IMAGE CAPTION GENERATION

INTRODUCTION:

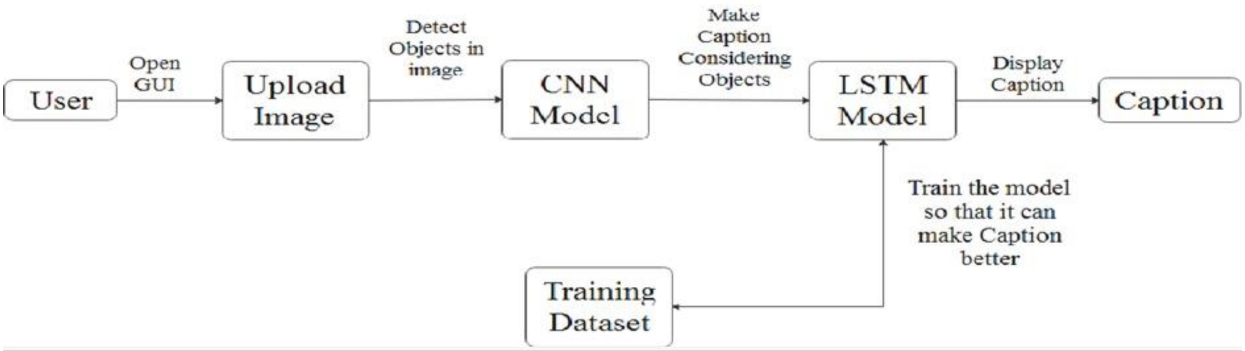
What is image captioning?

Creating image captions is a fundamental aspect of computer vision, central to the goal of understanding scenes. The challenge is twofold: caption models must not only identify objects in an image but also convey their relationships in natural language. This task is renowned for its complexity, requiring machine learning algorithms to emulate the human ability to distill vast visual information into descriptive language.

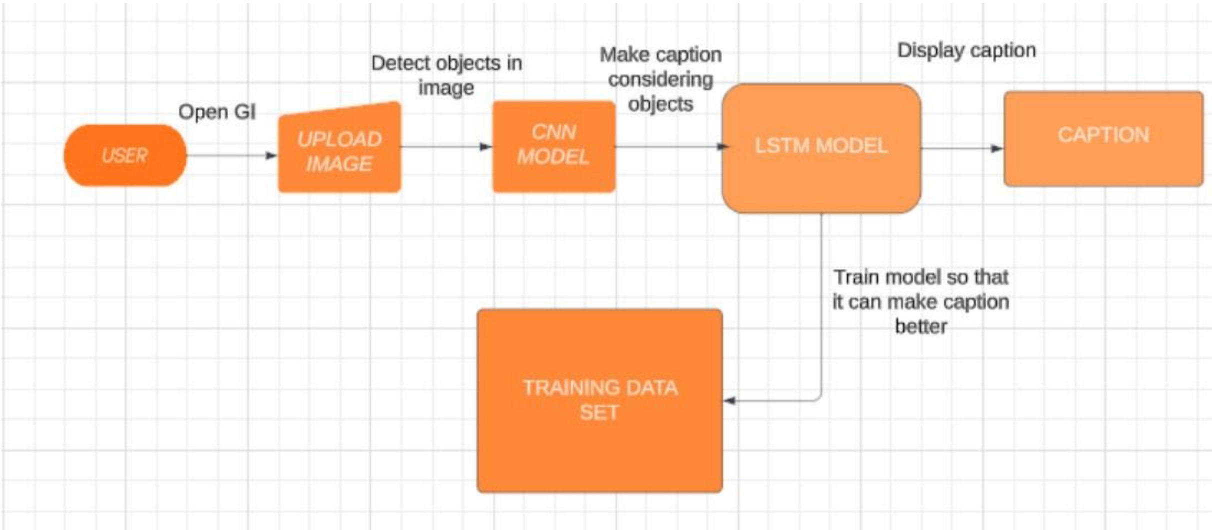
Our project aims to address this challenge through a user-friendly web interface. Users can upload images and receive detailed descriptions, while a classification system differentiates images based on these descriptions. The primary focus is aiding the visually impaired, offering them a tool to comprehend their surroundings and make informed decisions. However, existing approaches tend to generate generic descriptions, often overlooking crucial background details.

By navigating this complexity, our project seeks to empower individuals with visual impairments, providing them with nuanced and contextually rich image descriptions through an accessible online platform.

Technical Architecture:



Data Flow Diagram (DFD):



Pre-requisites:

To get this project rolling, you'll need to set up some software and tools. One essential is Anaconda Navigator, a free and open-source platform that supports Python and various languages for machine learning and deep learning tasks. It works on different devices, so choose the version that suits your gear. Installing Anaconda is a must for our project. We'll also make use of Jupyter notebook, a handy tool for problem-solving.

Next up, there's Visual Studio Code, a free code editor that plays well with Windows, macOS, and Linux. It's a go-to for many developers thanks to its compatibility with different operating systems. If you're not a fan of Jupyter notebook, Visual Studio Code is a solid alternative for our project. So, get these tools installed, and you'll be all set to dive into the project!

To build Machine learning models you must require and install the following packages.

1. Installation of Python packages
2. Installation of Numpy which it is used for the mathematical operations.it is an open source python library

Deep Learning Concepts:

CNN- CNN means Convolutional Neural networks are specialized deep neural networks that process the data that has input shape like a 2D matrix. CNN works well with images and is easily represented as a 2D matrix. Image identification is often easily done using CNN.

LSTM- LSTM means long-short term memory. LSTM is a type of RNN (Recurrent Neural Network) that is well suited for sequence finding problems. One can find the next words based on the previous text which is used.

FLASK- Flask is designed to be a lightweight framework, providing only the essentials for building web applications.

Project objectivies

Image caption generation is a field that involves computer vision, machine learning. The goal is to develop models that can understand and describe the visual world in a way that is useful to humans. These models have applications in various fields including image content. You can learn the some techniques and models of deep learning and know about automotive generation.

Project Flow:

1. User should request with interface and after that user should select the image in their device.
2. The chosen image analyzed by the model which is integrated console prompt.
3. LSTM is used to process the captions in form of text, and prediction is showcased on the console prompt.

To accomplish this, we have to complete all the activities and tasks listed below

1. Data Collection.
2. Data Pre-processing.
3. Model Building
4. Testing the model

Project Structure:

1. Create a Project folder which contains files as shown below

.ipynb_checkpoints	06-11-2023 14:47	File folder	
Flicker8k_Datadet	06-11-2023 15:11	File folder	
Flickr_8k_text	06-11-2023 15:01	File folder	
models	06-11-2023 14:49	File folder	
descriptions	06-11-2023 14:51	Text Document	3,072 KB
features.p	06-11-2023 14:52	P File	65,477 KB
model	06-11-2023 14:53	PNG File	48 KB
testing_caption_generator	06-11-2023 14:53	Python Source File	3 KB
tokenizer.p	06-11-2023 14:53	P File	286 KB
training_caption_generator	06-11-2023 14:53	Jupyter Source File	33 KB

2. The Dataset folder contains the training and testing images for training our model.
3. We need the model which is saved as model.h5 and the captions as tokenizer.pkl the templates folder contains index.html and prediction.html pages.

Step-1:

Data Collection

We can download the data from kaggle website, there are nearly 8000 images associated with the 5 captions for each image. The given dataset has 40000 high quality human readable text captions. After downloading datasets you should create a folder and insert datasets into folder as Flicker8k_Dataset and Flickr_8k_text.

Step-2:

Data Pre-processing

Clean the text captions and mapping each together. Then it's time to build our Vgg16 model which contains an input layer CNN model and the LSTM model.

Task-1

First we have to import all the necessary packages

```
import string
import numpy as np
from PIL import Image
import os
from pickle import dump, load
import numpy as np

from keras.applications.xception import Xception, preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers.merge import add
from keras.models import Model, load_model
from keras.layers import Input, Dense, LSTM, Embedding, Dropout

# small library for seeing the progress of loops.
from tqdm import tqdm_notebook as tqdm
tqdm().pandas()
```

Task-2

Getting and performing data cleaning

```
File  Edit  Format  Run  Options  Window  Help
{
'3461437556_cc5e97f3ac.jpg': ['dogs on grass',
                              'three dogs are running on the grass',
                              'three dogs one white and two brown are running together
                              'three dogs run along grassy yard',
                              'three dogs run together in the grass'
                              ],

'3461583471_2b8b6b4d73.jpg': ['buy is grinding rail on snowboard',
                              'person is jumping ramp on snowboard',
                              'snowboarder goes down ramp',
                              'snowboarder going over ramp',
                              'snowboarder performs jump on the clean white snow'
                              ],
'997722733_0cb5439472.jpg' : ['man in pink shirt climbs rock face',
                              'man is rock climbing high in the air',
                              'person in red shirt climbing up rock face covered in as
                              'rock climber in red shirt',
                              'rock climber practices on rock climbing wall'
                              ]

}
```

This function takes all descriptions and performs data cleaning. This is an important step when we work with textual data, according to our goal, we decide what type of cleaning we want to perform on the text. In our case, we will be removing punctuations it will converting all text to lowercase and removing words that contain numbers. So, a caption like “A man riding on a three-wheeled wheelchair” will be transformed into “man riding on three wheeled wheelchair”.

```python
# get all imgs with their captions
def all_img_captions(filename):
    file = load_doc(filename)
    captions = file.split('\n')
    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)
    return descriptions
```

```python
##Data cleaning- lower casing, removing puntuations and words containing numbers
def cleaning_text(captions):
    table = str.maketrans('','',string.punctuation)
    for img,caps in captions.items():
        for i,img_caption in enumerate(caps):

            img_caption.replace("-"," ")
            desc = img_caption.split()

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

Task-3

Extracting the features

This technique is also called transfer learning, we don’t have to do everything on our own, and we use the pre-trained model that have been already trained on large datasets and extract the features from these models and use them for our tasks. We are using the Xception model which has been trained on imagenet dataset that had 1000 different classes to classify. We can directly import this model from the keras.applications . Make sure you are connected to the internet as the weights get automatically downloaded. Since the Xception model was originally built for imagenet, we will do little changes for integrating with our model. One thing to notice is that the Xception model takes 299*299*3 image size as input. We will remove the last classification layer and get the 2048 feature vector.

```
def extract_features(directory):
        model = Xception( include_top=False, pooling='avg' )
        features = {}
        for img in tqdm(os.listdir(directory)):
            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"))
```

Task-4

Loading dataset for Training the model

In our Flickr_8k_test folder, we have Flickr_8k.trainImages.txt file that contains a list of 6000 image names that we will use for training. This function will create a dictionary that contains captions for each photo from the list of photos.

```
#load the data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos


def load_clean_descriptions(filename, photos):
    #loading clean_descriptions
    file = load_doc(filename)
    descriptions = {}
    for line in file.split("\n"):

        words = line.split()
        if len(words)<1 :
            continue

        image, image_caption = words[0], words[1:]

        if image in photos:
            if image not in descriptions:
                descriptions[image] = []
            desc = '<start> ' + " ".join(image_caption) + ' <end>'
            descriptions[image].append(desc)

    return descriptions


def load_features(photos):
    #loading all features
    all_features = load(open("features.p","rb"))
    #selecting only needed features
    features = {k:all_features[k] for k in photos}
    return features
```

```python
filename = dataset_text + "/" + "Flickr_8k.trainImages.txt"

#train = loading_data(filename)
train_imgs = load_photos(filename)
train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)
train_features = load_features(train_imgs)
```

```python
#converting dictionary to clean list of descriptions
def dict_to_list(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

#creating tokenizer class
#this will vectorise text corpus
#each integer will represent token in dictionary

from keras.preprocessing.text import Tokenizer

def create_tokenizer(descriptions):
    desc_list = dict_to_list(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(desc_list)
    return tokenizer
```

Task-5

Tokenizing the vocabulary

Computers don’t understand English words, for computers, we will have to represent them with numbers. So, we will map each word of the vocabulary with a unique index value. Keras library provides us with the tokenizer function that we will use to create tokens from our vocabulary and save them to a “tokenizer.p” pickle file.

```python
# give each word a index, and store that into tokenizer.p pickle file
tokenizer = create_tokenizer(train_descriptions)
dump(tokenizer, open('tokenizer.p', 'wb'))
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

```
7577
```

```python
#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)
max_length
```

```
32
```

Task-6

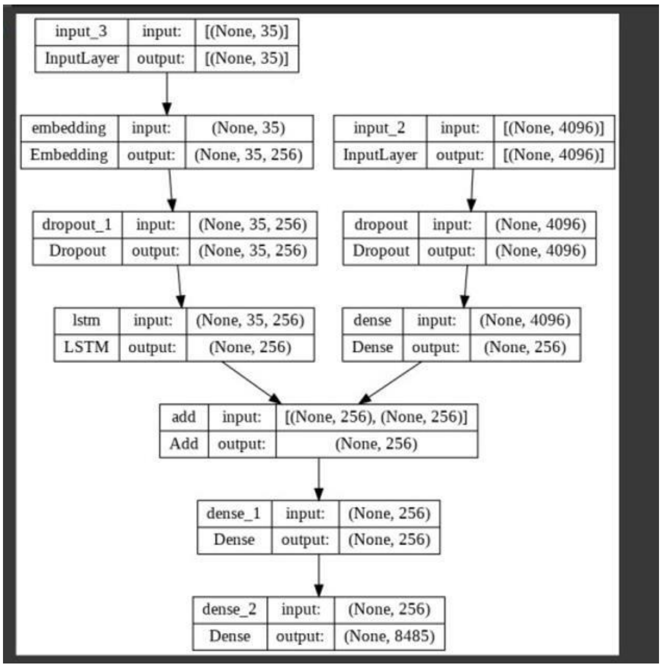
Data Generation

In Flicker8k-datset we are having lot of images approximately 8000 so we are creating a data generation where the images can store into memory it so we can run our model by images and description captions.

```
#data generator, used by model.fit_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)
            yield [[input_image, input_sequence], output_word]

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:
        # encode the sequence
        seq = tokenizer.texts_to_sequences([desc])[0]
        # 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]
            # pad input sequence
            in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
            # encode output sequence
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            # store
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)
```

By merging the output from the above two layers, we will process by the dense layer to make the final prediction. The final layer will contain the number of nodes equal to our vocabulary size.



```python
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])
    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-3:

Model Building

Now we can train our image data set

Train the model

To train the model, we will be using the 6000 training images by generating the input and output sequences in batches and fitting them to the model using model.fit_generator() method. We also save the model to our models folder. This will take some time depending on your system capability.

```python
# train our model
print('Dataset: ', len(train_imgs))
print('Descriptions: train=', len(train_descriptions))
print('Photos: train=', len(train_features))
print('Vocabulary Size:', vocab_size)
print('Description Length: ', max_length)

model = define_model(vocab_size, max_length)
epochs = 10
steps = len(train_descriptions)
# making a directory models to save our models
os.mkdir("models")
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
    model.fit_generator(generator, epochs=1, steps_per_epoch= steps, verbose=1)
    model.save("models/model_" + str(i) + ".h5")
```

```
Dataset:  6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length:  32
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_2 (InputLayer)            (None, 32)           0
________________________________________________________________________________________________
input_1 (InputLayer)            (None, 2048)         0
________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 32, 256)      1939712     input_2[0][0]
________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 2048)         0           input_1[0][0]
________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 32, 256)      0           embedding_1[0][0]
________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          524544      dropout_1[0][0]
________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 256)          525312      dropout_2[0][0]
________________________________________________________________________________________________
add_1 (Add)                     (None, 256)          0           dense_1[0][0]
                                                                 lstm_1[0][0]
________________________________________________________________________________________________
...
Trainable params: 5,002,649
Non-trainable params: 0
```

Step-4:

Testing the model

While we are testing the model we should check whether it is perfectly fit into to the model or not. The model has been trained, now, we will make a separate file testing_caption_generator.py which will load the model and can generate the predictions.

```
D: > Project- image caption generator > testing_caption_generator.py > extract_features
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.applications.xception import Xception
from keras.models import load_model
from pickle import load
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import argparse


ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required=True, help="Image Path")
args = vars(ap.parse_args())
img_path = args['image']

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def extract_features(filename, model):
        try:
            image = Image.open(filename)

        except:
            print("ERROR: Couldn't open image! Make sure the image path and extension is correct")
        image = image.resize((299,299))
        image = np.array(image)
        # for images that has 4 channels, we convert them into 3 channels
        if image.shape[2] == 4:
            image = image[..., :3]
        image = np.expand_dims(image, axis=0)
        image = image/127.5
        image = image - 1.0
        feature = model.predict(image)
        return feature

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def word_for_id(integer, tokenizer):
 for word, index in tokenizer.word_index.items():
```

training_caption_generator.ipynb ● testing_caption_generator.py 7 ×

D: > Project- image caption generator > testing_caption_generator.py > extract_features

```
        if index == integer:
            return word
    return None


Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
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


#path = 'Flicker8k_Dataset/111537222_07e56d5a30.jpg'
max_length = 32
tokenizer = load(open("tokenizer.p","rb"))
model = load_model('models/model_9.h5')
xception_model = Xception(include_top=False, pooling="avg")

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)
```

Inserting an image as input and getting caption prediction in the output .checking the results

```
start man is standing on rock overlooking the mountains end
```