Image caption generator

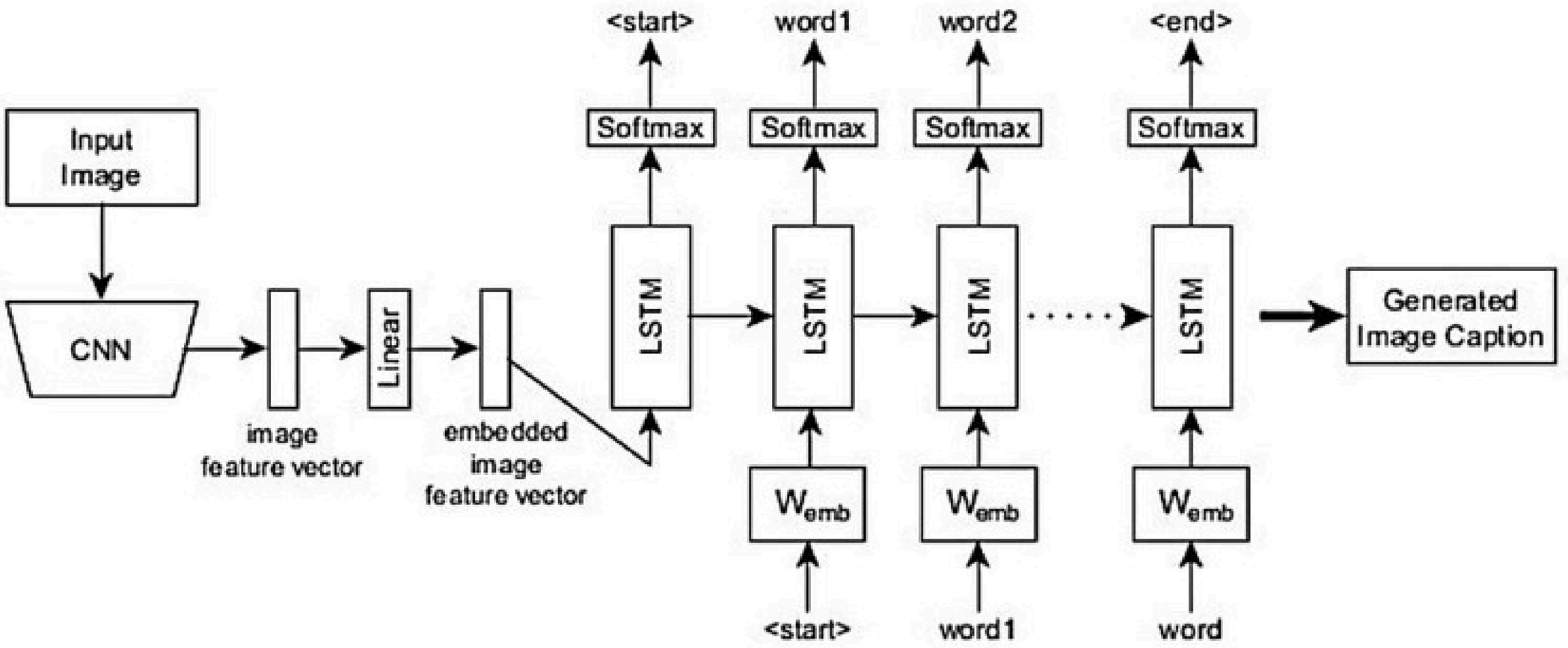
Input

Predicted Output

Black dog is running through the grass.

Methodologies To Be Used

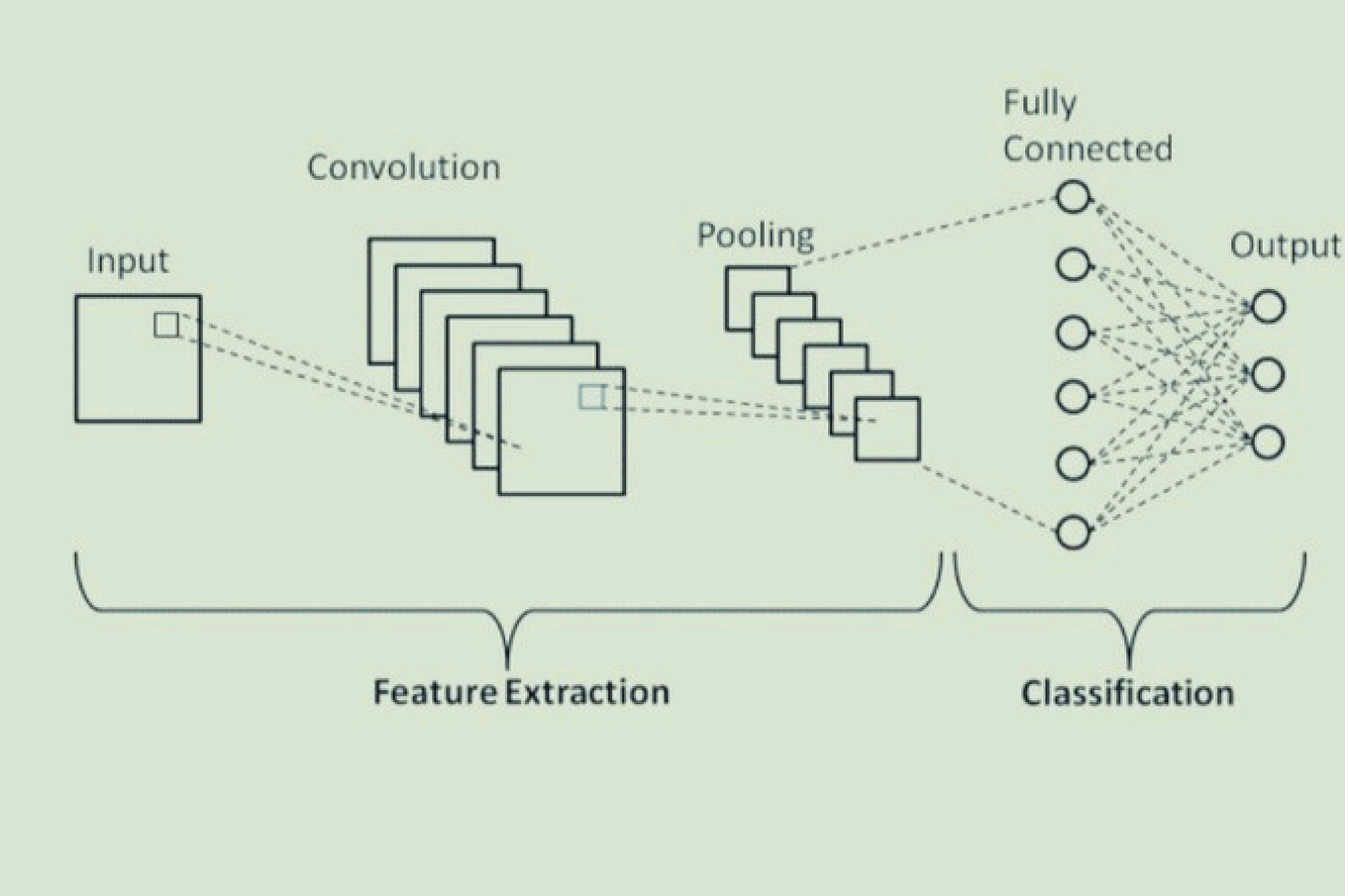
Working of Image Caption Generator



Convolutional Neural Network

- **A Convolutional Neural Network (CNN) is a type of deep learning algorithm**
- **Here it is use for image recognition and processing tasks.**
- **It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers.**

CNN - Architecture



A CNN architecture consists of two key components:

- A convolution tool that separates and identifies the distinct features of an image for analysis in a process known as Feature Extraction.
- A fully connected layer that takes the output of the convolution process and predicts the image's class based on the features retrieved earlier.

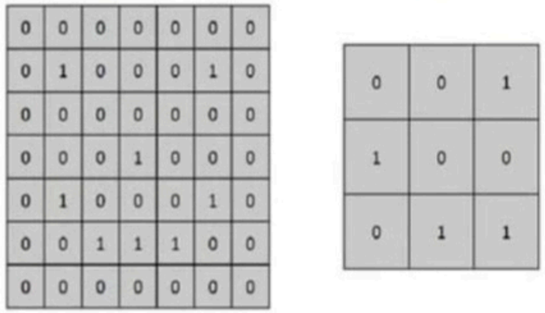
Convolutional layer

This is the very first layer in the CNN that is responsible for the extraction of the different features from the input images.

The convolution mathematical operation is done between the input image and a filter of a specific size MxM in this layer.

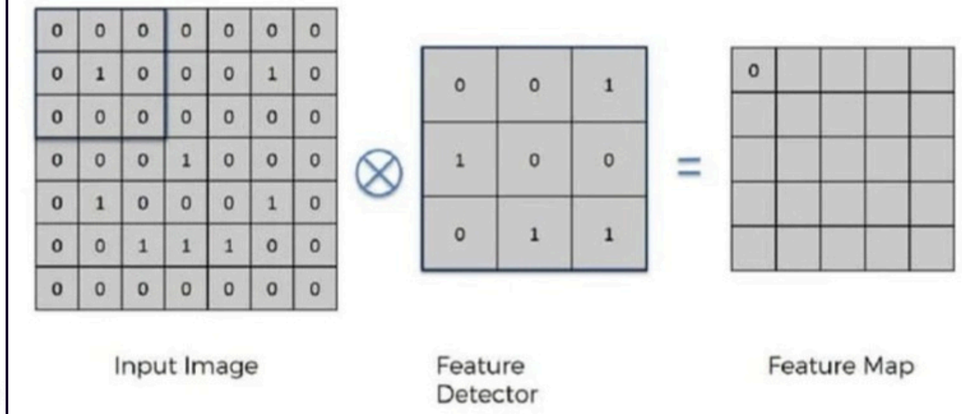
Convolutional layer

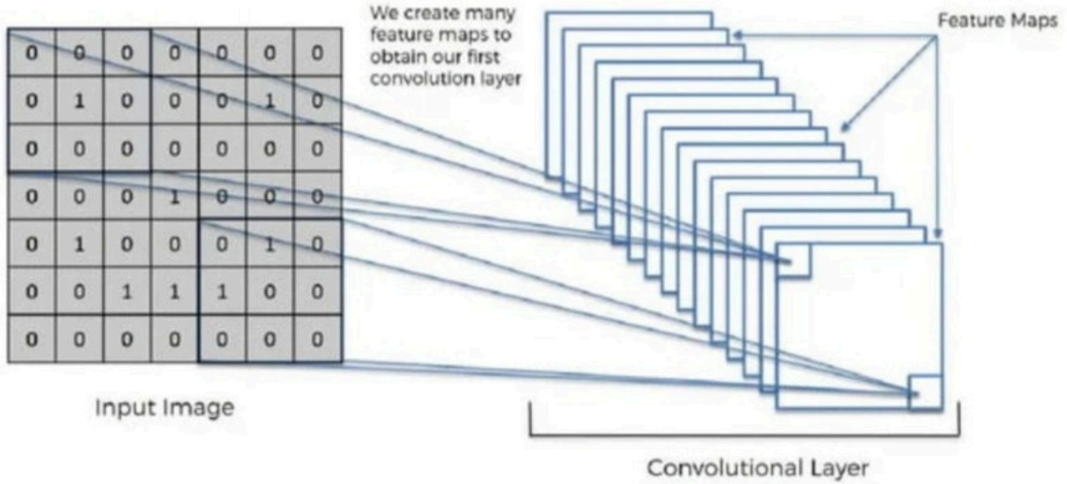
Example -



Feature Detector/Filter/Kernel:
It extracts some features from our image, stores in separate 2D array and compress the image.

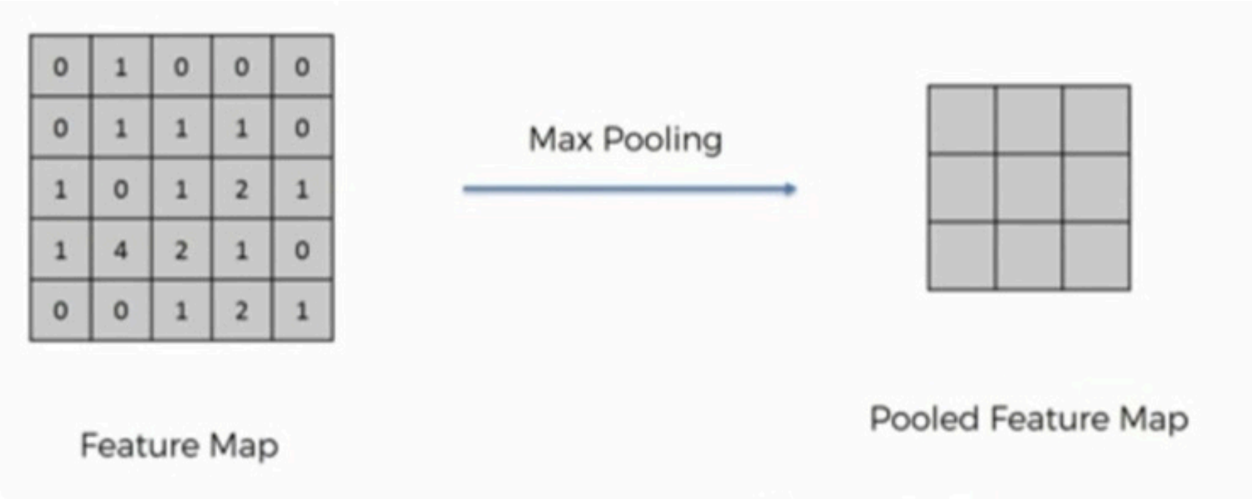
HOW?
We are going to match Feature detector with original image to compress it.





Pooling layer

The Pooling layer is responsible for the reduction of the size(spatial) of the Convolved Feature. It is usually applied after a Convolutional Layer. The Pooling Layer is typically used to connect the Convolutional Layer and the FC Layer.

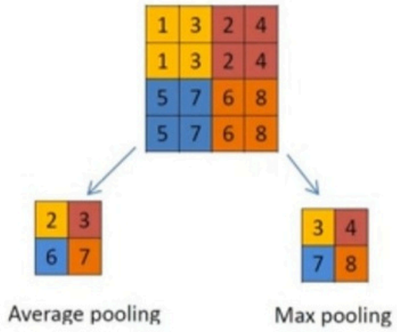


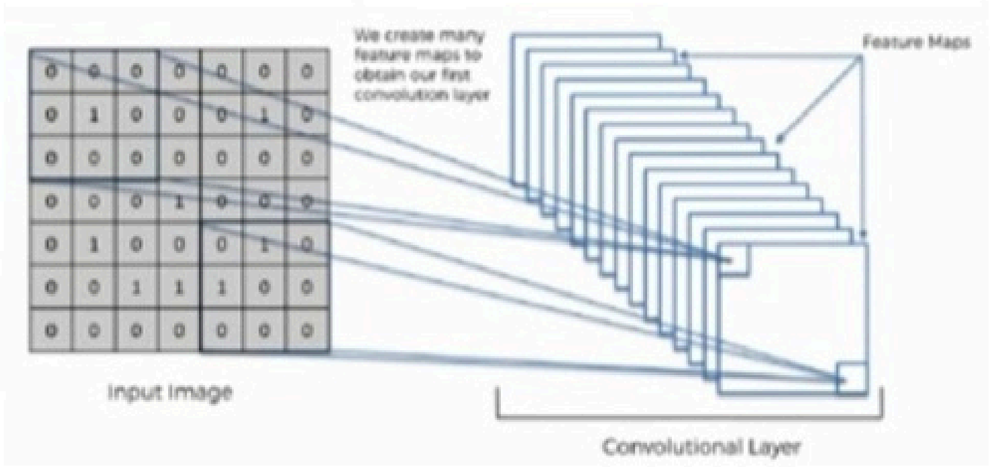
Pooling layer

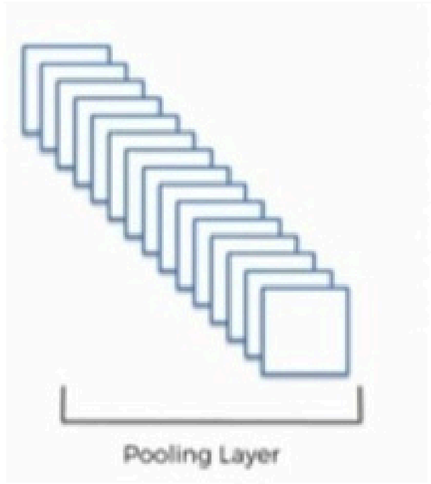
Max / Avg. Pooling

There are two types of pooling-

1. Max pooling - In this pooling the largest element is obtained from the feature map.
2. Average pooling - In this pooling the average of the elements in a predefined sized Image segment is calculated .

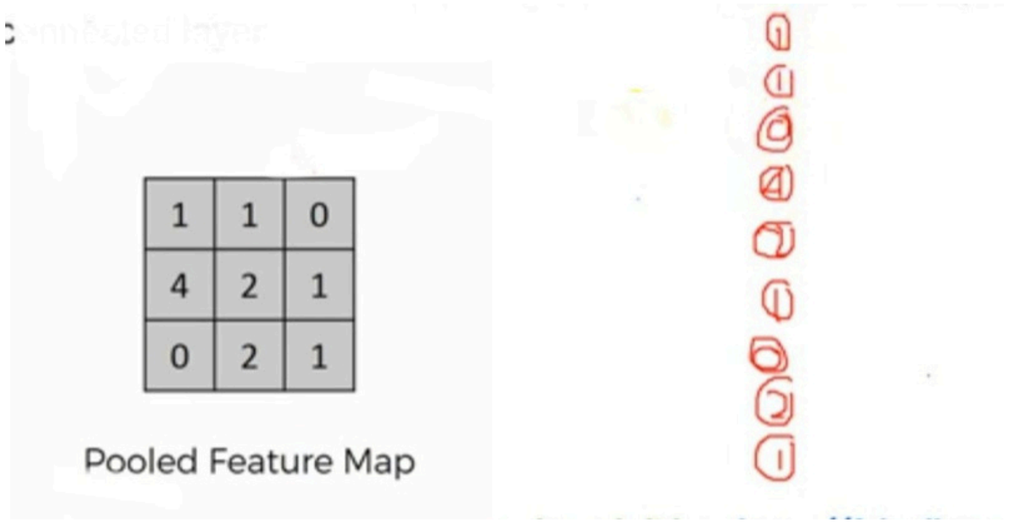






Fully Connected layer

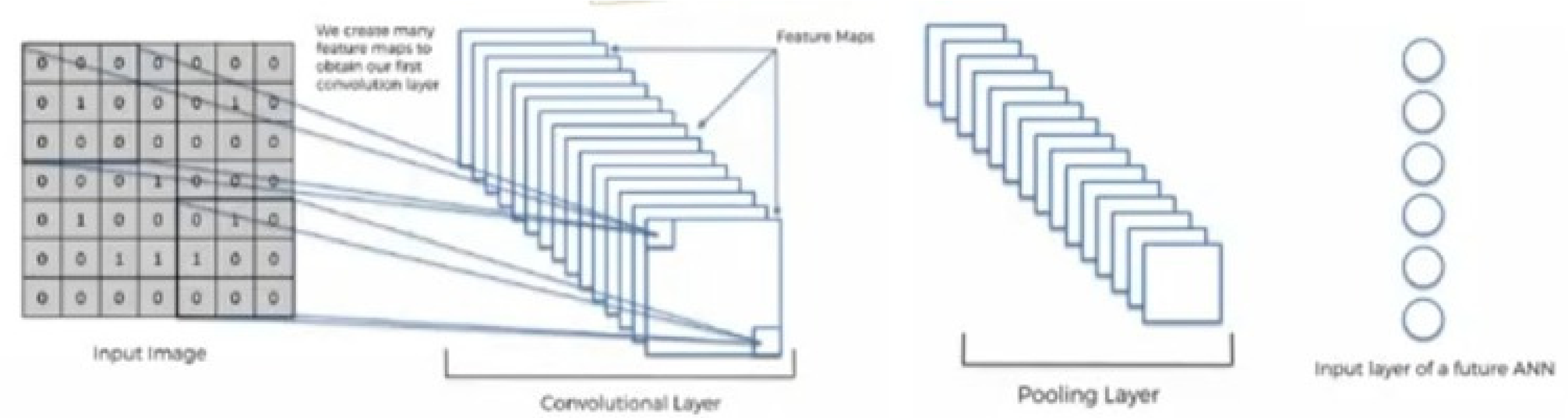
Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer.



Fully Connected layer

Fully Connected (FC) layer comprises the weights and biases together with the neurons and is used to connect the neurons between two separate layers.

The last several layers of a CNN Architecture are usually positioned before the output layer.



LSTM(Long Short Term Memory)

- LSTM is a type of RNN (Recurrrent Neural Netwok)
- Here, it is used for sequence prediction tasks.
- It incorporates feedback connections, which allow it to process entire sequence of data, not just individual data points.

Why LSTM over RNN?

- LSTM can handle vanishing gradient problem
- RNN can handle only short term dependencies, for long term dependencies LSTM is preferred.

For example,

SENTENCE 1 *The colour of the sky is _____.*

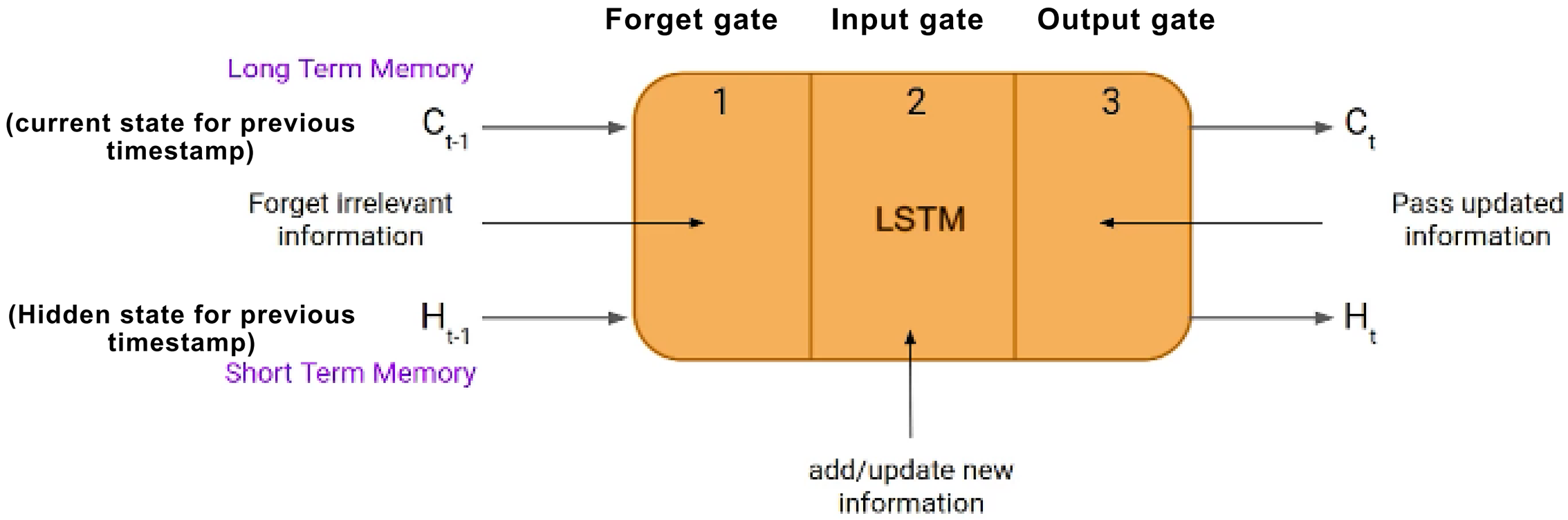
SENTENCE 2 *I spent 20 long years working for the under-privileged kids in Spain. I then moved to Africa.*

.....

I can speak fluent _____.

On sentence 1, RNN will work fluently but on sentence 2 RNN won't give correct results due to long term dependencies.

LSTM - Architecture



LSTM - Forget Gate

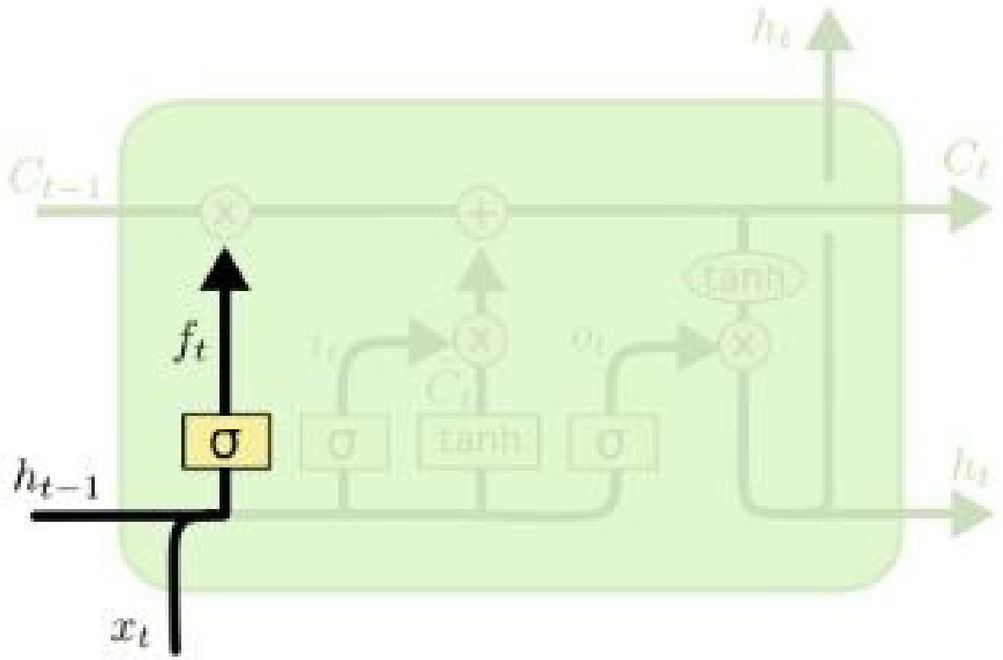
- It is used to decide whether we should keep the information from the previous time step or forget it.
- It is responsible for removing information from the cell state

For example,

Bob is a nice person. Dan on the other hand is evil.

As soon as the first full stop after “person” is encountered, the forget gate realizes that there may be a change of context in the next sentence. As a result of this, the subject of the sentence is forgotten and the place for the subject is vacated.

LSTM - Forget Gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

W is the weight vector

h(t-1) symbolizes hidden state at timestamp t-1

b(f) stands for bias

x(i) is the input

$C_{t-1} * f_t = 0$ ...if $f_t = 0$ (forget everything)

$C_{t-1} * f_t = C_{t-1}$ ...if $f_t = 1$ (forget nothing)

LSTM - Input Gate

- It is used to quantify the importance of the new information carried by the input.
- It is responsible for adding new information to the cell state.

For example,

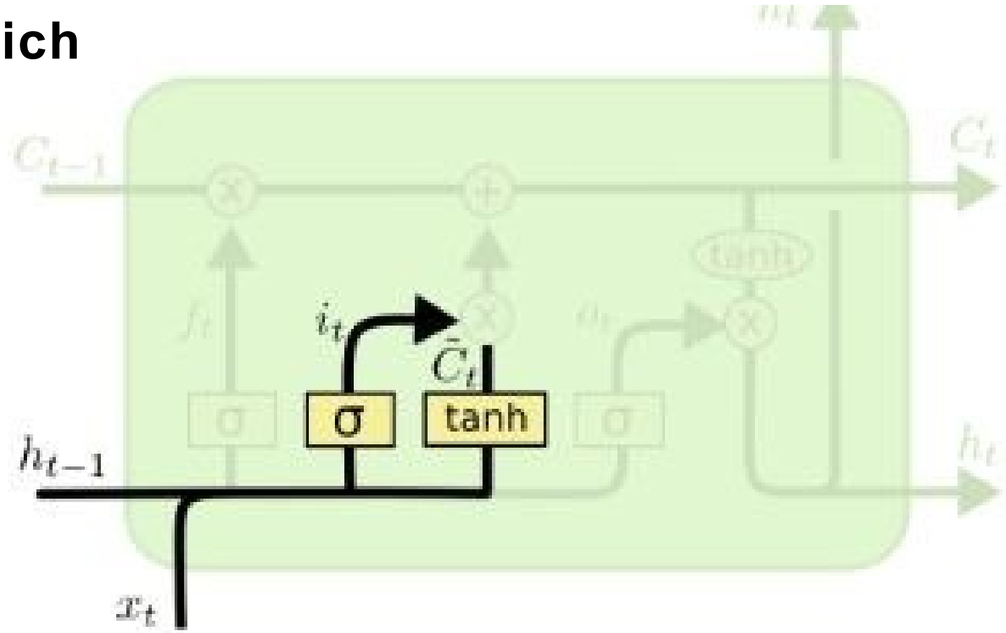
Bob knows swimming. He told me over the phone that he had served the navy for 4 long years.

Here, the important information here is that “Bob” knows swimming and that he has served the Navy for four years. This can be added to the cell state, however, the fact that he told all this over the phone is a less important fact and can be ignored.

LSTM - Input Gate

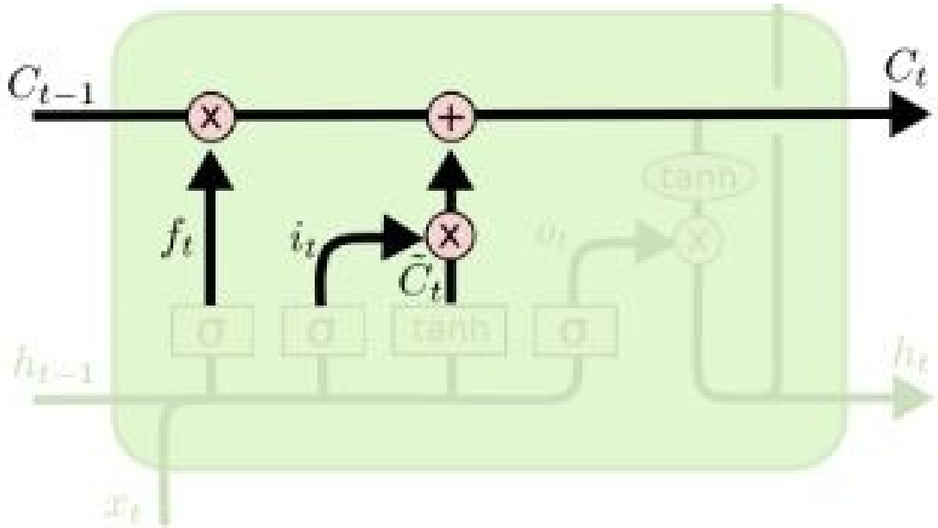
sigmoid layer called the “input gate layer” decides which values to update.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$



tanh layer creates a vector of new candidate values, C(t)hat, that could be added to the state

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



combining these two to create an update to the state

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM - Output Gate

- It selects useful information from the current cell state and show it out as an output

For example,

Bob fought single handedly with the enemy and died for his country. For his contributions brave _____.

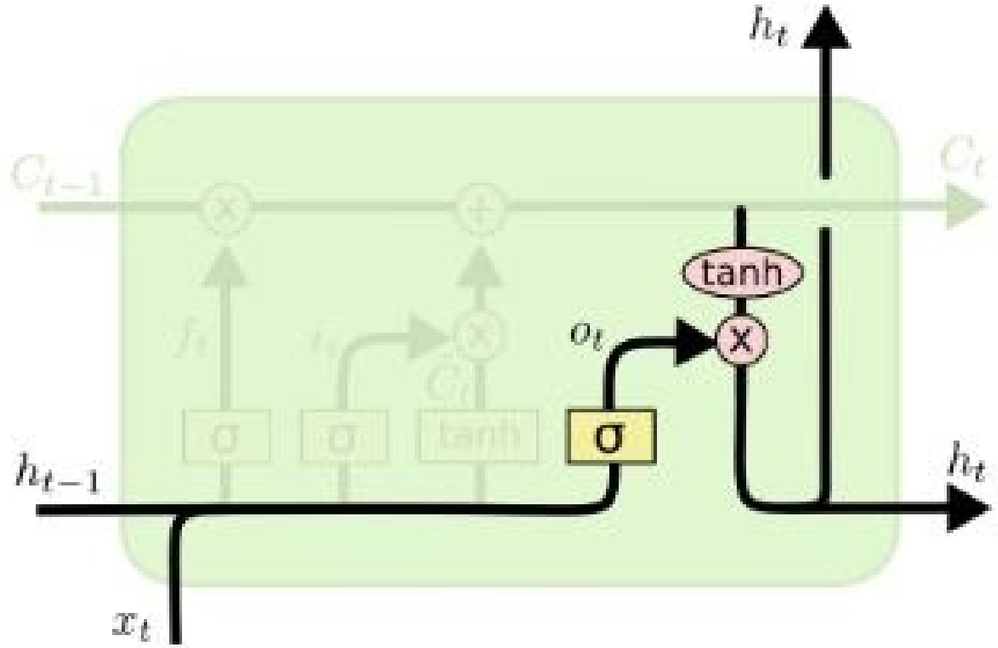
In this phrase, there could be a number of options for the empty space. But we know that the current input of ‘brave’, is an adjective that is used to describe a noun. Thus, whatever word follows, has a strong tendency of being a noun. And thus, Bob could be an apt output.

LSTM - Output Gate

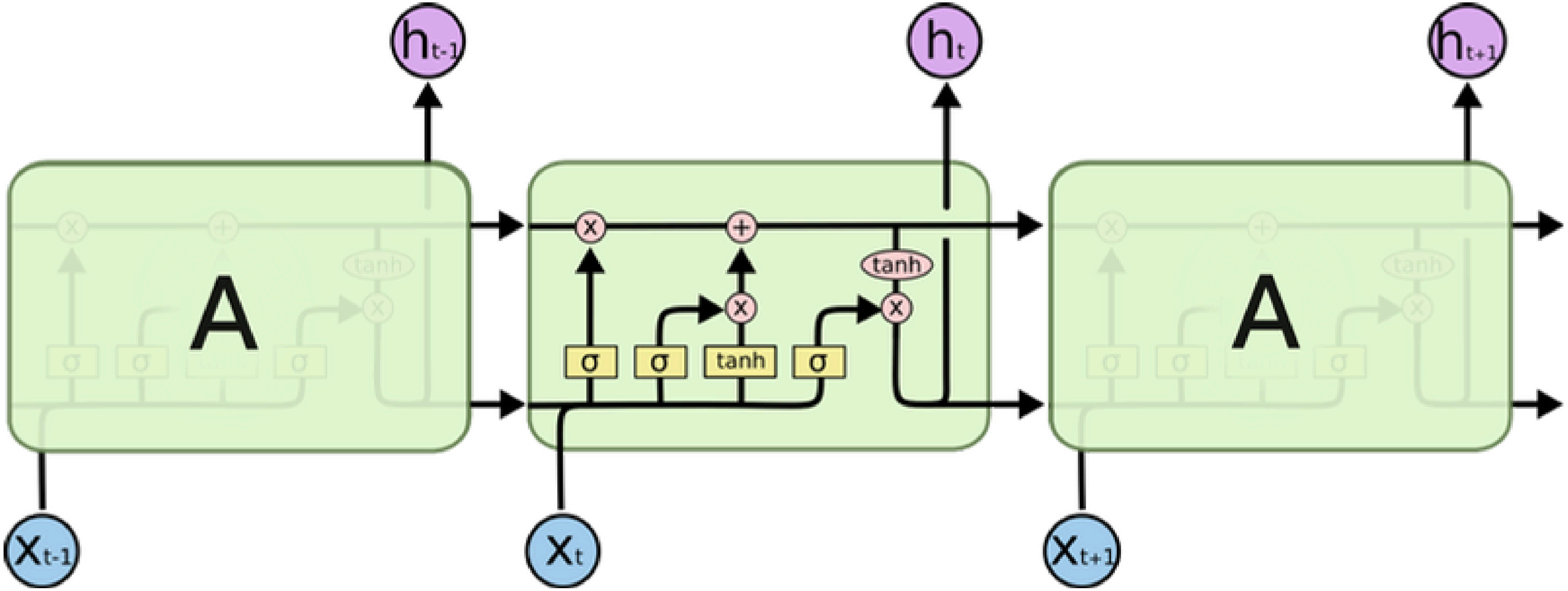
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

calculate the current hidden state

$$h_t = o_t * \tanh\left(C_t\right)$$



LSTM



Setting path of text and image folder

```
[1]  # Set these path according to project folder in you system
     dataset_text = "/content/drive/MyDrive/Flickr8k_text"
     dataset_images = "/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset"
```

```
[2]  print(dataset_text)
```

```
/content/drive/MyDrive/Flickr8k_text
```

```
[3]  #accesing flicker8k.token.txt
     #flicker8k.token.txt has 5 captions corresponding to each image
     filepath = dataset_text + "/" + "Flickr8k.token.txt"
     #file name has path of flicker8k.token.txt
     print(filepath)
```

```
/content/drive/MyDrive/Flickr8k_text/Flickr8k.token.txt
```

Step 1 :- Made dictionary with key as image name and value as five captions corresponding to that image.

Making a dictinoray with key as each unique image and value as the corresponding five captions to that image

```
[ ]  def all_img_captions(filepath):
         #loading the contents of given filepath into file var
         file = open(filepath, 'r')
         text = file.read()
         file.close()
         #captions store the file content splitted into sentences
         captions = text.split('\n')
         print("\n\nAfter splitting the given file content by delimitor new line\nDisplaying top 5 entries",captions[:5])
         descriptions ={}
         for caption in captions[:-1]:
             img, caption = caption.split('\t')
             if img[:-2] not in descriptions:
                 descriptions[img[:-2]] = [ caption ]
             else:
                 descriptions[img[:-2]].append(caption)
         print("\n\nDesciptions ",descriptions)
         return descriptions
```

```
[ ]  descriptions = all_img_captions(filepath)
     print("Length of descriptions =" ,len(descriptions))
```

```
descriptions = all_img_captions(filepath)
print("Length of descriptions =" ,len(descriptions))
```

```
After splitting the given file content by delimitor new line
Displaying top 5 entries ['1000268201_693b08cb0e.jpg#0\tA child in a pink dress is climbing up a set of stairs in an entry way .', '1000268201_693b08cb0e.jpg
```

```
Desciptions  {'1000268201_693b08cb0e.jpg': ['A child in a pink dress is climbing up a set of stairs in an entry way .', 'A girl going into a wooden building
Length of descriptions = 8092
```

Step 2 :- Cleaned the captions and saved in a file descriptions.txt

```
#Data cleaning- lower casing, removing puntuations and words containing numbers
def cleaning_text(descriptions):
  #maketrans is used to make mapping table
  #table contains , map all punctuations charchters to none
    table = str.maketrans('','',string.punctuation)
    for img,caps in descriptions.items():
        #img has image name
        #caps has cooresponding five caption for that image
        for i,img_caption in enumerate(caps):
            #for each image caption of the five caption iterating
            img_caption.replace("-"," ")
            #splitting text to words
            words_caption = img_caption.split()

            #converts to lowercase
            words_caption = [word.lower() for word in words_caption]
            #remove punctuation from each token
            words_caption = [word.translate(table) for word in words_caption]
            #remove hanging 's and a
            words_caption = [word for word in words_caption if(len(word)>1)]
            #remove tokens with numbers in them
            desc = [word for word in words_caption if(word.isalpha())]
            #convert back to string

            img_caption = ' '.join(desc)
            descriptions[img][i]= img_caption
    return descriptions
```

```
#cleaning the descriptions
clean_descriptions = cleaning_text(descriptions)
print("\n\nClean descriptions ",clean_descriptions)
print("Length of clean descriptions =" ,len(clean_descriptions))
```

```
Clean descriptions  {'1000268201_693b08cb0e.jpg': ['child in pink dress is climbing up set of stairs in an entry way',
Length of clean descriptions = 8092
```

Step 3 :- Created Vocabulary from text of the captions

Creating vocabulary from the text of the captions

```
[9]  def text_vocabulary(clean_descriptions):
         #  vocabulary has unique words
         vocab = set()

         for key in clean_descriptions.keys():
             [vocab.update(d.split()) for d in clean_descriptions[key]]

         return vocab
```

```
[10] vocabulary = text_vocabulary(clean_descriptions)
     print("\n\nVocabulary ",vocabulary)
     print("Length of vocabulary = ", len(vocabulary))
```

```
Vocabulary  {'alligator', 'tubing', 'boogieboard', 'bathtub', 'granite', 'brother', 'vike', 'geological', 'vert',
Length of vocabulary =  8763
```

Saving the cleaned descriptions in a file descriptions.txt

```
[11] def save_descriptions(descriptions, filename):
         lines = list()
         for key, desc_list in descriptions.items():
             for desc in desc_list:
                 lines.append(key + '\t' + desc )
         data = "\n".join(lines)
         file = open(filename,"w")
         file.write(data)
         file.close()
```

```
[12] #saving each description to file
     save_descriptions(clean_descriptions, "descriptions.txt")
```

```
from keras.applications.xception import Xception, preprocess_input
from PIL import Image
from pickle import dump, load
import os
import numpy as np
```

Step 4 :- Extracted features from given images using Xception and saved the extracted features in a file features.txt

```
def extract_features(directory):
        model = Xception( include_top=False, pooling='avg' )
        features = {}
        for img in os.listdir(directory):
          #access each image
            filename = directory + "/" + img
            image = Image.open(filename)
            image = image.resize((299,299))
            image = np.expand_dims(image, axis=0)
            #image = preprocess_input(image)
            image = image/127.5
            image = image - 1.0

            feature = model.predict(image)
            features[img] = feature
        return features
```

```
#2048 feature vector
features = extract_features(dataset_images)
dump(features, open("features.p","wb"))
```

```
features = load(open("features.p","rb"))
print("\n\nFeatures",features)
```

```
0.1539738 ]], dtype=float32), '2275372714_017c269742.jpg': array([[0.01864058, 0.00488617, 0.07784692, ..., 0.        , 0.08814
0.71809554]], dtype=float32), '2518508760_68d8df7365.jpg': array([[0.32870728, 0.05296158, 0.21066357, ..., 0.207057  , 0.00054
0.        ]], dtype=float32), '700884207_d3ec546494.jpg': array([[0.40770125, 0.01982971, 0.8455947 , ..., 0.01840207, 0.039328
0.80744904]], dtype=float32), '3289433994_4c67aab384.jpg': array([[0.10761029, 0.00047564, 0.        , ..., 0.        , 0.04604
0.        ]], dtype=float32), '519059913_4906fe4050.jpg': array([[0.01082998, 0.02830978, 0.00112282, ..., 0.07313032, 0.
```

```
features = load(open("features.p","rb"))
print("\n\nFeatures",features)
```

```
0.1539738 ]], dtype=float32), '2275372714_017c269742.jpg': array([[0.01864058, 0.00488617, 0.07784692, ..., 0.        , 0.08814
0.71809554]], dtype=float32), '2518508760_68d8df7365.jpg': array([[0.32870728, 0.05296158, 0.21066357, ..., 0.207057  , 0.00054
0.        ]], dtype=float32), '700884207_d3ec546494.jpg': array([[0.40770125, 0.01982971, 0.8455947 , ..., 0.01840207, 0.039328
0.80744904]], dtype=float32), '3289433994_4c67aab384.jpg': array([[0.10761029, 0.00047564, 0.        , ..., 0.        , 0.04604
0.        ]], dtype=float32), '519059913_4906fe4050.jpg': array([[0.01082998, 0.02830978, 0.00112282, ..., 0.07313032, 0.
```

Step 5:- Loaded the images of training dataset. (Flickr_8k.trainImages .txt)

Using train image data set to generate captions

```
[20] filename = dataset_text + "/" + "Flickr_8k.trainImages.txt"
```

Loading the images from training dataset

```
[21] def load_doc(filename):
         # Opening the file as read only
         file = open(filename, 'r')
         text = file.read()
         file.close()
         return text
```

```
[22] def load_photos(filename):
         file = load_doc(filename)
         photos = file.split("\n")[:-1]
         return photos
     train_imgs = load_photos(filename)
```

Step 6:- Loaded descriptions corresponding to the images in training dataset.

Loading corresponding descriptions of the image in training dataset

```
def load_clean_descriptions(filename, photos):
    #loading clean_descriptions
    file = load_doc(filename)
    descriptions = {}
    #for each sentence
    for line in file.split("\n"):
        words = line.split()
        if len(words)<1 :
            continue
        image, image_caption = words[0], words[1:]
        #if image is in training dataset
        if image in photos:
          #but its decription is not in description
            if image not in descriptions:
                descriptions[image] = []
            desc = '<start> ' + " ".join(image_caption) + ' <end>'
            descriptions[image].append(desc)

    return descriptions
```

```
[24] train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)
```

Loading features of the images in the training dataset

```
[25] def load_features(photos):
         #loading all features
         all_features = load(open("features.p","rb"))
         #creating dictionary of features of photos in training dataset
         features = {k:all_features[k] for k in photos}
         return features
```

```
[26] train_features = load_features(train_imgs)
```

Step 7:- Created sequence from tokenised words

Giving each word an index and storing it in tokenizer.p

```
[27] def dict_to_list(descriptions):
         all_desc = []
         for key in descriptions.keys():
             [all_desc.append(d) for d in descriptions[key]]
         return all_desc
```

```
[28] from keras.preprocessing.text import Tokenizer

     def create_tokenizer(descriptions):
         desc_list = dict_to_list(descriptions)
         tokenizer = Tokenizer()
         #To create a dictionary for the entire corpus (a map
         tokenizer.fit_on_texts(desc_list)
         return tokenizer
```

Step 8:- Generated a vocabulary from training image captions.

```
# give each word an index, and store that into tokenizer.p pickle file
tokenizer = create_tokenizer(train_descriptions)
#dump is used to write data to file
dump(tokenizer, open('tokensizer.txt', 'wb'))
vocab_size = len(tokenizer.word_index) + 1
print("\nVocabulary size of training dataset : ", vocab_size)
```

```
Vocabulary size of training dataset :  7577
```

```
#calculate maximum length of descriptions
def max_length(descriptions):
    desc_list = dict_to_list(descriptions)
    return max(len(d.split()) for d in desc_list)

max_length = max_length(descriptions)
print("\nMaximum length of description : ",max_length)
```

```
Maximum length of description :  32
```

Steps done in implementation

Step 9:- Created Data Generator

[x1, x2] are the input of our model, and y act as output, where x1 shows 2048 feature vectors of the image, x2 shows the input text sequence and y shows the output text sequence that is predicted by the model.

x1(feature vector)	x2(Text sequence)	y(word to predict)
feature	start,	two
feature	start, two	dogs
feature	start, two, dogs	drink
feature	start, two, dogs, drink	water
feature	start, two, dogs, drink, water	end

Creating sequence from the tokenised words

```
[ ]  from keras.preprocessing.sequence import pad_sequences
     from keras.utils import to_categorical
```

```
def create_sequences(tokenizer, max_length, desc_list, feature):
    X1, X2, y = list(), list(), list()
    # walk through each description for the image
    for desc in desc_list:
        #print("\n\ntokenizer.texts_to_sequences([desc])\n",tokenizer.texts_to_sequences([desc]))
        # encode the sequence
        seq = tokenizer.texts_to_sequences([desc])[0]
        #print("\n\nseq ",seq)
        # split one sequence into multiple X,y pairs
        for i in range(1, len(seq)):
            # split into input and output pair
            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]
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)
```

Creating a data generator

```
[ ]  def data_generator(descriptions, features, tokenizer, max_length):
         while 1:
             for key, description_list in descriptions.items():
                 #retrieve photo features
                 feature = features[key][0]
                 input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, feature)
                 #The yield keyword in Python is similar to a return statement used for returning values in Python which returr
                 yield [[input_image, input_sequence], output_word]
```

```
[ ]  #`next` keyword is used to retrieve the next item from an iterator.
     [a,b],c = next(data_generator(train_descriptions, features, tokenizer, max_length))
```

```
from keras.models import Model, load_model
from tensorflow.keras import layers
from keras.layers import Input, Dense, LSTM, Embedding, Dropout
import tensorflow as tf
```

```
from keras.utils import plot_model
```

Steps done in implementation

Step 10:- Defined CNN-RNN model

Feature Extractor –With a dense layer, it will extract the feature from the images of size 2048 and we will decrease the dimensions to 256 nodes.

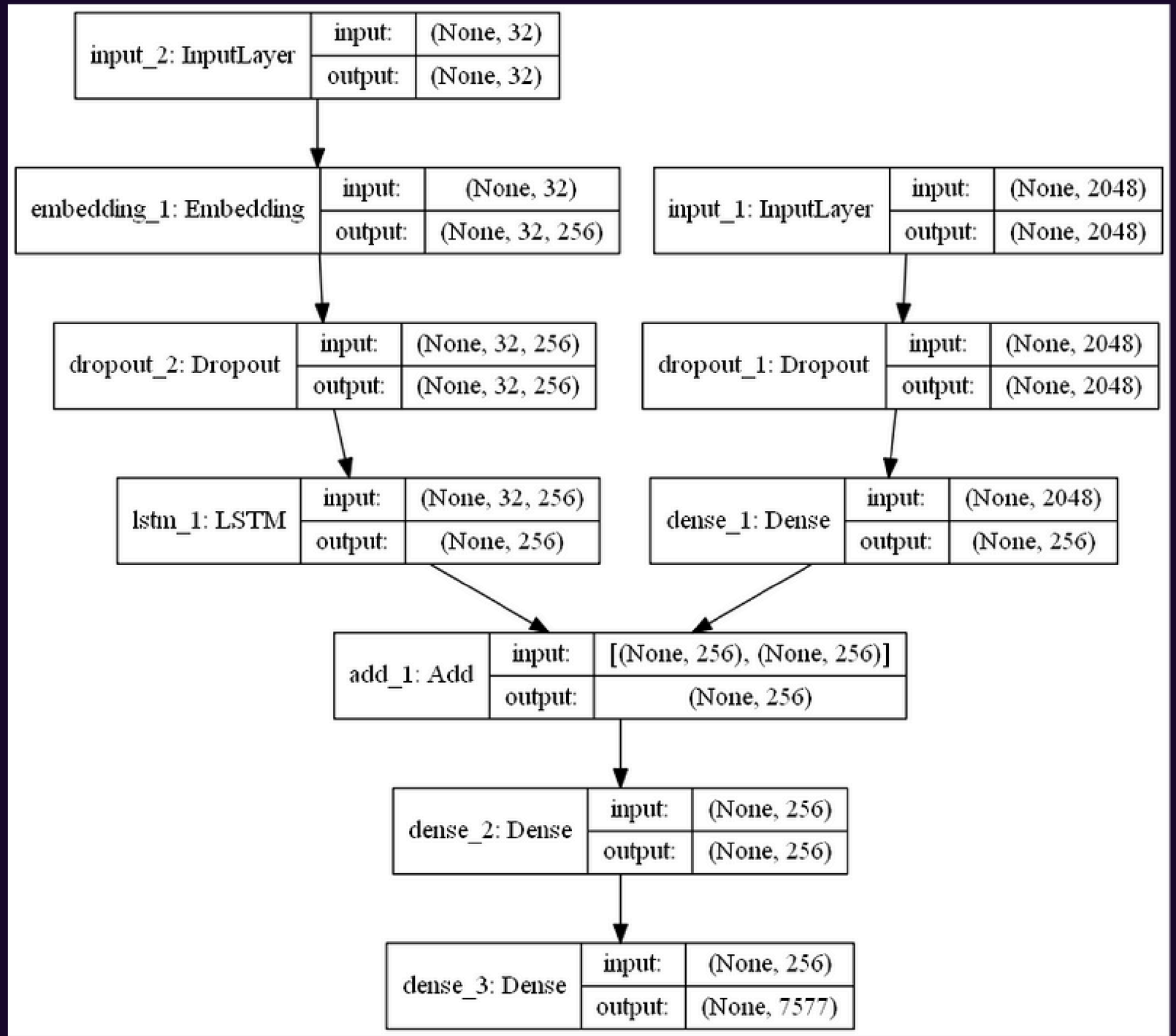
Sequence Processor – Followed by the LSTM layer, the textual input is handled by this embedded layer.

Decoder – We will merge the output of the above two layers and process the dense layer to make the final prediction.

```
from keras.utils import plot_model

# define the captioning model
def define_model(vocab_size, max_length):
    # features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    # LSTM sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)
    # Merging both models
    #decoder1 = add([fe2, se3])
    decoder1 = tf.keras.layers.Add()([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    # summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)
    return model
```

Step 11:- Trained the model on training dataset

```
model = define_model(vocab_size, max_length)
steps = len(train_descriptions)
# making a directory models to save our models
os.mkdir("models4")
#for i in range(epochs):
generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
model.fit(generator)
model.save("models4/model_")
```

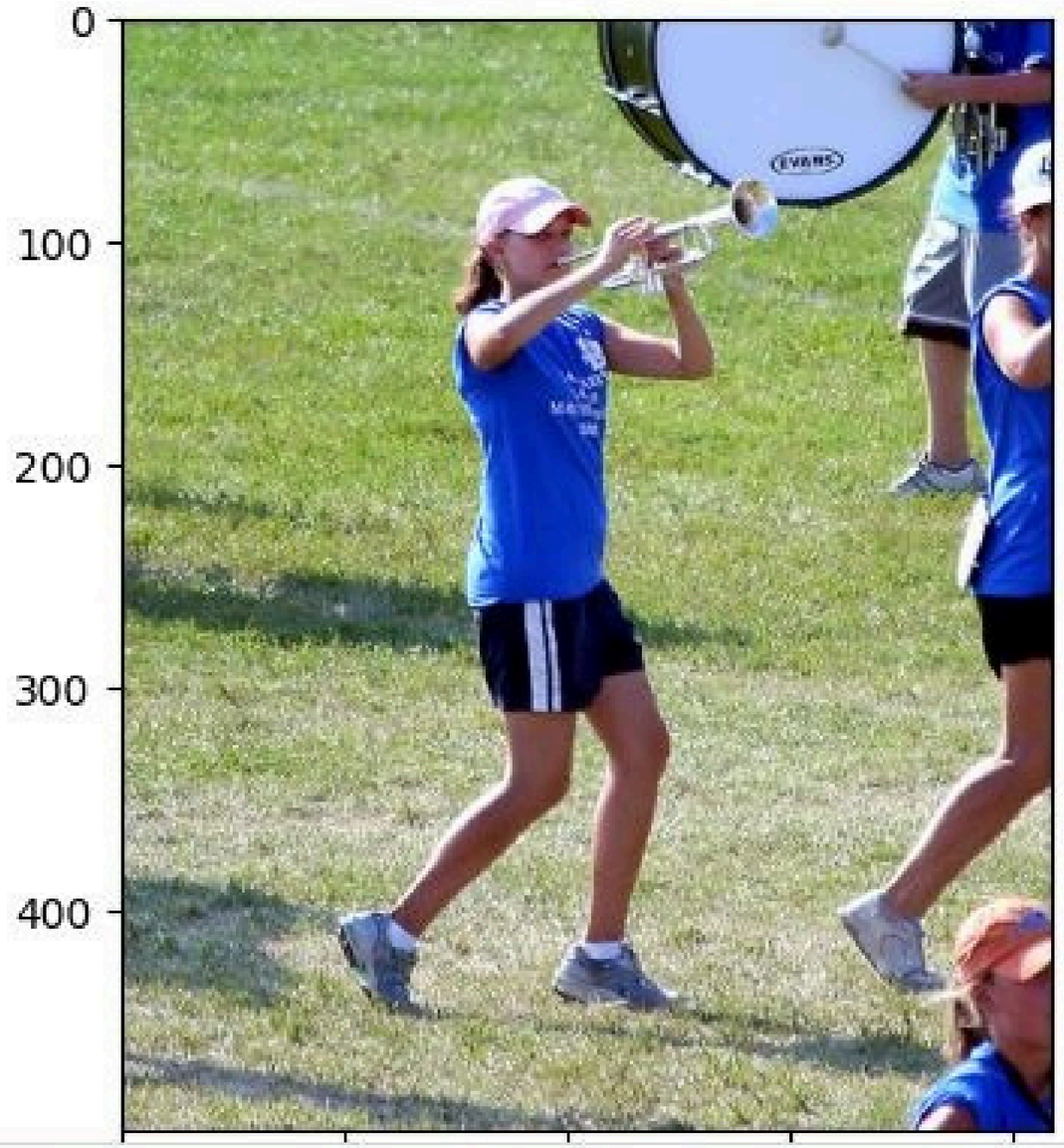

input_2: InputLayer
input: (None, 32)
output: (None, 32)
embedding_1: Embedding
input: (None, 32)
output: (None, 32, 256)
input_1: InputLayer
input: (None, 2048)
output: (None, 2048)
dropout_2: Dropout
input: (None, 32, 256)
output: (None, 32, 256)
dropout_1: Dropout
input: (None, 2048)
output: (None, 2048)
lstm_1: LSTM
input: (None, 32, 256)
output: (None, 256)
dense_1: Dense
input: (None, 2048)
output: (None, 256)
add_1: Add
input: [(None, 256), (None, 256)]
output: (None, 256)
dense_2: Dense
input: (None, 256)
output: (None, 256)
dense_3: Dense
input: (None, 256)
output: (None, 7577)

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


def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'start'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        pred = model.predict([photo,sequence], verbose=0)
        pred = np.argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'end':
            break
    return in_text
```

Step 12 : - Tested the trained model

```
#path = 'Flicker8k_Dataset/111537222_07e56d5a30.jpg'
max_length = 32
tokenizer = load(open("tokenizer.p","rb"))
model = load_model('models2/model_9.h5')
xception_model = Xception(include_top=False, pooling="avg")
img_path = '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1153704539_542f7aa3a5.jpg'
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)

description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
start two girls are playing basketball end
<matplotlib.image.AxesImage at 0x7e2cced897e0>
```



BIBLIOGRAPHY

- https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- IMAGE CAPTION GENERATOR USING DEEP LEARNING MODEL By Namitha O and Kavitha D
- https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/

THANK YOU