framework

This network model has three important gates additional to the one operation in RNN (cell state). These four gates Input gate, Forget gate and Output gate helps in remembering the information that is most needed and forgetting the information which is no longer needed

3.1 Architecture:

Proposed System architecture consists of,

Photo Feature Extractor: This is a 16-layer VGG model pre-trained on the ImageNet dataset. We have pre-processed the photos with the VGG model (without the output layer) and will use the extracted features predicted by this model as input.

Sequence Processor: This is a word embedding layer for handling the text input, followed by a LongShort-TermMemory(LSTM) recurrentneuralnetworklayer.

Decoder: Both the feature extractor and sequence processor output a fixed-length vector. These are merged and processed by a Dense layer to make a final prediction.

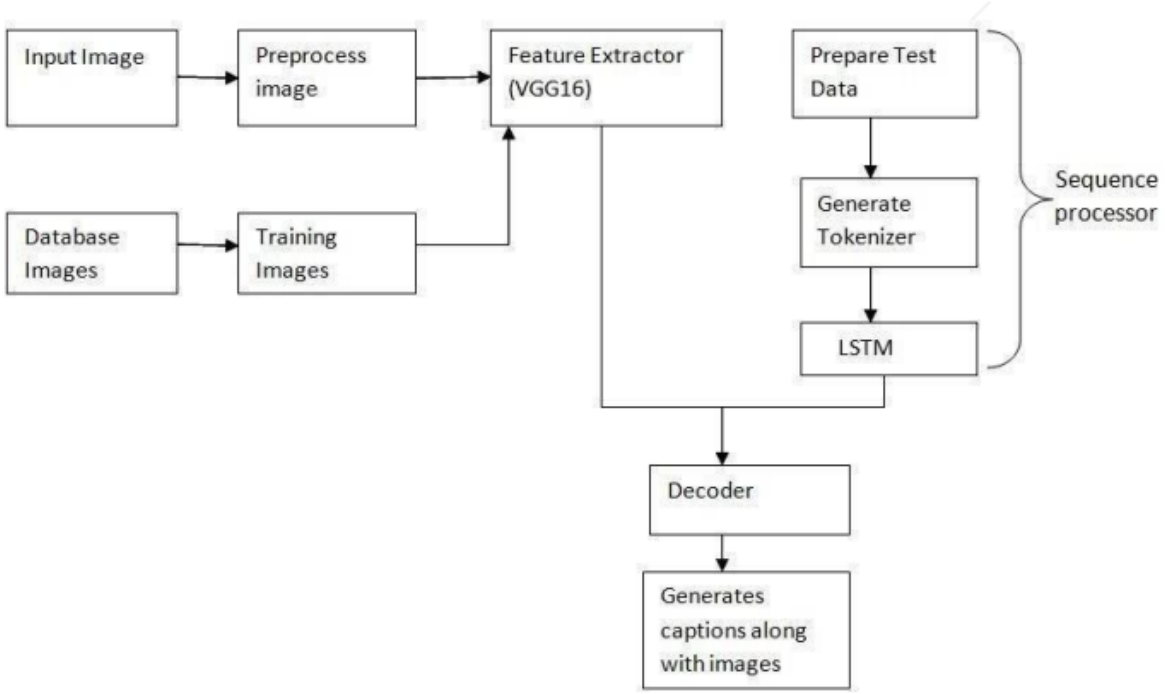


Fig.3: ICDL Architecture

3.2 Methodology

For feature extraction, we use a pre-trained model to interpret the content of the photos. There are many models to choose from. In this case, we will use model called VGG16, because of the 16 convolutional layers.

Keras provides this pre-trained model directly. The first time we use this model, Keras will download the model weights from the Internet, which are about 500 Megabytes. We could use this model as part of a broader image caption model. The problem is, it is a large model and running each photo through the network every time we want to test a new language model configuration (downstream) is redundant.

Instead, we pre-compute the "photo features" using the pre-trained model and save them to file. We can then load these features later and feed them into our model as the interpretation of a given photo in the dataset. It is no different to running the photo through the full VGG model; it is just we will have done it once in advance. This is an optimization that will make training our models faster and consume less memory. The function named extract_features() that, given a

directory name, will load each photo, prepare it for VGG, and collect the predicted features from the VGG model. The image features are a 1-dimensional 4,096 element vector. The function returns a dictionary of image identifier to image features. We call this function to prepare the photo data for testing our models, then save the resulting dictionary to a file named 'features.pkl'.

3.2.1 Preprocessing Text Data

The dataset contains multiple descriptions for each photograph and the text of the descriptions requires some minimal cleaning. First, we load the file containing all of the descriptions. Each photo has a unique identifier. This identifier is used on the photo filename and in the text file of descriptions.

The function load_descriptions() that, given the loaded document text, will return a dictionary of photo identifiers to descriptions. Each photo identifier maps to a list of one or more textual descriptions. Next, we need to clean the description text. The descriptions are already tokenized and easy to work with.

We cleaned the text in the following ways in order to reduce the size of the vocabulary of words

- Convert all words to lowercase.
- Remove all punctuation.
- Remove all words that are one character or less in length (e.g. 'a').
- Remove all words with numbers in them.

The clean_descriptions() function that, given the dictionary of image identifiers to descriptions, steps through each description and cleans the text. Once cleaned, we can summarize the size of the vocabulary.Ideally, we want a vocabulary that is both expressive and as small as possible. A smaller vocabulary will result in a smaller model that will train faster. The save_descriptions() function that, given a dictionary containing the mapping of identifiers to descriptions and a filename, saves the mapping to file. Running the example first prints the number of loaded photo descriptions (8,092) and the size of the clean vocabulary (8,763 words) Finally we saved the dictionary of image identifiers and descriptions to a new file named 'descriptions.txt', with one image identifier and description per line.

3.2.2 Developing Deep Learning Model

Load Data: First, we load the prepared photo and text data so that we can use it to fit the model. We are going to train the data on all of the photos and captions in the training dataset The train and development dataset have been predefined in theFlickr_8k.trainImages.txt and Flickr_8k.devImages.txt files respectively, that both contain lists of photo file names. From these file names, we can extract the photo identifiers and use these identifiers to filter photos and descriptions for each set

The function load_clean_descriptions() loads the cleaned text descriptions from 'descriptions.txt' for a given set of identifiers and returns a dictionary of identifiers to lists of text descriptions. The model we will develop will generate a caption given a photo, and the caption will be generated one word at a time.

Next, we can load the photo features for a given dataset. The function named load_photo_features() that loads the entire set of photo descriptions, then returns the subset of interest for a given set of photo identifiers. The create_tokenizer() function that will fit a Tokenizer given the loaded photo description text

3.3 Modular Approach

This is a 16-layer VGG model pre-trained on the ImageNet dataset. We have pre-processed the photos with the VGG model (without the output layer) and will use the extracted features predicted by this model as input. Sequence Processor: This is a word embedding layer for handling the text input, followed by a Long Short-Term Memory (LSTM) recurrent neural network layer. Both the feature extractor and sequence processor output a fixed -length vector. These are merged and processed by a Dense layer to make a final prediction.

• The Photo Feature Extractor model expects input photo features to be a vector of 4,096 elements. These are processed by a Dense layer to produce a 256-element representation of the photo.

• The Sequence Processor model expects input sequences with a pre-defined length (34 words) which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by an LSTM layer with 256 memory units.

• Both the input models produce a 256 element vector. Further, both input models use regularization in the form of 50% dropout. This is to reduce overfitting the training dataset, as this model configuration learns very fast.

• The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a Dense 256 neuron layer and then to a

final output Dense layer that makes a Softmax prediction over the entire output vocabulary for the next word in the sequence.

The function named define_model() defines and returns the model ready to be fit.

Train Model: The first step is to define a function that we can use as the data generator. The function data_generator() will be the data generator and will take the loaded textual descriptions, photo features, tokenizer and max length. Finally, we can use the fit_generator() function on the model to train the model with this data generator. We trained the model for 20 epochs and simply save the model after each training epoch. The models with the lowest loss is considered as the best model and that model is used for evaluation and testing.

Evaluation of Model: Once the model is fit, we evaluate the skill of its predictions on the holdout test dataset. We will evaluate a model by generating descriptions for all photos in the test dataset and evaluating those predictions with a standard cost function. We will generate predictions for all photos in the test dataset and in the train dataset.

The function named evaluate_model() will evaluate a trained model against a given dataset of photo descriptions and photo features. The actual and predicted descriptions are collected and evaluated collectively using the corpus BLEU score that summarizes how close the generated text is to the expected text.

BLEU SCORES are used in text translation for evaluating translated text against one or more reference translations. We compared each generated description against all of the reference descriptions for the photograph. We then calculate BLEU scores for 1, 2, 3 and 4 cumulative n grams. The NLTK Python library implements the BLEU score calculation in the corpus_bleu() function. A higher score close to 1.0 is better, a score closer to zero is worse.

CNN A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the

Receptive Field. A collection of such fields overlaps to cover the entire visual area.

RNNs are designed to identify data with sequential characteristics and predict the next likely scenario. They are used in models that simulate the activity of neurons in the human brain, such as deep learning and machine learning. This type of RNN has a memory that enables it to remember important events that have happened many times in the past (steps). RNNs are images that can be broken down into a series of patches and treated as sequences. By using the temporal dependence of the learned input data, we are able to distinguish the sequences we learn from other regression and classification tasks

4. EXPERIMENTAL RESULTS

DATASET

The dataset we are using is Flickr8k Dataset. The datasets are of the following size

- Flickr8k_Dataset.zip (1 Gigabyte) An archive of all photographs.
- Flickr8k_text.zip (2.2 Megabytes) An archive of all text descriptions for photographs.
- After downloading the datasets and extracting them. Extracted directories are
- Flickr8k_Dataset: Contains 8092 photographs in JPEG format.
- Flickr8k_text: Contains a number of files containing different sources of descriptions for the photographs.

Photo Feature Extraction VGG16 Layers

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
```

Preprocessing Text Data

```
Loaded: 8092
Vocabulary Size: 8763
```

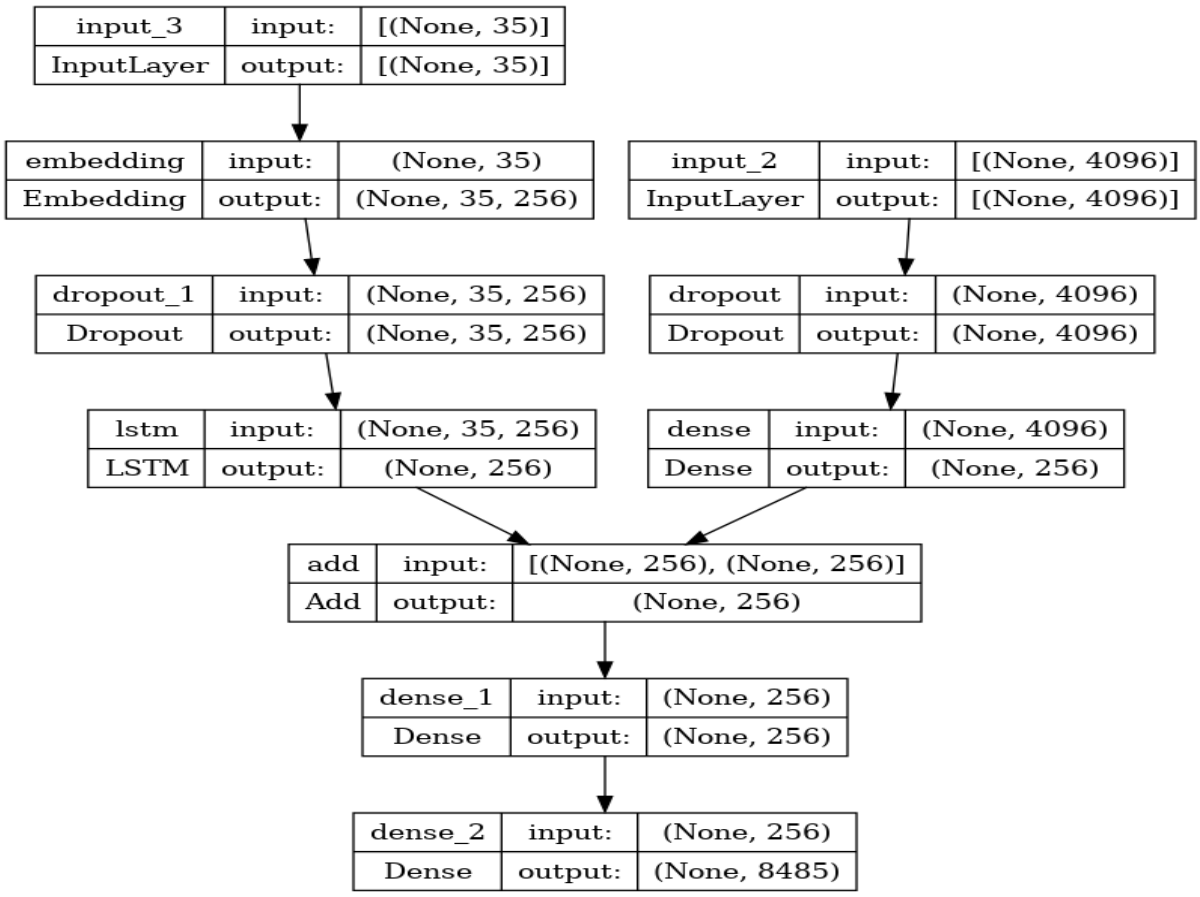
Training Model

```
Dataset: 6000
Descriptions: train=6000
Photos: train=6000
Vocabulary Size: 10400
Description Length: 32
Model: "model_3"
```

Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	[(None, 32)]	0	
input_6 (InputLayer)	[(None, 4096)]	0	
embedding_2 (Embedding)	(None, 32, 256)	2662400	input_7[0][0]
dropout_4 (Dropout)	(None, 4096)	0	input_6[0][0]
dropout_5 (Dropout)	(None, 32, 256)	0	embedding_2[0][0]
dense_6 (Dense)	(None, 256)	1048832	dropout_4[0][0]
lstm_2 (LSTM)	(None, 256)	525312	dropout_5[0][0]
add_2 (Add)	(None, 256)	0	dense_6[0][0] lstm_2[0][0]
dense_7 (Dense)	(None, 256)	65792	add_2[0][0]
dense_8 (Dense)	(None, 10400)	2672800	dense_7[0][0]

```
Total params: 6,975,136
Trainable params: 6,975,136
Non-trainable params: 0
```

Model Architecture



5. CODE

```
import streamlit as st
import numpy as np
import pickle
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, load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load the saved model
model = load_model('best_model.h5')

# Load the tokenizer
with open('tokenizer.pkl', 'rb') as tokenizer_file:
    tokenizer = pickle.load(tokenizer_file)

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

# Define functions for image processing and caption generation
 def load_and_preprocess_image(image_path):
# Load and preprocess the image for VGG
  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)
    # Extract features
    feature = vgg_model.predict(image, verbose=0)
    return feature

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

def generate_caption(model, image, tokenizer):
    # Add the start tag for the generation process
    in_text = 'startseq'
    max_length = 35

    # Iterate over the max length of the sequence
    for _ in range(max_length):
        # Encode the input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # Pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # Predict the next word using the model
        yhat = model.predict([image, sequence], verbose=0)
        # Get the index with the highest probability
        yhat = np.argmax(yhat)
        # Convert the index to a word
```

```
        word = idx_to_word(yhat, tokenizer)
        # Stop if the word is not found
        if word is None:
            break
        # Append the word as input for generating the next word
        in_text += " " + word
        # Stop if we reach the end tag
        if word == 'endseq':
            break

    return in_text

st.title("Image Caption Generator")

uploaded_image = st.file_uploader("Upload an image...", type=["jpg", "jpeg",
"png"])

if uploaded_image is not None:
    st.image(uploaded_image, caption="Uploaded Image",
use_column_width=True)
    feature = load_and_preprocess_image(uploaded_image)

    if st.button("Generate Caption"):
        caption = generate_caption(model, feature, tokenizer )

        st.write("Generated Caption:", caption)
```

6. CONCLUSION

Image captioning has made significant advances in recent years. Recent work based on deep learning techniques has resulted in a breakthrough in the accuracy of image captioning. The text description of the image can improve the content-based image retrieval efficiency, the expanding application scope of visual understanding in the fields of medicine, security, military and other fields, which has a broad application prospect. At the same time, the theoretical framework and research methods of image captioning can promote the development of the theory and application of image annotation and visual question answering (VQA), cross media retrieval, video captioning and video dialog, which has important academic and practical application value. At last during the training the model loss was decreasing with each epoch and at last the training loss was 2.1828. We attained BLEU-1 score of 0.536631 and BLEU-2 score of 0.313440. We will still be working on to reduce the loss and increase accuracy with high GPU and Tensor flow Object detection API using other advanced algorithms such as Faster RCNN, SSD, MASK RCNN etc. to detect and segment the forest land covers on the image.

7. REFERENCES

1. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method forautomatic evaluation of machine translation,” in Proceedings of the 40thannual meeting on association for computational linguistics. Association for Computational Linguistics, 2002, Conference Proceedings, pp.311–318.

2. Image Captioning Based on Deep Neural NetworksShuang Liu, Liang Bai, Yanli Hu and Haoran Wang, MATEC Web Conf., 232 (2018) 01052.

3. He, Kaiming, et al. "Deep Residual Learning for Image Recognition." IEEE Conference on Computer Vision and Pattern Recognition IEEE Computer Society, 770-778. (2016).

4. Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." IEEE Conference on Computer Vision and Pattern Recognition IEEE Computer Society, 3156-3164. (2015).

5. Xu, Kelvin, et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention." Computer Science ,2048-2057. (2015).

6. Anderson, Peter, et al. "SPICE: Semantic Propositional Image Caption Evaluation." Adaptive Behavior 11.4 382-398. (2016).

7. Ranzato, Marc'Aurelio, et al. "Sequence Level Training with Recurrent Neural Networks." Computer Science (2015).

8. Kalchbrenner, Nal, E. Grefenstette, and P. Blunsom. "A Convolutional Neural Network for Modelling Sentences." Eprint Arxiv (2014).

9. Aneja, Jyoti, A. Deshpande, and A. Schwing. "Convolutional Image Captioning." (2017) 24. Gu, Jiuxiang, et al. "Stack-Captioning: Coarse-to-Fine Learning for Image Captioning." (2018).

10. Krizhevsky, Alex, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks." International Conference on Neural Information Processing Systems Curran Associates Inc. 1097-1105. (2012).

11. Devlin, Jacob, et al. "Language Models for Image Captioning: The Quirks and What Works." Computer Science (2015).

12. Hochreiter, Sepp, and J. Schmidhuber. "Long ShortTermMemory."Neural Computation 9.8: 1735-1780. (1997).

13. Karpathy, Andrej, and F. F. Li. "Deep visual-semantic alignments for generating image descriptions." Computer Vision and Pattern Recognition IEEE, 3128-3137. (2015).