CHAPTER-1
INTRODUCTION

1.1 Overview

The first intention of trying to "understand the scene" is one of the base ideas in computer vision that leads to a continuous increase in the need to apprehend the high level context in images regarding object recognition and image classification. Images have become ubiquitous in a variety of fields as so many people and systems extract vast amounts of information from imagery hence rapidly growing the field of computer vision.

Image classification is the task of taking an input image and outputting a class or a probability of classes that best describes the image. For humans, this task is one of the first skills we learn and it comes naturally and effortlessly as adults.

Information that can be vital in areas such as robotics, hospitals, self-driving cars, surveillance or building 3D representations of objects. While each of the abovementioned applications differs by numerous factors, they share the common process of correctly annotating an image with one or a probability of labels that correlates to a series of classes or categories. This procedure is known as image classification and, combined with machine learning, it has become an important research topic in the field, on account of the focus on the understanding of what an image is representative of. The complex process of identifying the type of materials in diverse tasks linked to image-based scene perspectives has taken advantage of the combination of machine learning techniques applied to the up-to-date development of neural networks. This outlines the challenging problem of image classification due to the variety of the definite features of image. The state of-the-art solutions rely massively on the attention that Computer Vision systems have received, which led to a series of algorithms being developed and images being collected in datasets.

Object detection is a general term for computer vision techniques for locating and labelling objects. Object detection techniques can be applied both to static images or moving images. Computer vision techniques are already in wide use in every field today, as they can offer valuable insight about movement of individual object or whole items to training, help in tracking and help visualizing progression during object movement.

Open-source libraries such as OpenCV's DNN library and TensorFlow Image classification API offer easy-to-use, Open-source frameworks where pre-trained models

for Image Recognition (such as ones downloadable from the TensorFlow model zoo) reach high accuracy in detecting various object from humans to tv monitors.

Object detection is a term related to computer vision and image processing techniques for locating and annotating all the potential objects in the images. The image scan be either static pictures or moving frames. The task of object detection is closely related to two other problems: image classification and object localization. Image classification also referred to as image recognition, is about classifying an image to a particular class out of all the possible classes. Object localization is more advanced than image recognition, and it demands to localize the labelled object in an image. Object detection is the most complex of all—it involves labelling and localizing multiple objects in an image. Object detection techniques are extensively used for applications related to face detection, locating pedestrian on streets or players on football grounds, and detecting vehicles on roads.

Image classification is a task where a computer will predict an image belongs to which class. Before deep learning starts booming, tasks like image classification cannot achieve human-level performance. It is because the machine learning model cannot learn the neighbour information of an image. The model only gets pixel-level information. Thanks to the power of deep learning, image classification task can reach a human level performance using a model called Convolutional Neural Network (CNN). CNN is a type of deep learning model that learns representation from an image. This model can learn from low to high-level features without human involvement. That is why I will use CNN and Faster R-CNN algorithm in this project.

The model learns not only information on a pixel level. The model also learns the neighbour information from an image by a mechanism called convolution. Convolution will aggregate neighbourhood information by multiplying the collection of pixels in a region and sum them into a value. Those features will be used to classify the image into a class, and then the evaluation done.

1.2 INTRODUCTION TO CURRENT SYSTEM

There have been several recent systems that use Transformers for image captioning. Here are some examples:

- VilBERT: VilBERT (short for Vision and Language BERT) is a Transformer-based model that jointly learns representations of images and text. It was introduced in a 2019 paper by Lu et al. VilBERT uses two different types of self-attention

mechanisms to capture the relationships between image regions and words in the caption.

- LXMERT: LXMERT (short for Language and Vision Mesh Transformer) is another Transformer-based model for joint image and text representation learning. It was introduced in a 2019 paper by Tan and Bansal. LXMERT uses a meshed multi-head attention mechanism to integrate visual and linguistic features.

- UNITER: UNITER (short for Universal Image-Text Representation) is a Transformer-based model that learns a joint image-text embedding. It was introduced in a 2020 paper by Chen et al. UNITER uses cross-modal attention mechanisms to align visual and textual features and can perform tasks such as image-text retrieval and image captioning.

- OSCAR: OSCAR (short for Object-Semantics Aligned Pre-training) is another Transformer-based model for joint image and text representation learning. It was introduced in a 2020 paper by Li et al. OSCAR uses a multi-task pre-training framework to learn both object-level and semantic-level visual features and can perform tasks such as image captioning, visual question answering, and image-text retrieval.

Overall, these systems demonstrate the effectiveness of using Transformers for image captioning tasks. By capturing the relationships between image regions and the corresponding words in the caption using self-attention mechanisms, these models can generate more informative and coherent captions.

1.3 NEED OF PROPOSED SYSTEM

One of the main objectives of this project is to train a machine learning model on any of the Machine learning datasets like COCO2017 or CIFAR-10/100 or any other image dataset with the help of machine learning algorithms (CNN and Transformer). This whole project or I say under the hood of the Deep Neural Network for this project is Developed with the help of TENSORFLOW and Keras library of Python.

The proposed algorithm is consisting of four main steps.

- Creating dataframe of images with their captions for training set
- Feeding the Tokenized images to CNN (feature vector are generated)
- Giving the feature vectors to the Transformer's Incoder to get Context Vector
- Giving Context vector to the Transformer's Decoder to get caption sequence

1.3.1 Need of Neural Networks

A computational model that works in a similar way to the neurons in the human brain. Each neuron takes an input, performs some operations then passes the output to the following neuron.

Convolutional Neural Network: A special type Neural Networks that works in the same way of a regular neural network except that it has a convolution layer at the beginning. The main problem with image classification is the difficulty to find useful features. The manual handcraft of creating features from images like shapes, edges, regions is not an easy one even if there are important progresses in this field.

However, a neural network, along with learning a model for classification can create and select automatically useful features. Modern approaches like convolutional networks, deep learning networks, tangential networks are also able to select features which are also invariant to some types of transformations like rotations, transpositions, scaling.

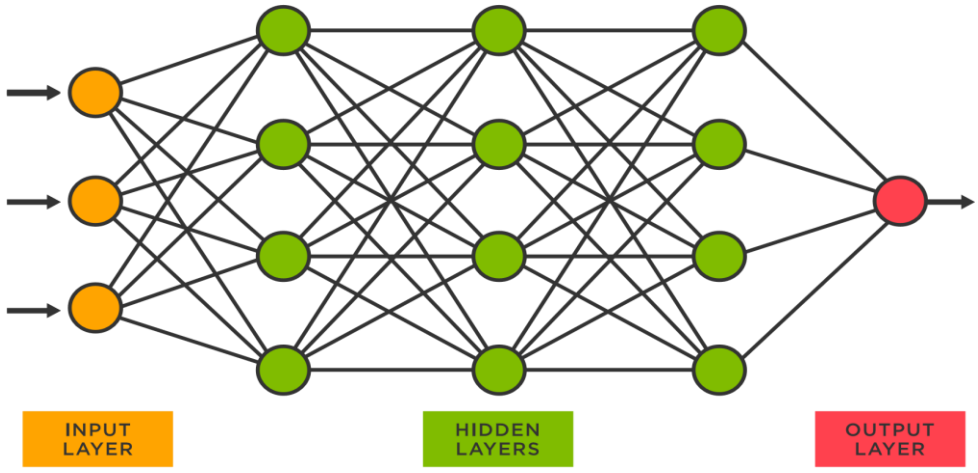


Fig – 1.2 : Illustration of Neural networks

1.3.2 Need of Transformer

Traditionally, for things like Image captioning (Vector to Sequence), Image classification and more, RNNs (Recurrent Neural Network) are used. RNN is a Sequence Generating model based on the LSTM (Long Short-Term Memory) to overcome the problem of "Long term Dependency". But the Drawback of RNN is that the sequence generated by it will be generated Word-by-Word.

Transformers, on the other hand, are a type of neural network architecture that was introduced in 2017 by Vaswani et al. They do not have any recurrence or feedback mechanism, and instead rely on self-attention to process sequences of input. The Transformer architecture consists of an encoder and a decoder, both of which contain multiple layers of self-attention and feedforward neural networks. Self-attention allows the network to attend to different parts of the

input sequence when computing the output at each position hence, relates to different positions (or different words) in a single sequence, in order to compute a representation of that sequence. This makes Transformers well-suited for parallel processing, as each output position can be computed independently of the others.

Long-range dependencies: Image captioning requires understanding of the entire image, not just the local context. Transformers can capture long-range dependencies between image features and the corresponding words in the caption. In contrast, RNNs may struggle with long-term dependencies and can suffer from vanishing gradients when processing long input sequences.

Attention mechanism: Transformers are designed to incorporate an attention mechanism, which allows them to selectively focus on different parts of the image when generating each word in the caption. This attention mechanism enables the network to attend to the most relevant parts of the image and to generate captions that are more descriptive and coherent.

Overall, Transformers are better suited for image captioning tasks because they can more effectively capture the long-range dependencies between the image features and the corresponding words in the caption, and they can do so more efficiently than RNNs.

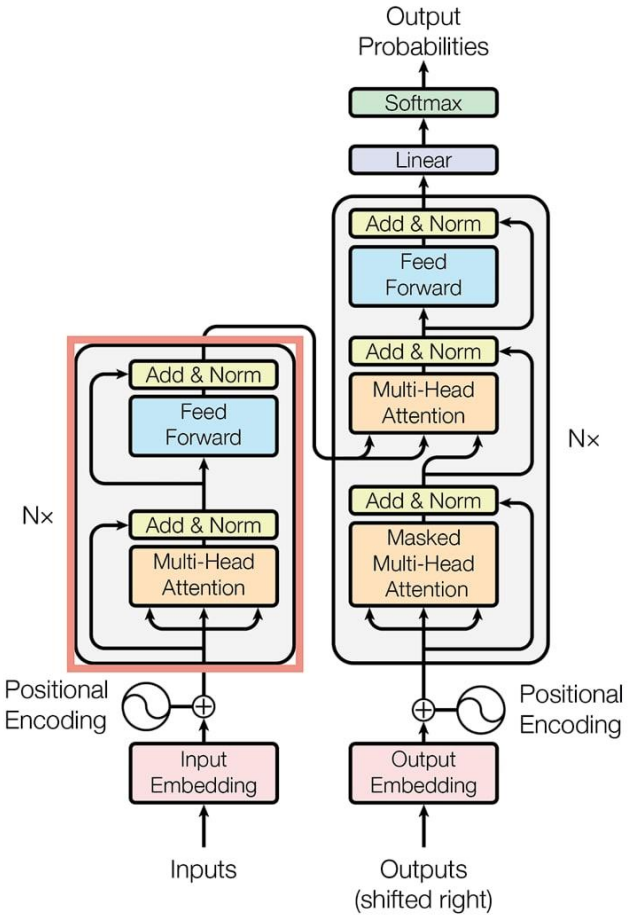


Fig 1.2

CHAPTER-2
Software Development Life Cycle

From quality, accuracy and precision, the Software Development Life cycle acts as a methodical, systematic process for building software or a mobile application. Regarded as an industry benchmark, SDLC basically ensures the quality part of work and emphasizes smooth flow and correctness of the product. The end results down the SDLC process is high-quality software as expected by customers. Abiding by the SDLC rules while shaping the product also affects or determines the time and cost taken to complete the project. Due to detailed plan and road map suggested by SDLC, one can easily look ahead and plan and build software of their vision. Each phase of SDLC cycle has its own value and deliverables, which eventually impacts the coming phases.

Importance of Software Development Life Cycle (SDLC)

- It acts as a guide to the project and meet client's objectives.
- It helps in evaluating, scheduling, and estimating deliverables.
- It provides a framework for a standard set of activities.
- It ensures correct and timely delivery to the client.

2.1 Phases of SDLC

Various phases of Software Development Life Cycle (SDLC) are: -

Fig-2.1- Phases of SDLC

1. **Planning And Requirement Analysis**: - Planning is the crucial step in everything and so as in software development. In this same stage, requirement analysis is also

performed by the developers of the organization. This is attained from the inputs from the customers, sales department/market surveys. The information from this analysis forms the building block of a basic project. The quality proof of the project is a result of planning. Thus, in this stage, the basic project is designed with all the available information.

2. **Requirement Analysis: -** In this stage, all the requirements for the target software are specified. These requirements get approval from the customers, market analysts, and stakeholders. This is fulfilled by utilizing SRS. This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.

3. **Designing Architecture:** SRS is a reference for software designers to come out with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS). This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for the development.

4. **Implementation: -** At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

5. **Testing: -** After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC. Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS.

6. **Deployment and Maintenance: -** After detailed testing, the conclusive product is released in phases as per the organization's strategy. Then it is tested in a real industrial environment. Because it is important to ensure its smooth performance. If it performs well, the organization sends out the product. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers.

2.2 SDLC Model Used

There are different models that can be used in the development process of a software. As for this project, waterfall model is used.

The waterfall model is the basic software development life cycle model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model. The classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after the completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus, the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure

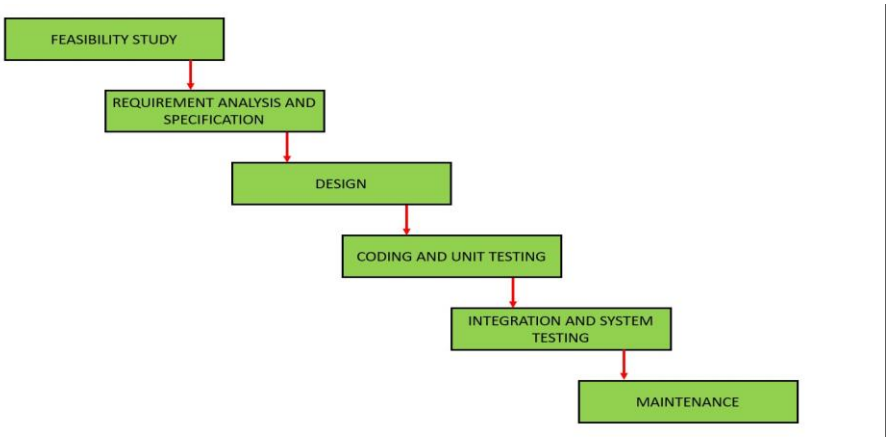


Fig-2.2: diagram of waterfall model

1. Feasibility Study: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software. The feasibility study involves understanding the problem and then determining the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution.

2. Requirements analysis and specification: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

- Requirement gathering and analysis: Firstly, all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete

requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (an inconsistent requirement is one in which some part of the requirement contradicts some other part).

- Requirement specification: These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

3. Design: The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture. A Software Design Document is used to document all this effort (SDD)

4. Coding and Unit testing: In the coding phase software design is translated into source code using any suitable programming language. Thus, each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

5. Integration and System testing: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over several steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.

6.Maintenance: Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software.

2.2.1 Benefits of Waterfall model

- This model is simple to implement also the number of resources that are required for it is minimal.

- The requirements are simple and explicitly declared; they remain unchanged during the entire project development.

- The start and end points for each phase is fixed, which makes it easy to cover progress.
- The release date for the complete product, as well as its final cost, can be determined before development

CHAPTER-3 ANALYSIS

3.1 REQUIREMENT ANALYSIS

A **software requirements specification** (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. A reasonable product delivery demands correct requirements gathering, the efficient examination of requirements gathered, and clear requirement documentation as a precondition. This entire process is known as Requirement **Analysis in Software Development Life Cycle (SDLC).**

Requirement Analysis begins with: Requirement Gathering, which is also known as Elicitation. It is followed by Analyzing the collected requirements to understand the feasibility and correctness of converting the requirements into a possible product. Then, Documenting the gathered requirements.

Requirements analysis process:

The software requirements analysis process involves the following steps/phases:

1- **Eliciting requirements**

The process of gathering requirements by communicating with the customers is known as eliciting requirements.

2- **Analyzing requirements**

This step helps to determine the quality of the requirements. It involves identifying whether the requirements are unclear, incomplete, ambiguous, and contradictory. These issues resolved before moving to the next step.

3- **Requirements modeling**

In Requirements modeling, the requirements are usually documented in different formats such as use cases, user stories, natural-language documents, or process specification.

4- **Review and retrospective**

This step is conducted to reflect on the previous iterations of requirements gathering in a bid to make improvements in the process going forward.
There are several requirements for this project design. The main requirement is that the system should be as easy to use as possible and should allow the user to access all

functionality via app interface and must be designed in a modular fashion. A modular system allows for easier implements as well as easier modification at earlier stage.

3.1.1 HARDWARE REQUIREMENTS

- Processor: Any Processor above 5000 MHz
- RAM: 4 GB(minimum)
- Hard Disk: 250 GB
- GPU
- Output device: High Resolution Monitor.
- A CPU with the Advanced Vector Extensions (AVX) instruction set. In general, any CPU after 2011 will contain this instruction set.

3.1.2 SOFTWARE REQUIREMENTS

- ML notebook IDE (like Kaggle, Google Colab).
- Machine learning libraries.
- TensorFlow.
- Keras.
- Datasets like (COCO2017 and CIFAR-10).

3.2 REQUIREMENT SPECIFICATIONS

3.2.1 FUNCTIONAL REQUIREMENTS

The proposed application should be able to identify input image type. Functional requirements define a function of a system and its components. A function is described as a set of inputs, the behavior, and its outputs. The project is developed in such a way that it requires minimal maintenance. The software used are open source and easy to install. The user can give image input and classify it.

3.2.2 NON-FUNCTIONAL REQUIREMENT: -

Non-functional requirements determine the resources required, time interval, transaction rates, throughput and everything that deals with the performance of the system. Easy maintenance is one among the features that makes this proposal most usable.

3.3 USECASE ANALYSIS

Definition: Use case analysis is a way of gathering information about how a system would interact with users or other system.

Description: Use case analysis, in simple terms, represents the various ways a software would react based upon the input that it receives. A use case analysis is used to design a system from the viewpoint of the end user, the person using the site or software. It is used to determine and convey system behaviour information. The use case analysis attempts to convey information on the system requirements and usage, the role of the user, the system actions in response to the user, and what the user will receive from the system. The use cases are analysed so that they can be converted into more technical requirements for the software developers. Later, when the software is developed, the use case is analyzed to develop the testing scenarios, also referred to as test cases, and so that they can be included in the user documentation.

3.3.1 USECASE DIAGRAM: --

Definition: A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. In this context, the "system" refers to something being developed or operated. Use case diagrams are employed in UML (Unified Modelling Language), a standard notation for the modelling of real-word objects and systems.

3.3.2 Use-Case Description:

A use case is a written description of how users will perform tasks on your website. It outlines, from a user's point of view, a system's behavior as it responds to a request. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled.

Elements of a Use-Case:

Depending on how in depth and complex you want or need to get, use case describe a combination of the following elements:

- **Actor** – anyone or anything that performs a behavior (who is using the system)
- **Primary Actor** – stakeholder who initiates an interaction with the system to achieve a goal.
- **Preconditions** – what must be true or happen before and after the use case runs.
- **Triggers** – this is the event that causes the use case to be initiated.
- **Main success scenarios** [Basic Flow] – use case in which nothing goes wrong.

- **Alternative paths [**Alternative Flow] – these paths are a variation on the main theme. These exceptions are what happen when things go wrong at the system level.

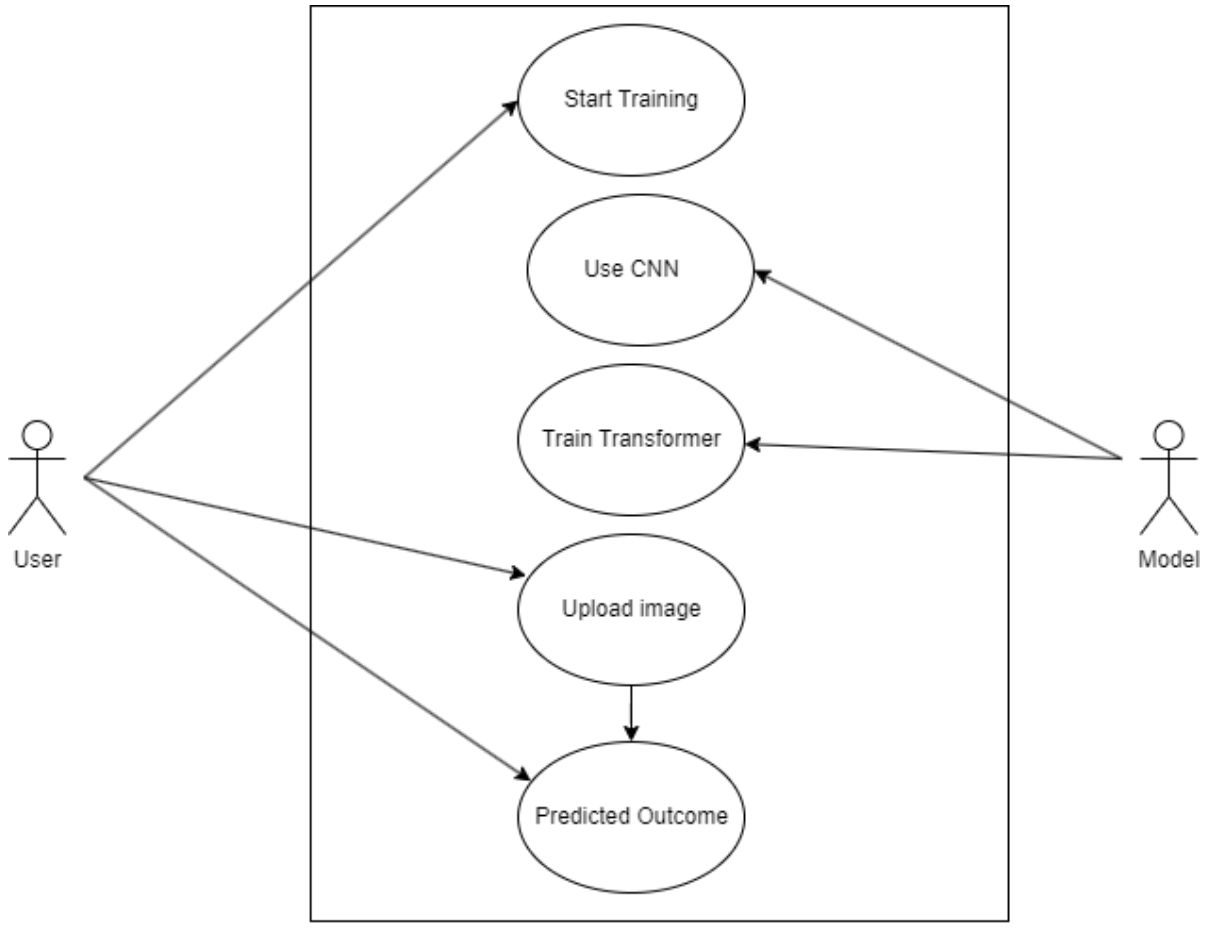


Fig-3.1: Use case diagram

In the above diagram, the user can use any open-source image_caption pair dataset for the training of model. After the tokenized dataset is created, this dataset then passed to the prebuild CNN model(Inception V3) which creates Tensors(Feature vectors), then this tensors passed to the transformer encoder(Acts as a fully connected CNN with self-attention) and create context vectors with low dimension, this vectors then passed to the transformer decoder to generate sequence(Whole sentence) And the training loop goes according to the epochs and the model is ready to give caption of any random image.

3.3.3 Activity Diagram

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning (an initial state) and an end (a final state).

In UML, an activity diagram provides a view of the behaviour of a system by describing the sequence of actions in a process.

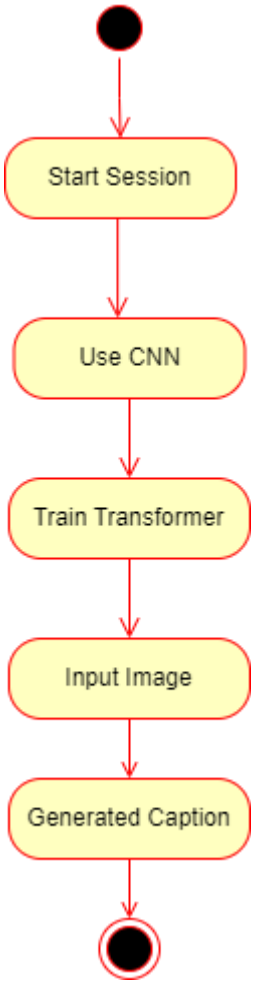


Fig-3.2: Activity Diagram

The Activity Diagram for this project is shown above, Here the process starts by using pre-build CNN model, the output generated by the CNN is taken by the Encoder and its output is taken by the Decoder, this model is trained using the training set after training completes its ready to use .

CHAPTER-4
DESIGN

Once the requirements document for the software to be developed is available, the software design phase begins. While the requirement specification activity deals entirely with the problem domain, design is the first phase of transforming the problem into a solution. In the design phase, the customer and business requirements and technical considerations all come together to formulate a product or a system.

The design process comprises a set of principles, concepts, and practices, which allow a software engineer to model the system or product that is to be built. This model, known as design model, is assessed for quality, and reviewed before a code is generated and tests are conducted.

The design model provides details about software data structures, architecture, interfaces, and components which are required to implement the system. This chapter discusses the design elements that are required to develop a software design model. It also discusses the design patterns and various software design notations used to represent a software design.

4.1 System Flow Diagram

The system flow diagram is one of the graphical representations of the flow of data in a system in software engineering. The diagram consists of several steps that identify where the input is coming to the system and output going out of the system.

With the help of the diagram, it is possible to control the event decisions of the system and how data is flowing to the system. Therefore, the system flow diagram is basically a visual representation of data flow, excluding the minor parts and including the major parts of the system in a sequential manner.

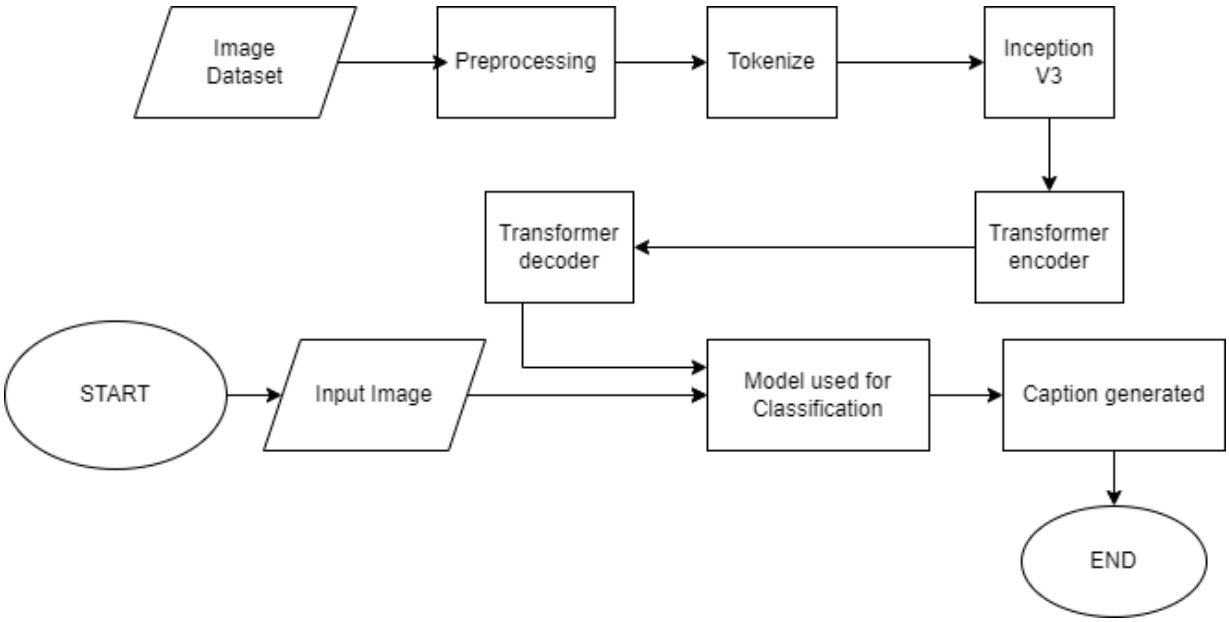


Fig-4.1: System flow diagram

In the above diagram of system flow, the flow starts by the preprocessing of dataset and then tokenization of dataset which then converted to feature vectors by the CNN and Attention mechanism is applied to the vectors, this vectors then taken by the decoder to learn captioning and then the model is trained to used by the user.

4.2 Module Identified

1. Project Image Classifier: – This is the main module which consists of files and other modules which are required for our project. Files such as source code files and dataset modules.
2. Dataset: – This module consists of dataset files which are required for the model training and testing. This consist of ImageNet and COCO 2017 image dataset collections. This module is placed inside the Project Image Classifier module.
3. Dependencies: -- like Tensorflow, keras, pandas, numpy.

4.3 Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

high-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams).

Sequence Diagrams show elements as they interact over time and they are organized according to object (horizontally) and time (vertically).

The sequence diagram for this project is shown below. The user request for the Jupyter notebook session to start in kaggel environment, all the preprocessing, vectorization and training completes and then the random image captioner runs to give the caption of a random image from the validation set also the user can upload a image link to get its caption.

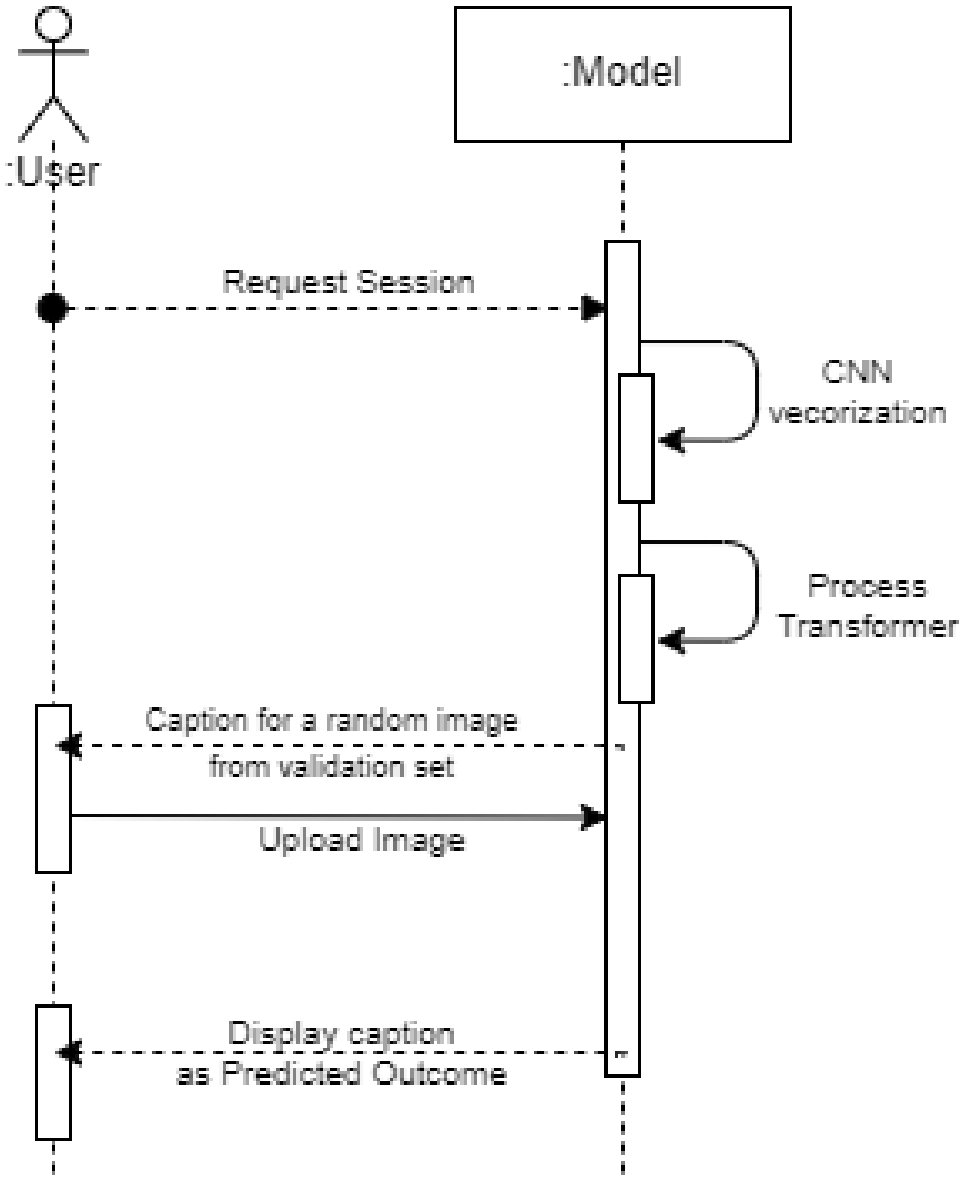


Fig-4.4: Sequence diagram

Chapter 5

IMPLEMENTATION & TESTING

5.1 Implementation

The implementation of an Deep Learning model as a minor project can be broken down into several components:

Planning: Define the scope of the project, including the target audience and other key features.

Design: Designing the Architecture of the model.

Development: Python language is used for programming, Jupyter notebook Environment by Kaggle is used, Tensorflow framework for implementing deep learning model

Testing: A Validation set is used to test the models accuracy.

5.2 Platform used:

5.2.1 HARDWARE PLATFORM:

Workplace Application - Laptop

GPU – GPU P100

Monitor- 14'' colour monitor.

RAM – 10 GB

5.2.2 SOFTWARE PLATFORM

Notebook environment – Kaggle jupyter notebook

Language – Python

Tool – Tensorflow, COCO 2017 dataset, Transformer

5.3 Testing Strategy Adapted

Software testing is the process of evaluating a software item to detect differences between given input and expected output. Also, to assess the feature of A software item. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words, software testing is a verification and validation process.

There are various types of Testing Techniques:

Functional Testing types include:

- Unit Testing
- Integration Testing
- System Testing
- Interface Testing
- Beta/Acceptance Testing

Non-functional Testing types include:

- Performance Testing
- Load Testing
- Stress Testing

5.4 Piece of Code

```
import tensorflow as tf
import os
import json
import pandas as pd
import re
import numpy as np
import time
import matplotlib.pyplot as plt
import collections
import random
import requests
import json
from math import sqrt
from PIL import Image
from tqdm.auto import tqdm

BASE_PATH = '../input/coco-2017-dataset/coco2017'

with open(f'{BASE_PATH}/annotations/captions_train2017.json', 'r') as f:
    data = json.load(f)
    data = data['annotations']

img_cap_pairs = []

for sample in data:
    img_name = '%012d.jpg' % sample['image_id']
    img_cap_pairs.append([img_name, sample['caption']])

captions = pd.DataFrame(img_cap_pairs, columns=['image', 'caption'])
captions['image'] = captions['image'].apply(
    lambda x: f'{BASE_PATH}/train2017/{x}'
)
captions = captions.sample(70000)
captions = captions.reset_index(drop=True)
captions.head()

def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
```

```
    text = re.sub('\s+', ' ', text)
    text = text.strip()
    text = '[start] ' + text + ' [end]'
    return text

captions['caption'] = captions['caption'].apply(preprocess)
captions.head()

MAX_LENGTH = 40
VOCABULARY_SIZE = 15000
BATCH_SIZE = 64
BUFFER_SIZE = 1000
EMBEDDING_DIM = 512
UNITS = 512
EPOCHS = 5

tokenizer = tf.keras.layers.TextVectorization(
    max_tokens=VOCABULARY_SIZE,
    standardize=None,
    output_sequence_length=MAX_LENGTH)

tokenizer.adapt(captions['caption'])
tokenizer.vocabulary_size()
import pickle
pickle.dump(tokenizer.get_vocabulary(), open('vocab_coco.file', 'wb'))
import pickle

word2idx = tf.keras.layers.StringLookup(
    mask_token="",
    vocabulary=tokenizer.get_vocabulary())

idx2word = tf.keras.layers.StringLookup(
    mask_token="",
    vocabulary=tokenizer.get_vocabulary(),
    invert=True)

img_to_cap_vector = collections.defaultdict(list)
for img, cap in zip(captions['image'], captions['caption']):
    img_to_cap_vector[img].append(cap)

img_keys = list(img_to_cap_vector.keys())
```

```
random.shuffle(img_keys)

slice_index = int(len(img_keys)*0.8)
img_name_train_keys, img_name_val_keys = (img_keys[:slice_index],
                                          img_keys[slice_index:])

train_imgs = []
train_captions = []
for imgt in img_name_train_keys:
    capt_len = len(img_to_cap_vector[imgt])
    train_imgs.extend([imgt] * capt_len)
    train_captions.extend(img_to_cap_vector[imgt])

val_imgs = []
val_captions = []
for imgv in img_name_val_keys:
    capv_len = len(img_to_cap_vector[imgv])
    val_imgs.extend([imgv] * capv_len)
    val_captions.extend(img_to_cap_vector[imgv])

def load_data(img_path, caption):
    img = tf.io.read_file(img_path)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.keras.layers.Resizing(299, 299)(img)
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    caption = tokenizer(caption)
    return img, caption

train_dataset = tf.data.Dataset.from_tensor_slices(
    (train_imgs, train_captions))

train_dataset = train_dataset.map(
    load_data, num_parallel_calls=tf.data.AUTOTUNE
    ).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

val_dataset = tf.data.Dataset.from_tensor_slices(
    (val_imgs, val_captions))

val_dataset = val_dataset.map(
    load_data, num_parallel_calls=tf.data.AUTOTUNE
    ).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

```
image_augmentation = tf.keras.Sequential(
    [
        tf.keras.layers.RandomFlip("horizontal"),
        tf.keras.layers.RandomRotation(0.2),
        tf.keras.layers.RandomContrast(0.3),
    ]
)

def CNN_Encoder():
    inception_v3 = tf.keras.applications.InceptionV3(
        include_top=False,
        weights='imagenet'
    )

    output = inception_v3.output
    output = tf.keras.layers.Reshape(
        (-1, output.shape[-1]))(output)

    cnn_model = tf.keras.models.Model(inception_v3.input, output)
    return cnn_model

class TransformerEncoderLayer(tf.keras.layers.Layer):

    def __init__(self, embed_dim, num_heads):
        super().__init__()
        self.layer_norm_1 = tf.keras.layers.LayerNormalization()
        self.layer_norm_2 = tf.keras.layers.LayerNormalization()
        self.attention = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.dense = tf.keras.layers.Dense(embed_dim, activation="relu")


    def call(self, x, training):
        x = self.layer_norm_1(x)
        x = self.dense(x)

        attn_output = self.attention(
            query=x,
            value=x,
            key=x,
```

```
            attention_mask=None,
            training=training
        )

        x = self.layer_norm_2(x + attn_output)
        return x


class Embeddings(tf.keras.layers.Layer):

    def __init__(self, vocab_size, embed_dim, max_len):
        super().__init__()
        self.token_embeddings = tf.keras.layers.Embedding(
            vocab_size, embed_dim)
        self.position_embeddings = tf.keras.layers.Embedding(
            max_len, embed_dim, input_shape=(None, max_len))


    def call(self, input_ids):
        length = tf.shape(input_ids)[-1]
        position_ids = tf.range(start=0, limit=length, delta=1)
        position_ids = tf.expand_dims(position_ids, axis=0)

        token_embeddings = self.token_embeddings(input_ids)
        position_embeddings = self.position_embeddings(position_ids)

        return token_embeddings + position_embeddings

class TransformerDecoderLayer(tf.keras.layers.Layer):

    def __init__(self, embed_dim, units, num_heads):
        super().__init__()
        self.embedding = Embeddings(
            tokenizer.vocabulary_size(), embed_dim, MAX_LENGTH)

        self.attention_1 = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim, dropout=0.1
        )
        self.attention_2 = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim, dropout=0.1
        )
```

```
        self.layernorm_1 = tf.keras.layers.LayerNormalization()
        self.layernorm_2 = tf.keras.layers.LayerNormalization()
        self.layernorm_3 = tf.keras.layers.LayerNormalization()

        self.ffn_layer_1 = tf.keras.layers.Dense(units, activation="relu")
        self.ffn_layer_2 = tf.keras.layers.Dense(embed_dim)

        self.out = tf.keras.layers.Dense(tokenizer.vocabulary_size(),
activation="softmax")

        self.dropout_1 = tf.keras.layers.Dropout(0.3)
        self.dropout_2 = tf.keras.layers.Dropout(0.5)


    def call(self, input_ids, encoder_output, training, mask=None):
        embeddings = self.embedding(input_ids)

        combined_mask = None
        padding_mask = None

        if mask is not None:
            causal_mask = self.get_causal_attention_mask(embeddings)
            padding_mask = tf.cast(mask[:, :, tf.newaxis], dtype=tf.int32)
            combined_mask = tf.cast(mask[:, tf.newaxis, :], dtype=tf.int32)
            combined_mask = tf.minimum(combined_mask, causal_mask)

        attn_output_1 = self.attention_1(
            query=embeddings,
            value=embeddings,
            key=embeddings,
            attention_mask=combined_mask,
            training=training
        )

        out_1 = self.layernorm_1(embeddings + attn_output_1)

        attn_output_2 = self.attention_2(
            query=out_1,
            value=encoder_output,
            key=encoder_output,
```

```
            attention_mask=padding_mask,
            training=training
        )

        out_2 = self.layernorm_2(out_1 + attn_output_2)

        ffn_out = self.ffn_layer_1(out_2)
        ffn_out = self.dropout_1(ffn_out, training=training)
        ffn_out = self.ffn_layer_2(ffn_out)

        ffn_out = self.layernorm_3(ffn_out + out_2)
        ffn_out = self.dropout_2(ffn_out, training=training)
        preds = self.out(ffn_out)
        return preds


    def get_causal_attention_mask(self, inputs):
        input_shape = tf.shape(inputs)
        batch_size, sequence_length = input_shape[0], input_shape[1]
        i = tf.range(sequence_length)[:, tf.newaxis]
        j = tf.range(sequence_length)
        mask = tf.cast(i >= j, dtype="int32")
        mask = tf.reshape(mask, (1, input_shape[1], input_shape[1]))
        mult = tf.concat(
            [tf.expand_dims(batch_size, -1), tf.constant([1, 1], dtype=tf.int32)],
            axis=0
        )
        return tf.tile(mask, mult)

class ImageCaptioningModel(tf.keras.Model):

    def __init__(self, cnn_model, encoder, decoder, image_aug=None):
        super().__init__()
        self.cnn_model = cnn_model
        self.encoder = encoder
        self.decoder = decoder
        self.image_aug = image_aug
        self.loss_tracker = tf.keras.metrics.Mean(name="loss")
        self.acc_tracker = tf.keras.metrics.Mean(name="accuracy")
```

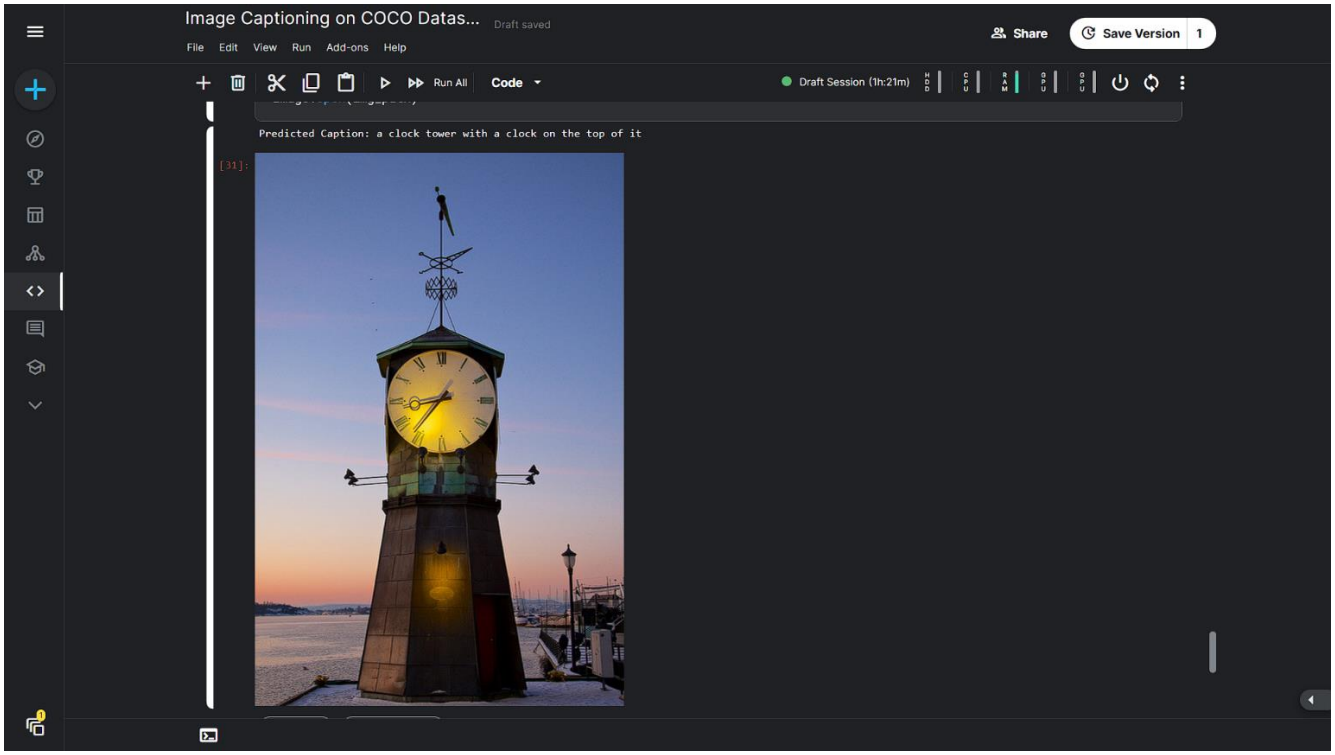
```
def calculate_loss(self, y_true, y_pred, mask):
    loss = self.loss(y_true, y_pred)
    mask = tf.cast(mask, dtype=loss.dtype)
    loss *= mask
    return tf.reduce_sum(loss) / tf.reduce_sum(mask)


def calculate_accuracy(self, y_true, y_pred, mask):
    accuracy = tf.equal(y_true, tf.argmax(y_pred, axis=2))
    accuracy = tf.math.logical_and(mask, accuracy)
    accuracy = tf.cast(accuracy, dtype=tf.float32)
    mask = tf.cast(mask, dtype=tf.float32)
    return tf.reduce_sum(accuracy) / tf.reduce_sum(mask)


def compute_loss_and_acc(self, img_embed, captions, training=True):
    encoder_output = self.encoder(img_embed, training=True)
    y_input = captions[:, :-1]
    y_true = captions[:, 1:]
    mask = (y_true != 0)
    y_pred = self.decoder(
        y_input, encoder_output, training=True, mask=mask
    )
    loss = self.calculate_loss(y_true, y_pred, mask)
    acc = self.calculate_accuracy(y_true, y_pred, mask)
    return loss, acc


def train_step(self, batch):
    imgs, captions = batch

    if self.image_aug:
        imgs = self.image_aug(imgs)

    img_embed = self.cnn_model(imgs)

    with tf.GradientTape() as tape:
        loss, acc = self.compute_loss_and_acc(
            img_embed, captions
        )
```

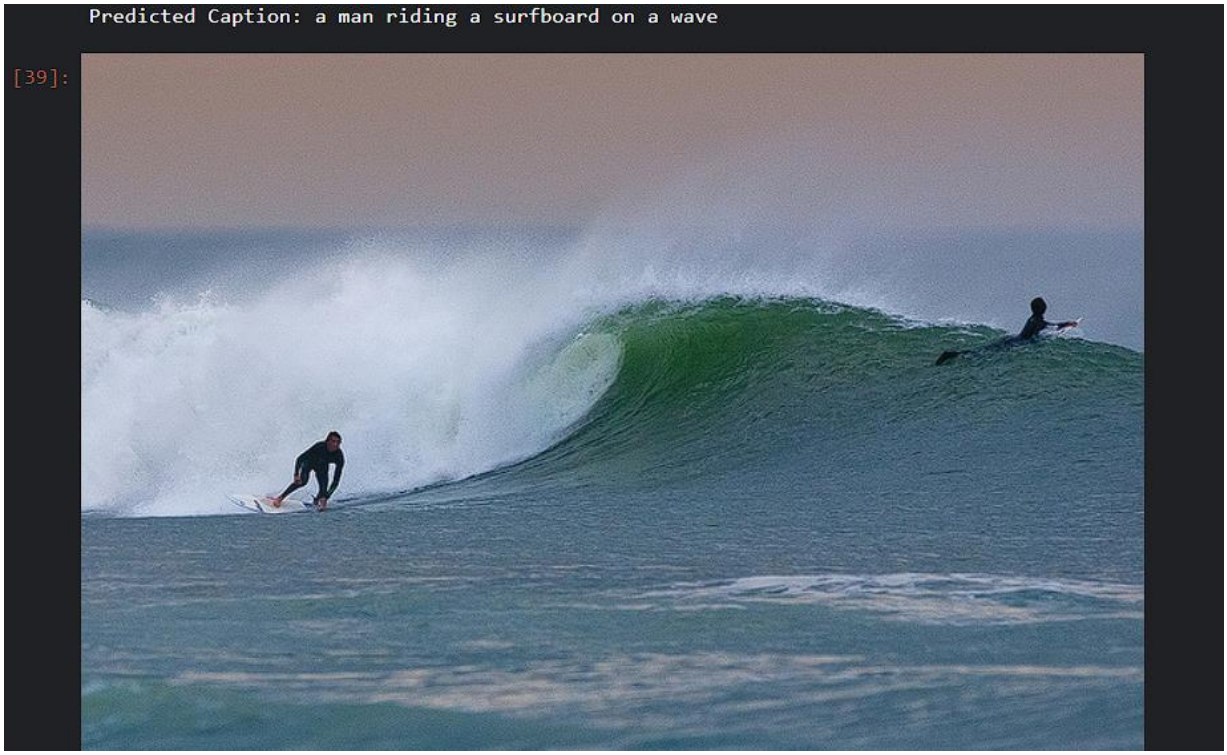
```
        train_vars = (
            self.encoder.trainable_variables + self.decoder.trainable_variables
        )
        grads = tape.gradient(loss, train_vars)
        self.optimizer.apply_gradients(zip(grads, train_vars))
        self.loss_tracker.update_state(loss)
        self.acc_tracker.update_state(acc)

        return {"loss": self.loss_tracker.result(), "acc": self.acc_tracker.result()}


    def test_step(self, batch):
        imgs, captions = batch

        img_embed = self.cnn_model(imgs)

        loss, acc = self.compute_loss_and_acc(
            img_embed, captions, training=False
        )

        self.loss_tracker.update_state(loss)
        self.acc_tracker.update_state(acc)

        return {"loss": self.loss_tracker.result(), "acc": self.acc_tracker.result()}

    @property
    def metrics(self):
        return [self.loss_tracker, self.acc_tracker]

encoder = TransformerEncoderLayer(EMBEDDING_DIM, 1)
decoder = TransformerDecoderLayer(EMBEDDING_DIM, UNITS, 8)

cnn_model = CNN_Encoder()
caption_model = ImageCaptioningModel(
    cnn_model=cnn_model,          encoder=encoder,          decoder=decoder,
image_aug=image_augmentation,
)

cross_entropy = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=False, reduction="none"
)
```

```
early_stopping = tf.keras.callbacks.EarlyStopping(patience=3,
restore_best_weights=True)

caption_model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=cross_entropy
)

history = caption_model.fit(
    train_dataset,
    epochs=EPOCHS,
    validation_data=val_dataset,
    callbacks=[early_stopping]
)

def load_image_from_path(img_path):
    img = tf.io.read_file(img_path)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.keras.layers.Resizing(299, 299)(img)
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    return img


def generate_caption(img_path, add_noise=False):
    img = load_image_from_path(img_path)

    if add_noise:
        noise = tf.random.normal(img.shape)*0.1
        img = img + noise
        img = (img - tf.reduce_min(img))/(tf.reduce_max(img) - tf.reduce_min(img))

    img = tf.expand_dims(img, axis=0)
    img_embed = caption_model.cnn_model(img)
    img_encoded = caption_model.encoder(img_embed, training=False)

    y_inp = '[start]'
    for i in range(MAX_LENGTH-1):
        tokenized = tokenizer([y_inp])[:, :-1]
        mask = tf.cast(tokenized != 0, tf.int32)
        pred = caption_model.decoder(
```

```
            tokenized, img_encoded, training=False, mask=mask)

        pred_idx = np.argmax(pred[0, i, :])
        pred_idx = tf.convert_to_tensor(pred_idx)
        pred_word = idx2word(pred_idx).numpy().decode('utf-8')
        if pred_word == '[end]':
            break

        y_inp += ' ' + pred_word

    y_inp = y_inp.replace('[start] ', '')
    return y_inp

idx = random.randrange(0, len(captions))
img_path = captions.iloc[idx].image

pred_caption = generate_caption(img_path)
print('Predicted Caption:', pred_caption)
print()
Image.open(img_path)


img_url = "Img_link"
im = Image.open(requests.get(img_url, stream=True).raw)
im = im.convert('RGB')
im.save('tmp.jpg')

pred_caption = generate_caption('tmp.jpg', add_noise=False)
print('Predicted Caption:', pred_caption)
print()
im
```

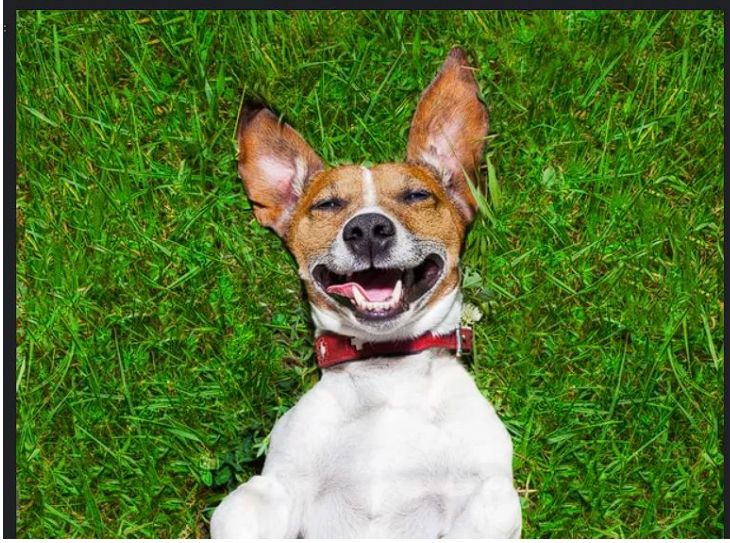
5.2 Project Output Screenshots:





```
img_url = "https://st.depositphotos.com/1146092/4811/i/450/depositphotos_48114905-stock-photo-very-funny-dog.jpg"

im = Image.open(requests.get(img_url, stream=True).raw)
im = im.convert('RGB')
im.save('tmp.jpg')

pred_caption = generate_caption('tmp.jpg', add_noise=False)
print('Predicted Caption:', pred_caption)
print()
im
```

```
Predicted Caption: a dog is laying on a lush green field
```



CHAPTER 6 CONCLUSION

6.1 Important Features:

- In this project, I expressed my idea to automatically classify objects in images using the latest deep learning neural network frameworks
- Can be used to classify distorted and blurry images.
- Feature extraction and model classification process is very fast.

6.2 Limitation:

- The proposed system will work only on the colored images.
- Cannot recognize and extract text from the image

6.3 Future Scope:

In future, I want to work on greyscale images that are given as input to the system. I would like to extend the system so that the accuracy of the predicted test image would be literally high. The above system is for only single label classification. Therefore, in future, I want to work for the multi label classification.

REFERENCES

- https://github.com/nicknochnack
- https://github.com/kying18
- https://www.kaggle.com/ - Open-source Community for data science
- https://www.tensorflow.org/community
- https://github.com/vorisjonov
- https://blog.paperspace.com/faster-r-cnn-explained-object-detection/
- https://www.ibm.com/cloud/learn/machine-learning
- https://www.v7labs.com/blog/yolo-object-detection/
- https://www.mygreatlearning.com/blog/object-detection-using-tensorflow/
- https://www.youtube.com/playlist?list=PLWKjhJtqVAblStefaz_YOVpDWqcRScc2s
- https://www.youtube.com/@murtazasworkshop/videos - for computer vision projects
- https://www.youtube.com/playlist?list=PLBZBJbE_rGRWeh5mIBhDhhDwSEDxogDg
- https://www.youtube.com/c/Fireship/featured - • https://paperswithcode.com/task/image-classification
- https://github.com/jgraph - Desktop application I used to create UML diagrams.