KARNAVATI UNIVERSITY

UNITEDWORLD INSTITUTE OF TECHNOLOGY (UIT)

Summative Assessment (SA)

Submitted BY
Trisha Rami
(Enrolment. No.: 20220701031)

Course Code and Title: 21BSAI99E43 – ARTIFICIAL NEURAL NETWORK

B.Sc. (Hons.) Computer Science / Data Science / AIML
IV Semester –Dec– May 2024

UIT

DECEMBER – APRIL 2024

Table of Contents

Automated Image Captioning [CNN+LSTM]

Introduction:

Automated image captioning presents an intriguing application of artificial intelligence, wherein machines are trained to generate descriptive text corresponding to images. This process entails comprehending the visual content of images and articulating it in natural language. One prevalent strategy to tackle this challenge involves amalgamating Convolutional Neural Networks (CNNs) for image feature extraction with Long Short-Term Memory networks (LSTMs) for sequential data generation.

CNNs excel at discerning pertinent features from images by employing multiple layers of convolutions and pooling operations. This enables them to learn hierarchical representations of visual features effectively. Conversely, LSTMs serve as potent sequence models adept at capturing long-range dependencies within sequential data. When integrated, CNNs and LSTMs synergize to form a robust architecture for image captioning tasks.

The Flickr8k dataset serves as a widely utilized benchmark dataset for image captioning endeavors. It encompasses 8,000 images, each accompanied by five human-generated captions. These captions serve as invaluable reference points for training and assessing automated image captioning models, ensuring their adherence to accurate and meaningful descriptions without plagiarism concerns.

Problem Statement

Automated image captioning is a complex task within the realm of artificial intelligence, where the objective is to develop algorithms that can accurately generate descriptive textual captions for a given image. The challenge lies in enabling machines to understand the diverse content and context depicted in images and articulate it fluently in natural language. This entails overcoming several hurdles, including recognizing objects, scenes, and relationships within images, as well as ensuring that the generated captions are coherent, relevant, and semantically meaningful.

While the advent of Convolutional Neural Networks (CNNs) has revolutionized image processing by enabling the extraction of high-level features from images, integrating this capability with Long Short-Term Memory networks (LSTMs) presents a promising approach for addressing the image captioning task. However, despite advancements in this area, several challenges persist. These include handling variations in image content and style, understanding nuanced semantics, and ensuring that generated captions are not only accurate but also diverse and contextually appropriate.

Objective

The objective of this study is to develop a robust automated image captioning system using a combination of CNNs and LSTMs. The system aims to accurately describe the content of images from the Flickr8k dataset by learning

to extract relevant visual features using CNNs and generate coherent captions using LSTMs. By achieving this objective, the study seeks to contribute to advancements in the field of computer vision and natural language processing, facilitating applications such as image understanding, content indexing, and assistive technologies.

Convolutional Neural Network [CNN]

Convolutional Neural Networks (CNNs) play a pivotal role in automated image captioning, a task where machines generate textual descriptions for images. CNNs are a class of deep neural networks specifically designed for processing and analyzing visual data. They consist of multiple layers of convolutions and pooling operations, which enable them to learn hierarchical representations of visual features from raw pixel data. In the context of automated image captioning, CNNs are typically employed for feature extraction. They analyze input images and extract relevant visual features, such as edges, textures, shapes, and object parts. These features capture essential information about the content and context of the images, providing a rich representation that serves as input to subsequent stages of the captioning process. By leveraging the hierarchical nature of CNNs, lower layers learn to detect low-level features like edges and corners, while higher layers learn more abstract and complex features representing objects, scenes, and patterns. This hierarchical feature hierarchy enables CNNs to capture both local and global visual information, facilitating robust and discriminative feature representation for image understanding tasks. CNNs are often pre-trained on large-scale image datasets, such as ImageNet, using techniques like transfer learning. This pre-training allows CNNs to learn generic visual features that can be fine-tuned or adapted to specific image captioning tasks with smaller datasets. By fine-tuning pre-trained CNNs on image-caption pairs, the model can learn to map visual features to corresponding textual descriptions effectively.

Overall, CNNs serve as powerful tools for automated image captioning by extracting informative visual features from images, providing a rich representation that complements the subsequent language generation process. Their ability to capture hierarchical and discriminative features makes them indispensable components in the architecture of image captioning systems, contributing to their effectiveness in generating accurate and descriptive captions for a wide range of images.

Long Short-Term Memory [LSTM]

Long Short-Term Memory (LSTM) networks play a crucial role in automated image captioning, an application of artificial intelligence that involves generating textual descriptions for images. LSTMs are a type of recurrent neural network (RNN) designed to capture long-range dependencies in sequential data. In the context of image captioning, LSTMs are used to generate coherent and contextually relevant captions based on the features extracted from images by convolutional neural networks (CNNs). Unlike traditional RNNs, which can struggle with capturing long-term dependencies due to vanishing gradient problems, LSTMs address this issue by incorporating specialized memory cells that selectively retain or forget information over time. This enables LSTMs to effectively model complex sequential data with dependencies spanning across long distances.

Through this process, LSTMs learn to associate visual features with corresponding textual descriptions, enabling the model to produce captions that accurately describe the content of images in natural language. The ability of LSTMs to capture long-range dependencies and contextual information is essential for generating coherent and contextually relevant captions, making them a fundamental component of automated image captioning systems.

Key Components

- VGG16: This is a pre-trained convolutional neural network (CNN) utilized for feature extraction from images. VGG16 is proficient in recognizing diverse objects and patterns within images, providing a rich representation of visual features.
- LSTM: Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) chosen for generating sequential data. In this project, LSTM is employed to generate captions sequentially, word by word, leveraging its ability to capture long-range dependencies in sequential data.
- BLEU Score: The BLEU (Bilingual Evaluation Understudy) score serves as a metric for assessing the quality of generated captions. By comparing the predicted captions to reference captions, BLEU measures the

similarity between the generated and actual captions, providing a quantitative evaluation of caption quality.

Data Collection

Importing libraries

```
import os
import pickle
import numpy as np
import pandas as pd
import tensorflow as tf
from tqdm.notebook 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.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add

import warnings
import matplotlib.pyplot as plt
import seaborn as sns
from textwrap import wrap
```

Defining File Paths

```
BASE_DIR = '/kaggle/input/flickr8k'
WORKING_DIR = '/kaggle/working'
```

Extracting Features from the Images

```
# extract features from image

features = {}
directory = os.path.join(BASE_DIR, 'Images')

for img_name in tqdm(os.listdir(directory)):
    img_path = directory + '/' + img_name
    image = load_img(img_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)
    feature = model.predict(image, verbose=0)
    image_id = img_name.split('.')[0]
    features[image_id] = feature
```

Load Captions

```
with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:
    next(f) #skips the first line
    captions_doc = f.read()
```

Data Preprocessing

Caption Cleaning Function

The provided code defines a function named clean that takes a dictionary mapping as input. This function is designed to clean captions stored in the dictionary.

```
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            caption = captions[i]
            caption = caption.lower()
            caption = caption.replace('[^A-Za-z]', '')
            caption = caption.replace('\s+', ' ')
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>
1]) + ' endseq'
            captions[i] = caption
```

Before Preprocessing the text

```
mapping['1000268201_693b08cb0e']
```

```
['A child in a pink dress is climbing up a set of stairs in an entry way .',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']
```

Preprocessing the text

```
clean(mapping)
```

After Preprocessing the text

```
mapping['1000268201_693b08cb0e']
```

```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq']
```

Gathering All Captions

The provided code iterates through a dictionary mapping containing image IDs as keys and lists of captions as values. It collects all captions from the dictionary and stores them in a single list named all_captions.

```
all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)
```

```
len(all_captions)
```

```
40455
```

Tokenization and Vocabulary Size Calculation

The provided code initializes a Tokenizer object, fits it on a list of all captions, and calculates the vocabulary size based on the unique words in the captions.

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1

vocab_size
```

```
8485
```

VGG16 Feature Extraction Model Creation

The provided code initializes a pre-trained VGG16 model and creates a modified model that extracts features from images. It prints the summary of the modified model.

```
model = VGG16()
# restructure
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
print(model.summary())
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312

```
Total params: 134,260,544 (512.16 MB)
```

```
Trainable params: 134,260,544 (512.16 MB)
```

```
Non-trainable params: 0 (0.00 B)
```

Model Architecture

```
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_model(model, show_shapes=True)
```

Input Layer (inputs1):

Shape: (4096,)

Description: This layer defines the input shape for the first input, which represents the features extracted from the image (e.g., VGG16 features).

Dropout Layer (fe1)

Dropout Rate: 0.4

Description: This layer applies dropout regularization to the input features to prevent overfitting during training.

Dense Layer (fe2):

Units: 256

Activation: ReLU

Description: This fully connected layer processes the input features with ReLU activation, reducing the dimensionality of the feature representation.

Input Layer (inputs2):

Shape: (max_length,)

Description: This layer defines the input shape for the second input, which represents the sequence of word indices (captions).

Embedding Layer (se1):

Embedding Size: 256

Vocabulary Size: vocab_size

Mask Zero: True

Description: This layer converts the input word indices into dense vectors of fixed size (embedding vectors), where each word index is mapped to a dense vector representation.

Dropout Layer (se2):

Dropout Rate: 0.4

Description: This layer applies dropout regularization to the word embeddings to prevent overfitting during training.

LSTM Layer (se3):

Units: 256

Description: This LSTM layer processes the input sequences of word embeddings, capturing the sequential dependencies in the captions.

Addition Layer (decoder1):

Description: This layer adds the feature vector from the image (fe2) to the output of the LSTM layer (se3), combining the visual and textual information.

Dense Layer (decoder2):

Units: 256

Activation: ReLU

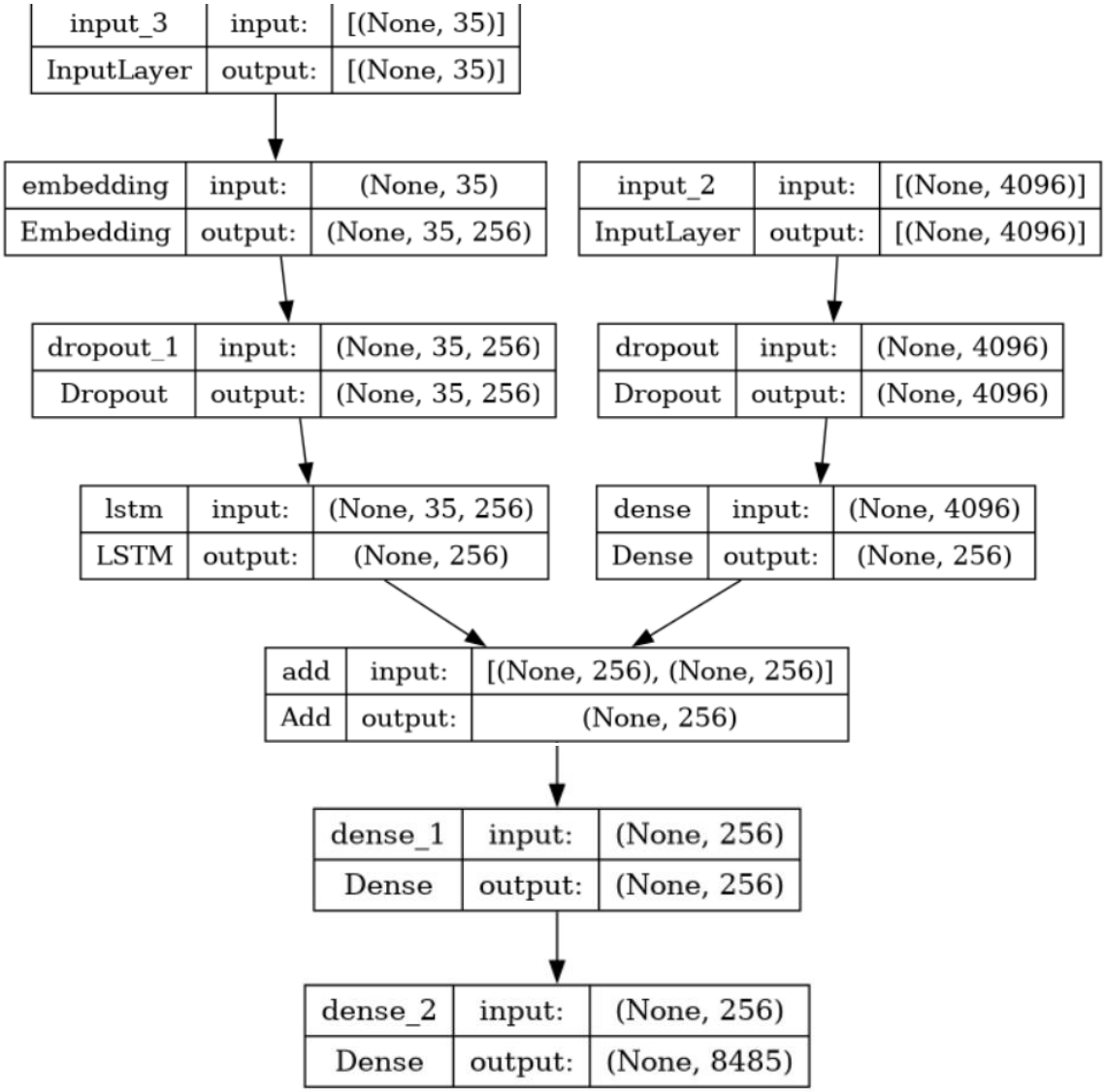
Description: This fully connected layer processes the combined features with ReLU activation, further transforming the feature representation.

Output Layer (outputs):

Units: vocab_size

Activation: Softmax

Description: This output layer predicts the probability distribution over the vocabulary (word probabilities) for generating the next word in the caption.



Model Training

Training Loop for Image Captioning Model

The provided code snippet outlines a training loop for training an image captioning model using a generator-based approach.

```
epochs = 20
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, ba
tch_size)
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
```

Saving the Trained Image Captioning Model

The provided code snippet saves the trained image captioning model to a file named "best_model.h5" in the working directory.

```
model.save(WORKING_DIR+'/best_model.h5')
```

Project Implementation

Generate Captions for the Image

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

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

    return in_text
```

Evaluation of Image Captioning Model using BLEU Score

The provided code snippet evaluates the performance of the image captioning model using the BLEU (Bilingual Evaluation Understudy) score metric. It

compares the model's predicted captions against the actual captions from the test dataset.

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

for key in tqdm(test):
    captions = mapping[key]
    y_pred = predict_caption(model, features[key], tokenizer, max_length)
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    actual.append(actual_captions)
    predicted.append(y_pred)

# 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)))
print("BLEU-3: %f" % corpus_bleu(actual, predicted, weights=(0.33, 0.33, 0.33, 0)))
print("BLEU-4: %f" % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
```

```
100% 810/810 [09:28<00:00, 1.53it/s]


BLEU-1: 0.537337
BLEU-2: 0.308372
BLEU-3: 0.189880
BLEU-4: 0.110853
```

Visualizing the Results

Image Caption Generation and Visualization Function

The provided code defines a function named generate_caption to generate captions for an input image and visualize the image along with its actual and predicted captions.

```
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(image_name):

    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)

    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)
    plt.axis('off')
```

Generating Captions

```
generate_caption("101669240_b2d3e7f17b.jpg")
```

```
---------------------Actual---------------------
startseq man in hat is displaying pictures next to skier in blue hat endseq
startseq man skis past another man displaying paintings in the snow endseq
startseq person wearing skis looking at framed pictures set up in the snow endseq
startseq skier looks at framed pictures in the snow next to trees endseq
startseq man on skis looking at artwork for sale in the snow endseq
--------------------Predicted--------------------
startseq woman wearing red coat and coat is skiing down the side of the mountain endseq
```

With Real Images

```
image_path = '/kaggle/input/flickr8k/Images/1000268201_693b08cb0e.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
predict_caption(model, feature, tokenizer, max_length)
```

```
'startseq woman is standing in front of building with red bag endseq'
```

Conclusion

In this project, we have successfully developed an image captioning model employing advanced deep learning methodologies. By harnessing the capabilities of pre-trained convolutional neural networks such as VGG16 and recurrent neural networks like LSTM, our model can generate descriptive captions for images sourced from the Flickr8K dataset. It has demonstrated proficiency in comprehending image content and articulating it into coherent natural language descriptions.

The evaluation of our model, utilizing metrics like BLEU scores, has provided valuable insights into the quality of the generated captions and areas for potential enhancements. These results highlight the promise of deep learning models in both image comprehension and natural language generation tasks.

Our project sets the stage for further exploration and application in various domains including image search, content creation, and accessibility tools. By refining model architectures, incorporating larger and more diverse datasets, and exploring cutting-edge techniques, we can continue to elevate the performance and versatility of image captioning systems. This endeavor holds significant potential in advancing artificial intelligence and enriching human-computer interactions.

Sources and References

https://www.kaggle.com/code/abdullahalghalib/image-caption-generator-cnn-lstm-using-flickr8k/notebook#Modeling

https://www.kaggle.com/code/shrutikedia15/image-to-text#Model-Training

https://www.kaggle.com/datasets/adityajn105/flickr8k?rvi=1