Project Manual

Introduction:

The fusion of computer vision and natural language processing has ushered in a new era of AI-driven comprehension, enabling machines to interpret and describe visual content. Image captioning, at the intersection of these domains, represents a complex yet intriguing challenge. It involves understanding the intricate details within an image and expressing them eloquently in natural language.

This project embarks on the task of automating image description generation, a feat achieved through a sophisticated blend of deep learning architectures. By leveraging Convolutional Neural Networks (CNNs) for extracting image features, Long Short-Term Memory (LSTM) networks for language modeling, and the formidable ResNet-50 architecture for robust image understanding, this endeavor seeks to bridge the gap between visual perception and linguistic expression.

Human-level understanding of visual content and the ability to describe it in textual form has long been an innate skill. However, teaching machines to emulate this cognitive process necessitates the development of advanced algorithms capable of interpreting visual features and converting them into coherent sentences.

The challenge lies in comprehensively representing the visual nuances, contextual details, and semantic richness inherent in images. Furthermore, it requires seamlessly integrating these visual cues with language generation, ensuring the produced captions not only describe the image content accurately but also convey the essence captured by the visual data.

The motivation behind this project is rooted in the practical applications of image captioning across various domains. From aiding visually impaired individuals by providing auditory descriptions of images to enhancing content accessibility on social media platforms, the potential applications are diverse and impactful.

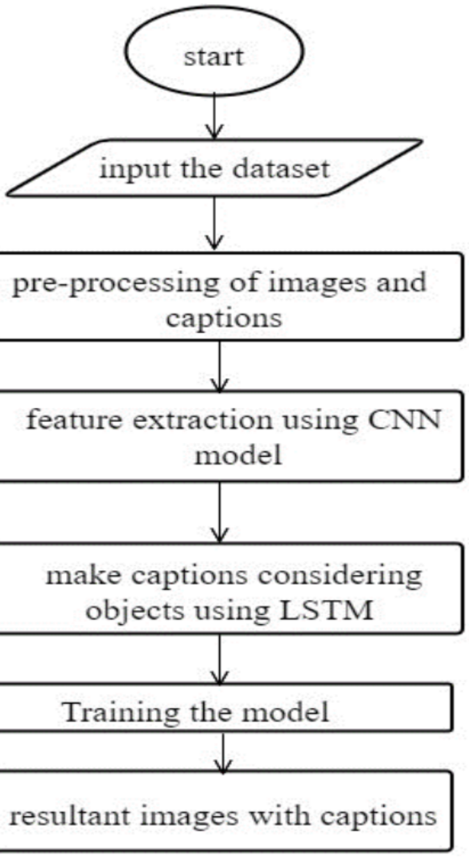
Moreover, the project aligns with the broader scope of advancing multimodal AI systems that can comprehend and communicate across different modalities—visual and textual. By pushing the boundaries of image understanding and language modeling, the aim is to contribute to the development of AI systems capable of nuanced perception and communication.

The primary objective of this project is to design an end-to-end deep learning architecture that seamlessly integrates visual understanding with language generation. By utilizing ResNet-50 for extracting image features, LSTM networks for language modeling, and CNNs for overall architecture enhancement, the goal is to create a model that generates descriptive and contextually relevant captions for a wide array of images.

The significance of this endeavor lies in its potential to advance the state-of-the-art in multimodal AI systems. It aims not only to produce accurate image descriptions but also to lay the groundwork for more comprehensive models that can comprehend, interpret, and communicate across diverse data modalities.

To complement the prowess of the image captioning model, this project aims to present a user-friendly interface for seamless interaction. Leveraging the Flask web framework, a user interface will be developed to enable users to upload images and receive real-time generated captions. Flask's simplicity and flexibility make it an ideal choice for building a responsive and intuitive web application. Users will have the convenience of uploading images through the interface, which will then be processed by the underlying deep learning model. The generated captions will be displayed, fostering a more interactive and engaging experience, ultimately showcasing the capabilities of the image captioning system.

Technical Architecture:


start
input the dataset
pre-processing of images and captions
feature extraction using CNN model
make captions considering objects using LSTM
Training the model
resultant images with captions

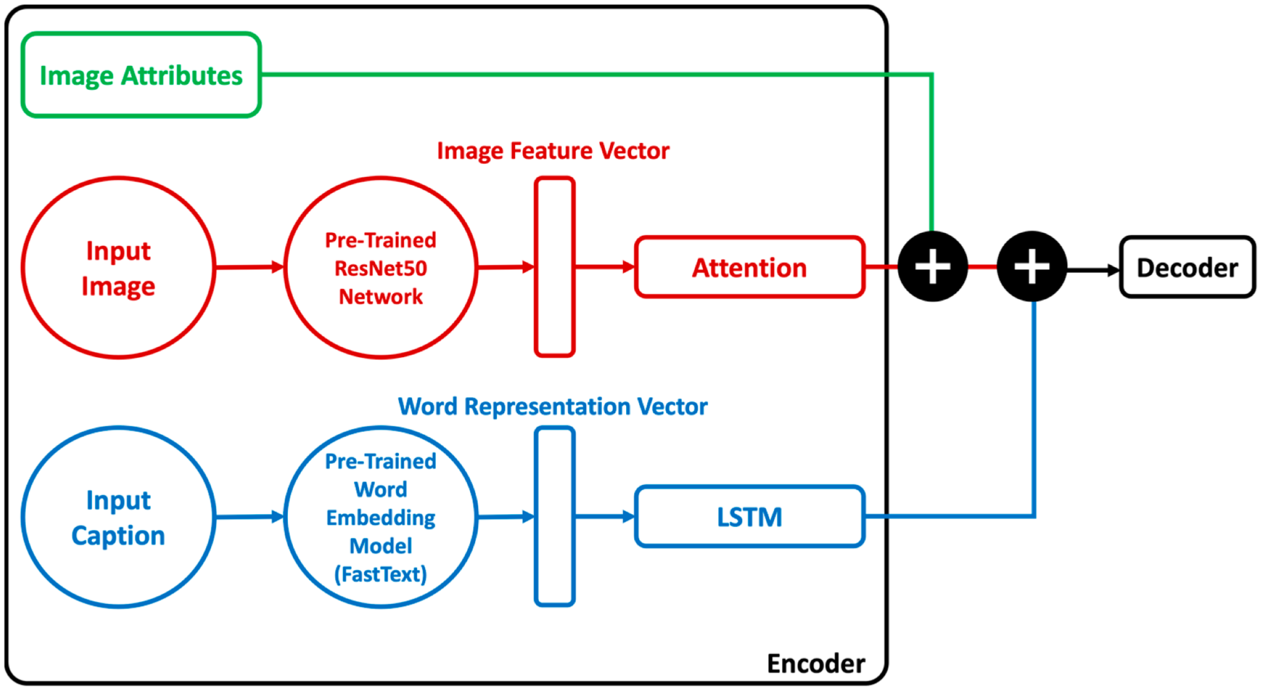


Image Attributes
Image Feature Vector
Input Image
Pre-Trained ResNet50 Network
Attention
Decoder
Word Representation Vector
Input Caption
Pre-Trained Word Embedding Model (FastText)
LSTM
Encoder

Pre-requisites:

1. Kaggle Account:
Sign up for a Kaggle account if you haven't already. This will allow you to access datasets, kernels (Jupyter notebooks), and competitions.

2. Anaconda Distribution:
- Download and install Anaconda, which includes Python and essential libraries for data science.
- Anaconda Navigator provides a user-friendly interface to manage environments and launch Spider (a development environment similar to Jupyter notebooks).

3. Python Libraries:
Install necessary Python libraries using Anaconda or pip within your Anaconda environment. These could include:
- TensorFlow or PyTorch for deep learning.
- OpenCV or PIL for image processing.
- NumPy, Pandas, Matplotlib for data manipulation and visualization.
- Flask for web development (if creating a web interface).

4. Kaggle API:
Install the Kaggle API to access datasets directly from Kaggle. You can do this by following the instructions provided on the Kaggle website for API setup.

5. Dataset and Models:
- Identify or download the image captioning dataset you plan to use from Kaggle or other reputable sources.
- Pre-trained models like ResNet-50 may need to be downloaded or accessed through libraries like TensorFlow Hub or PyTorch Hub.

6. Jupyter or Spider Environment:
Launch Spider from the Anaconda Navigator to start developing your project in a familiar Jupyter-like interface.

7. GPU (optional):
If you plan to train deep learning models, having access to a GPU through Kaggle kernels or local hardware can significantly speed up model training.

8. Understanding of Deep Learning and Image Captioning:
Familiarize yourself with deep learning concepts, CNNs, LSTMs, and their application in image captioning tasks through online courses, tutorials, or documentation.

9. Project Structure and Planning:
Outline your project structure, including directories for data, code, and model checkpoints. Plan your workflow for data preprocessing, model development, and evaluation.

Milestone 1: Collection of Data

To assemble the dataset for the image captioning project, a collection of images capturing various events will be compiled. Each image will be associated with a set of five captions that describe the content within the image. The dataset will be structured by organizing the images into specific subdirectories named according to their respective events as indicated in the project structure. This dataset encompasses over 7,000 diverse images capturing different events, accompanied by a collection of more than 40,000 high-quality, human-readable text captions.

The process involves creating folders to contain the images and compiling a text file that catalogs the five captions linked to each image. This structured dataset will serve as the foundation for training and evaluating the image captioning model, facilitating a comprehensive understanding of event-centric visual content matched with descriptive textual annotations.

This revised version provides a clearer description of the data collection process, emphasizing the creation of a dataset comprising images and associated captions, along with the organizational structure required for the project.

Download the Dataset-https://www.kaggle.com/datasets/shadabhussain/flickr8k

Milestone 2: Image Pre-processing and Model Building

Activity 1: Importing the Model Building Libraries

```
# Load libraries
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import numpy as np
import os
from keras.applications.resnet50 import ResNet50
from keras.applications import DenseNet201
from keras.optimizers import Adam
from keras.layers import Dense, Flatten,Input, Convolution2D, Dropout, LSTM, TimeDistributed,
Embedding, Bidirectional, Activation, RepeatVector,Concatenate
from keras.models import Sequential, Model
from keras.utils import np_utils
import random
from keras.preprocessing import image, sequence
import matplotlib.pyplot as plt
```

Activity 2: Exploratory Data

```
# Load data
images_dir = os.listdir("../input/flickr_data/Flickr_Data/")

images_path = '../input/flickr_data/Flickr_Data/Images/'
captions_path = '../input/flickr_data/Flickr_Data/Flickr_TextData/Flickr8k.token.txt'
train_path = '../input/flickr_data/Flickr_Data/Flickr_TextData/Flickr_8k.trainImages.txt'
val_path = '../input/flickr_data/Flickr_Data/Flickr_TextData/Flickr_8k.devImages.txt'
test_path = '../input/flickr_data/Flickr_Data/Flickr_TextData/Flickr_8k.testImages.txt'

captions = open(captions_path, 'r').read().split("\n")
x_train = open(train_path, 'r').read().split("\n")
x_val = open(val_path, 'r').read().split("\n")
x_test = open(test_path, 'r').read().split("\n")
```

Activity 3: Loading captions and displaying the image

```
# Loading captions as values and images as key in dictionary
tokens = {}

for ix in range(len(captions)-1):
    temp = captions[ix].split("#")
    if temp[0] in tokens:
        tokens[temp[0]].append(temp[1][2:])
    else:
        tokens[temp[0]] = [temp[1][2:]]
```

```
# displaying an image and captions given to it
temp = captions[10].split("#")
from IPython.display import Image, display
z = Image(filename=images_path+temp[0])
display(z)

for ix in range(len(tokens[temp[0]])):
    print(tokens[temp[0]][ix])
```

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

Activity 4: Initializing the model

```
model = ResNet50(include_top=False,weights='imagenet',input_shape=(224,224,3),pooling='avg')
model.summary()
```

Activity 5: Configure the Learning Process

```
# Helper function to process images
def preprocessing(img_path):
    im = image.load_img(img_path, target_size=(224,224,3))
    im = image.img_to_array(im)
    im = np.expand_dims(im, axis=0)
    return im
```

```
train_data = {}
ctr=0
for ix in x_train:
    if ix == "":
        continue
    if ctr >= 3000:
        break
    ctr+=1
    if ctr%1000==0:
        print(ctr)
    path = images_path + ix
    img = preprocessing(path)
    pred = model.predict(img).reshape(2048)
    train_data[ix] = pred
```

```
# Creating a list of all unique words
unique = []
for i in words:
    unique.extend(i)
unique = list(set(unique))

print(len(unique))

vocab_size = len(unique)
```

```
8253
```

```
# Vectorization
word_2_indices = {val:index for index, val in enumerate(unique)}
indices_2_word = {index:val for index, val in enumerate(unique)}
```

```
word_2_indices['UNK'] = 0
word_2_indices['raining'] = 8253
```

```
indices_2_word[0] = 'UNK'
indices_2_word[8253] = 'raining'
```

1. Image Preprocessing:

- The function `preprocessing(img_path)` takes an image path as input, loads the image using `image.load_img` from a library (likely Keras), resizes it to (224, 224, 3), converts it to an array, and expands its dimensions.
- It prepares images for processing by a neural network model.

Feature Extraction:

- There's a loop (`for ix in x_train`) iterating over a training set (`x_train`). For each image path in the training set:
- If the path is empty, it continues to the next iteration.
- If the counter `ctr` reaches 3000, the loop breaks.

- It preprocesses the image using the `preprocessing` function defined earlier.
- Utilizes a pre-trained `model` to extract features from the image. The extracted features are reshaped into a vector of size 2048.
- These image features are stored in a dictionary `train_data` with th image path as the key and the extracted features as the value.

3.Saving Extracted Features:
After processing, the extracted features are saved into a file named `"train_encoded_images.p"` using Python's `pickle` module.

4. Loading Image-Caption Dataset:
Loads a dataset from a file named `"flickr_8k_train_dataset.txt"` using Pandas, containing image IDs and their corresponding captions.

5. Data Preparation:
- Creates a list `sentences` containing all the captions from the dataset.
- Splits each caption into words and stores them in `words` (a list of lists).
- Creates a list of unique words from all captions and assigns each unique word an index.

6. Vocabulary Creation and Vectorization:
- Determines the size of the vocabulary (`vocab_size`) based on the unique words.
- Creates two dictionaries: `word_2_indices` and `indices_2_word`, mapping words to indices and vice versa.
- Assigns indices for special tokens like 'UNK' (unknown words) and 'raining'.

Activity 6: Train The model

Padding:

```
padded_sequences, subsequent_words = [], []

for ix in range(ds.shape[0]):
    partial_seqs = []
    next_words = []
    text = ds[ix, 1].split()
    text = [word_2_indices[i] for i in text]
    for i in range(1, len(text)):
        partial_seqs.append(text[:i])
        next_words.append(text[i])
    padded_partial_seqs = sequence.pad_sequences(partial_seqs, max_len, padding='post')

    next_words_1hot = np.zeros([len(next_words), vocab_size], dtype=np.bool)

    #Vectorization
    for i,next_word in enumerate(next_words):
        next_words_1hot[i, next_word] = 1

    padded_sequences.append(padded_partial_seqs)
    subsequent_words.append(next_words_1hot)

padded_sequences = np.asarray(padded_sequences)
subsequent_words = np.asarray(subsequent_words)

print(padded_sequences.shape)
print(subsequent_words.shape)
```

Saving Captions and array conversions:

```
for ix in range(num_of_images):#img_to_padded_seqs.shape[0]):
    captions = np.concatenate([captions, padded_sequences[ix]])
    next_words = np.concatenate([next_words, subsequent_words[ix]])

np.save("captions.npy", captions)
np.save("next_words.npy", next_words)

print(captions.shape)
print(next_words.shape)
```

```
(25493, 40)
(25493, 8254)
```

```
with open('../input/train_encoded_images.p', 'rb') as f:
    encoded_images = pickle.load(f, encoding="bytes")
```

```
imgs = []

for ix in range(ds.shape[0]):
    if ds[ix, 0].encode() in encoded_images.keys():
#         print(ix, encoded_images[ds[ix, 0].encode()])
        imgs.append(list(encoded_images[ds[ix, 0].encode()]))

imgs = np.asarray(imgs)
print(imgs.shape)
```

```
(15000, 2048)
```

Training the model:

```
image_model = Sequential()

image_model.add(Dense(embedding_size, input_shape=(2048,), activation='relu'))
image_model.add(RepeatVector(max_len))

image_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               262272
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 40, 128)           0
=================================================================
Total params: 262,272
Trainable params: 262,272
Non-trainable params: 0
_________________________________________________________________
```

```
language_model = Sequential()

language_model.add(Embedding(input_dim=vocab_size, output_dim=embedding_size, input_length=max_len))
language_model.add(LSTM(256, return_sequences=True))
language_model.add(TimeDistributed(Dense(embedding_size)))

language_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 40, 128)           1056512
_________________________________________________________________
lstm_1 (LSTM)                (None, 40, 256)           394240
_________________________________________________________________
time_distributed_1 (TimeDist (None, 40, 128)           32896
=================================================================
Total params: 1,483,648
Trainable params: 1,483,648
Non-trainable params: 0
_________________________________________________________________
```

```
conca = Concatenate()([image_model.output, language_model.output])
x = LSTM(128, return_sequences=True)(conca)
x = LSTM(512, return_sequences=False)(x)
x = Dense(vocab_size)(x)
out = Activation('softmax')(x)
model = Model(inputs=[image_model.input, language_model.input], outputs = out)

# model.load_weights("../input/model_weights.h5")
model.compile(loss='categorical_crossentropy', optimizer='RMSprop', metrics=['accuracy'])
model.summary()
```

```
hist = model.fit([images, captions], next_words, batch_size=512, epochs=200)
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.
python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/200
25493/25493 [==============================] - 12s 462us/step - loss: 5.7067 - acc: 0.0744
Epoch 2/200
25493/25493 [==============================] - 9s 341us/step - loss: 5.2456 - acc: 0.1052
Epoch 3/200
25493/25493 [==============================] - 9s 344us/step - loss: 4.9945 - acc: 0.1372
Epoch 4/200
25493/25493 [==============================] - 9s 343us/step - loss: 4.7792 - acc: 0.1699
Epoch 5/200
25493/25493 [==============================] - 9s 342us/step - loss: 4.5912 - acc: 0.2041
Epoch 6/200
25493/25493 [==============================] - 9s 341us/step - loss: 4.4034 - acc: 0.2345
Epoch 7/200
25493/25493 [==============================] - 9s 351us/step - loss: 4.2948 - acc: 0.2427
```

```
25493/25493 [==============================] - 9s 342us/step - loss: 0.2637 - acc: 0.9009
Epoch 190/200
25493/25493 [==============================] - 9s 341us/step - loss: 0.2719 - acc: 0.8979
Epoch 191/200
25493/25493 [==============================] - 9s 339us/step - loss: 0.2706 - acc: 0.8994
Epoch 192/200
25493/25493 [==============================] - 9s 343us/step - loss: 0.2723 - acc: 0.8987
Epoch 193/200
25493/25493 [==============================] - 9s 342us/step - loss: 0.2665 - acc: 0.8990
Epoch 194/200
25493/25493 [==============================] - 9s 340us/step - loss: 0.2663 - acc: 0.8999
Epoch 195/200
25493/25493 [==============================] - 9s 345us/step - loss: 0.2661 - acc: 0.8989
Epoch 196/200
25493/25493 [==============================] - 9s 342us/step - loss: 0.2702 - acc: 0.8985
Epoch 197/200
25493/25493 [==============================] - 9s 339us/step - loss: 0.2598 - acc: 0.9033
Epoch 198/200
25493/25493 [==============================] - 9s 340us/step - loss: 0.2625 - acc: 0.8997
Epoch 199/200
25493/25493 [==============================] - 9s 343us/step - loss: 0.2595 - acc: 0.9019
Epoch 200/200
25493/25493 [==============================] - 9s 340us/step - loss: 0.2687 - acc: 0.8986
```

Activity 7: Save the Model:

```python
import json
with open('word_2_indices.json', 'w') as f:
    json.dump(word_2_indices, f)

# Saving indices_2_word to JSON file
with open('indices_2_word.json', 'w') as f:
    json.dump(indices_2_word, f)
```

```python
model.save_weights("model_weights.h5")
```

```python
model.save('my_combined_model.h5')
```

```python
model.save('ImageCaptioning.h5')
```

The model is saved with .h5 extension as follows An H5 file is a data file saved in the

Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

Activity 8: Test The model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data. Load the saved model.

```python
def preprocessing(img_path):
    im = image.load_img(img_path, target_size=(224,224,3))
    im = image.img_to_array(im)
    im = np.expand_dims(im, axis=0)
    return im
```

```python
def get_encoding(model, img):
    image = preprocessing(img)
    pred = model.predict(image).reshape(2048)
    return pred
```

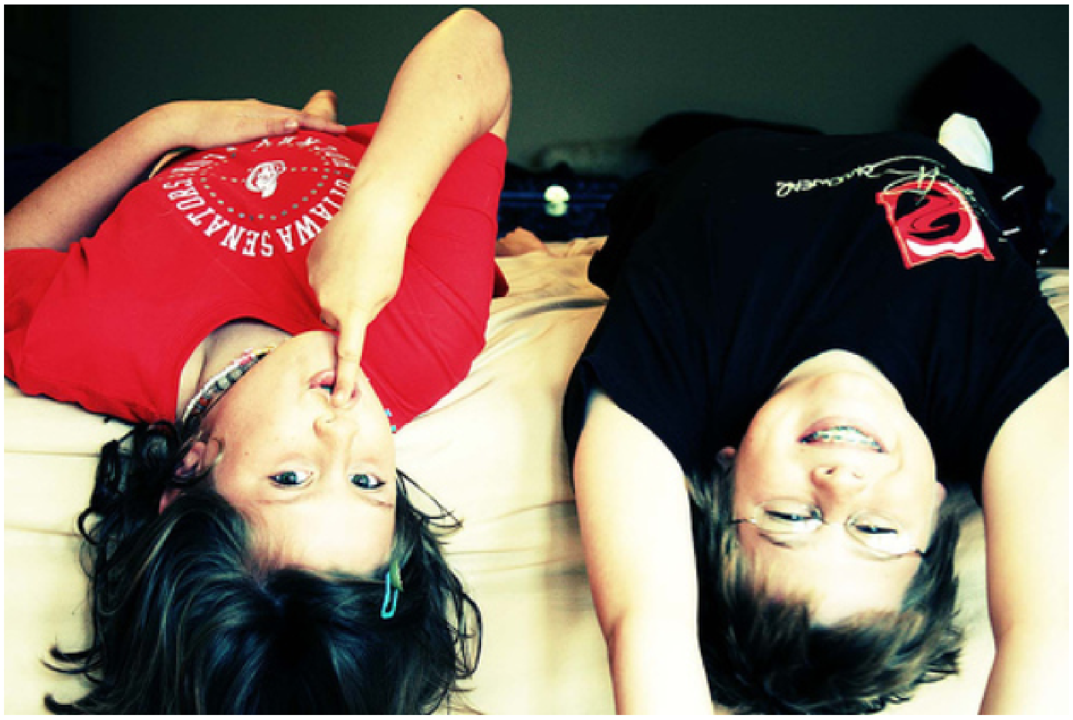
```python
resnet = ResNet50(include_top=False,weights='imagenet',input_shape=(224,224,3),pooling='avg')
```

```python
img = "../input/flickr_data/Flickr_Data/Images/1453366750_6e8cf601bf.jpg"

test_img = get_encoding(resnet, img)
```

```python
def predict_captions(image):
    start_word = ["<start>"]
    while True:
        par_caps = [word_2_indices[i] for i in start_word]
        par_caps = sequence.pad_sequences([par_caps], maxlen=max_len, padding='post')
        preds = model.predict([np.array([image]), np.array(par_caps)])
        word_pred = indices_2_word[np.argmax(preds[0])]
        start_word.append(word_pred)

        if word_pred == "<end>" or len(start_word) > max_len:
            break

    return ' '.join(start_word[1:-1])

Argmax_Search = predict_captions(test_img)
```

Taking an image as input and checking the results.

```
z = Image(filename=img)
display(z)

print(Argmax_Search)
```



```
Two children are laying upside down on their bed .
```

Milestone 4: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface. In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Image” button, the next page is opened where the user chooses the image and predicts the output.

Activity 1: Create HTML Pages

- We use HTML to create the front end part of the web page.
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
- Link :CSS , JS

Create app.py (Python Flask) file: -

Write below code in Flask app.py python file script to run Imagae Caption Generation Project.

Final Result :

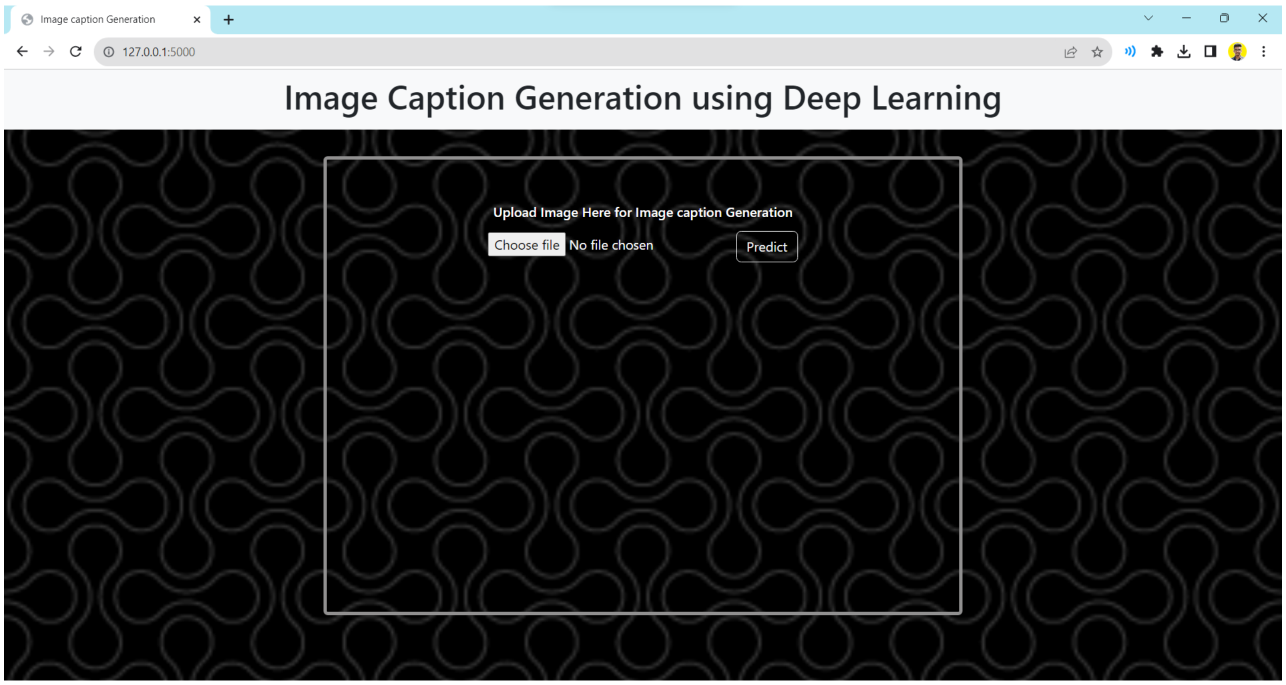


Image caption Generation
127.0.0.1:5000
Image Caption Generation using Deep Learning
Upload Image Here for Image caption Generation
Choose file
No file chosen
Predict

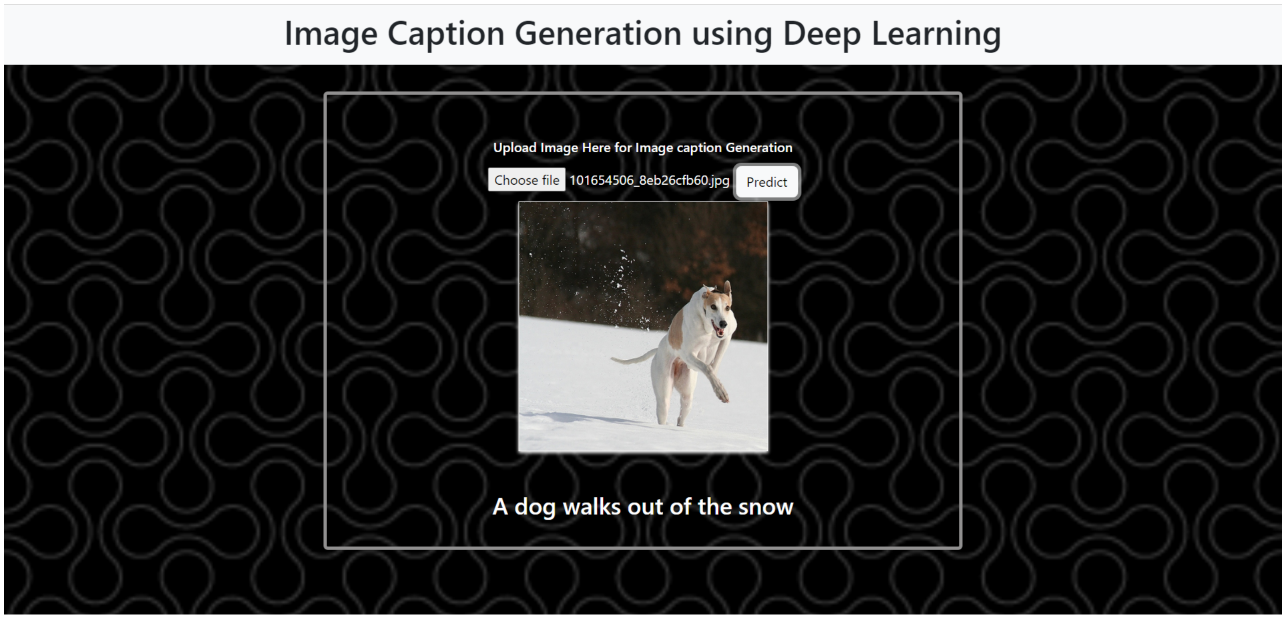


Image Caption Generation using Deep Learning

Upload Image Here for Image caption Generation

A dog walks out of the snow