Cairo University

Faculty Of Computers and Artificial Intelligence

Information Technology Department

Automatic Image Captioning

By :

Youssef Bahgat	20180339
Mahmoud Ahmed	20180249
Mikhael elkess Youhanna	20180301
Mohamed Mahmoud Abdelaliem	20170255
Mahmoud Ragab Hanafy	20170265

Supervised by: DR. Mona Solyman

July 2022

Abstract

Image captioning aims to automatically generate a sentence description for an image. Our project model will take an image as input and generate an English sentence as output, describing the contents of the image. It has attracted much research attention in cognitive computing in recent years. The task is rather complex, as the concepts of both computer vision and natural language processing domains are combined together. We have developed a model using the concepts of a Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) model and build a working model of Image caption generator by implementing CNN with LSTM. CNN works as an encoder to extract features from images and LSTM works as a decoder to generate words describing images. After the caption generation phase, we useBLEUScores to evaluate the efficiency of our model. Thus, our system helps the user to get descriptive caption for the given input image

Table of Contents

List of figures

1. Introduction

1.1 Overview and Problem Definition

Captioning images automatically is one of the heart of the human visual system. There are various advantages if there is an application which automatically caption the scenes surrounded by them and revert back the caption as a plain message.

In this paper, we present a model based on CNN-LSTM neural networks which automatically detects the objects in the images and generates descriptions for the images. It uses various pre-trained models to perform the task of detecting objects and uses CNN and LSTM to generate the captions. It uses Transfer Learning based pre-trained models for the task of object Detection. This model can perform two operations. The first one is to detect objects in the image using Convolutional Neural Networks (CNN) and the other is to caption the images using RNN based LSTM(Long Short Term Memory). Interface of the model is developed using flask rest API, which is a web development framework of python. The main use case of this project is to help the visually impaired to understand the surrounding environment and act according to that. Also, self- driving cars are one of the biggest challenges.

Caption generation is one of the interesting and focussed areas of Artificial Intelligence which has many challenges to pass on. Caption generation involves various complex scenarios starting from picking the dataset, training the model, validating the model, creating pre-trained models to test the images ,detecting the images and finally generating the captions.

1.2. Motivation and Objective

We can use auto image captioning to describe images to **aid blind people** or visually impaired people as they can rely on sound and texts for the description of a scene, **automatic driving (Self Driving Car)** is one of the biggest challenges and if we can properly caption the scene around the car, it can give a boost to the self driving system, and automatic captioning can help make **google image search** as good as google search, as then every image could be first converted into a caption and then search can be performed based on the caption.
Auto-caption can help with E-commerce. Artificial intelligence (AI)-assisted Product Information Management systems can analyze photos and provide rich and detailed attributes for web catalogs automatically. Image captioning software can evaluate product photos and automatically recommend relevant qualities and categories, which saves time and money. The system may, for example, detect the type of fashion item, its material, color, design, etc.
In security applications, Annotating data can be used in many applications, such as: classification of objects on security monitors so we can raise alarms as soon as any malicious activity is detected, Detection of weapons and/or dangerous things and Annotation for face recognition

2. Data Collection

There are many open source datasets available for this problem, like Flickr8K dataset (Containing 8k images) it is good to use this dataset, reason why it is small and realistic so that you can download it and build your model on your machine using the CPU, flickr30k dataset (Containing 30k images)

can also use MS COCO dataset (Containing over 180k images) COCO stands for Common Objects in Context, it is a large-scale object detection, segmentation, and captioning dataset published by Microsoft.

We have mainly two types of data: Images and Captions (text).

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

two dogs on pavement moving toward each other .
two dogs of different breeds looking at each other on the road .
a black dog and a white dog with brown spots are staring at each other in the street .
a black dog and a tri-colored dog playing with each other on the road .
a black dog and a spotted dog are fighting

young girl with pigtails painting outside in the grass .
there is a girl with pigtails sitting in front of a rainbow painting .
a small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
a little girl is sitting in front of a large painted rainbow .
a little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .

man laying on bench holding leash of dog sitting on ground
a shirtless man lies on a park bench with his dog .
a man sleeping on a bench outside with a white and black dog sitting next to him .
a man lays on the bench to which a white dog is also tied .
a man lays on a bench while his dog sits by him .

Fig .1: shows some of the pictures together with the captions.

3. System Requirement

You must have python 3 and install some libraries like Scipy, Statsmodels, Numpy, Pandas, Matplotlib and Scikit-learn and you must have keras installed with TensorFlow backend.

- ❖ Python Environment
  - ➢ Install Python 3
  - ➢ Download Anaconda
  - ➢ Installed Anaconda
  - ➢ Install deep learning libraries
    - ■ conda install -c conda-forge keras
    - ■ conda install -c conda-forge tensorflow

- ❖ Be familiar with Google Colab and Jupyter notebook
- ❖ Application Front End
  - ➢ HTML, CSS and Bootstrap

- ❖ Application Back End
  - ➢ Flask Python
  - ➢ Flask Rest API

4. Background

4.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (or ConvNet) are very similar to ordinary Neural Networks, they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. and they still have a loss function (e.g. Softmax) on the last fully-connected layer

So what changes..? the architecture of a ConvNet is designed to take advantage of the 2D structure of an input image

This enables us to encode specific properties into the architecture. As a result, the forward function is more efficient to implement, and the number of parameters in the network is greatly reduced.

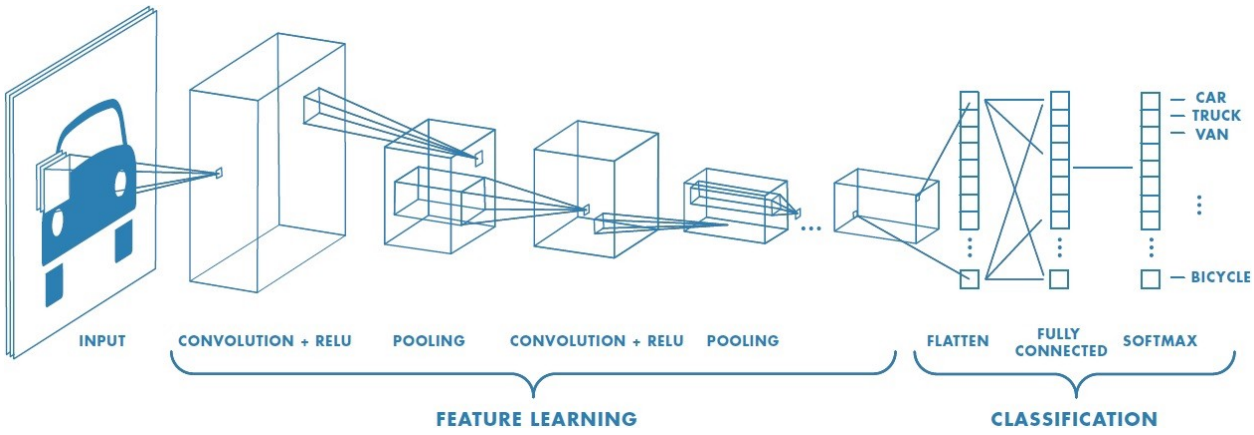


Fig.2: show the CNN Architecture Diagram

A convNet is comprised of one or more convolutional layers (often with a pooling step) and then followed by one or more fully connected layers as in a standard multilayer neural network

4.1.2 Convolution Layer :

The CONV layer's parameters consist of a set of learnable filters. every filter is small spatially (along width and height), but extends through the full depth of the input volume. during the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot product between the entries of the filter and the input at any position

4.1.3 ReLU Layer :

The Rectified Linear Unit (or ReLU) is not a separate component of the convolutional neural It's a supplementary step to the convolution operation

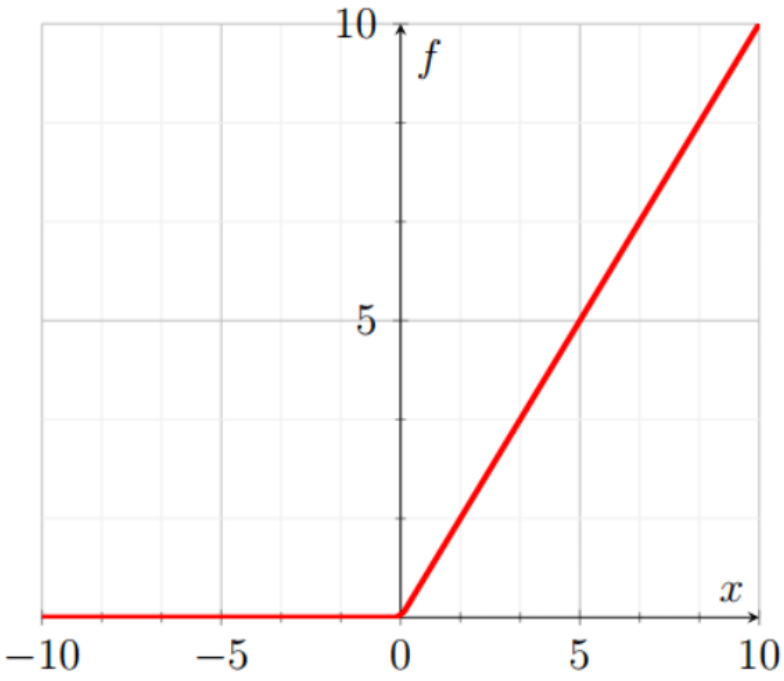


Fig .3: shows the RULE functions

The ReLU layer applies the function f(x) = max(0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0.This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the conv layer.

4.1.4. Pooling layer

The Pooling layer is responsible for reducing the spatial size of the Convolved Feature, It is inserted often in between convolutional layers to reduce dimensionality. Dimension reduction can be obtained by using stride when convolving, leading to dilution of receptive fields overlap

This is to decrease the computational power required to process the data by reducing the dimensions.

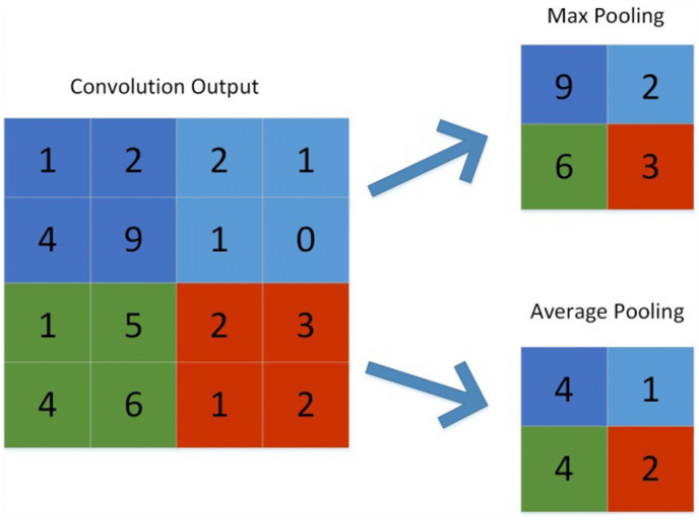


Fig .4: shows the pooling - max / avg

So what we do in Max Pooling is we find the maximum value of a pixel an input is partitioned into non-overlapping rectangles and the layer simply outputs a grid of maximum values of each rectangle.

On the other hand, Average Pooling returns the average of all the values from the portion of the image.

4.1.5 Case Study : VGG-16 net

VGG 16 abbreviation for (Visual Geometry Group) is a convolutional neural network model proposed by Karen Simonyan and Andrew Zisserman from the University of Oxford, in this paper Very Deep Convolutional Networks for Large-Scale Image Recognition. The model archives 92.7% top-5 test accuracy in ImageNet Challenge in 2014, which is a dataset of over 14 million images belonging to 1000 class

It made improvements over AlexNet architecture by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.

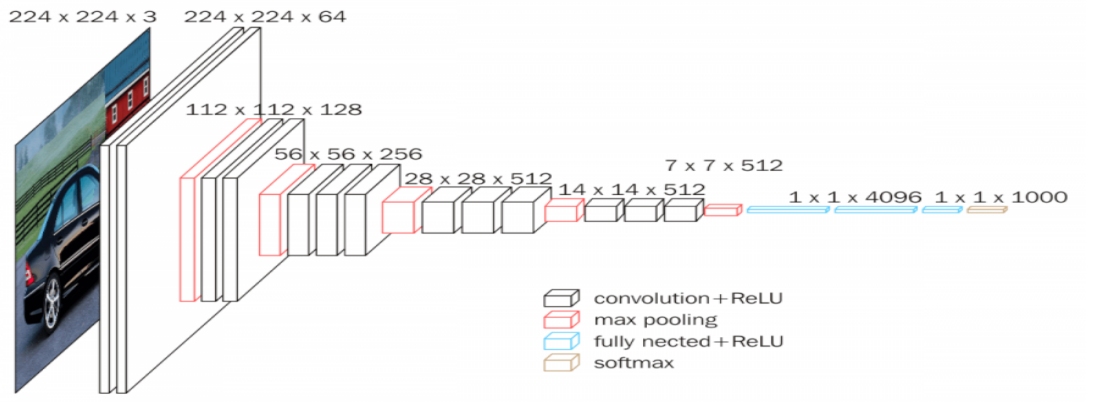


Fig .5: shows the VGG-16 Network Architecture

The input to VGG-16 is an image with size 224 × 224 × 3. Because the padding is one in the convolution kernels (meaning one row or column is added outside of the four edges of the input), convolution will not change the spatial extent. The pooling layers will reduce the input size by a factor of 2. Hence, the output after the last (5th) pooling layer has spatial extent 7 × 7 (and 512 channels). We may interpret this tensor as 7 × 7 × 512 = 25088 "features". The first fully connected layer converts them into 4096 features.

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

Fig .6: shows re_structure of the model

In our case we are going to remove the final fully connected layer that classifies the image

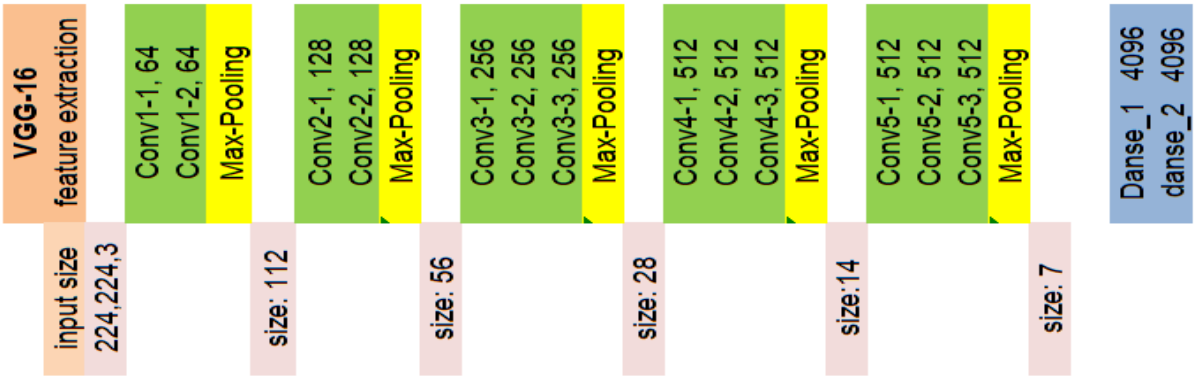


Fig .7: shows VGG-16 feature extraction diagram

Input size of image	Trainable Params	Image features	Extracted Features
224x224x3	134,260,544	1x4096	8091

4.2 RNN model overview :

Recurrent neural networks (RNN) are the state-of-the-art algorithm for sequential data and are used by Apple's Siri and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. It is one of the algorithms behind the scenes of the amazing achievements seen in deep learning over the past few years. In this post, we'll cover the basic concepts of how recurrent neural networks work, what the biggest issues are and how to solve them. Because of their internal memory, RNN's can remember important things about the input they received, which allows them to be very precise in predicting what's coming next. This is why they're the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more. Recurrent neural networks can form a much deeper understanding of a sequence and its context compared to other algorithms.

Simply Recurrent neural networks (RNN) are a class of neural networks that are helpful in modeling sequence data. Derived from feedforward networks, RNNs exhibit similar behavior to how human brains function. Simply put: recurrent neural networks produce predictive results in sequential data that other algorithms can’t.

How Does RNN Work?

You'll need a working knowledge of "regular" feed-forward neural networks and sequential data to fully comprehend RNNs.

Sequential data is simply ordered data in which related items appear one after the other. Financial data or the DNA sequence are two examples. Perhaps the most common sort of sequential data is time series data, which is just a list of data points in chronological order.

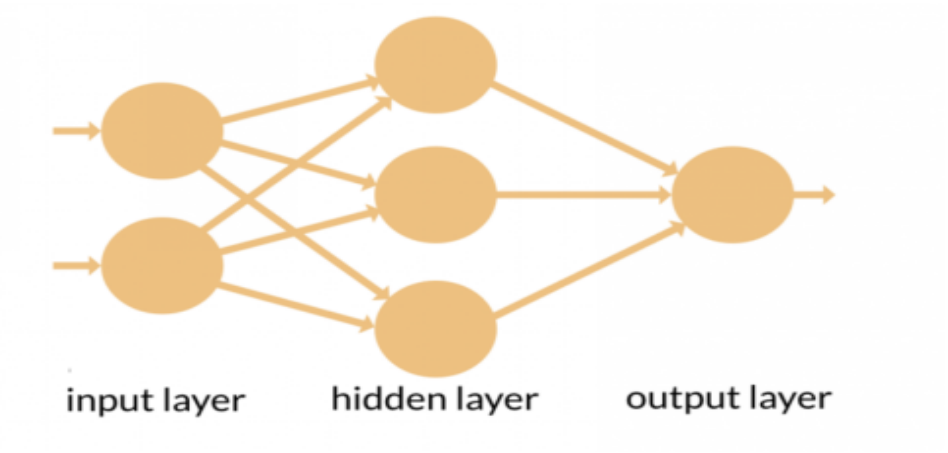


Fig .8: RNN and FEED-FORWARD NEURAL NETWORKS

The way they channel information gives RNNs and feed-forward neural networks their names.

The information in a feed-forward neural network flows in just one direction: from the input layer to the output layer, passing via the hidden layers. The

data travels across the network in a straight line, never passing through the same node twice.

Feed-forward neural networks have no recollection of the data they receive and are poor predictors of what will happen next. A feed-forward network has no sense of order in time since it just examines the current input. Except for its training, it has no recollection of anything that transpired in the past.

The information in an RNN cycles via a loop. When it makes a judgment, it takes into account the current input as well as what it has learned from prior inputs.

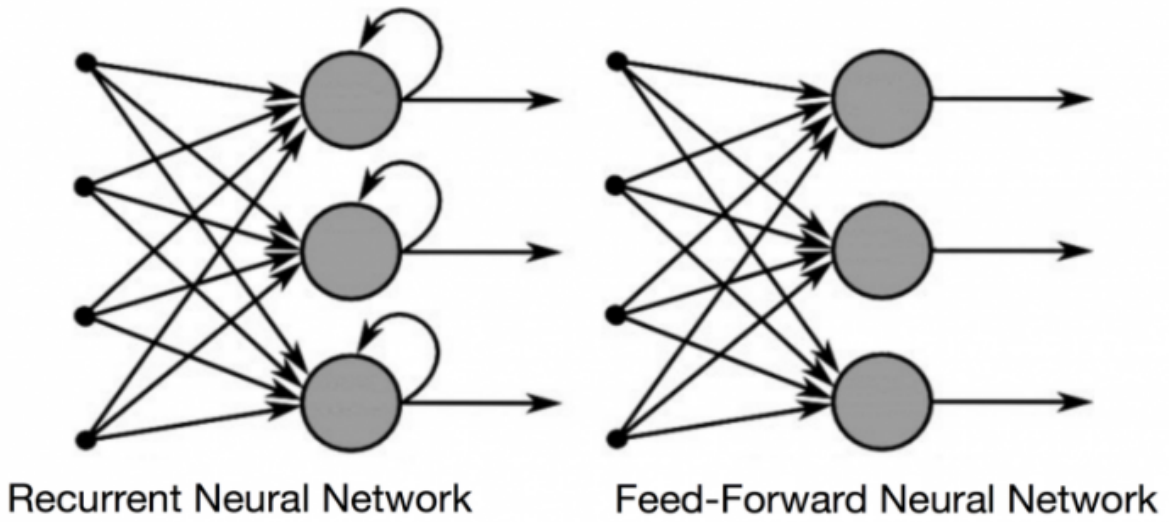


Fig. 9: The information flow between an RNN and a feed-forward neural network.

Simply, recurrent neural networks combine information from the past with information from the present. As a result, there are two inputs to an RNN: the present and the recent past. This is significant because the data sequence provides critical information about what will happen next, which is why an RNN can perform tasks that other algorithms cannot. Like all other deep learning algorithms, a feed-forward neural network applies a weight matrix to its inputs before producing the output. It's worth noting that RNNs apply weights to both the current

and prior inputs. In addition, a recurrent neural network will adjust the weights over time via gradient descent and backpropagation (BPTT).

What's the type of RNN..?

- ❖ One to One.
- ❖ One to Many.
- ❖ Many to One
- ❖ Many to Many.

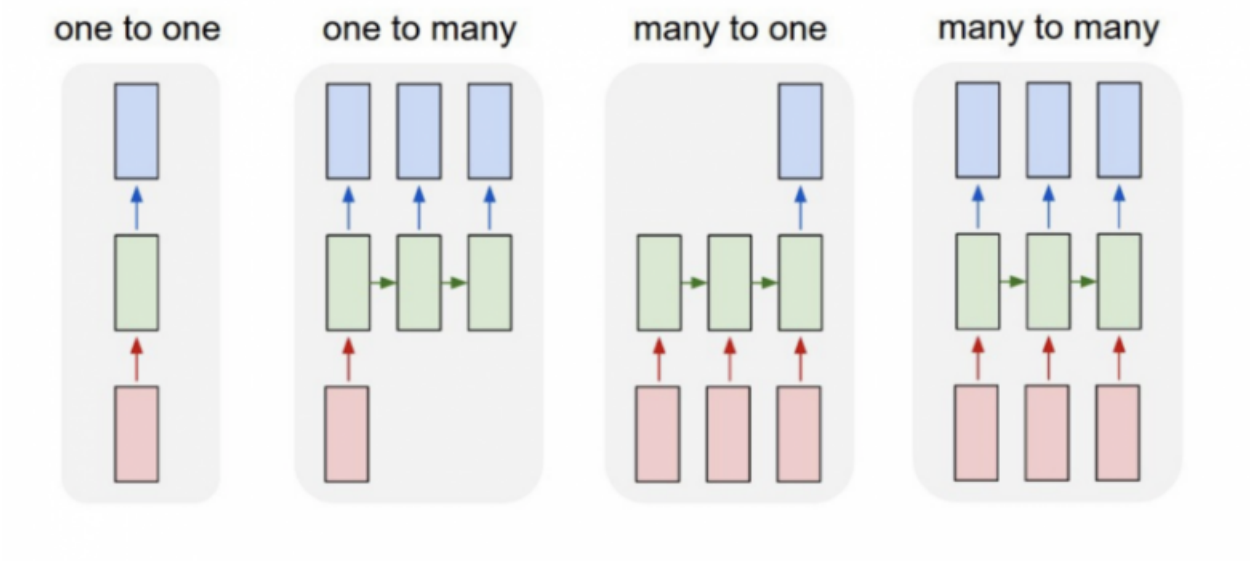


Fig .10: shows the RNN types.

4.2.1 Long short-term memory (LSTM)

Long short-term memory networks (LSTMs) are an extension of recurrent neural networks that expands the memory capacity. As a result, it is ideally adapted to learning from critical experiences that occur over lengthy periods of time, extending the memory. The layers of an RNN are built using LSTM as the building blocks. LSTMs assign "weights" to data, allowing RNNs to either let new information in, forget it, or give it enough relevance to affect the output. The layers of an RNN, sometimes referred to as an LSTM network, are built using the units of an LSTM. RNNs can recall inputs for a long time because of LSTMs. This is due to the fact that LSTMs store information in a memory similar to that of a computer.

The LSTM has the ability to read, write, and erase data from its memory. This memory may be thought of as a gated cell, with gated indicating that the cell selects whether or not to store or erase information (i.e., whether or not to open the gates) based on the value it gives to the data. Weights, which are also learnt by the algorithm, are used to allocate importance. This basically implies that it learns what information is important over time and what information is not. There are three gates in an LSTM: input, forget, and output. These gates decide whether fresh input should be allowed (input gate), whether it should be deleted because it isn't significant (forget gate), or whether it should have an influence on the output at the current timestep (impact gate).

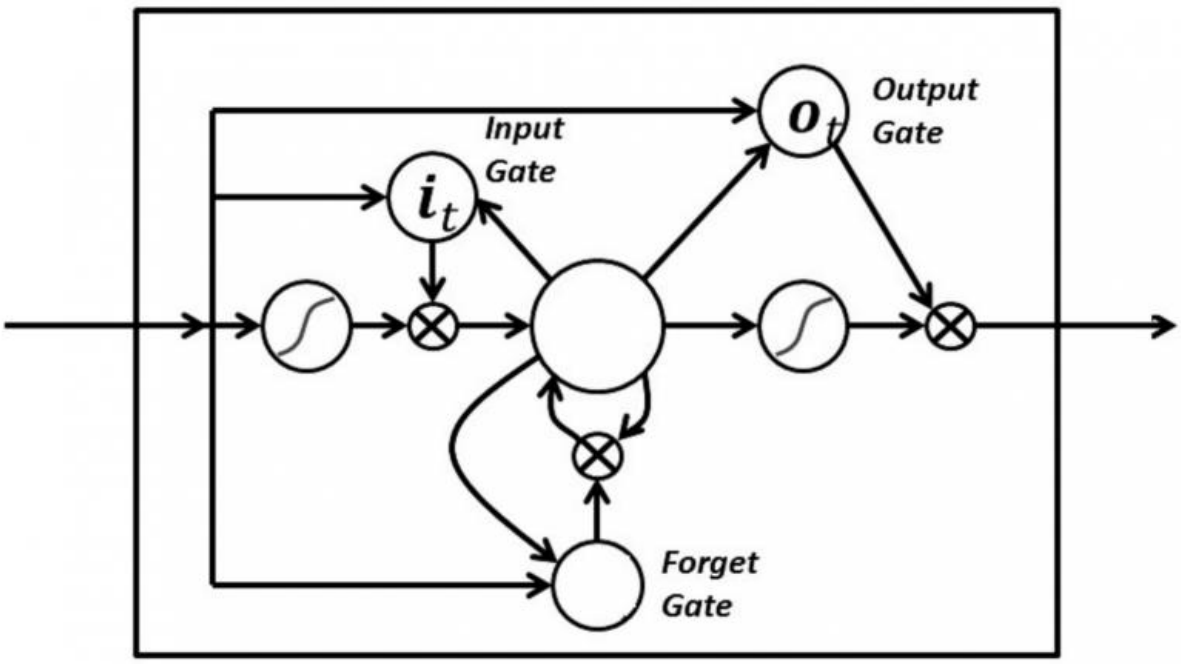


Fig .11: shows LSTM model

An LSTM's gates are analogue in the form of sigmoid, which means they range from zero to one. Because they are analogue, they may do backpropagation.

The problem of disappearing gradients is overcome by LSTM because it keeps the gradients steep enough, resulting in a quick training period and good accuracy.

5. system design

5.1. System architecture

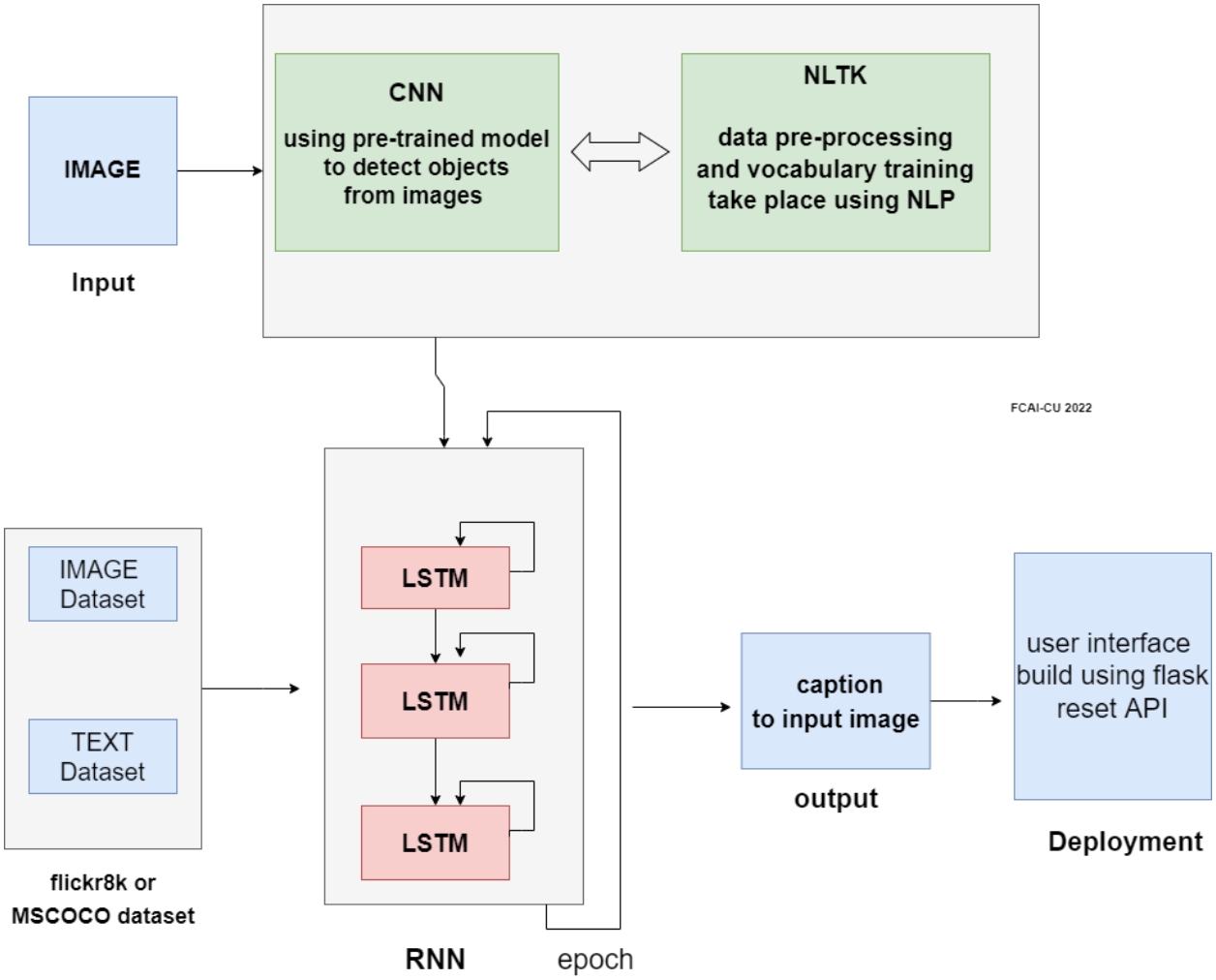


Fig .12: System architecture

Image captioning system would encode the image, using a pre-trained Convolutional Neural Network **VGG16** (ENCODER) that would produce a hidden state **h.**

Then, it would decode this hidden state by using a **LSTM** (DECODER) and generate recursively each word of the caption.

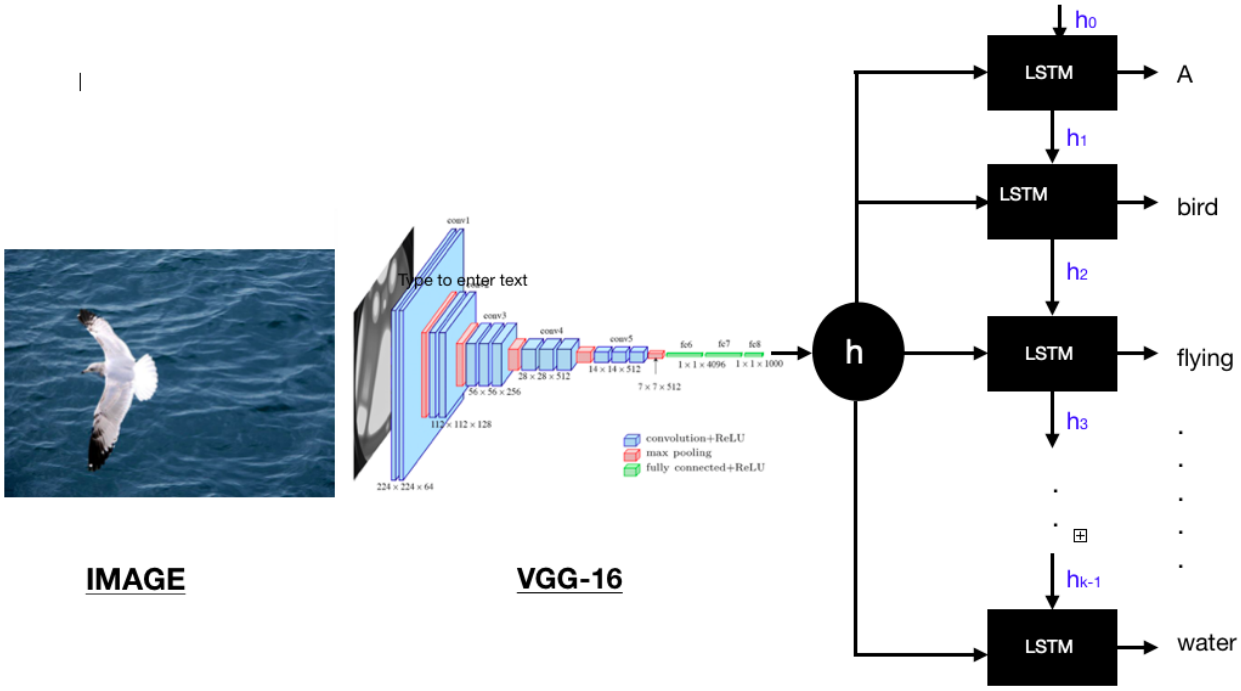


Fig.13: shows how the system works.

5.2. UML diagrams

5.2.1. Use Case Diagram

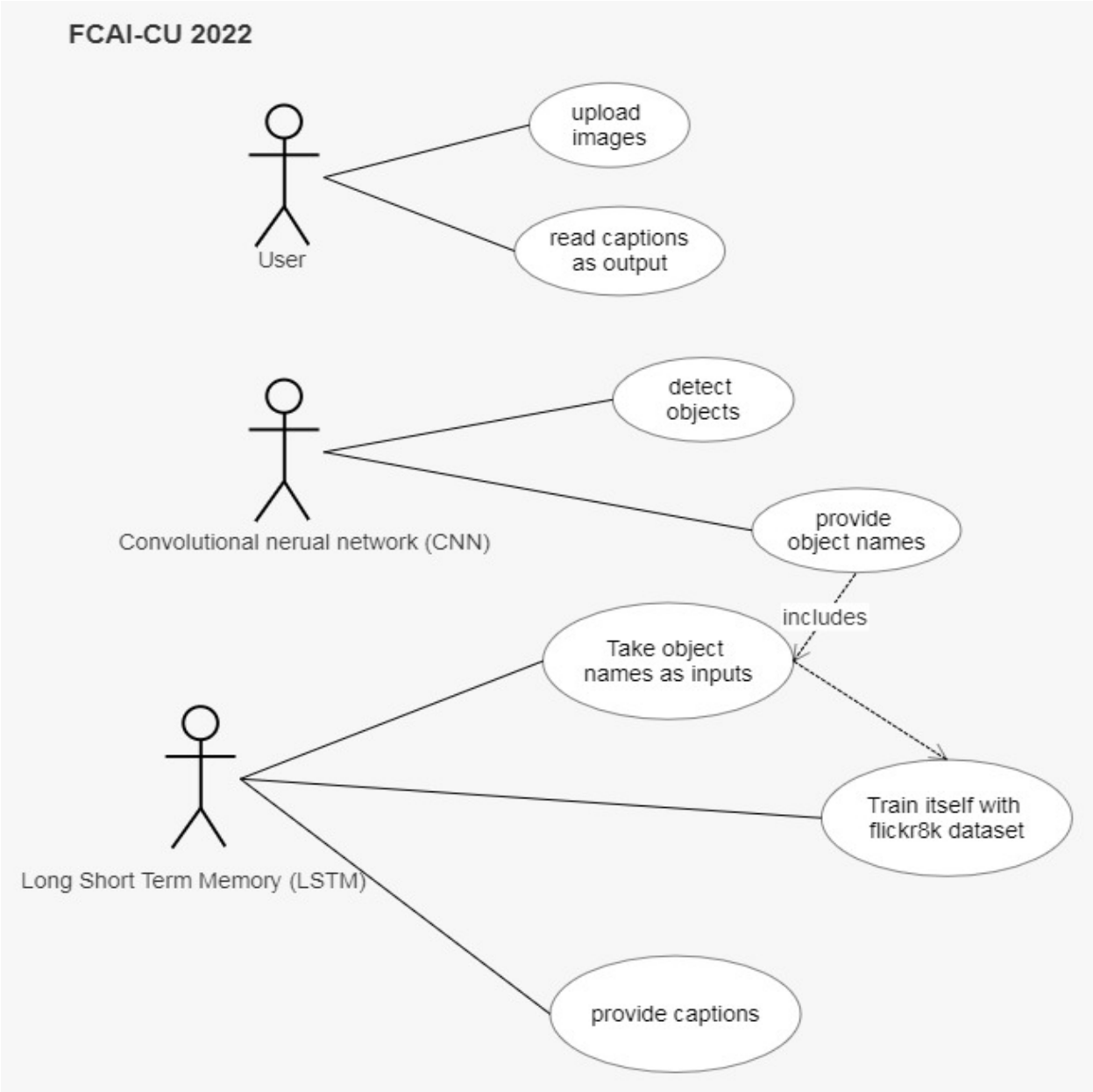


Fig .14: Use Case Diagram

5.2.2. Class Diagram

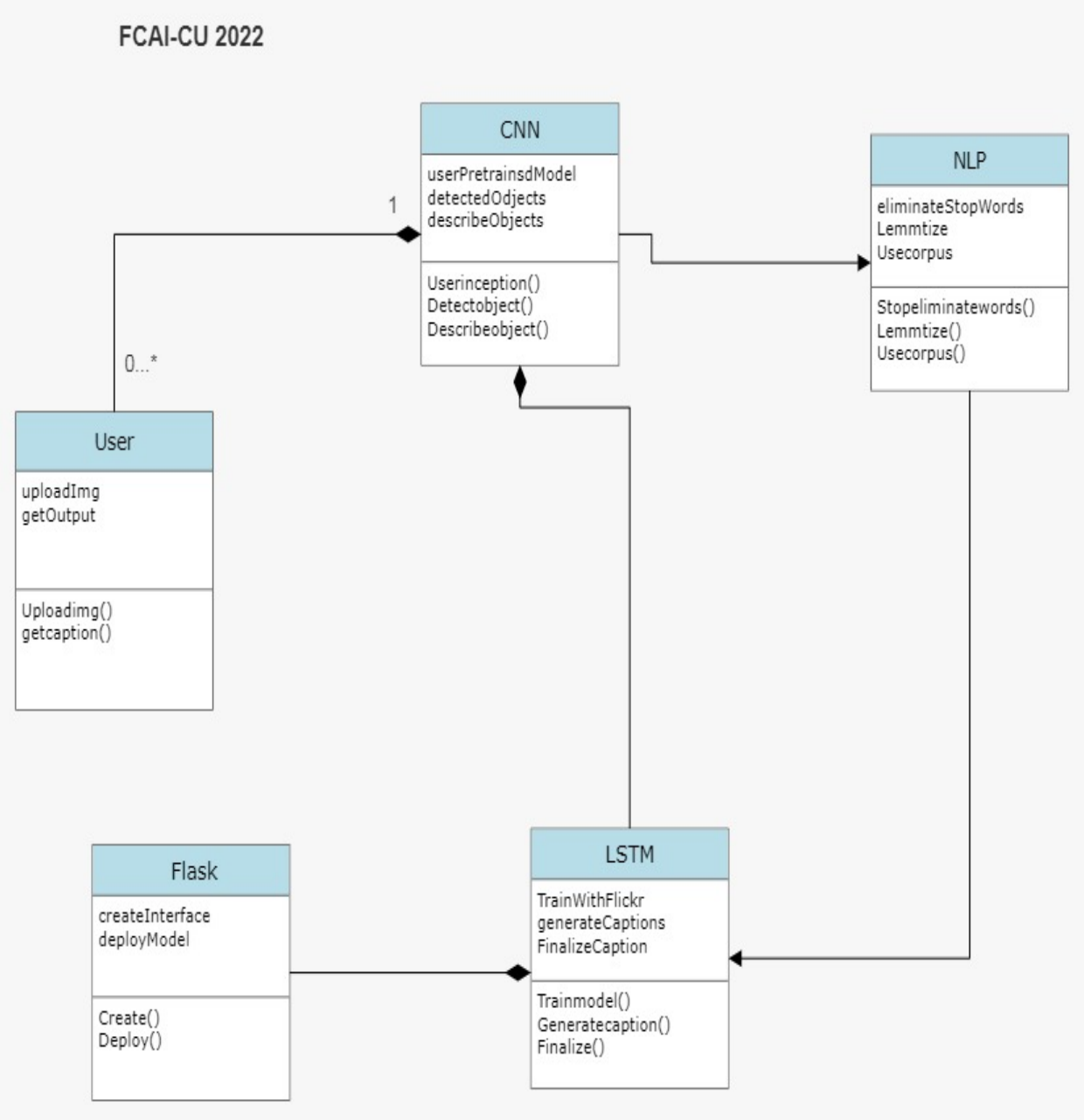


Fig .15: Class Diagram

5.2.3. Sequence Diagram

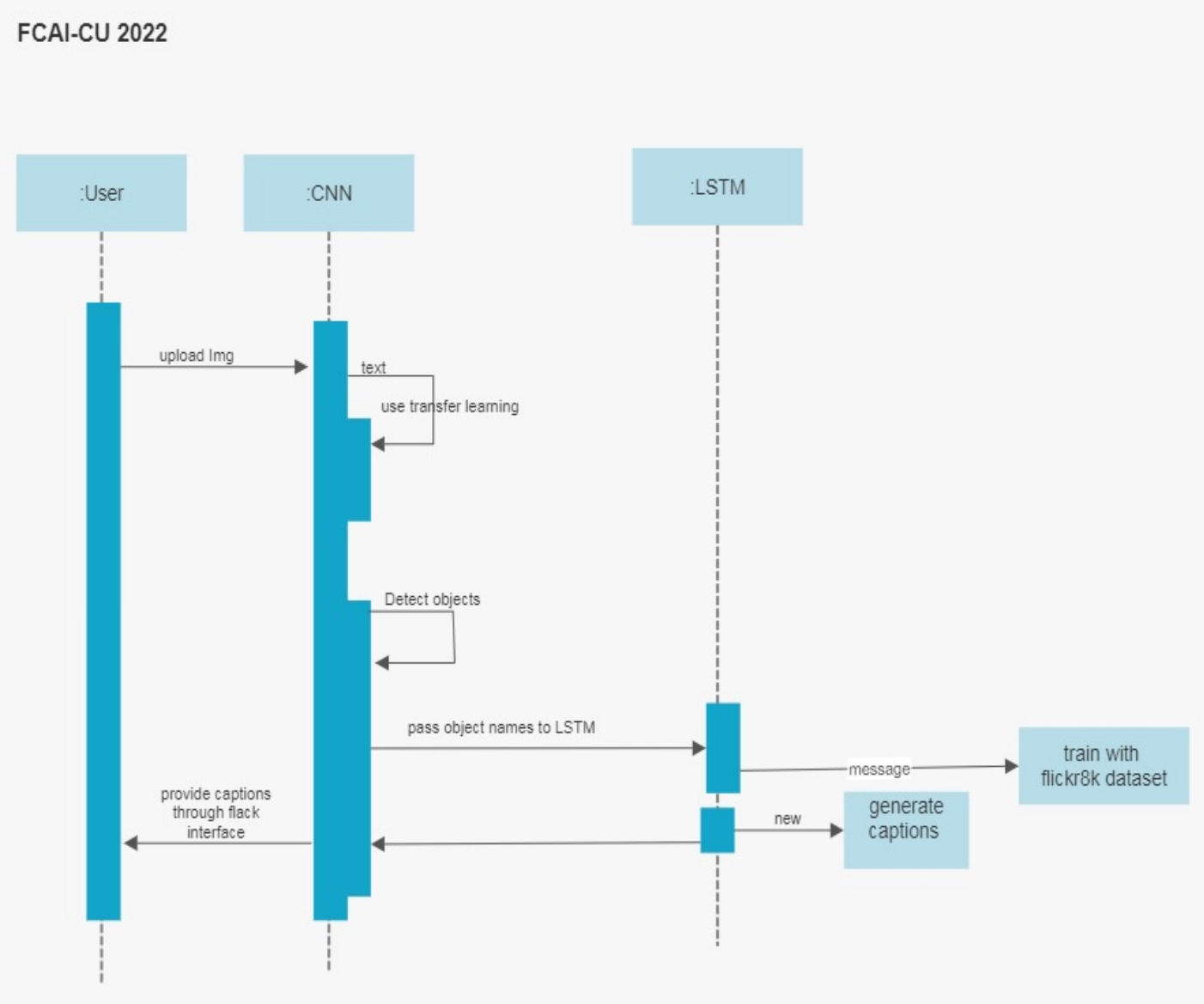


Fig .16: Sequence Diagram

5.2.4. DataFlow Diagram

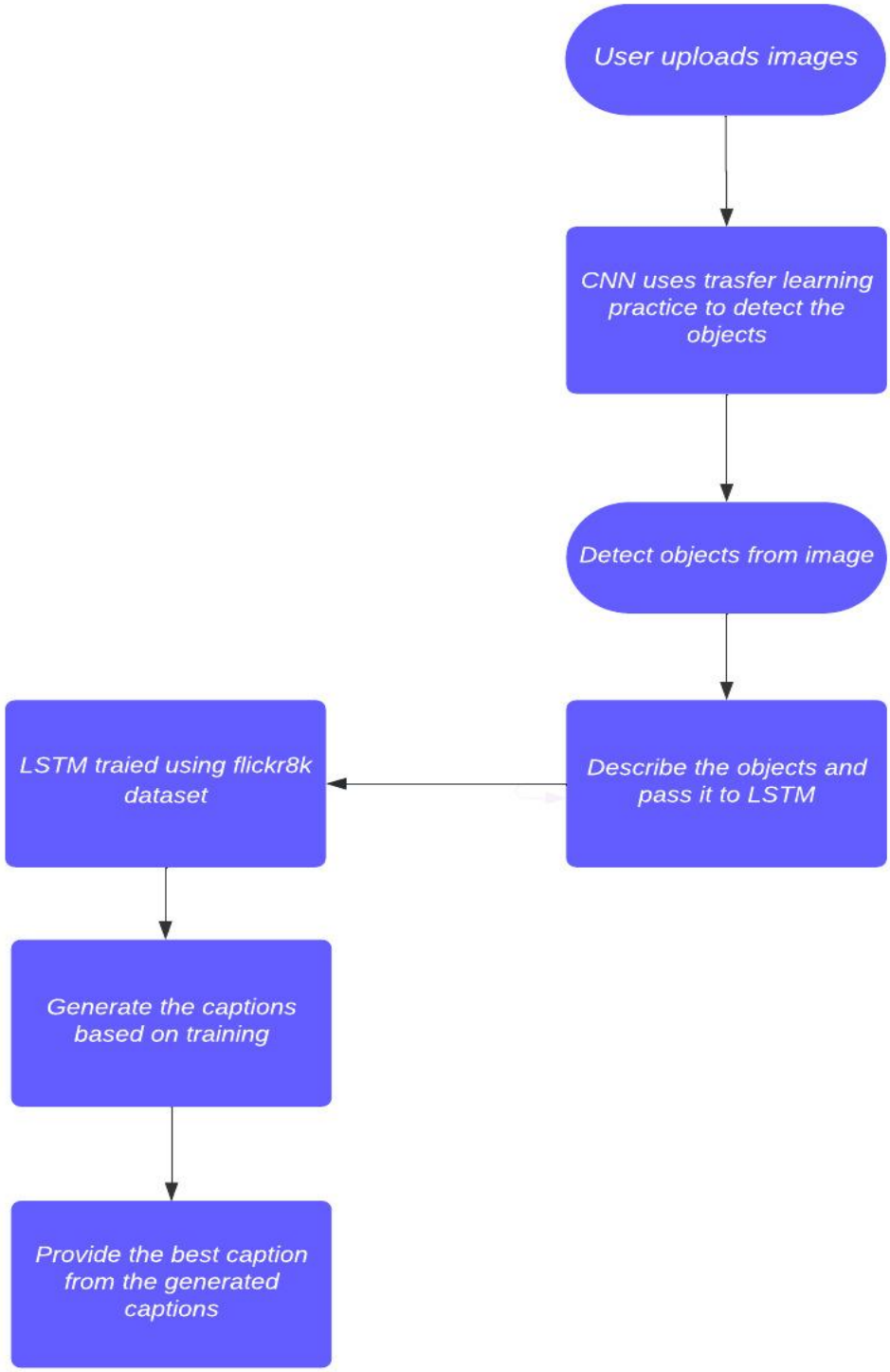


Fig .17: DataFlow Diagram

6. Implementation

6.1. Image features extractions

We can load the VGG model in Keras using the VGG class. We will remove the last layer from the loaded model, as this is the model used to predict a classification for a photo. We are not interested in classifying images, but we are interested in the internal representation of the photo right before a classification is made. These are the “features” that the model has extracted from the photo.

Keras also provides tools for reshaping the loaded photo into the preferred size for the model (e.g. 3 channel 224 x 224 pixel image).

Below is a 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 a 1-dimensional 4,096 element vector.

```
from os import listdir
from pickle import dump
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model

def extract_features(directory):
  model = VGG16()
  model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
  print(model.summary())

  features = dict()

  for name in listdir(directory):
    filename = directory + '/' + name
    image = load_img(filename, 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 = name.split('.')[0]
    features[image_id] = feature
    print('>%s' % name)
  return features

directory = '/content/Flicker8k_Dataset'
features = extract_features(directory)
print('Extracted Features: %d' % len(features))
dump(features, open('features.pkl', 'wb'))
```

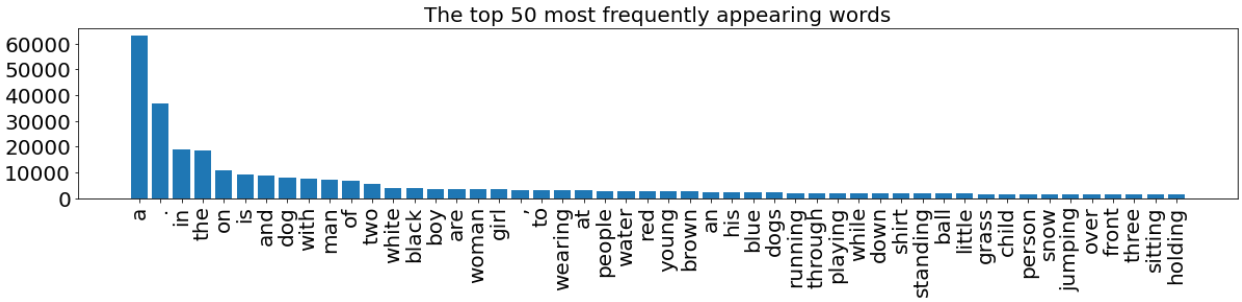
Fig .18: function shows image features extractions

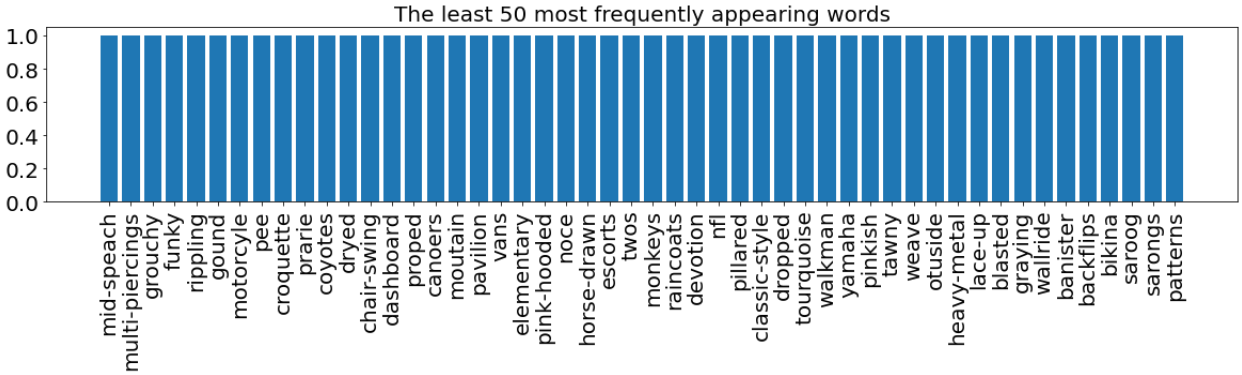
6.2. Text preparation

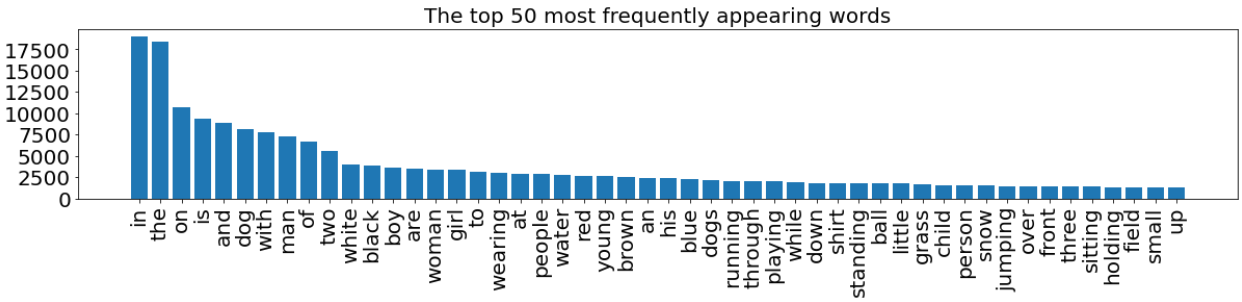
In order to clean the data, I will create the function to :

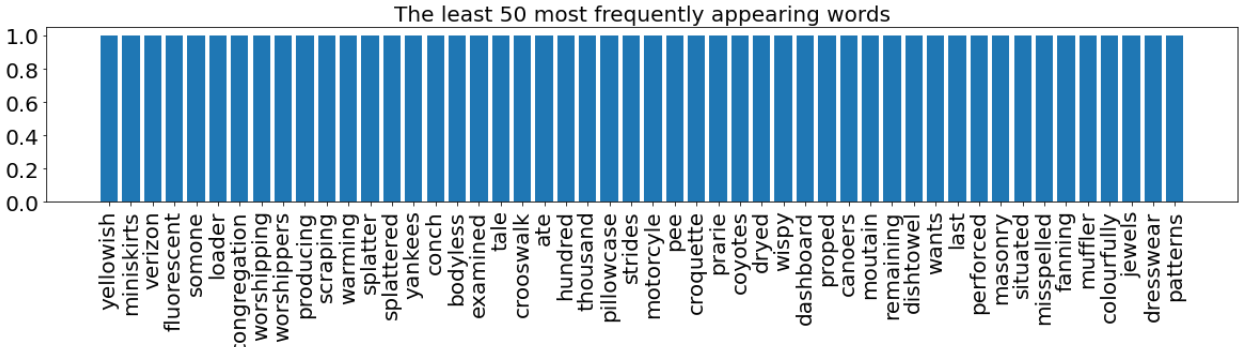
- remove punctuation
- remove numeric character
- remove single character

Vocabulary Size	Vocabulary Size after cleaning
8915	8357









Before Cleaning

> I ate 1000 apples and a banana. I have python v2.7. It's 2:30 pm. Could you buy me an iphone7?

After cleaning

> ate apples and banana have python Its pm Could you buy me

6.3. Caption Preprocessing

Since there are 5 captions for each image and we have preprocessed and encoded them in below format

“ **start seq** ” + caption + “ **end seq** ”

The model we will develop will generate a caption given a photo, and the caption will be generated one word at a time. The sequence of previously generated words will be provided as input. Therefore, we will need a ‘*first word*’ to kick-off the generation process and a ‘*last word*‘ to signal the end of the caption.

We will use the strings 'start seq‘ and ‘*endseq*‘ for this purpose.

start seq : Will act as our first word when a feature extracted image vector is fed to the decoder. It will kick-start the caption generation process.

end seq : This will tell the decoder when to stop. We will stop predicting words as soon as endseq appears or we have predicted all words from the train dictionary, whichever comes first.

filename	index	caption
1000268201_693b08cb0e.jpg	0	startseq child in pink dress is climbing up set of stairs in an entry way endseq
1000268201_693b08cb0e.jpg	1	startseq girl going into wooden building endseq
1000268201_693b08cb0e.jpg	2	startseq little girl climbing into wooden playhouse endseq
1000268201_693b08cb0e.jpg	3	startseq little girl climbing the stairs to her playhouse endseq
1000268201_693b08cb0e.jpg	4	startseq little girl in pink dress going into wooden cabin endseq
1001773457_577c3a7d70.jpg	0	startseq black dog and spotted dog are fighting endseq
1001773457_577c3a7d70.jpg	1	startseq black dog and dog playing with each other on the road endseq
1001773457_577c3a7d70.jpg	2	startseq black dog and white dog with brown spots are staring at each other in the street endseq
1001773457_577c3a7d70.jpg	3	startseq two dogs of different breeds looking at each other on the road endseq
1001773457_577c3a7d70.jpg	4	startseq two dogs on pavement moving toward each other endseq
1002674143_1b742ab4b8.jpg	0	startseq little girl covered in paint sits in front of painted rainbow with her hands in bowl endseq
1002674143_1b742ab4b8.jpg	1	startseq little girl is sitting in front of large painted rainbow endseq
1002674143_1b742ab4b8.jpg	2	startseq small girl in the grass plays with fingerpaints in front of white canvas with rainbow on it endseq
1002674143_1b742ab4b8.jpg	3	startseq there is girl with pigtails sitting in front of rainbow painting endseq
1002674143_1b742ab4b8.jpg	4	startseq young girl with pigtails painting outside in the grass endseq

Fig.22: shows filename and captions

6.4. Sequential Data preparation :

Caption: startseq girl going into wooden building endseq

```
Image_vector + ['startseq']
Image_vector + ['startseq', 'girl']
Image_vector + ['startseq', 'girl', 'going']
Image_vector + ['startseq', 'girl', 'going', 'into']
Image_vector + ['startseq', 'girl', 'going', 'into', 'wooden']
Image_vector + ['startseq', 'girl', 'going', 'into', 'wooden','building']
Image_vector + ['startseq', 'girl', 'going', 'into', 'wooden','building','endseq']
```

Fig.23: show the sequence form

Change character vector to integer vector using **Tokenizer**

will transform the data into input-output pairs of data for training the model. There are two input arrays to the model: one for photo features and one for the encoded text. There is one output for the model which is the encoded next word in the text sequence. The input text is encoded as integers, which will be fed to a word embedding layer.

	Xi		Yi
i	Image feature vector	Partial Caption	Target word
1	Image_1	startseq	the
2	Image_1	startseq the	black
3	Image_1	startseq the black	cat
4	Image_1	startseq the black cat	sat
5	Image_1	startseq the black cat sat	on
6	Image_1	startseq the black cat sat on	grass
7	Image_1	startseq the black cat sat on grass	endseq
8	Image_2	startseq	the
9	Image_2	startseq the	white
10	Image_2	startseq the white	cat
11	Image_2	startseq the white cat	is
12	Image_2	startseq the white cat is	walking
13	Image_2	startseq the white cat is walking	on
14	Image_2	startseq the white cat is walking on	road
15	Image_2	startseq the white cat is walking on road	endseq

data points corresponding to image 1 and its caption

data points corresponding to image 2 and its caption

Fig .24: Data Matrix for both the images and captions

	Xi		Yi
i	Image feature vector	Partial Caption	Target word
1	Image_1	[9, 0, 0 …., 0]	10
2	Image_1	[9, 10, 0, 0 …., 0]	1
3	Image_1	[9, 10, 1, 0, 0 …., 0]	2
4	Image_1	[9, 10, 1, 2, 0, 0 …., 0]	8
5	Image_1	[9, 10, 1, 2, 8, 0, 0 …., 0]	6
6	Image_1	[9, 10, 1, 2, 8, 6, 0, 0 …., 0]	4
7	Image_1	[9, 10, 1, 2, 8, 6, 4, 0, 0 …., 0]	3
8	Image_2	[9, 0, 0 …., 0]	10
9	Image_2	[9, 10, 0, 0 …., 0]	12
10	Image_2	[9, 10, 12, 0, 0 …., 0]	2
11	Image_2	[9, 10, 12, 2, 0, 0 …., 0]	5
12	Image_2	[9, 10, 12, 2, 5, 0, 0 …., 0]	11
13	Image_2	[9, 10, 12, 2, 5, 11, 0, 0 …., 0]	6
14	Image_2	[9, 10, 12, 2, 5, 11, 6, 0, 0 …., 0]	7
15	Image_2	[9, 10, 12, 2, 5, 11, 6, 7, 0, 0 …., 0]	3

Fig.25: Appending zeros to each sequence to make them all of same length 34

we need to make sure that each sequence is of equal length. Hence we need to append 0's (zero padding) at the end of each sequence. But how many zeros should we append in each sequence?

This is the reason we had calculated the maximum length of a caption, which is 34. So we will append those many zeros which will lead to every sequence having a length of 34.

6.5. Model

```
def define_model(vocab_size, max_length):
  inputs1 = Input(shape=(4096,))
  fe1 = Dropout(0.5)(inputs1)
  fe2 = Dense(256, activation='relu')(fe1)

  inputs2 = Input(shape=(max_length,))
  se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
  se2 = Dropout(0.5)(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=[inputs1, inputs2], outputs=outputs)
  model.compile(loss='categorical_crossentropy', optimizer='adam')
  print(model.summary())
  return model
```

Fig.26: define model

```
__________________________________________________________________________________________________
 Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
 input_8 (InputLayer)         [(None, 34)]              0           []

 input_7 (InputLayer)         [(None, 4096)]            0           []

 embedding_2 (Embedding)      (None, 34, 256)           1940224     ['input_8[0][0]']

 dropout_4 (Dropout)          (None, 4096)              0           ['input_7[0][0]']

 dropout_5 (Dropout)          (None, 34, 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, 7579)              1947803     ['dense_7[0][0]']

==================================================================================================
Total params: 5,527,963
Trainable params: 5,527,963
Non-trainable params: 0
```

Fig.27: show Summary of the parameters in the model

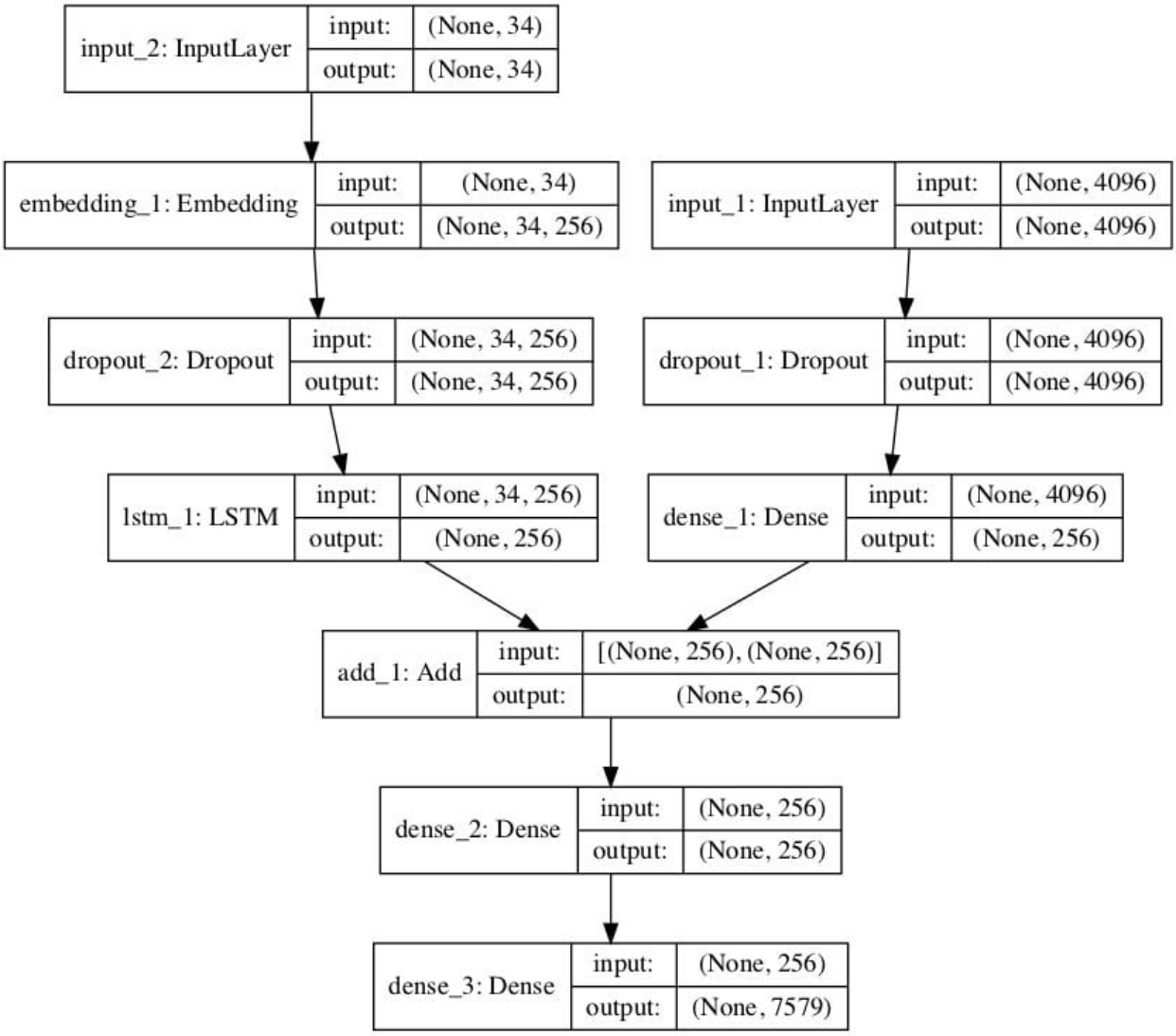


Fig.28: show flowchart the structure of the network

The model takes two input :

- 4096-dimensional image features from a pre-trained VGG16 model.
    - The image feature is passed to a fully connected layer with 256 hidden units.

- tokenized captions up to t word.
    - The tokenized caption up to tth time point is passed to embedding layer

    - Then passed to LSTM with 256 hidden states, and then a single output at the final time point is passed to the higher layer.

- The networks are merged by simply adding the two vectors of size 256.

- The vector of length 256 is passed to two dense layers and the final dense layer (softmax) returns probability that the t+1st word is kth word in the vocabulary (k=1,...,7579).

Given the caption prediction up to the tth word, the model can predict the t+1st word in the caption, and then the input caption can be augmented with the predicted word to contain the caption up to the t+1th word. The augmented caption up to the t+1st word can, in turn, be used as input to predict the t+2nd word in caption. The process is repeated until the "endseq" is predicted.

6.6. Validation loss and training loss

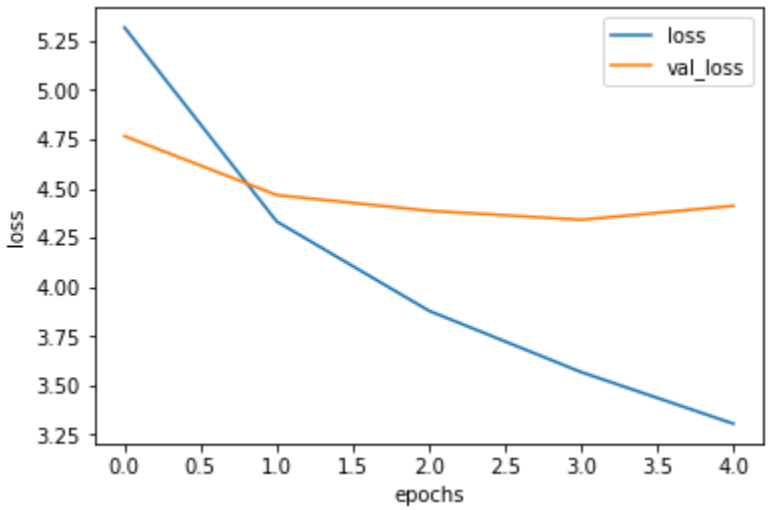


Fig.29: show validation loss and training loss over 5 epochs

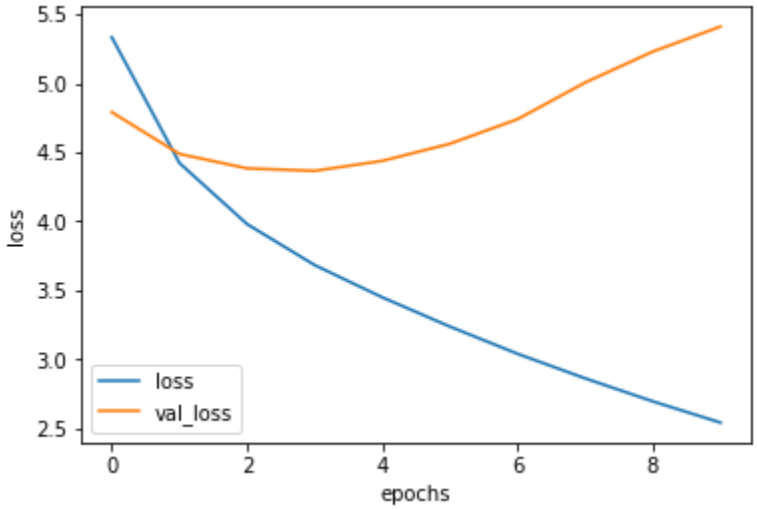


Fig.30: show validation loss and training loss over 10 epochs

The validation loss falls upto 5th epoch and then increases after 5th epoch in most cases even though the training loss decreases over time. This indicates that the model is overfitting

6.7. Predicting on the test dataset

After the model is trained, it is tested on test_dataset to see how it performs on caption generation for just 5 images.to see If the captions are acceptable or not.


startseq little blonde girl in pink top is playing with white hat endseq


startseq brown dog is running over the grass endseq


startseq brown dog fetching stick in its mouth endseq


startseq boy in black shirt is walking along city street endseq


startseq black dog is chasing an orange stick in its mouth endseq

Fig.31: shows Prediction of testing images

startseq brown dog is playing in the grass endseq

startseq boy in red shirt is playing in the water endseq

startseq man in red shirt is sitting on rock endseq

startseq man in red jacket is climbing rock endseq

startseq man in red shirt is on the the road endseq

Fig.32: shows Prediction of testing images

6.8. Evaluating model using BLEU scores

What is BLEU?

BLEU stands for Bilingual Evaluation Understudy, is a score for comparing a candidate translation of text to one or more reference translations.

A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0.

Although developed for **translation**, it can be used to evaluate text generated for a suite of natural language processing tasks.

Image captioning: Evaluate how well generated caption matches with a reference caption generated by human

The BLEU algorithm compares consecutive phrases of the automatic translation with the consecutive phrases it finds in the reference translation, and counts the number of matches, in a weighted fashion. These matches are position independent. A higher match degree indicates a higher degree of similarity with the reference translation, and higher score. Intelligibility and grammatical correctness aren't taken into account.

The BLEU score's strength is that it correlates well with human judgment. BLEU averages out individual sentence judgment errors over a test corpus, rather than attempting to devise the exact human judgment for every sentence.

The score is for comparing sentences, but a modified version that normalizes n-grams by their occurrence is also proposed for better scoring blocks of multiple sentences.

The Python Natural Language Toolkit library, or NLTK, provides an implementation of the BLEU score that you can use to evaluate your generated text against a reference.

BLEU score results in our model

BLEU-1	0.529691
BLEU-2	0.277107
BLEU-3	0.182869
BLEU-4	0.080318

true: black dog and spotted dog are fighting

pred: black and white dog is playing in the grass

BLEU: 0.7598356856515925

true: man drilling hole in the ice

pred: man in blue shirt is jumping on the air

BLEU: 0.7598356856515925

true: man and baby are in yellow kayak on water

pred: man in blue wetsuit is playing in the water

BLEU: 0.7598356856515925

true: man and woman pose for the camera while another man looks on

pred: man in black shirt and blue shirt is standing in the street

BLEU: 0.7071067811865476

true: the children are playing in the water

pred: girl in blue shirt is playing on the beach

BLEU: 0.7598356856515925

Fig.33: shows good captions with higher BLUE scores.

true: child in pink dress is climbing up set of stairs in an entry way

pred: woman is sitting next to fence in front of column

BLEU: 9.037968939610782e-232

true: black dog and spotted dog are fighting

pred: black and white dog is herding ball in the sand

BLEU: 1.3483065280626046e-231

true: little girl covered in paint sits in front of painted rainbow with her hands in bowl

pred: group of birds sitting on the grass

BLEU: 3.0963867043779474e-232

true: man lays on bench while his dog sits by him

pred: man in white shirt is throwing while woman in shorts is jumping while another dog in shorts is jumping while another dog in the park

BLEU: 1.072268715173723e-231

true: man in an orange hat starring at something

pred: man with winter hat holds their head in mirror

BLEU: 1.384292958842266e-231

Fig.34: shows bad caption with low BLUE scores.

7. Results

```
img_to_test = '/content/drive/MyDrive/Classroom/test2.jpg'
img = plt.imread(img_to_test)
plt.imshow(img)
description = ' '.join(description.split()[1:-1])
print()
print(description)
```

```
boy in blue bathing suit is jumping into swimming pool
```

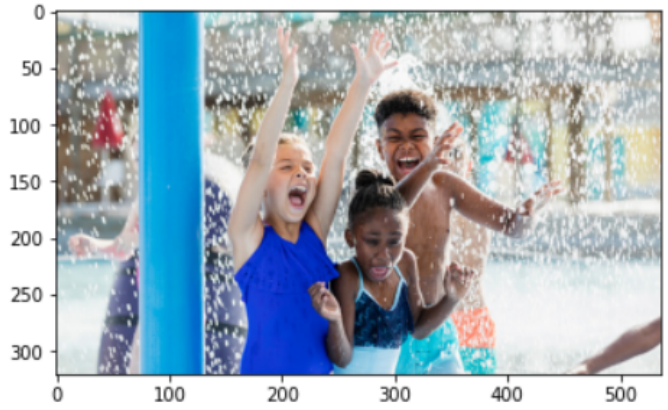


Fig.35: Example image 1 with generated caption.

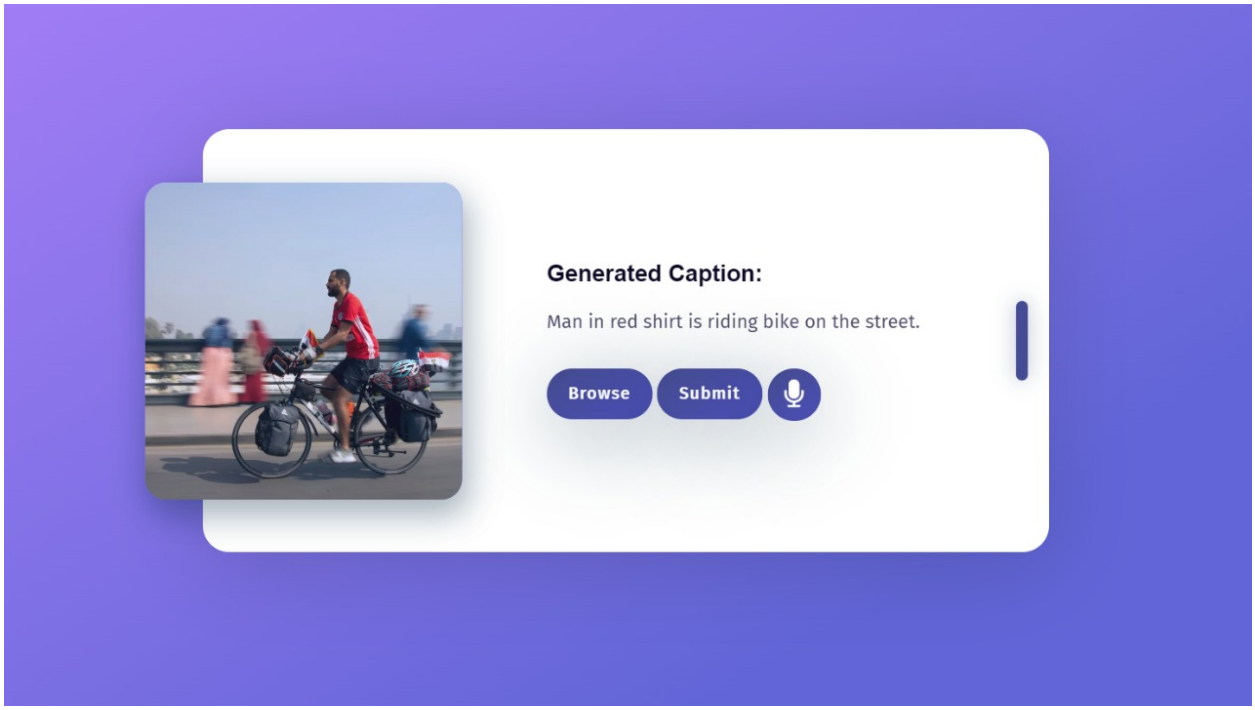


Fig.36: Example image 2 with generated caption.

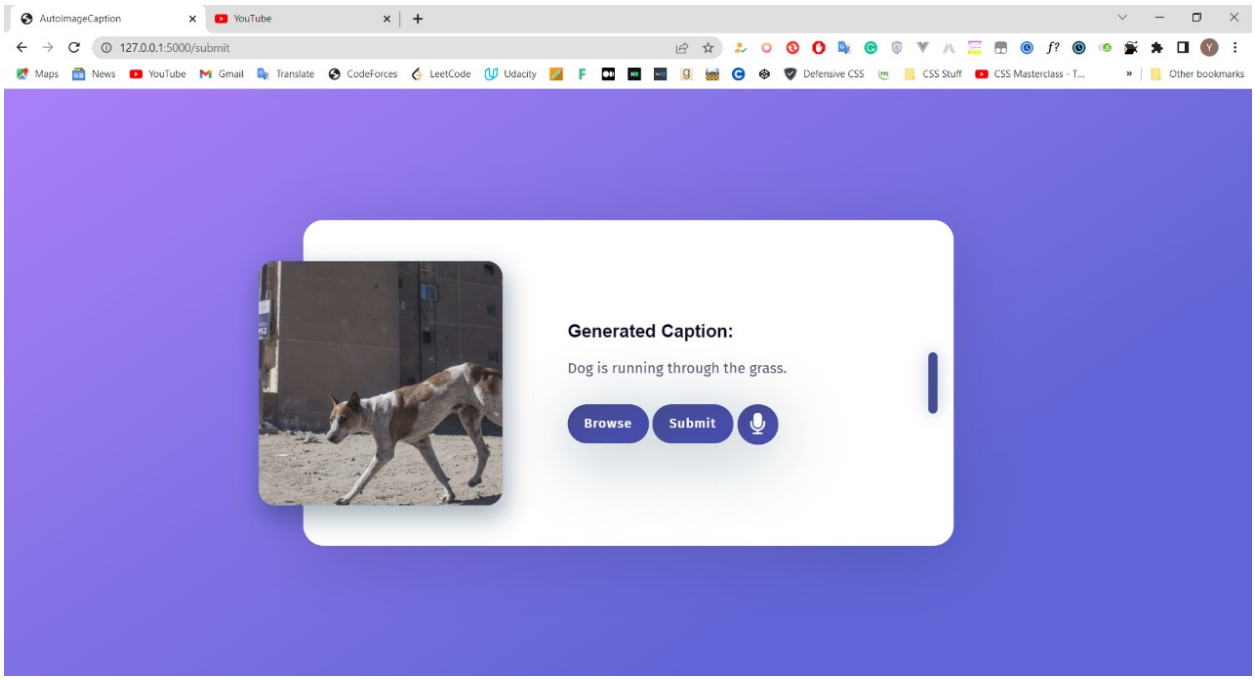


Fig.36: Example image 3 with generated caption.

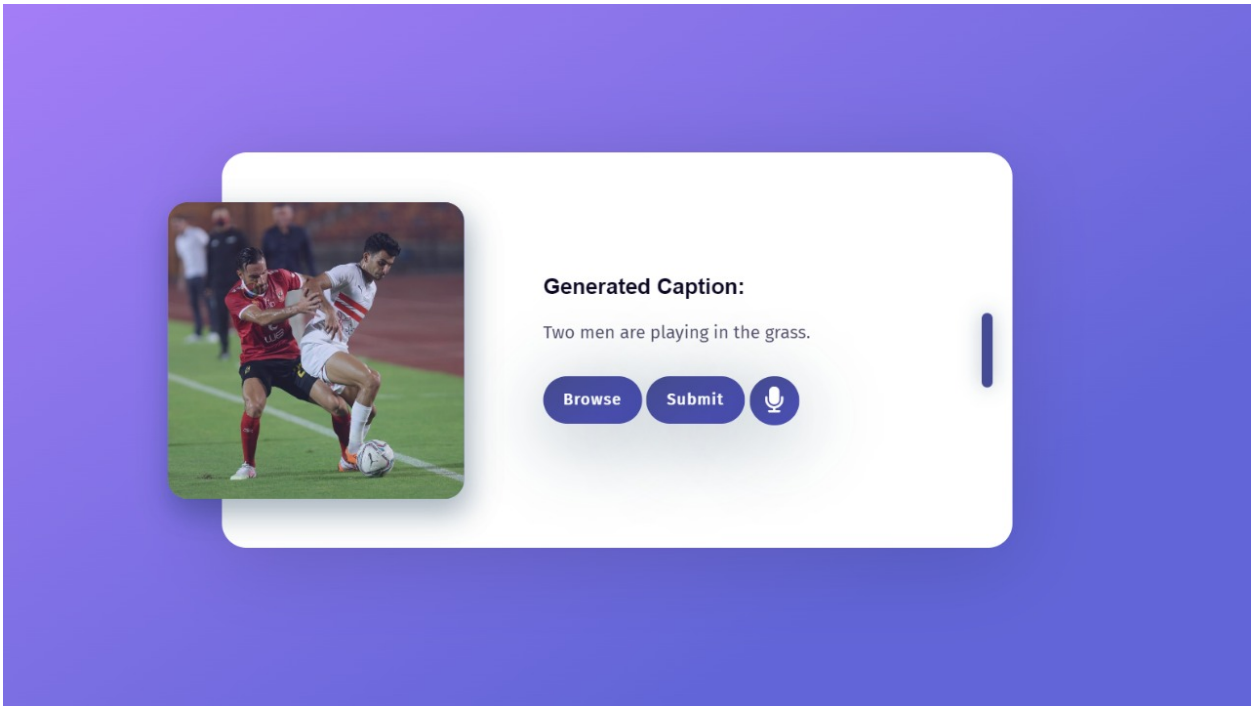


Fig.37: Example image 4 with generated caption.

8. Conclusion

Image captioning has many advantages in almost every complex area of Artificial Intelligence. The main use case of our model is to help the visually impaired to understand the environment and make it easy to act according to the environment. As this is a complex task to do, with the help of pre-trained models and powerful deep learning frameworks like Tensorflow and Keras, we made it possible. This is completely a Deep Learning project, which makes use of multiple Neural Networks like Convolutional Neural Network and Long Short Term Memory to detect objects and caption the images. To deploy our model as a web application, we have used Flask, which is a powerful Python's web framework.

Some key

aspects about our project note are that our model depends on the data, so, it cannot predict the words that are out of its vocabulary. A dataset consisting of 8000 images is used here. But for production-level models i.e. higher accuracy models, we need to train the model on larger than 150,000 images datasets so that better accuracy models can be developed.

9. Future Work

We will work on improving our model by using big dataset and using attention mechanisms. We are going to extend our work to the next higher level by enhancing our model to generate captions even for the live video frame. Our present model generates captions only for the image, which itself is a complex task and captioning live video frames is much more complex to create. This is completely GPU based and captioning live video frames cannot be possible with the general CPUs. Video captioning is a popular research area in which it is going to change the lifestyle of the people with the use cases being widely usable in almost every domain. It automates the major tasks like video surveillance and other security tasks.

9. References

1. https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip
2. https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_text.zip
3. https://cs231n.github.io/convolutional-networks/
4. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/
5. https://cs.nju.edu.cn/wujx/paper/CNN.pdf
6. https://www.hindawi.com/journals/cin/2020/3062706/
7. https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-1b-relu-layer
8. https://www.robots.ox.ac.uk/~vgg/research/very_deep/
9. https://keras.io/api/applications/#vgg16
10. https://iq.opengenus.org/maxpool-vs-avgpool/
11. https://ieeexplore.ieee.org/abstract/document/7505636
12. https://www.hindawi.com/journals/wcmc/2020/8909458/
13. http://karpathy.github.io/2015/05/21/rnn-effectiveness/
14. https://ieeexplore.ieee.org/abstract/document/9418234
15. http://cs231n.stanford.edu/reports/2016/pdfs/364_Report.pdf
16. https://fuqichen1998.github.io/pdfs/eecs442_report.pdf
17. https://towardsdatascience.com/deploying-a-deep-learning-model-using-flask-3ec166ef59fb
18. https://www.tutorialspoint.com/flask/flask_http_methods.htm
19. https://arxiv.org/pdf/1411.4555v2.pdf
20. https://www.researchgate.net/publication/337311437_Automatic_Image_Captioning_Using_Convolution_Neural_Networks_and_LSTM