

## Group Members

Vengadesh S (2021fc04424)

Aiswarya S Parvathi (2021fc04430)

Nipun Gupta (2021fc04426)

Problem Statement: Generate Image Captions using CNN+LSTM

For this problem statement, we have worked on the Flickr dataset

We will first install and import the opendatasets library to download the dataset

```
1 !pip install opendatasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from opendatasets) (4.65.0)
Requirement already satisfied: kaggle in /usr/local/lib/python3.9/dist-packages (from opendatasets) (1.5.13)
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages (from opendatasets) (8.1.3)
Requirement already satisfied: certifi in /usr/local/lib/python3.9/dist-packages (from kaggle->opendatasets) (2022)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from kaggle->opendatasets) (2.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.9/dist-packages (from kaggle->opendatasets) (1.
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.9/dist-packages (from kaggle->opendatasets) (1.
Requirement already satisfied: python-slugify in /usr/local/lib/python3.9/dist-packages (from kaggle->opendatasets)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/dist-packages (from kaggle->opendatasets) (1.26)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.9/dist-packages (from python-slugify-
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.9/dist-packages (from requests->kaggle-
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->kaggle->open
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
```

```
1 import opendatasets as od
```

```
1 # URL of the dataset on kaggle website
2 dataset = r"https://www.kaggle.com/datasets/adityajn105/flickr8k"
3 WORKING_DIR = '/kaggle/working'
```

```
1 # Downloading the dataset
2 od.download(dataset)
3 #Kaggle creds - {"username":"nipungupta26","key":"7800b6f2327a2db633fe1d14f04280a5"}
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: nipungupta26
Your Kaggle Key: .....
Downloading flickr8k.zip to ./flickr8k
100%[██████████] 1.04G/1.04G [00:48<00:00, 22.8MB/s]
```

```
1 # Directory of the downloaded datasets in
2 data_dir = r'flickr8k/Images'
```

Importing the required libraries

```
1 import numpy as np
2 import pandas as pd
3 from matplotlib import pyplot as plt
```

```

4 import seaborn as sns
5 import os
6
7 import warnings
8 warnings.filterwarnings('ignore')

1 # Listing out the images in the dataset
2 images = os.listdir(data_dir)

1 import pickle
2 from tqdm.notebook import tqdm
3 import tensorflow as tf
4 import cv2
5
6 from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
7 from tensorflow.keras.preprocessing.image import load_img, img_to_array
8 from tensorflow.keras.preprocessing.text import Tokenizer
9 from tensorflow.keras.preprocessing.sequence import pad_sequences
10 from tensorflow.keras.models import Model
11 from tensorflow.keras.utils import to_categorical, plot_model
12 from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add

1 # Checking the GPU available
2 tf.test.gpu_device_name()

    '/device:GPU:0'

1 # Loading the captions
2 with open(r'flickr8k/captions.txt', 'r') as f:
3     next(f)
4     captions_doc = f.read()

1 # create mapping of image to captions
2 mapping = {}
3
4 count = 0
5
6 # process lines
7 for line in tqdm(captions_doc.split('\n')):
8     count = count + 1
9     tokens = line.split(',')
10    if len(line) < 2:
11        continue
12    image_id, caption = tokens[0], tokens[1:]
13    image_id = image_id.split('.')[0]
14    caption = ' '.join(caption)
15    if image_id not in mapping:
16        mapping[image_id] = []
17    mapping[image_id].append(caption)

100% 40456/40456 [00:00<00:00, 284485.00it/s]

1 len(list(mapping.keys()))

8091

1 mapping['1001773457_577c3a7d70']

```

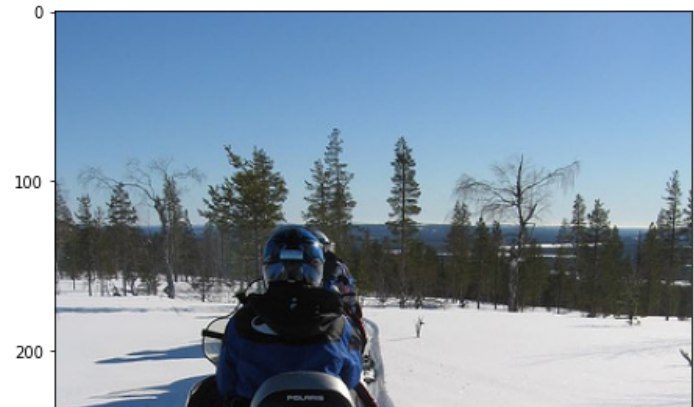
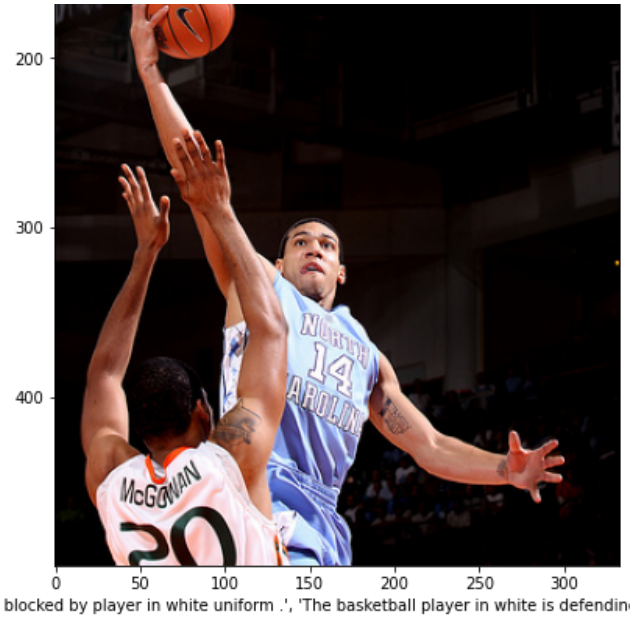
```

['A black dog and a spotted dog are fighting',
 'A black dog and a tri-colored dog playing with each other on the road .',
 'A black dog and a white dog with brown spots are staring at each other in the street .',

```

```
'Two dogs of different breeds looking at each other on the road .',  
'Two dogs on pavement moving toward each other .']
```

```
1 # Plotting at least two samples and their captions (use matplotlib/seaborn/any other library)  
2 for i in range(3):  
3     plt.figure(figsize=(10,10))  
4     img = cv2.imread(r'flickr8k/Images/' + images[i])  
5     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  
6     for j in list(mapping.keys()):  
7         if images[i][-4] in j:  
8             x = str(images[i][-4])  
9             break  
10    plt.xlabel(mapping[x])  
11    plt.imshow(img)
```



Extracting Image features



```
1 # load the pretrained VGG16 model
2 model = VGG16()
3 #VGG16 is a convolutional neural network trained on a subset of the ImageNet dataset, a collection of over 14 million

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_and_
553467096/553467096 [=====] - 24s 0us/step
```



```
1 # Restructure the model
2 model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
3 # We have written -2 because we don't need to fully connected layer of VGG16 model. We just need the previous layers
```



```
1 # summarize the model
2 print(model.summary())
```

Model: "model"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856

block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312

```

=====
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0

```

---

None

```

1 #extract features from image
2 features = {}
3 directory = os.path.join(data_dir, 'train2017')

```

### Converting the data into the correct format which could be used for the DL model

```

1 count = 0
2
3 for img_name in tqdm(os.listdir(data_dir)):
4     count = count + 1
5     # load the image from file
6     img_path = data_dir + '/' + img_name
7     image = load_img(img_path, target_size=(224, 224))
8     # (224, 224) will be the size of the resized image
9
10    # Converting the image pixel to a numpy array
11    image= img_to_array(image)
12
13    # Reshaping the data for the model to extract the features
14    image = image.reshape(1, image.shape[0], image.shape[1], image.shape[2])
15
16    # Preparing the image for the VGG model
17    image = preprocess_input(image)
18
19    # Extracting the features

```

```
20 feature = model.predict(image, verbose=0)
21
22 # getting the image id
23 image_id = img_name.split('.')[0]
24
25 features[image_id] = feature
```

100%

8091/8091 [10:40&lt;00:00, 14.76it/s]

```
1 # Storing/Pickling the features in a file
2 pickle.dump(features, open('features.pk1','wb' ))
```

```
1 # Unpickling the features
2 with open('features.pk1', 'rb') as f:
3     features = pickle.load(f)
```

```
1 len(features)
```

8091

```
1 features
```

```
'2b98b14194_b4e6e11d77': array([[0.          , 0.          , 0.          , ..., 0.          , 0.65814966,
0.          ]], dtype=float32),
'2922807898_b5a06d5c70': array([[0.14262903, 0.          , 0.9791787 , ..., 0.          , 1.6309462 ,
0.          ]], dtype=float32),
'2646615552_3aeeb2473b': array([[0.          , 0.          , 0.          , ..., 0.          , 2.3836195,
2.1790028]], dtype=float32),
'2586911841_41b7a48c91': array([[0.          , 3.3675995 , 2.4686646 , ..., 0.          , 0.          ,
0.10566723]], dtype=float32),
'3295391572_cbfde03a10': array([[0.          , 0.          , 0.2531849, ..., 0.          , 2.7493427,
1.3197509]], dtype=float32),
'2831314869_5025300133': array([[0., 0., 0., ..., 0., 0., 0.]], dtype=float32),
'3564148252_aa4cb36a32': array([[0.          , 0.          , 0.          , ..., 0.          , 1.0996966,
4.1292043]], dtype=float32),
...
}
```

## Preprocess Text Data

```
1 # Now we will preprocess the captions
2 def clean(mapping):
3     for key, captions in mapping.items():
4         for i in range(len(captions)):
5             #Take one caption at a time
6             caption = captions[i]
7             #Preprocessing steps
8             #Convert to lower case
9             caption = caption.lower()
10            #delete digit, special characters
11            caption = caption.replace('[^A-Za-z]', '')
12            # Remove additional spaces
13            caption = caption.replace('\s+', ' ')
14            # Add start and end tags to the caption
15            caption = 'startseq ' + ' '.join(word for word in caption.split() if len(word)>1) + ' endseq'
16            captions[i] = caption
```

```
1 # Before preprocess of text
2 mapping['1000268201_693b08cb0e']

['A child in a pink dress is climbing up a set of stairs in an entry way .',
'A girl going into a wooden building .',
'A little girl climbing into a wooden playhouse .',
'A little girl climbing the stairs to her playhouse .',
'A little girl in a pink dress going into a wooden cabin .']
```

```
1 # preprocess the text
2 clean(mapping)
```

```
1 # After preprocess of text
2 mapping['1000268201_693b08cb0e']

['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
'startseq girl going into wooden building endseq',
'startseq little girl climbing into wooden playhouse endseq',
'startseq little girl climbing the stairs to her playhouse endseq',
'startseq little girl in pink dress going into wooden cabin endseq']
```

```
1 all_captions = []
2 for key in mapping:
3     for caption in mapping[key]:
4         all_captions.append(caption)
```

```
1 len(all_captions)
```

```
40455
```

```

1 all_captions[:5]

['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq']

1 # tokenize the text
2 tokenizer = Tokenizer()
3 tokenizer.fit_on_texts(all_captions)
4 vocab_size = len(tokenizer.word_index) + 1

1 vocab_size

8485

1 # get maximum length of the caption available
2 max_length = max(len(caption.split()) for caption in all_captions)
3 max_length

35

1 # Train test split
2 image_ids = list(mapping.keys())
3 split = int(len(image_ids) * 0.90)
4 split
5 train = image_ids[:split]
6 test = image_ids[split:]

1 # create data generator to get data in batch (avoids sessions crash)
2 def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batch_size):
3     # loop over images
4     X1, X2, y = list(), list(), list()
5     n = 0
6     while 1:
7         for key in data_keys:
8             n += 1
9             captions = mapping[key]
10            # process each caption
11            for caption in captions:
12                # encode the sequence
13                seq = tokenizer.texts_to_sequences([caption])[0]
14                # split the sequence into X, y pairs
15                for i in range(1, len(seq)):
16                    # split into input and output pairs
17                    in_seq, out_seq = seq[:i], seq[i]
18                    # pad input sequence
19                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
20                    # encode the output sequence
21                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
22
23                # store the sequences
24                X1.append(features[key][0])
25                #print('features[key] : ', features[key])
26                #print('features[key][0] : ', features[key][0])
27                X2.append(in_seq)
28                y.append(out_seq)
29            if n == batch_size:
30                X1, X2, y = np.array(X1), np.array(X2), np.array(y)
31                yield [X1, X2], y
32                X1, X2, y = list(), list(), list()
33                n = 0

```



```

1 from keras.models import Sequential
2 from keras import layers
3 from keras.layers import Dense, LeakyReLU, PReLU, ELU, Dropout

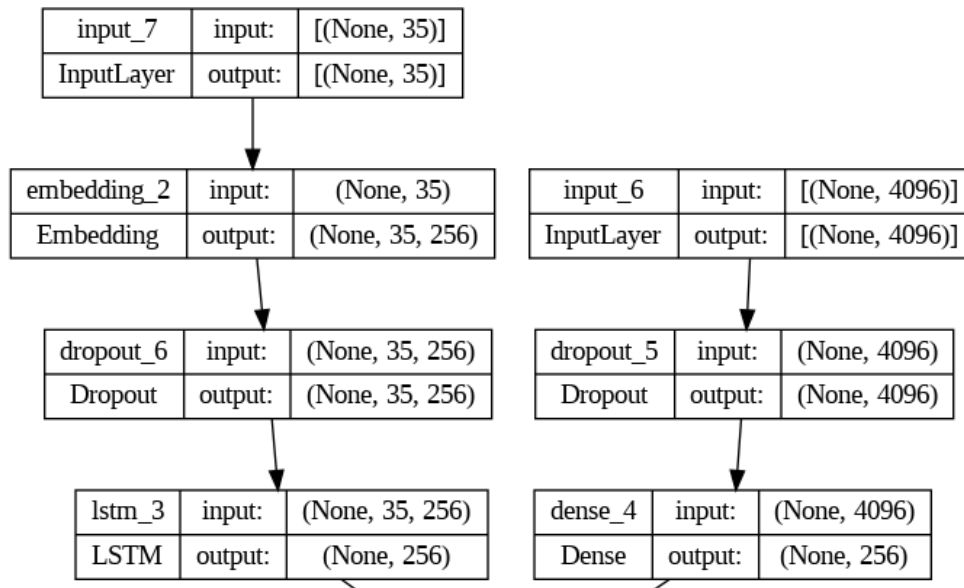
```

## Model Creation

```

1 # Encoder model
2 # image feature layers
3 # If we see the output of the VGG16 model that was created earlier, the output shape is (None, 4096). So the input sh
4 inputs1 = Input(shape = (4096, ))
5
6 # Introducing a dropout of 0.4 to prevent overfitting
7 fe1 = Dropout(0.4)(inputs1)
8 fe2 = Dense(256, activation = 'relu')(fe1)
9 # The relu activation function prevents vanishing gradient problem
10
11 # sequence feature layers
12 inputs2 = Input(shape=(max_length,))
13 se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
14 # We have set mask_zero = True because we are padding the sequences
15 se2 = Dropout(0.4)(se1)
16 # Introducing 2 layers of LSTM
17 se3 = LSTM(256)(se2)
18
19
20 # decoder model
21 decoder1 = add([fe2, se3])
22 decoder2 = Dense(256, activation='relu')(decoder1)
23 outputs = Dense(vocab_size, activation = 'softmax')(decoder2)
24 # The softmax acitvation function gives probabilities of the respective target classes as output
25
26 model = Model(inputs=[inputs1, inputs2], outputs=outputs)
27 model.compile(loss='categorical_crossentropy', optimizer='adam')
28 '''The adam optimization algorithm is a further extension of stochastic gradient descent to update network weights du
29 a single learning rate through training in SGD, Adam optimizer updates the learning rate for each network weight indi
30
31 Categorical_crossentropy: Used as a loss function for multi-class classification model where there are two or more ou
32 The output label is assigned one-hot category encoding value in form of 0s and 1. The output label, if present in int
33 is converted into categorical encoding'''
34
35
36 # plot the model
37 plot_model(model, show_shapes=True)

```



```

1 # Train the model
2 epochs = 2
3 batch_size = 64
4 steps = len(train) // batch_size
5
6 for i in range(epochs):
7     # create data generator
8     generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, batch_size)
9     # fit for one epoch
10    classifier_nn = model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)

113/113 [=====] - 65s 519ms/step - loss: 5.6055
113/113 [=====] - 53s 464ms/step - loss: 4.4865

```

### Generate captions for the image

```

1 def idx_to_word(integer, token):
2     for word, index in tokenizer.word_index.items():
3         if index == integer:
4             return word
5     return None

1 # generate caption for an image
2 def predict_caption(model, image, tokenizer, max_length):
3     # add start tag for generation process
4     int_text = '<start>'
5     #print(int_text)
6     # iterate over the max length of sequence
7     for i in range(max_length):
8         # encode input sequence
9         sequence = tokenizer.texts_to_sequences([int_text])[0]
10        # pad the sequence
11        sequence = pad_sequences([sequence], max_length)
12        # predict next word
13        yhat = model.predict([image, sequence], verbose=0)
14        # get index with high probability
15        yhat = np.argmax(yhat)
16        # convert index to word
17        word = idx_to_word(yhat, tokenizer)
18        # stop if word not found
19        if word is None:
20            break
21        # append word as input for generating next word

```

```

22     int_text = int_text + " " + word
23     # stop if we reach end tag
24     if word == 'endseq':
25         break
26     return int_text

1 from nltk.translate.bleu_score import corpus_bleu
2
3 # Validate with test data
4 actual, predicted = list(), list()
5
6 for key in tqdm(test):
7     # get actual caption
8     captions = mapping[key]
9     # predict the caption for image
10    y_pred = predict_caption(model, features[key], tokenizer, max_length)
11
12    # split into words
13    actual_captions = [caption.split() for caption in captions]
14    y_pred = y_pred.split()
15    actual.append(actual_captions)
16    predicted.append(y_pred)
17
18
19 # calculate BLEU score
20 print(f'BLEU-1: {corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0))}')
21 print(f'BLEU-2: {corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0))}')

100%
810/810 [12:41<00:00, 1.20s/it]

BLEU-1: 0.2583941605839416
BLEU-2: 0.1445543056273581

```

## Visualize the results

```

1 from PIL import Image
2 import matplotlib.pyplot as plt
3
4 def generate_caption(image_name):
5     #load the image
6
7     image_id = image_name.split('.')[0]
8     img_path = os.path.join(r'flickr8k/', "Images", image_name)
9     image = Image.open(img_path)
10    captions = mapping[image_id]
11    print('-----Actual-----')
12    for caption in captions:
13        print(caption)
14    #predict the caption
15    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)
16    print('-----Predicted-----')
17    print(y_pred)
18    plt.imshow(image)

1 image_name = "1001773457_577c3a7d70.jpg"
2 generate_caption(image_name)

```



-----Actual-----

startseq black dog and spotted dog are fighting endseq

startseq black dog and tri-colored dog playing with each other on the road endseq

startseq black dog and white dog with brown spots are staring at each other in the stre

startseq two dogs of different breeds looking at each other on the road endseq

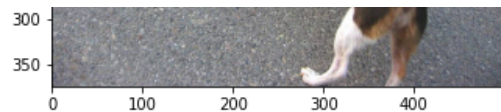
startseq two dogs on pavement moving toward each other endseq

-----Predicted-----

<start> two two dogs dogs playing on the grass endseq



1



✓ 2s completed at 10:08 AM

● ✕