**DS Lab Experiment - 8**

**Sushant Tulasi**

**D20B**

**60**

**AIM:** To implement RNN Deep Learning Applications like Image Caption generator.

# THEORY:

A Recurrent Neural Network (RNN) is a type of neural network designed specifically to handle sequential data, where the output depends not only on the current input but also on the previous inputs in the sequence. Unlike traditional feedforward neural networks, RNNs have connections that form cycles, allowing them to maintain a "memory" of previous inputs, making them

well-suited for tasks involving sequences, such as time series prediction, natural language processing, and speech recognition.

Key Features of RNNs:

* Memory: RNNs have internal states (often referred to as hidden states) that retain information from previous time steps, giving the network a form of memory. This is especially useful for tasks where context is important, such as generating sentences or analyzing time-based data.
* Recurrent Connections: Each neuron in an RNN is connected not only to the next layer but also to itself, which allows the network to pass information across time steps. This enables RNNs to model dependencies in sequences.
  + Weight Sharing: The weights used to process input data are shared across all time steps, making the network efficient when handling variable-length sequences.

Implementation of RNN Deep Learning Applications: Image Caption Generator The task of generating captions for images involves combining computer vision and natural language processing (NLP), leveraging both convolutional neural networks (CNNs) and recurrent neural networks (RNNs). In this context, a Image Caption Generator system uses a pre-trained CNN, such as VGG16 or ResNet, to extract visual features from an image, which are then fed into an RNN (typically an LSTM or GRU) to generate a sequence of words that form a caption.

# CODE :

**Step 1 : Import the required libraries**

# linear algebra import numpy as np

# data processing, CSV file I / O (e.g. pd.read\_csv) import pandas as pd

import os

import tensorflow as tf

from keras.preprocessing.sequence import pad\_sequences

from keras.preprocessing.text import Tokenizer from keras.models import Model

from keras.layers import Flatten, Dense, LSTM, Dropout, Embedding, Activation from keras.layers import concatenate, BatchNormalization, Input from keras.layers.merge import add

from keras.utils import to\_categorical, plot\_model

from keras.applications.inception\_v3 import InceptionV3, preprocess\_input import matplotlib.pyplot as plt # for plotting data

import cv2

# Step 2: Load the descriptions

def load\_description(text): mapping = dict()

for line in text.split("\n"):

token = line.split("\t")

if len(line) < 2: # remove short descriptions continue

img\_id = token[0].split('.')[0] # name of the image

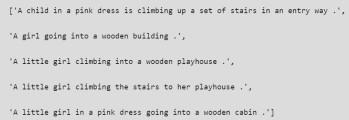
img\_des = token[1] # description of the image if img\_id not in mapping:

mapping[img\_id] = list() mapping[img\_id].append(img\_des)

return mapping

token\_path = '/kaggle / input / flickr8k / flickr\_data / Flickr\_Data / Flickr\_TextData / Flickr8k.token.txt'

text = open(token\_path, 'r', encoding = 'utf-8').read() descriptions = load\_description(text) print(descriptions['1000268201\_693b08cb0e'])



# Step 3: Cleaning the text

def clean\_description(desc):

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

for i in range(len(des\_list)): caption = des\_list[i]

word.isalpha()]

caption = [ch for ch in caption if ch not in string.punctuation] caption = ''.join(caption)

caption = caption.split(' ')

caption = [word.lower() for word in caption if len(word)>1 andcaption = ' '.join(caption) des\_list[i] = caption

clean\_description(descriptions) descriptions['1000268201\_693b08cb0e']

# Step 4: Generate the Vocabulary

def to\_vocab(desc):

words = set()

for key in desc.keys():

for line in desc[key]:

words.update(line.split())

return words

vocab = to\_vocab(descriptions)

# Step 5: Load the images

import glob

images = '/kaggle / input / flickr8k / flickr\_data / Flickr\_Data / Images/'

# Create a list of all image names in the directory img = glob.glob(images + '\*.jpg')

train\_path = '/kaggle / input / flickr8k / flickr\_data / Flickr\_Data / Flickr\_TextData / Flickr\_8k.trainImages.txt'

train\_images = open(train\_path, 'r', encoding = 'utf-8').read().split("\n") train\_img = [] # list of all images in training set

for im in img:

if(im[len(images):] in train\_images): train\_img.append(im)

# load descriptions of training set in a dictionary. Name of the image will act as ey def load\_clean\_descriptions(des, dataset):

dataset\_des = dict()

for key, des\_list in des.items(): if key+'.jpg' in dataset:

if key not in dataset\_des: dataset\_des[key] = list()

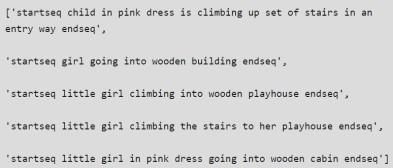
for line in des\_list:

desc = 'startseq ' + line + ' endseq'

return dataset\_des

dataset\_des[key].append(desc)

train\_descriptions = load\_clean\_descriptions(descriptions, train\_images) print(train\_descriptions['1000268201\_693b08cb0e'])



# Step 6: Extract the feature vector from all images

from keras.preprocessing.image import load\_img, img\_to\_array def preprocess\_img(img\_path):

# inception v3 excepts img in 299 \* 299 \* 3

img = load\_img(img\_path, target\_size = (299, 299)) x = img\_to\_array(img)

# Add one more dimension

x = np.expand\_dims(x, axis = 0) x = preprocess\_input(x)

return x

def encode(image):

image = preprocess\_img(image) vec = model.predict(image)

vec = np.reshape(vec, (vec.shape[1])) return vec

base\_model = InceptionV3(weights = 'imagenet')

model = Model(base\_model.input, base\_model.layers[-2].output) # run the encode function on all train images and store the feature vectors in a list encoding\_train = {}

for img in train\_img: encoding\_train[img[len(images):]] = encode(img)

# Step 7: Tokenizing the vocabulary

# list of all training captions all\_train\_captions = []

for key, val in train\_descriptions.items(): for caption in val:

all\_train\_captions.append(caption)

# consider only words which occur atleast 10 times vocabulary = vocab

threshold = 10 # you can change this value according to your need word\_counts = {}

for cap in all\_train\_captions: for word in cap.split(' '):

word\_counts[word] = word\_counts.get(word, 0) + 1

vocab = [word for word in word\_counts if word\_counts[word] >= threshold]

# word mapping to integers ixtoword = {}

wordtoix = {}

ix = 1

for word in vocab: wordtoix[word] = ix ixtoword[ix] = word ix += 1

# find the maximum length of a description in a dataset max\_length = max(len(des.split()) for des in all\_train\_captions) max\_length

# Step 8: Glove vector embeddings

X1, X2, y = list(), list(), list()

for key, des\_list in train\_descriptions.items(): pic = train\_features[key + '.jpg']

for cap in des\_list:

seq = [wordtoix[word] for word in cap.split(' ') if word in wordtoix] for i in range(1, len(seq)):

in\_seq, out\_seq = seq[:i], seq[i]

in\_seq = pad\_sequences([in\_seq], maxlen = max\_length)[0] out\_seq = to\_categorical([out\_seq], num\_classes = vocab\_size)[0]

# store X1.append(pic) X2.append(in\_seq)

y.append(out\_seq) X2 = np.array(X2)

X1 = np.array(X1) y = np.array(y)

# load glove vectors for embedding layer embeddings\_index = {}

golve\_path ='/kaggle / input / glove-global-vectors-for-word-representation / glove.6B.200d.txt' glove = open(golve\_path, 'r', encoding = 'utf-8').read()

for line in glove.split("\n"): values = line.split(" ") word = values[0]

indices = np.asarray(values[1: ], dtype = 'float32') embeddings\_index[word] = indices

emb\_dim = 200

emb\_matrix = np.zeros((vocab\_size, emb\_dim)) for word, i in wordtoix.items():

emb\_vec = embeddings\_index.get(word) if emb\_vec is not None:

emb\_matrix[i] = emb\_vec emb\_matrix.shape

# Step 9: Define the model

# define the model

ip1 = Input(shape = (2048, )) fe1 = Dropout(0.2)(ip1)

fe2 = Dense(256, activation = 'relu')(fe1) ip2 = Input(shape = (max\_length, ))

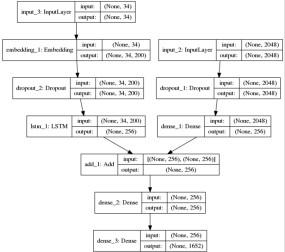
se1 = Embedding(vocab\_size, emb\_dim, mask\_zero = True)(ip2) se2 = Dropout(0.2)(se1)

se3 = LSTM(256)(se2)

decoder1 = add([fe2, se3])

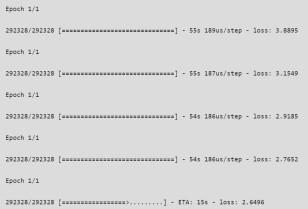
decoder2 = Dense(256, activation = 'relu')(decoder1)

outputs = Dense(vocab\_size, activation = 'softmax')(decoder2) model = Model(inputs = [ip1, ip2], outputs = outputs)



**Step 10: Training the model** model.layers[2].set\_weights([emb\_matrix]) model.layers[2].trainable = False

model.compile(loss = 'categorical\_crossentropy', optimizer = 'adam') model.fit([X1, X2], y, epochs = 50, batch\_size = 256)



# Step 11: Predicting the output

def greedy\_search(pic): start = 'startseq'

for i in range(max\_length):

seq = [wordtoix[word] for word in start.split() if word in wordtoix]

seq = pad\_sequences([seq], maxlen = max\_length) yhat = model.predict([pic, seq])

yhat = np.argmax(yhat) word = ixtoword[yhat] start += ' ' + word

if word == 'endseq': break

final = start.split() final = final[1:-1] final = ' '.join(final) return final





# CONCLUSION :

implemented a simple image caption generator model using an LSTM-based Recurrent Neural Network. The model was trained on a small dataset to predict the image caption.