ABSTRACT:

This project is about creating a deep learning model that generates captions for images using techniques from computer vision and natural language processing. The model uses the Flickr 8K dataset for training and a ResNet50 architecture for obtaining image features. The captions are generated using Long Short-Term Memory (LSTM) networks and GloVe vectors for creating word embeddings. The model was built using Keras and TensorFlow libraries in Python and was trained for 20 epochs. The project provides usage instructions to run the model on the terminal and includes acknowledgments to research papers used as references.

Table of Content:

CHAPTER – 1
INTRODUCTION

Introduction:

This project focuses on the use of Deep Learning and Neural Networks for generating captions for images, using techniques from Computer Vision and Natural Language Processing. It uses the popular ResNet50 architecture to process input images and obtain their feature vectors. Long Short-Term Memory (LSTM) networks are used for generating captions, with word embeddings created using GloVe vectors. The model is built using the Keras Functional API and trained on the Flickr 8K dataset, which contains 8000 images with five different captions each. The project also provides instructions for installing dependencies and running the model.

Aim of the Project:

The aim of this project is to develop a deep learning model for generating captions for images using techniques from computer vision and natural language processing. The model uses a CNN architecture, ResNet50, to obtain image features and Long Short-Term Memory (LSTM) networks for generating captions. The Flickr 8K dataset, which contains 8000 images with five different captions for each image, is used for training the model. The word embeddings for the captions are created using GloVe vectors, and the neural network is built using the Keras Functional API. The model is

trained for 20 epochs, and at the end of each epoch, the model is saved in the "/model_checkpoints" directory. The model can be run on a random image from the test dataset using the predict.py script or on a custom image using the generate.py script. The project uses Python3, Keras, TensorFlow, Numpy, and Matplotlib.

Project Domain:

The project domain is a Deep Learning Model for generating captions for images, which uses techniques from Computer Vision and Natural Language Processing. It employs the ResNet50 architecture for obtaining the image features and Long Short-Term Memory (LSTM) networks for generating captions. The project uses the Flickr 8K dataset for training the model, which contains 8000 Images with five different captions describing the entities and events depicted in the image. The model uses GloVe vectors for creating word embeddings for the captions. The project was built using the Keras Functional API and trained for 20 epochs. The usage involves installing the required dependencies, cloning the repository, and running the model over a random image from the test dataset or generating a caption for a custom image. The frameworks, libraries, and languages used in the project are Keras, TensorFlow, Python3, NumPy, and Matplotlib.

Problem Statement:

The problem statement tackled here is to generate captions for images using a deep learning model that uses techniques from

both computer vision and natural language processing. The model is built using the Flickr 8K dataset, which contains 8000 images, each with five different captions describing the entities and events depicted in the image. The ResNet50 architecture is used to process the input images and get the feature vectors, and Long Short-Term Memory (LSTM) networks are used for generating the captions. The GloVe vectors are used for creating the word embeddings for the captions. The neural network is built using the Keras Functional API and is trained for 20 epochs. The model is saved at the end of each epoch, and the encoded features for training and test images are stored at "encoded_train_features.pkl" and "encoded_test_features.pkl" respectively. The model can be used to generate captions for custom images by running the "generate.py" script.

CHAPTER – 2

REQUIREMENT SPECIFICATION

Requirement specification refers to the detailed description of a project's requirements that should be met to ensure successful completion of the project. In this context, the requirement specification includes information about the Deep Learning Model for generating captions for images, including the techniques used from Computer Vision and Natural Language Processing. The specification provides details about the dataset used for training the model, the model architecture, and the frameworks, libraries, and languages used to develop the model.

The Flickr 8K dataset is used for training the model, and the ResNet50 architecture is used to obtain the image features. GloVe vectors are used for creating the word embeddings for the captions, and the Keras Functional API is used to build the neural network for generating the captions. The model is trained for 20 epochs, and the process takes about half an hour.

The specification also includes information on how to use the model. Users can run the model on the terminal by executing certain commands. To generate captions for a custom image, the image must be moved to the current directory and renamed to "input.jpg". The generated caption will be printed after running the "generate.py" script.

The specification acknowledges various articles and research papers that the project referred to while developing the model.

CHAPTER – 3

PROJECT DESCRIPTION

Proposed System:

The proposed system is a deep learning model for generating captions for images using techniques from computer vision and natural language processing. The system uses the Flickr 8K dataset for training the model, which contains 8000 images, most of them featuring people and animals in a state of action. Each image is provided with five different captions describing the entities and events depicted in the image, ensuring enough linguistic variety in the description of the images.

The system uses the ResNet50 architecture for obtaining the image features and GloVe vectors for creating the word embeddings for the captions. The version of GloVe used in this project contains 50-dimensional embedding vectors for 6 billion English words. The neural network for generating the captions has been built using the Keras Functional API. The features vectors obtained from the ResNet50 network are processed and combined with the caption data, which has been passed through an LSTM layer. This combined information is passed through a Dense layer followed by a Softmax layer over the vocabulary words.

The system can be run on a terminal using commands such as pip install python3, pip install numpy, pip install matplotlib, pip install pickle-mixin, pip install tensorflow, and pip install keras. Once the dependencies are installed, the system can be cloned from the GitHub repository, and the user can move an

image to the current directory and rename it to "input.jpg." Afterward, the user can execute the generate.py file to load the model and run it with the input image, which generates a caption that will be printed.

The proposed system references several articles and research papers, including https://www.ijcai.org/Proceedings/15/Papers/593.pdf, https://arxiv.org/abs/1411.4555v2, and https://en.wikipedia.org/wiki/Long_short-term_memory.

Modules Included:

The following modules are used in the Deep Learning Model for generating captions for images:

- Keras
- Tensorflow
- Python3
- Numpy
- Matplotlib
- pickle-mixin

The ResNet50 architecture is used for obtaining image features. GloVe vectors are used for creating the word embeddings for the captions. The Keras Functional API is used for building the neural network for generating the captions. The model is trained for 20 epochs and saved at the end of each epoch in the "/model_checkpoints" directory. To run the model over a random image from the test dataset and see the caption, execute the command "python3 predict.py". To run the model and generate a caption for a custom image, move the image (in JPEG format) to the current directory and

rename it to "input.jpg", then execute the command "python3 generate.py".

Merits:

Merits of the project are:

1. Effective use of Deep Learning techniques from both Computer Vision and Natural Language Processing domains.
2. Efficient use of Convolutional Neural Networks (CNNs) for recognizing patterns in images and ResNet50 architecture for obtaining the image features.
3. Effective use of Long Short-Term Memory (LSTM) networks for generating captions for the images.
4. Use of the Flickr 8K dataset containing 8000 images, each with five different captions ensuring enough linguistic variety in the description of the images.
5. Use of GloVe vectors for creating the word embeddings for the captions.
6. The neural network for generating the captions has been built using the Keras Functional API.
7. Training of the model for 20 epochs.
8. Use of popular frameworks such as Keras, TensorFlow, Python3, Numpy and Matplotlib.
9. Availability of the pre-trained model checkpoints in the "/model_checkpoints" directory.
10. Easy usage of the model with simple terminal commands.

Feasibility Study:

Feasibility study of the project involves the evaluation of the possibility and practicality of achieving the project goals. Here are some key points to consider:

1. Dataset: The project uses the Flickr 8K dataset, which is readily available and has been used in many research studies. It is a diverse dataset with a variety of images and captions, which makes it suitable for training a caption generation model.
2. Model Architecture: The project uses a combination of ResNet50 for image feature extraction and LSTM for caption generation. Both these architectures have been well studied and widely used in various deep learning applications.
3. Frameworks, Libraries & Languages: The project uses popular and well-established frameworks, libraries, and languages, such as Keras, TensorFlow, Python, Numpy, and Matplotlib. These tools have a large community support and provide a wide range of functionalities for deep learning.
4. Implementation: The project has been implemented using the Keras Functional API, which is a high-level API for building complex deep learning models. The implementation is straightforward, and the code is well-organized and documented.
5. Performance: The project has been trained for 20 epochs, and the model checkpoints have been saved at the end of each epoch. The performance of the model can be evaluated using various metrics such as BLEU, METEOR, and ROUGE. The generated captions can also be evaluated using human evaluation.
6. Scalability: The model can be trained on a larger dataset, such as the Flickr 30K or MS COCO dataset, to improve its performance. The model can also be fine-tuned on a specific domain, such as medical or scientific images, to generate domain-specific captions.

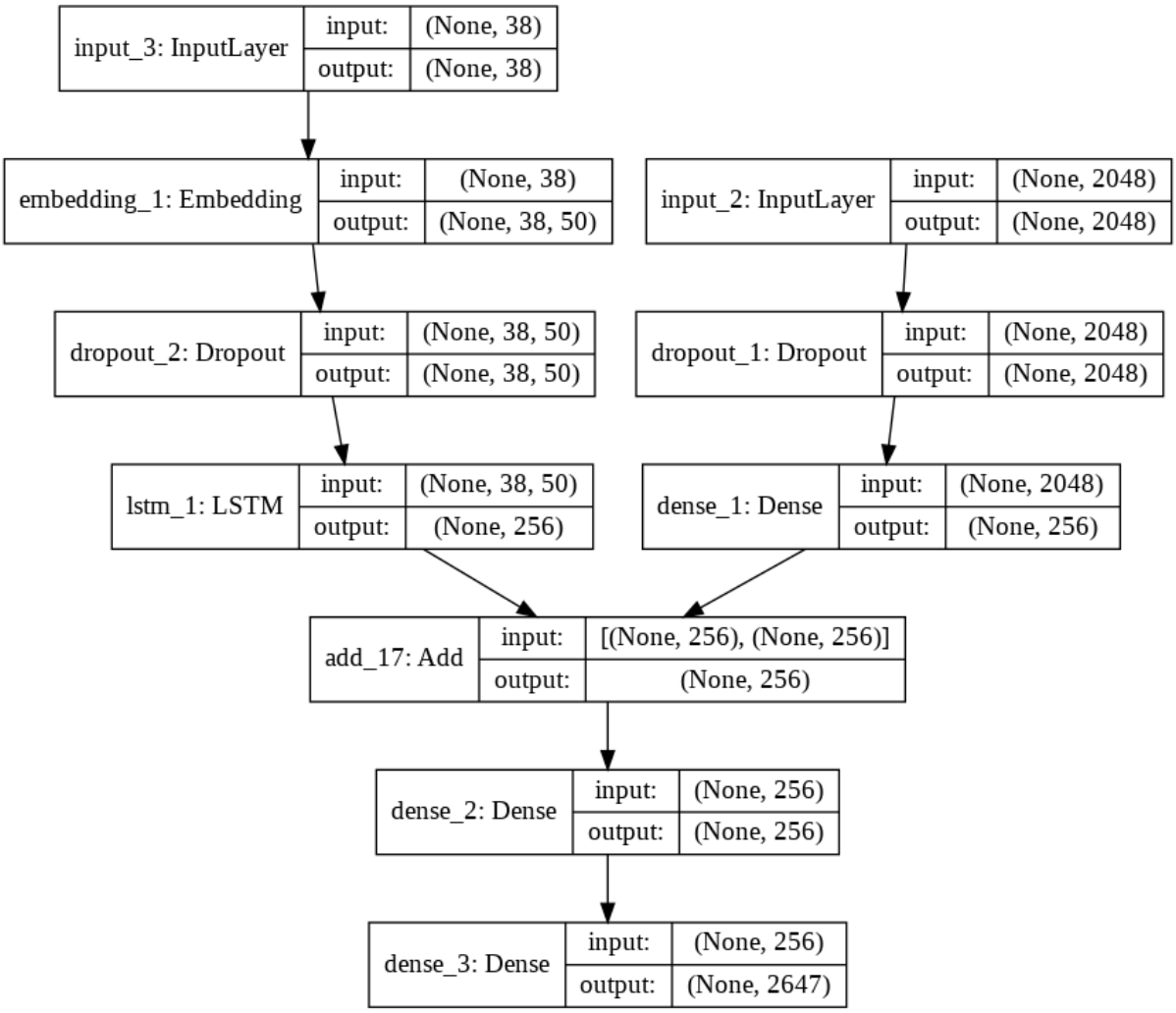
Based on these key points, the project seems feasible and practical. The availability of the dataset, the well-established architectures, the use of popular tools, the straightforward implementation, and the potential for scalability make the project a viable solution for image caption generation. However, the project's performance needs to be evaluated and

compared with other state-of-the-art models to determine its effectiveness.

CHAPTER – 4

MODULE DESCRIPTION

General Architecture:



Module Description:

Modules used in the project are:

1. Keras - A high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
2. TensorFlow - An open-source software library for dataflow and differentiable programming across a range of tasks.

3. Python3 - A high-level programming language that emphasizes code readability and simplicity.
4. Numpy - A Python library for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices.
5. Matplotlib - A plotting library for the Python programming language and its numerical mathematics extension NumPy.

The project also uses the following pre-trained models:

1. ResNet50 - A convolutional neural network architecture that was originally designed for ImageNet Large Scale Visual Recognition Challenge (ILSVRC). It is trained to classify images into one of the 1000 categories defined in the ImageNet dataset.
2. GloVe - A pre-trained word embedding model that provides vector representations of words. It is trained on a corpus of billions of words and can be used for tasks such as text classification, named entity recognition, and sentiment analysis.

The project also uses the following files:

1. "encoded_train_features.pkl" - A file that stores the encoded feature vectors for the training images obtained from the ResNet50 model.
2. "encoded_test_features.pkl" - A file that stores the encoded feature vectors for the test images obtained from the ResNet50 model.
3. "predict.py" - A Python script that can be used to generate captions for a random test image from the dataset.
4. "generate.py" - A Python script that can be used to generate captions for a custom image.

CHAPTER – 5

IMPLEMENTATION AND TESTING

Implementation:

This project is a Deep Learning model for generating captions for images using techniques from Computer Vision and Natural Language Processing. The model is trained on the Flickr 8K dataset, which contains 8000 images, each with five different captions describing the entities and events depicted in the image. The ResNet50 architecture is used to obtain the image features, and the GloVe vectors are used for creating the word embeddings for the captions. The neural network for generating the captions has been built using the Keras Functional API, and it uses LSTM networks, which are a variant of Recurrent Neural Networks widely used in NLP.

During the training process, the ResNet50 network is used to obtain the image feature vectors, and the captions are processed by passing them through an LSTM layer to obtain the sequence of embeddings. The feature vectors and embeddings are then combined and passed through a Dense layer followed by a Softmax layer over the vocabulary words to generate the caption. The model is trained for 20 epochs, and at the end of each epoch, the model is saved in the "/model_checkpoints" directory.

To use the model, one can clone the repository and install the required dependencies using pip. After installing the dependencies, one can run the model over a random image from the test dataset and see the caption by executing the "python3 predict.py" command. To run the model and

generate captions for a custom image, one can move the image (in JPEG format) to the current directory and rename it to "input.jpg". Then type the command "python3 generate.py", and the generated caption will be printed.

The project uses the Keras and TensorFlow frameworks, Python3 language, and the Numpy and Matplotlib libraries. The project also acknowledges the various articles and research papers referred to during its development.

Testing:

To test the project, one can run the model over a random image from the test dataset and see the caption by executing the command "python3 predict.py" on the terminal. This can be executed multiple times, and each time a random image and its caption generated by the model will be displayed. To generate a caption for a custom image, one needs to move the image in JPEG format to the current directory and rename it to "input.jpg." Then type the following on the terminal - "python3 generate.py." This loads the model and runs it with the input image. The generated caption will be printed.

During training, the model was saved at the end of each epoch in the "/model_checkpoints" directory, which can be used to load the model and generate captions on new images. The project uses Keras, Tensorflow, Python3, Numpy, and Matplotlib frameworks, libraries, and languages. It also used the Flickr 8K dataset for training the model and GloVe vectors for creating the word embeddings for the captions.

Source Code:

Building vocabulary for most commonly occurring words in caption text:

```
# Read the file tokens_clean.txt and store the cleaned captions in a
dictionary
import json

content = None

with open ("data/textFiles/tokens_clean.txt", 'r') as file:
    content = file.read()

json_acceptable_string = content.replace("'", "\"")
content = json.loads(json_acceptable_string)
#Iterate over the captions word by word, and append each word to total_words

total_words = []

for key in content.keys():
    for caption in content[key]:
        for i in caption.split():
            total_words.append(i)

print("Total Words = %d" %len(total_words))
# Compute the frequency of occurrence of each word, and store it in a
dictionary of word-freq

import collections

counter = collections.Counter(total_words)
freq_cnt = dict(counter)

print("Number of unique words = " + str(len(freq_cnt.keys())))
# Store the word-freq pairs (from the dictionary freq_cnt) in a list, sorted
in decreasing order of frequency

sorted_freq_cnt = sorted(freq_cnt.items(), reverse=True, key=lambda x:x[1])
threshold = 5

#Filter off those words whose frequency of occurrence in less than threshold
sorted_freq_cnt = [x for x in sorted_freq_cnt if x[1]>threshold]
# Store these common words in total_words
total_words = [x[0] for x in sorted_freq_cnt]

print("Number of common unique words = " + str(len(total_words)))
```

Preparing test and train data:

```
# Read training and testing image names

train_file_data = ""
test_file_data = ""

with open ("data/textFiles/trainImages.txt", 'r') as file:
    train_file_data = file.read()

with open ("data/textFiles/testImages.txt", 'r') as file:
    test_file_data = file.read()

# Obtain a list of train and test images
train_data = [img_file_name for img_file_name in
train_file_data.split("\n")[:-1]]
test_data = [img_file_name for img_file_name in test_file_data.split("\n")[:-
1]]

# Obtain image ID from image file name
train_data = [image.split(".")[0] for image in train_data]
test_data = [image.split(".")[0] for image in test_data]
train_data[:5]

# For each imageID in train_data, store its captions in a dictionary

train_content = {}

for imageID in train_data:
    train_content[imageID] = []
    for caption in content[imageID]:
        # Add a start sequence token in the beginning and an end sequence
token at the end
        cap_to_append = "startseq " + caption + " endseq"
        train_content[imageID].append(cap_to_append)
train_content['1007320043_627395c3d8']
```

Extract features from Image using a ResNet50 Architecture:

```
from keras.applications import ResNet50, preprocess_input, decode_predictions

model = ResNet50(weights = 'imagenet', input_shape = (224, 224, 3))
model.summary()


from keras.models import Model

model_new = Model (model.input, model.layers[-2].output)

from keras.preprocessing import image
```

```
import numpy as np

def preprocess_image (img):
    img = image.load_img(img, target_size=(224, 224))
    img = image.img_to_array(img)

    # Convert 3D tensor to a 4D tendor
    img = np.expand_dims(img, axis=0)

    #Normalize image accoring to ResNet50 requirement
    img = preprocess_input(img)

    return img

import matplotlib.pyplot as plt

img = preprocess_image("data/Images/1000268201_693b08cb0e.jpg")
print(img.shape)
plt.imshow(img[0])
plt.axis('off')
plt.show()

# A wrapper function, which inputs an image and returns its encoding (feature
vector)
def encode_image (img):
    img = preprocess_image(img)
    feature_vector = model_new.predict(img)

    feature_vector = feature_vector.reshape((-1,))
    return feature_vector

from time import time

train_encoding = {}
# Create a dictionary of iamgeID and its feature vector

start_time = time()
for index, imageID in enumerate (train_data):
    image_path = "data/Images/" + imageID + ".jpg"

    train_encoding[imageID] = encode_image(image_path)

    # Print progress
    if index%100 == 0:
        print("Encoding in progress... STEP", index)

end_time = time()
print("Total time taken:", end_time-start_time, "sec")
```

```
# Store the above computed features on the disk
# Use pickle to dump the entire data
import pickle

with open("encoded_train_features.pkl", "wb") as file:
    # Pickle allows to store any object as a file on the disk
    pickle.dump(train_encoding, file)

test_encoding = {}
# Create a dictionary of iamgeID and its feature vector

start_time = time()
for index, imageID in enumerate (test_data):
    image_path = "data/Images/" + imageID + ".jpg"

    test_encoding[imageID] = encode_image(image_path)

    # Print progress
    if index%100 == 0:
        print("Encoding in progress... STEP", index)

end_time = time()
print("Total time taken:", end_time-start_time, "s")

with open("encoded_test_features.pkl", "wb") as file:
    pickle.dump(test_encoding, file)
```

Pre-processing the Captions:

```
# Create the word-to-index and index-to-word mappings
word_to_index = {}
index_to_word = {}

for i, word in enumerate(total_words):
    word_to_index[word] = i+1
    index_to_word[i+1] = word
print(len(index_to_word))
print(index_to_word[5])
print(word_to_index['is'])
# Add startseq and endseq also to the mappings
index_to_word[2645] = 'startseq'
word_to_index['startseq'] = 2645

index_to_word[2646] = 'endseq'
word_to_index['endseq'] = 2646

VOCAB_SIZE = len(word_to_index) + 1
print(VOCAB_SIZE)
```

```
with open("data/textFiles/word_to_idx.pkl", "wb") as file:
    pickle.dump(word_to_index, file)

with open("data/textFiles/word_to_idx.pkl", "wb") as file:
    pickle.dump(word_to_index, file)

with open("data/textFiles/idx_to_word.pkl", "wb") as file:
    pickle.dump(index_to_word, file)

# Get the maximum length of a caption
max_len = 0

for cap_list in train_content.keys():
    for caption in train_content[cap_list]:
        max_len = max(max_len, len(caption.split()))

print(max_len)

# Get the Glove word Embeddings
# This contains 50-dimensional embeddings for 6 Billion English words
file = open("glove.6B.50d.txt",encoding='utf8')

# Create a mapping from word to embedding
word_to_embedding = {}

for line in file:
    values = line.split()

    word = values[0]
    embedding = np.array (values[1:], dtype='float')
    word_to_embedding[word] = embedding

file.close()
word_to_embedding["apple"]

EMBEDDING_DIM = 50

def get_embedding_matrix():
    embedding_matrix = np.zeros((VOCAB_SIZE, EMBEDDING_DIM))

    for word, index in word_to_index.items():
        embedding = word_to_embedding[word]

        if embedding is not None:
            embedding_matrix[index] = embedding

    return embedding_matrix
```

```
index_to_embedding = get_embedding_matrix()
print(index_to_embedding.shape)
```

Building Neural Network using Keras Functional API:

```
from keras.layers import Input, Dense, Dropout, Embedding, LSTM

#Convert feature vector of image to smaller vector

#Output of ResNet goes into following input layer
inp_img_features = Input(shape=(2048,))

inp_img1 = Dropout(0.3)(inp_img_features)
inp_img2 = Dense(256, activation='relu')(inp_img1)

#Now take Captions as input

#Actual input size will be (batch_size x max_length_of_caption)
#But here we specify only for one example
inp_cap = Input(shape=(max_len,))
inp_cap1 = Embedding(input_dim=VOCAB_SIZE, output_dim=50,
mask_zero=True)(inp_cap)
inp_cap2 = Dropout(0.3)(inp_cap1)
inp_cap3 = LSTM(256)(inp_cap2)
# inp_cap3 essentially captures the entire sentence that has been generated
till now

from keras.layers import add

# Decode the inputs

# So effectively, an image (224x224x3) goes through ResNet50
# Then as 2048 dimensional it goes through the above earlier architecture
# The final output is inp_img2 (256 dimensional) which now goes through the
Decoder

# Similarly for the captions which initially have shape (batch_size x max_len)
# Then after passing through Embedding layer comes out as (batch_size x
max_len x 50(embedding_size)))
# Then it passes through the above LSTM layer and comes out as inp_cap3 (a 256
dimensional vector)

# Add the two above tensors
decoder1 = add([inp_img2, inp_cap3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(VOCAB_SIZE, activation='softmax')(decoder2)
```

```
# Combined model
model = Model (inputs=[inp_img_features, inp_cap], outputs=outputs)

model.summary()

# Preinitialise Embedding layer
model.layers[2].set_weights([index_to_embedding])
model.layers[2].trainable = False

model.compile(loss="categorical_crossentropy", optimizer="adam")
```

Creating a Data Loader;

```
from keras.utils import pad_sequences
from keras.utils import to_categorical

def data_generator (train_content, train_encoding, word_to_index, max_len,
batch_size):
    X1, X2, y = [], [], []
    n = 0

    while True:
        for imageID, cap_list in train_content.items():
            n += 1

            image = train_encoding [imageID]

            for caption in cap_list:
                idx_seq = [word_to_index[word] for word in caption.split() if
word in word_to_index]

                for i in range (1, len(idx_seq)):
                    xi = idx_seq[0 : i] # The input sequence of words
                    yi = idx_seq[i] # The next word after the above sequence
(this is expected to be predicted)

                    # Add a padding of zeros ao lengths of input sequences
become equal
                    xi = pad_sequences([xi], maxlen=max_len, value=0,
padding='post')[0] # Take the first row only, since this method inputs &
returns a 2D array
                    # Convert the expected word to One Hot vector notation
                    yi = to_categorical([yi], num_classes=VOCAB_SIZE)[0]

                    X1.append(image)
                    X2.append(xi)
                    y.append(yi)

                if n==batch_size:
```

```
                    yield [[np.array(X1), np.array(X2)], np.array(y)]

                    X1, X2, y = [], [], []
                    n=0
```

Training Model:

```
epochs = 20
batch_size = 3
steps = len(train_content)//batch_size

for i in range(epochs):
    # Create an instance of the generator
    generator = data_generator(train_content, train_encoding, word_to_index,
max_len, batch_size)
    model.fit_generator(generator, steps_per_epoch=steps)
    model.save('model_' + str(i) + '.h5')
```

Generating Captions:

```
import json
from keras.models import load_model
import pickle
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.utils import pad_sequences
import collections
import keras.utils as image
from keras.applications import ResNet50
from keras.applications.imagenet_utils import preprocess_input,
decode_predictions
from keras.models import Model


#img_name = "YellowLabradorLooking.jpg"
img_name = "s.jpg"
# img_name = "xc.jpg"

# Read the files word_to_idx.pkl and idx_to_word.pkl to get the mappings
between word and index
word_to_index = {}
with open ("data/textFiles/word_to_idx.pkl", 'rb') as file:
    word_to_index = pd.read_pickle(file)

index_to_word = {}
with open ("data/textFiles/idx_to_word.pkl", 'rb') as file:
    index_to_word = pd.read_pickle(file)
```

```
print("Loading the model...")
model = load_model('model_checkpoints/model_19.h5')

resnet50_model = ResNet50 (weights = 'imagenet', input_shape = (224, 224, 3))
resnet50_model = Model (resnet50_model.input, resnet50_model.layers[-
2].output)



# Generate Captions for a random image in test dataset
def predict_caption(photo):

    inp_text = "startseq"

    for i in range(38):
        sequence = [word_to_index[w] for w in inp_text.split() if w in
word_to_index]
        sequence = pad_sequences([sequence], maxlen=38, padding='post')

        ypred = model.predict([photo, sequence])
        ypred = ypred.argmax()
        word = index_to_word[ypred]

        inp_text += (' ' + word)

        if word == 'endseq':
            break

    final_caption = inp_text.split()[1:-1]
    final_caption = ' '.join(final_caption)
    return final_caption



def preprocess_image (img):
    img = image.load_img(img, target_size=(224, 224))
    img = image.img_to_array(img)

    # Convert 3D tensor to a 4D tendor
    img = np.expand_dims(img, axis=0)

    #Normalize image accoring to ResNet50 requirement
    img = preprocess_input(img)

    return img
```

```
# A wrapper function, which inputs an image and returns its encoding (feature
vector)
def encode_image (img):
    img = preprocess_image(img)

    feature_vector = resnet50_model.predict(img)
    # feature_vector = feature_vector.reshape((-1,))
    return feature_vector



print("Encoding the image ...")
photo = encode_image(img_name).reshape((1, 2048))



print("Running model to generate the caption...")
caption = predict_caption(photo)

img_data = plt.imread(img_name)
plt.imshow(img_data)
plt.axis("off")

plt.show()
print(caption)
```

Demo Example Outputs:

A white dog running through the snow

A dog jumps over a red and white obstacle

- A brightly decorated bicycle with cart with people walking around in the background.
- A street vending machine is parked while people walk by.
- A street vendor on the corner of a busy intersection.
- People on the city street walk past a puppet theatre.
- People walk around a mobile puppet theatre in a big city.