Final Project Report

CSE 676B: Deep Learning

Team 19:

Name	UBITName	UBITNumber	Contribution
Suchit Gupta	suchitni	50518842	33%
Sonal Chaudhari	sonalash	50538663	33%
Rahul Kudva	rahulkar	50544982	34%

Title: Image Caption Generation

Introduction

Image captioning is the process of generating textual descriptions for images automatically. It combines computer vision techniques, which extract visual features from the image, with natural language processing (NLP) models to generate descriptive captions. The goal is to create captions that accurately describe the content and context of the image, simulating the way humans perceive and describe visual scenes.

Main Purpose

The need to solve image captioning arises from several reasons. Firstly, it enhances accessibility for visually impaired individuals by providing textual descriptions of images on the web or in documents. Secondly, it aids in image retrieval and organization, making it easier to search for specific images based on their content. Additionally, image captioning has applications in content generation for social media, advertising, and journalism. It also facilitates human-computer interaction, enabling systems to better understand and respond to visual content in a manner that is more intuitive for users. Overall, solving the problem of image captioning not only advances computer vision and NLP technologies but also improves the usability and accessibility of visual information in various domains.

Overview

The project aims to address the challenge of accurately and contextually generating captions for images, a task that involves both understanding the content of images and generating relevant textual descriptions. This problem is interesting because it sits at the

intersection of computer vision and natural language processing (NLP), requiring the model to not only accurately interpret visual data but also to generate coherent and contextually appropriate text. Improving image caption detection can significantly enhance accessibility, searchability, and user interaction in digital media platforms.

Dataset

For conceptualizing, building and training the model for this project, we are using Flickr 8k dataset.

The Flickr 8k dataset is a widely used benchmark dataset in the field of image captioning. It consists of 8,000 images sourced from the photo-sharing website Flickr. Each image in the dataset is given five different captions, providing diverse textual descriptions of the visual content. The dataset covers a wide range of scenes, objects, and activities, making it suitable for training and evaluating image captioning models acro

ss various domains.

Flickr8k serves as a standard dataset for researchers to develop and benchmark image captioning algorithms. It allows for the evaluation of the quality and diversity of generated captions, as well as the ability of models to understand and describe the content of different images accurately.

Sample Images:

man on a bicycle riding on only one wheel .

asian man in orange hat is popping a wheelie on his bike .

a man on a bicycle is on only the back wheel .

a man is doing a wheelie on a mountain bike .

a man does a wheelie on his bicycle on the sidewalk .

five people are sitting together in the snow .

five children getting ready to sled .

a group of people sit in the snow overlooking a mountain scene .

a group of people sit atop a snowy mountain .

a group is sitting around a snowy crevasse .

a white crane stands tall as it looks out upon the ocean .

a water bird standing at the ocean 's edge .

a tall bird is standing on the sand beside the ocean .

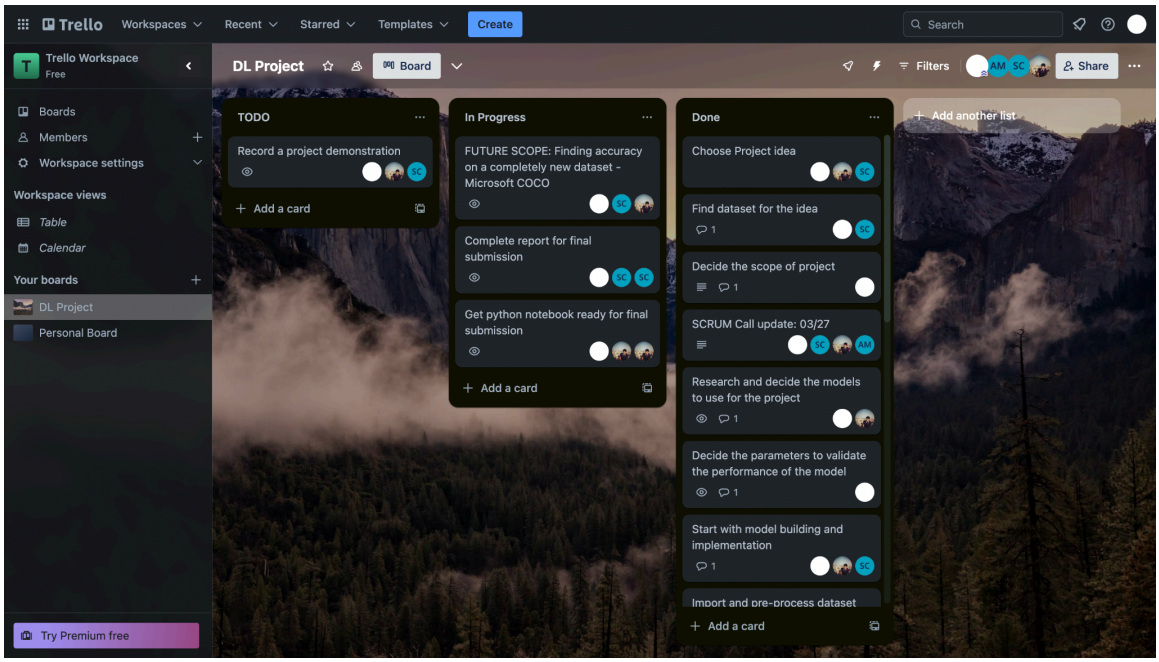
a large bird stands in the water on the beach .

a grey bird stands majestically on a beach while waves roll in .

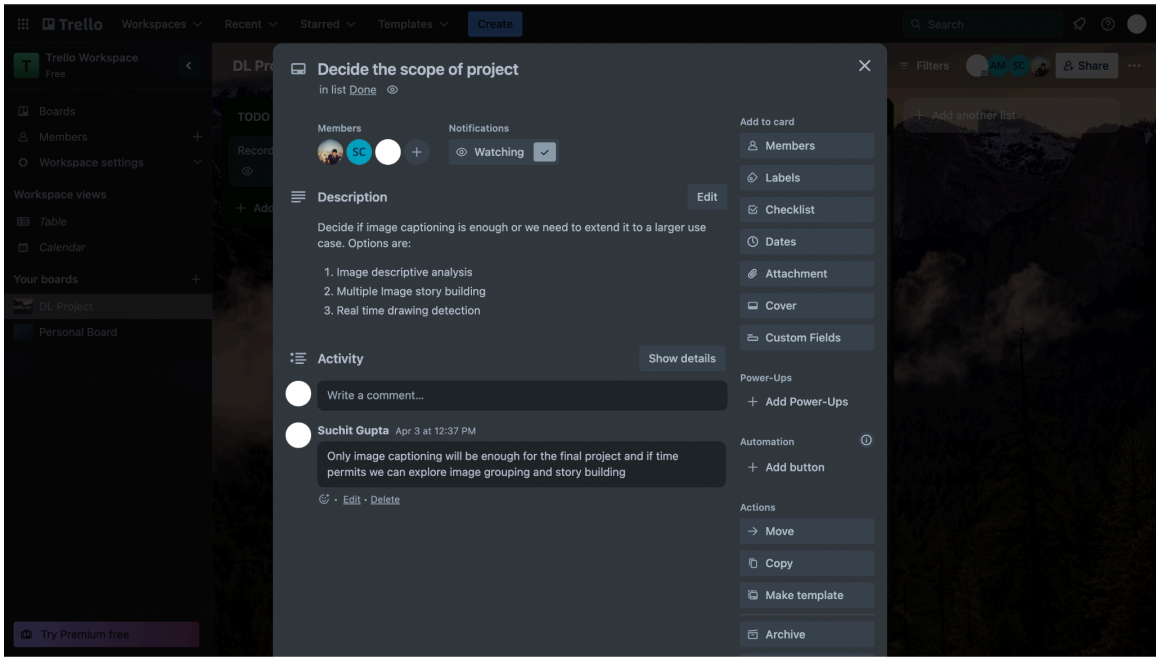
Project Management

We used Trello for managing our project and keep track of tasks and updates every week. We divided the tasks effectively amongst the 3 team members to ensure parallel working with minimal internal blockers.

View of the Trello Board



Each ticket as a more detail description of what needs to be done and time to time updates when a progress is made on the ticket



You can request access to the board using the UBIT name mentioned at the start of the report. To access the Trello board here.

Implementation

Overview

We performed the following steps for implementation of image captioning

a. Setting up the project
b. Caption preprocessing
c. VGG 16 Implementation: Image segmentation
d. Caption preprocessing for LSTM Model
e. LSTM Model Training
f. Performance evaluation

1. Setting up the project

We start by importing/unzipping our Flickr 8k dataset

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/flickr8k/Flickr8k/Flickr8k.token.txt
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/3226254560_2f8ac147ea.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/214543992_ce6c0d9f9b.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/2366643786_9c9a830db8.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/3368819708_0bfa0808f8.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/2190227737_6e0bde2623.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/2752809449_632cd991b3.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/3097776588_312932e438.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/1206506157_c7956accd5.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/1319634306_816f21677f.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/2465218087_fca77998c6.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/3351493005_6e5030f596.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/2949337912_beba55698b.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/534886684_a6c9f40fa1.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/3543600125_223747ef4c.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/2174206711_11cb712a8d.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/280706862_14c30d734a.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/2100735137_05c6079537.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/3131632154_098f86f4cb.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/2236016316_f476cbbf06.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/3335375223_b4da8df523.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/929679367_ff8c7df2ee.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/241347214_5f19e7998c.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/2998024845_1529c11694.jpg
/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset/1317292658_ba29330a0b.jpg
```

Following the successful set up/extraction of dataset, we import all the required packages.

```
import os
import time
import progressbar
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras

import sys
import warnings
import numpy as np
import pandas as pd
from collections import Counter
from keras.preprocessing.image import load_img
from nltk.tokenize import word_tokenize
from keras.preprocessing.image import load_img, img_to_array
from IPython.display import display
from PIL import Image
import string
import re
from tensorflow.python.keras import backend as K

from keras.applications.vgg16 import preprocess_input
from collections import OrderedDict
from tqdm.notebook import tqdm
```

Next we set up settings for out notebook like ignoring warnings and tensorflow to optimally use GPU

```
warnings.filterwarnings("ignore")

# Configure TensorFlow session
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True  # dynamically grow the memory used on the GPU
sess = tf.compat.v1.Session(config=config)
K.set_session(sess)
```

2. Caption preprocessing

We started by importing the captions counting the total number of images in the dataset

```
image_caption_folder_path = "/kaggle/input/flickr8k/Flickr8k/Flickr8k.token.txt"

image_folder_path = "/kaggle/input/flickr8k/Flickr8k/Flicker8k_Dataset"

jpgs = os.listdir(image_folder_path)
print("The number of jpg flies in Flicker 8k: {}".format(len(jpgs)))
```

```
The number of jpg flies in Flicker 8k: 8091
```

Next, we extracted the captions from text file, split the file name in actual name and index of caption and the caption itself.

```
#Finding the captions for each image.
file = open(image_caption_folder_path,'r', encoding='utf8')
caption_file = file.read()
file.close()


datatxt = []
for line in caption_file.split('\n'):
    col = line.split('\t')
    if len(col) == 1:
        continue
    image_id, image_index = col[0].split("#")
    image_caption = col[1].lower()
    datatxt.append([image_id, image_index, image_caption])

df_txt = pd.DataFrame(datatxt,columns=["filename","index","caption"])

unique_filenames = np.unique(df_txt.filename.values)
print("The number of unique file ids : {}".format(len(unique_filenames)))
df_txt.head(10)
```

```
The number of unique file ids : 8092
```

	filename	index	caption
0	1000268201_693b08cb0e.jpg	0	a child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	1	a girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	2	a little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	3	a little girl climbing the stairs to her playh...
4	1000268201_693b08cb0e.jpg	4	a little girl in a pink dress going into a woo...
5	1001773457_577c3a7d70.jpg	0	a black dog and a spotted dog are fighting
6	1001773457_577c3a7d70.jpg	1	a black dog and a tri-colored dog playing with...
7	1001773457_577c3a7d70.jpg	2	a black dog and a white dog with brown spots a...
8	1001773457_577c3a7d70.jpg	3	two dogs of different breeds looking at each o...
9	1001773457_577c3a7d70.jpg	4	two dogs on pavement moving toward each other .

In order to manually check the images and their corresponding captions we display a sample set of the images

```
npic = 5 # Displaying 5 images from the dataset
target_size = (224, 224, 3)
count = 1
fig = plt.figure(figsize=(10,15))

for image_id in unique_filenames[-5:]:
    filename = image_folder_path + '/' + image_id
    captions = list(df_txt["caption"].loc[df_txt["filename"] == image_id].values)
    image_load = load_img(filename, target_size=target_size)

    ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
    ax.imshow(image_load)
    count += 1

    ax = fig.add_subplot(npic,2,count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
    ax.set_ylim(0,len(captions))
    for i, caption in enumerate(captions):
        ax.text(0,i,caption,fontsize=10)
    count += 1
plt.show()
```

man on a bicycle riding on only one wheel .

asian man in orange hat is popping a wheelie on his bike .

a man on a bicycle is on only the back wheel .

a man is doing a wheelie on a mountain bike .

a man does a wheelie on his bicycle on the sidewalk .

five people are sitting together in the snow .

five children getting ready to sled .

a group of people sit in the snow overlooking a mountain scene .

a group of people sit atop a snowy mountain .

a group is sitting around a snowy crevasse .

a white crane stands tall as it looks out upon the ocean .

a water bird standing at the ocean 's edge .

a tall bird is standing on the sand beside the ocean .

a large bird stands in the water on the beach .

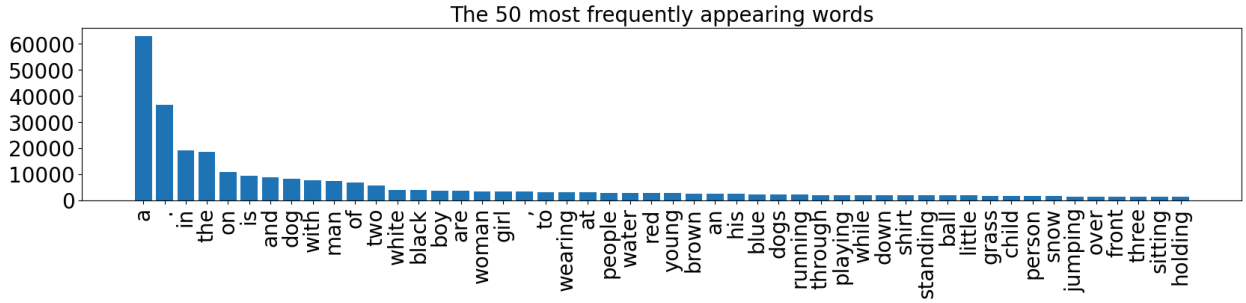
a grey bird stands majestically on a beach while waves roll in .

We verify that each image has 5 captions and captions encompass a lot of different but similar perspectives to look at the image. We will use these captions to find the BLEU score of the captions that we predict.

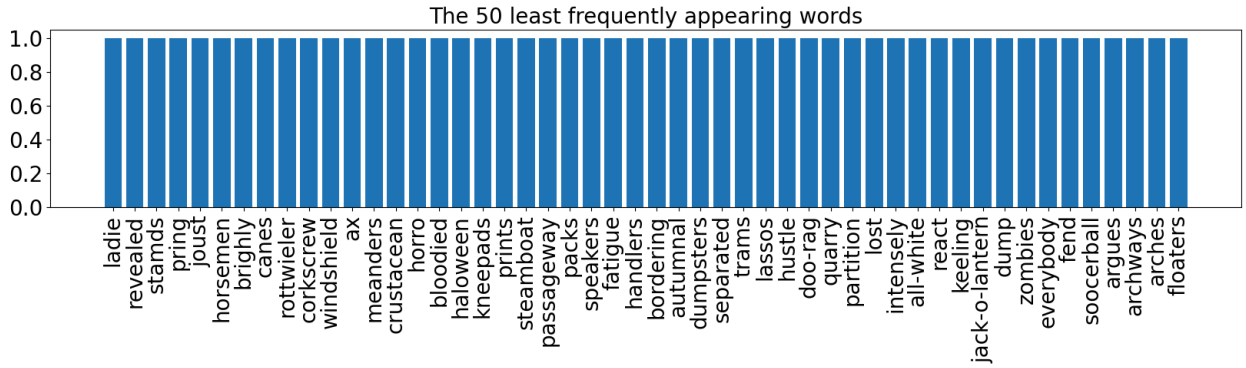
Next, we find some more information related to the captions of the images. We count the occurrence of every word in the caption to verify if the dataset is not biased and remove any existing ones from the get go.

```python
# Counts the occurance of all words in captions
def df_word(df_txt):
    vocabulary = []
    for i in range(len(df_txt)):
        temp=df_txt.iloc[i,2]
        vocabulary.extend(temp.split())
    print('Vocabulary Size: %d' % len(set(vocabulary)))
    ct = Counter(vocabulary)
    dfword = pd.DataFrame({"word":list(ct.keys()),"count":list(ct.values())})
    dfword = dfword.sort_values("count",ascending=False)
    dfword = dfword.reset_index()[["word","count"]]
    return(dfword)
```

We look at the 50 most frequently appearing words in the captions



Similarly, we look at the 50 least frequently appearing words in the captions



We observe that most commonly used words are generally stop words like articles and conjunctions. We also find that the word count includes punctuation that we will need to filter out.

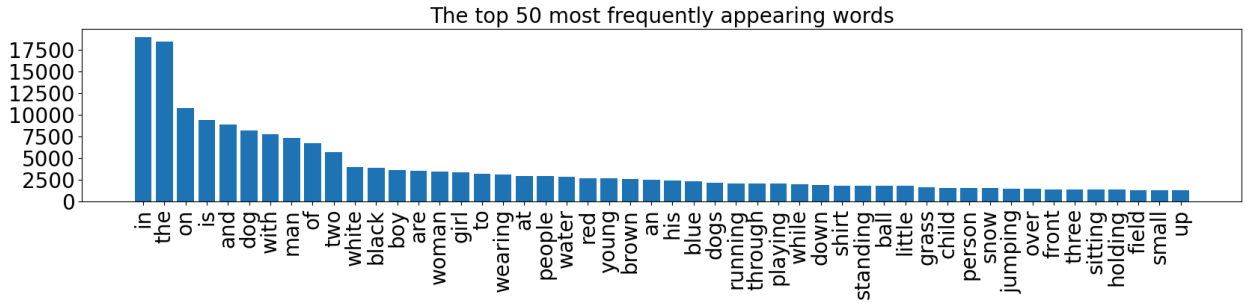
Hence, we clean our captions by

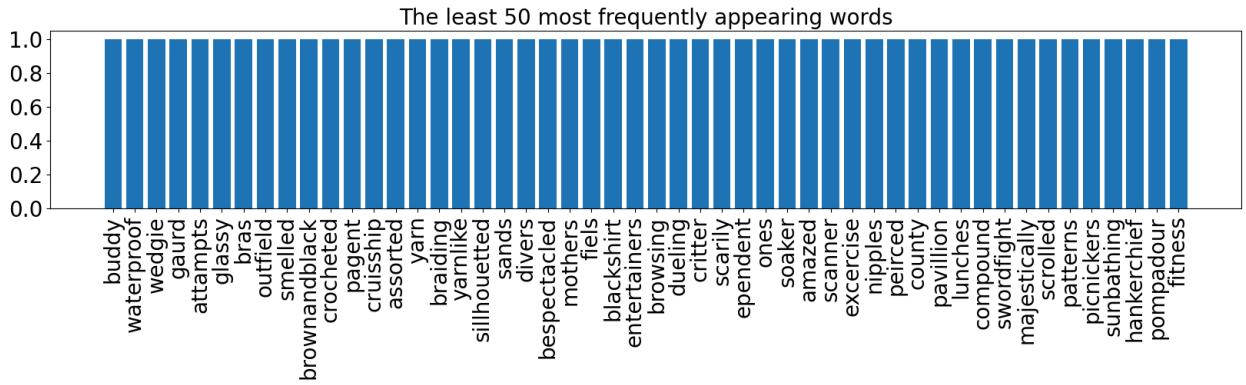
1. Removing punctuations
2. Removing single characters
3. Removing numbers

```python
def remove_punctuation(text_original):
    return text_original.translate(str.maketrans('','',string.punctuation))

def remove_single_character(text):
    words = text.split()

    filtered_words = [word for word in words if len(word) > 1]

    new_sentence = ' '.join(filtered_words)
    return new_sentence

def remove_numeric(text):
    return re.sub(r'\d+', '', text)

def text_clean(text_original):
    text = remove_punctuation(text_original)
    text = remove_single_character(text)
    text = remove_numeric(text)
    return text

with progressbar.ProgressBar(max_value=len(df_txt.caption.values)) as bar:
    for i, caption in enumerate(df_txt.caption.values):
        newcaption = text_clean(caption)
        df_txt["caption"].iloc[i] = newcaption
        bar.update(i)
```

```
100% (40460 of 40460) |##################| Elapsed Time: 0:00:15 Time:  0:00:150001
```

After cleaning the captions dataset, we again find the 50 most frequently and least frequently appearing words.





Finally, we add startseq and endseq tags to the start and the end of the caption as delimiters so that the model understands start and end of variable length captions

```
#adding start and end sequence
from copy import copy
def add_start_end_seq_token(captions):
    caps = []
    for txt in captions:
        txt = 'startseq ' + txt + ' endseq'
        caps.append(txt)
    return(caps)
df_txt0 = copy(df_txt)
df_txt0["caption"] = add_start_end_seq_token(df_txt["caption"])
# df_txt0.head(5)
# del df_txt
df_txt0.head()
```

	filename	index	caption
0	1000268201_693b08cb0e.jpg	0	startseq child in pink dress is climbing up se...
1	1000268201_693b08cb0e.jpg	1	startseq girl going into wooden building endseq
2	1000268201_693b08cb0e.jpg	2	startseq little girl climbing into wooden play...
3	1000268201_693b08cb0e.jpg	3	startseq little girl climbing the stairs to he...
4	1000268201_693b08cb0e.jpg	4	startseq little girl in pink dress going into ...

3. VGG 16 Implementation: Image segmentation

Now, since our captions are cleaning and processed, we need to extract features from our images. For this process we use CNN models. We specifically use VGG16 model, which is tested and proven to work well for image feature extraction.

We import the VGG16 model from keras and load pretrained VGG16 model weights.

```
# Loading VGG16 model and weights to extract features from the images
from keras.applications import VGG16

modelvgg = VGG16(include_top=True,weights=None)

modelvgg.load_weights("/kaggle/input/vgg16-weights/vgg16_weight1.h5")
modelvgg.summary()
```

Next, we pop out the last layer of VGG16 model, since the last layer is used for classification whereas we need all the features and their confidence level from the image

```
from keras import models
# Deleting the last layer of the model
# The last layer of the VGG-16 is excluded here because we are are just using it for extracting
# the features rather than using for object classification.
#modelvgg.layers.pop()
modelvgg = models.Model(inputs=modelvgg.inputs, outputs=modelvgg.layers[-2].output)
## show the deep learning model
modelvgg.summary()
```

We use the VGG16 model to get features from 224x224 size images as the model has been pre trained on that sized images.

```
# Feature extraction
# Here the features are extracted from all the images in the dataset.
# VGG-16 model gives out 4096 features from the input image of size 224 * 224

images = OrderedDict()
npix = 224 #image size is fixed at 224 because VGG16 model has been pre-trained to take that size.
target_size = (npix,npix,3)
data = np.zeros((len(jpgs),npix,npix,3))
tf.keras.config.disable_interactive_logging()

for i, image_id in enumerate(tqdm(jpgs)):
    # load an image from file
    filename = image_folder_path + '/' + image_id
    image = load_img(filename, target_size=target_size)
    # convert the image pixels to a numpy array
    image = img_to_array(image)
    image = image.reshape( (1,) + image.shape[:3])
    nimage = preprocess_input(image)

    y_pred = modelvgg.predict(nimage)
    images[image_id] = y_pred.flatten()

tf.keras.config.enable_interactive_logging()
```

VGG-16 model gives out 4096 features from the input image of size 224x224

4. Caption preprocessing for LSTM Model

For training of the LSTM model, we consider only the 1st caption of the 5 provided captions for every image.

```
# Merging the images and the captions for training
dimages, keepindex = [],[]
# Creating a datframe where only first caption is taken for processing
df_txt0 = df_txt0.loc[df_txt0["index"].values == "0",: ]
for i, fnm in enumerate(df_txt0.filename):
    if fnm in images.keys():
        dimages.append(images[fnm])
        keepindex.append(i)

#fnames are the names of the image files
fnames = df_txt0["filename"].iloc[keepindex].values
#dcaptions are the captions of the images
dcaptions = df_txt0["caption"].iloc[keepindex].values
#dimages are the actual features of the images
dimages = np.array(dimages)
df_txt0[:5]
```

	filename	index	caption
0	1000268201_693b08cb0e.jpg	0	startseq child in pink dress is climbing up se...
5	1001773457_577c3a7d70.jpg	0	startseq black dog and spotted dog are fightin...
10	1002674143_1b742ab4b8.jpg	0	startseq little girl covered in paint sits in ...
15	1003163366_44323f5815.jpg	0	startseq man lays on bench while his dog sits ...
20	1007129816_e794419615.jpg	0	startseq man in an orange hat starring at some...

Next we tokenize our captions using Keras Tokenizer, since LSTM works with tokens and not text data itself.

```
# Tokenizing the captions for further processing
# As the model can't take texts as an input, they need to converted into vectors.

from tensorflow.keras.preprocessing.text import Tokenizer
## the maximum number of words in dictionary
nb_words = 6000
tokenizer = Tokenizer(nb_words=nb_words)
tokenizer.fit_on_texts(dcaptions)
vocab_size = len(tokenizer.word_index) + 1
print("vocabulary size : {}".format(vocab_size))
dtexts = tokenizer.texts_to_sequences(dcaptions)
print(dtexts[:5])
```

```
vocabulary size : 4478
[[1, 38, 3, 66, 144, 7, 124, 52, 406, 9, 367, 3, 24, 2351, 522, 2], [1, 12, 8, 5, 752, 8, 17, 368, 2], [1, 48, 15, 170, 3, 584
```

Next, we split our caption dataset into training, validation and testing sets in a 60:20:20 ratio.

```
# Splitting the training and test data
prop_test, prop_val = 0.2, 0.2

N = len(dtexts)
Ntest, Nval = int(N*prop_test), int(N*prop_val)

def split_test_val_train(dtexts,Ntest,Nval):
    return(dtexts[:Ntest],
           dtexts[Ntest:Ntest+Nval],
           dtexts[Ntest+Nval:])

dt_test,  dt_val, dt_train   = split_test_val_train(dtexts,Ntest,Nval)
di_test,  di_val, di_train   = split_test_val_train(dimages,Ntest,Nval)
fnm_test,fnm_val, fnm_train  = split_test_val_train(fnames,Ntest,Nval)
# Finding the max length of the caption
maxlen = np.max([len(text) for text in dtexts])
print(maxlen)
```

```
30
```

In the previous step we all find the length of the longest caption which is 30 as seen.
Finally, we pad all the captions to convert them into standard length and make them categorical.

```
# Processing the captions and images as per the required shape by the model
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

def preprocessing(dtexts,dimages):
    N = len(dtexts)
    print("# captions/images = {}".format(N))

    assert(N==len(dimages)) # using assert to make sure that length of images and captions are always similar
    Xtext, Ximage, ytext = [],[],[]
    for text,image in zip(dtexts,dimages):
        # zip() is used to create a tuple of iteratable items
        for i in range(1,len(text)):
            in_text, out_text = text[:i], text[i]
            in_text = pad_sequences([in_text],maxlen=maxlen).flatten()# using pad sequence to make the length of
            out_text = to_categorical(out_text,num_classes = vocab_size) # using to_categorical to


            Xtext.append(in_text)
            Ximage.append(image)
            ytext.append(out_text)

    Xtext  = np.array(Xtext)
    Ximage = np.array(Ximage)
    ytext  = np.array(ytext)
    print(" {} {} {}".format(Xtext.shape,Ximage.shape,ytext.shape))
    return(Xtext,Ximage,ytext)


Xtext_train, Ximage_train, ytext_train = preprocessing(dt_train,di_train)
Xtext_val,   Ximage_val,   ytext_val   = preprocessing(dt_val,di_val)
# pre-processing is not necessary for testing data
#Xtext_test,  Ximage_test,  ytext_test  = preprocessing(dt_test,di_test)
```

5. LSTM Model Training

Finally after all the preprocessing and feature extraction, we use all this data to train our LSTM model

```
# Building the LSTM model
from keras import layers
from keras.models import Model

print(vocab_size)
## image feature

dim_embedding = 64

input_image = layers.Input(shape=(Ximage_train.shape[1],))
fimage = layers.Dense(256, activation='relu', name="ImageFeature")(input_image)
## sequence model
input_txt = layers.Input(shape=(maxlen,))
ftxt = layers.Embedding(vocab_size, dim_embedding)(input_txt)
ftxt = layers.LSTM(256, name="CaptionFeature", return_sequences=True)(ftxt)
se2 = layers.Dropout(0.04)(ftxt)
ftxt = layers.LSTM(256, name="CaptionFeature2")(se2)
## combined model for decoder
decoder = layers.add([ftxt, fimage])
decoder = layers.Dense(256, activation='relu')(decoder)
output = layers.Dense(vocab_size, activation='softmax')(decoder)
model = Model(inputs=[input_image, input_txt], outputs=output)

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

Following is the architecture of our LSTM Model

Model: "functional_13"

Layer (type)	Output Shape	Param #	Connected to
input_layer_13 (InputLayer)	(None, 30)	0	-
embedding_1 (Embedding)	(None, 30, 64)	286,592	input_layer_13[0…
CaptionFeature (LSTM)	(None, 30, 256)	328,704	embedding_1[0][0]
dropout_1 (Dropout)	(None, 30, 256)	0	CaptionFeature[0…
input_layer_12 (InputLayer)	(None, 4096)	0	-
CaptionFeature2 (LSTM)	(None, 256)	525,312	dropout_1[0][0]
ImageFeature (Dense)	(None, 256)	1,048,832	input_layer_12[0…
add_1 (Add)	(None, 256)	0	CaptionFeature2[… ImageFeature[0][…
dense_2 (Dense)	(None, 256)	65,792	add_1[0][0]
dense_3 (Dense)	(None, 4478)	1,150,846	dense_2[0][0]

Total params: 3,406,078 (12.99 MB)

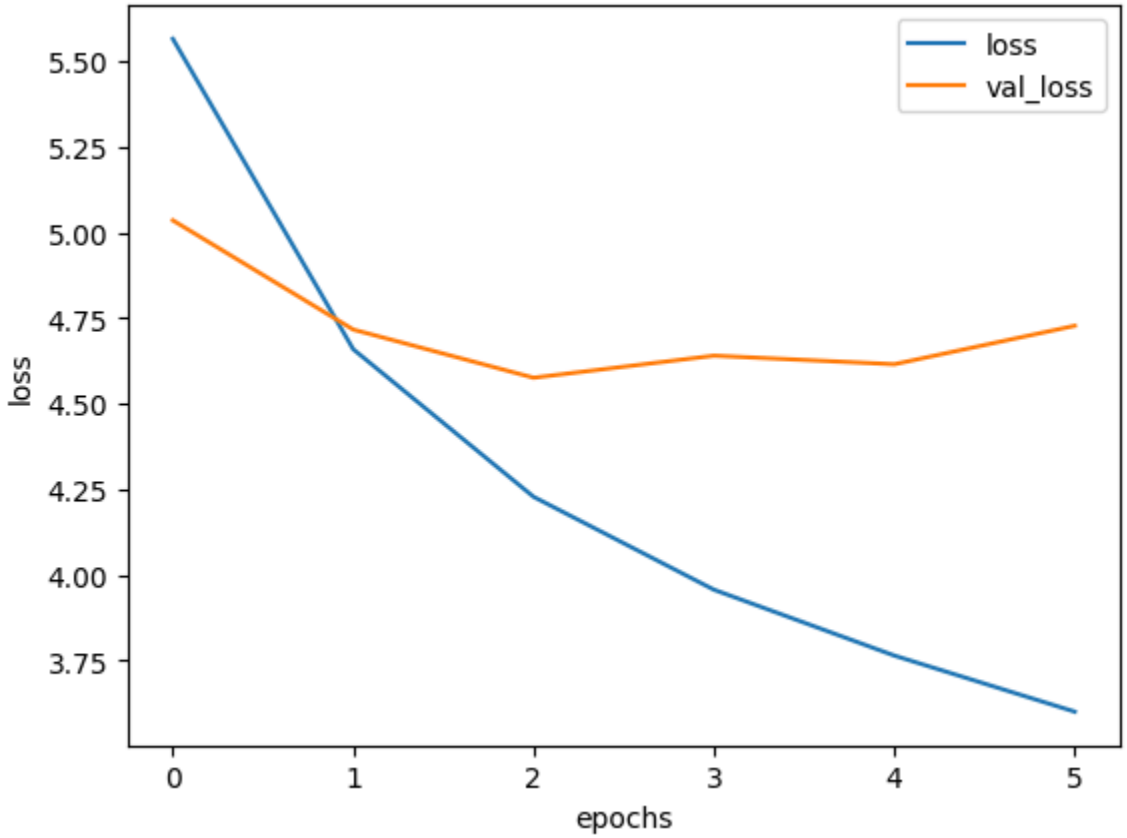
Trainable params: 3,406,078 (12.99 MB)

Non-trainable params: 0 (0.00 B)

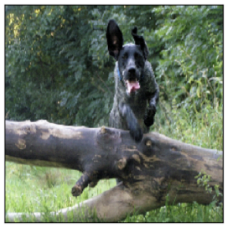
We train our LSTM Model for 6 epochs with a batch size of 32.
We use Categorical Cross Entropy Loss and Adam optimizer for training.

```
Epoch 1/6
1552/1552 - 22s - 14ms/step - accuracy: 0.1219 - loss: 5.5653 - val_accuracy: 0.1879 - val_loss: 5.0353
Epoch 2/6
1552/1552 - 18s - 11ms/step - accuracy: 0.2063 - loss: 4.6589 - val_accuracy: 0.2275 - val_loss: 4.7164
Epoch 3/6
1552/1552 - 18s - 11ms/step - accuracy: 0.2402 - loss: 4.2279 - val_accuracy: 0.2508 - val_loss: 4.5757
Epoch 4/6
1552/1552 - 18s - 11ms/step - accuracy: 0.2612 - loss: 3.9565 - val_accuracy: 0.2578 - val_loss: 4.6401
Epoch 5/6
1552/1552 - 18s - 11ms/step - accuracy: 0.2723 - loss: 3.7642 - val_accuracy: 0.2661 - val_loss: 4.6151
Epoch 6/6
1552/1552 - 18s - 12ms/step - accuracy: 0.2812 - loss: 3.6004 - val_accuracy: 0.2592 - val_loss: 4.7274
```

Training and Validation Loss across Epochs



Finally, we first run our model to predict captions for a small set of images to test it manually.



startseq black dog is jumping in the grass endseq

startseq black and white dog is running in the snow endseq

startseq man in red shirt is riding trick in the air endseq

startseq man in pink shirt is standing on the water endseq

On manual inspection of the sample captions, we find that some captions are decently accurate while some missed the mark. So, we run a deeper inspection to find out the mean BLEU scores of the model's predictions.

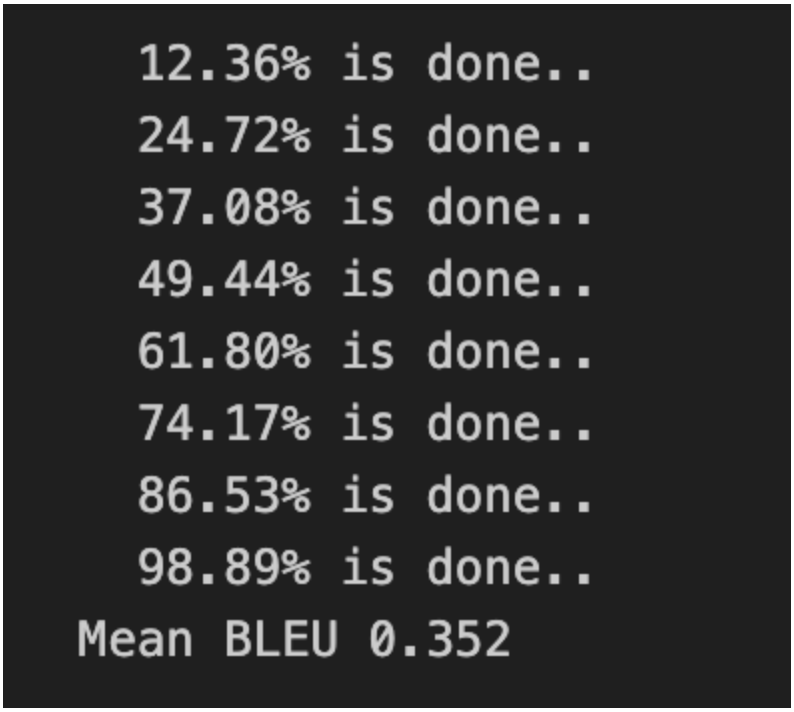
6. Performance evaluation

To start with performance evaluation, we find out the mean BLEU score of our model to Google cloud documentation following is the categorization of BLEU scores.

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

We run through the predicted captions of all test images and get an average BLEU score of 35.

```
# Evaluating the model performance
# Generating captions for the whole test data and finding BLEU score
from nltk.translate.bleu_score import sentence_bleu

index_word = dict([(index,word) for word, index in tokenizer.word_index.items()])


nkeep = 5
pred_good, pred_bad, bleus = [], [], []
count = 0
for jpgfnm, image_feature, tokenized_text in zip(fnm_test,di_test,dt_test):
    count += 1
    if count % 200 == 0:
        print("  {:4.2f}% is done..".format(100*count/float(len(fnm_test))))

    caption_true = [ index_word[i] for i in tokenized_text ]
    caption_true = caption_true[1:-1] ## remove startreg, and endreg
    ## captions
    caption = predict_caption(image_feature.reshape(1,len(image_feature)))
    caption = caption.split()
    caption = caption[1:-1]## remove startreg, and endreg

    bleu = sentence_bleu([caption_true],caption)
    bleus.append(bleu)
    if bleu > 0.7 and len(pred_good) < nkeep:
        pred_good.append((bleu,jpgfnm,caption_true,caption))
    elif bleu < 0.3 and len(pred_bad) < nkeep:
        pred_bad.append((bleu,jpgfnm,caption_true,caption))

print("Mean BLEU {:4.3f}".format(np.mean(bleus)))
```



We also dive one level deeper and find out the captions of images with best and worst BLEU scores.

Images with best BLEU Score

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

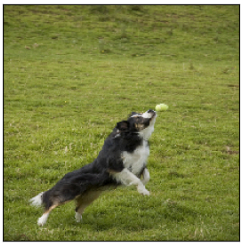
pred: man in white shirt is standing in front of people in front of building

BLEU: 0.7311104457090247

true: man drilling hole in the ice

pred: man is sitting on the water

BLEU: 0.7598356856515925



true: black and white dog catches toy in midair

pred: black dog is running in the air

BLEU: 0.7013967267997694

true: child with helmet on his head rides bike

pred: man is riding bike trick on the air

BLEU: 0.7071067811865476

true: bulldog sheep dog and boxer standing in yard

pred: black and white dog is standing on the beach

BLEU: 0.7598356856515925

Images with worst BLEU Score

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

pred: group of people are standing on the street

BLEU: 0.21874242445215208

true: man in an orange hat starring at something

pred: man in white shirt is standing in black and white and white and white and white

BLEU: 0.22277604606094104

true: black and white dog is running in grassy garden surrounded by white fence

pred: brown dog is jumping in the grass

BLEU: 0.21938936848339244

true: boy smiles in front of stony wall in city

pred: group of people are standing in front of people in front of people

BLEU: 0.2613022659677713

true: collage of one person climbing cliff

pred: man in pink shirt is standing on the water

BLEU: 0

Conclusion

After trying a bunch of different alternatives and combinations like DenseNet, Transformers along with different architectures for LSTM, we found that with the limited amount of training with the current combination gave us the best results. We are confident that, with higher computation power like more epochs and a bigger dataset like Flickr 32k or Microsoft COCO, our model will perform better.

Future Scope

We plan to extend our model to novel avenues like captioning live drawing of images or generating a story using a group of images. We concluded that the VGG16 and LSTM model combination worked best for us for image captioning and might go a long way for future ideas as well, only bound by compute power and training data.

References

1. Pandas documentation: https://pandas.pydata.org/docs/
2. Numpy documentation: https://numpy.org/doc/stable/user/index.html#user
3. Matplotlib documentation: https://matplotlib.org/stable/index.html
4. Tensorflow documentation: https://www.tensorflow.org/api_docs
5. Keras documentation: https://keras.io/guides/
6. CSE 676B: Deep Learning Lecture slides by Prof. Alina Vereshchaka