Image Caption Generation

Capstone Project Report

November 2022

Instructor: ***Albert Gyamfi***

Course: ***Artificial Intelligence and Data Analytics***

Group Members: ***Smitesh Tamboli & Lam To Ngan (Ivy)***

Table of Contents

I. Acknowledgment

Firstly, we would like to express our sincere thankfulness to our instructor - **Albert Gyamfi**, who has given us valuable guidance and feedback during the time we were working on this project.

We are **Smitesh Taboli** and **Lam To Ngan (Ivy)** in one group working together on this project. If you might any questions or further discussion related to this project, please feel free to contact us at our personal email addresses:

- Smitesh Taboli: smitesh.baroda@gmail.com
- Lam To Ngan (Ivy): tongan.lam@gmail.com

II. Abstract

The ambitious objective of this project is to build an application that can help the store owners auto-generate the easy-to-convert and eye-catching captions for their products to be listed on e-commerce platform. Initially, the project contains a website that takes a particular image as an input, and the website will produce different types of product tags related to the input image, and within the limited time frame of this project, the domain is narrowed down women's top clothes images.

In the background, the website will use different types of machine learning algorithms to generate image tags associated to the features of the clothes images such as: type of clothes (crop top or regular), type of sleeves (short, long, or cap) & neck shape (V-neck, round neck, etc.). If time is allowed, a full caption of that image should be built based on those product tags generated and adding some selling points according to those tags to make the caption more attractive and to increase the conversion rate for the store owners on e-commerce platforms.

III. Introduction

In any image, human is naturally capable to describe a large amount of information about the image within a fraction of seconds just by observing the image for few seconds. Making computers to intake the quality vision of human ability and to describe the visual world is a very large intention of the researchers in the world of robotic activities. Using different artificial techniques to automatically generate a natural language for an image, which can be considered as image captioning is quiet challenging task, but it is also one of the most attractive topics, which automatically generates natural language descriptions according to the content observed in an image.

Artificial intelligence (AI) has been developed rapidly in recent years and contributed huge value to many industries, and image caption generation are intensive and significant in some of practical use cases such as: helping blinded people to visualize things, detecting information from an image on social media such as Facebook (knowing where you are, what you wear or what you are doing based on the image posted on social media), generating caption for product images in e-commerce, etc.

In the e-commerce industry, product images are important to attract visitors, but the caption is even more important to help convert the visitors into customers. A skillfully written caption also gives an opportunity to hit home an important point about a product. In addition, informative captions also provide a search engine boost. Search engines can't duplicate the visual perception and interpretation performed by human eyes and brains. To understand the contents of a photo, search engines rely on captions.

IV. Problem Statement

Let's think as an e-commerce store owner, what if they have a huge number of products to be listed on the online store and every single product needs to have a very attractive and eye-catching captions to drive attention to the online shoppers. This is not to mention that the captions need to include the keywords which are the most associated to the products. It will be a time-consuming problem and ineffective way to write the caption manually one by one caption for each product they want to list on their online store if they might have thousands of products. This means the scalability of their business will be limited due to uncontrollable captions manually written for huge numbers of products coming.

With that demand, an AI application helping auto-generate high gravity captions in a timely manner for e-commerce products in a is intensively significant and necessary. This application can be extended to generate captions for all types of products that can be sold online, and even the captions can be added to some selling points that make them highly converted.

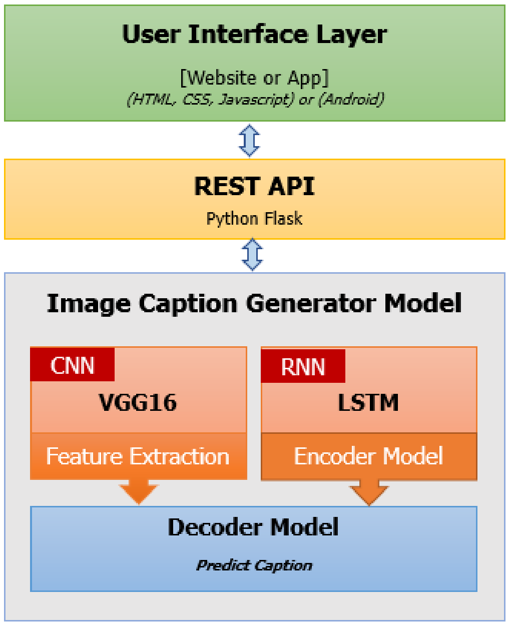
V. Methodology

In this project, we will be trying to develop an AI application which applies AI technology and methodologies to auto-generate e-commerce captions for clothes pictures. A machine learning model based on Convolutional Neural Networks (CNN) and Recurrent Neural Networks RNN) is required to develop the application. There will be two stages to building the application. The first stage is building the model to extract features from the image using CNN and generate possible texts using RNN. In the first stage, we will use Visual Geometry Group (VGG-16) which is a convolutional layer model used for object recognition. For the second stage, we need to train our features with the caption provided in the dataset. We will try to use two architectures Long Term Short Memory (LSTM) and Gated Recurrent Unit (GRU) to form sentences from the input image and observe which one provides the correct prediction.

Initially, the system uses Flickr 8k dataset which contains 8000 images with captions for training the model. We are also in the process of creating our own dataset of clothes and train the model to predict the clothing tags. Also, BLEU (Bilingual Evaluation Understudy) score is used to compare the performances between LSTM and GRU.

VI. System Models and Architecture

Our system architecture can be outlined as below diagram:



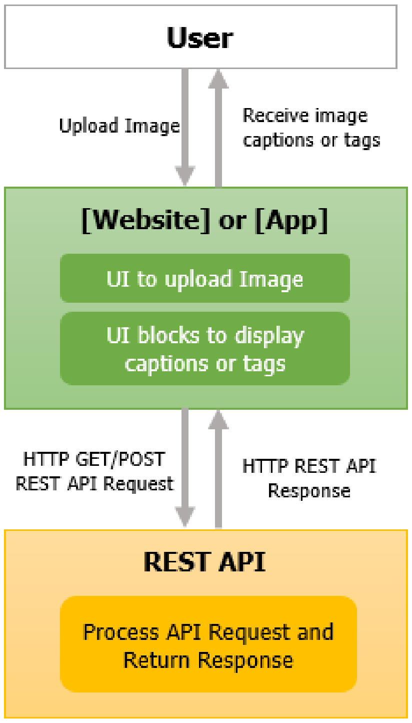
The complete system is divided into mainly three parts: 1) User Interface Layer; 2) REST API; 3) Image Caption Generator Model based on Artificial Intelligence.

1. **User Interface Layer**:

   This layer provides the facility for the end user to upload an image to generate a caption. The user interface will be either a website or a mobile application.

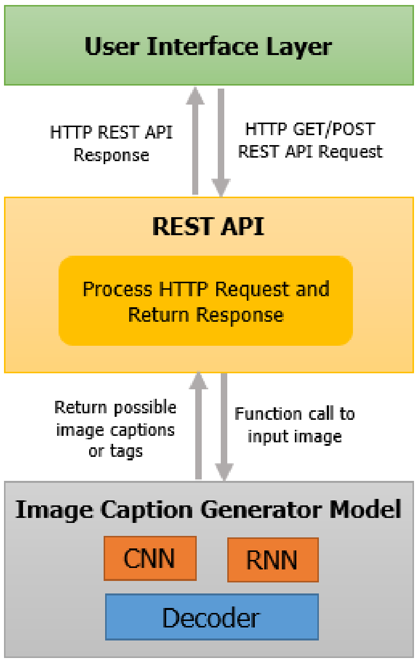
   Our priority is to create a website built using HTML, CSS, and Javascript technologies, where the webpage will ask to input the image and request appropriate Rest API for the caption generation. The webpage will receive captions or tags in response to Rest API requests.

   Optionally, we will also try to create an Android app that demands input image and display possible captions or tags as output.


User
Upload Image
Receive image captions or tags
[Website] or [App]
UI to upload Image
UI blocks to display captions or tags
HTTP GET/POST REST API Request
HTTP REST API Response
REST API
Process API Request and Return Response

2. **REST API Layer:**

   This layer receives the appropriate HTTP requests from the User Interface Layer and processes the request then returns the responses. The layer receives input images as an HTTP request form and returns input image captions or tags in the form of the HTTP response. The layer is also connected to the Image Caption Generator artificial intelligence model to generate captions. This layer will be built on the Python Flask module to accept HTTP requests/responses.


User Interface Layer
HTTP REST API Response
HTTP GET/POST REST API Request
REST API
Process HTTP Request and Return Response
Return possible image captions or tags
Function call to input image
Image Caption Generator Model
CNN
RNN
Decoder

3. Image Caption Generator Model:

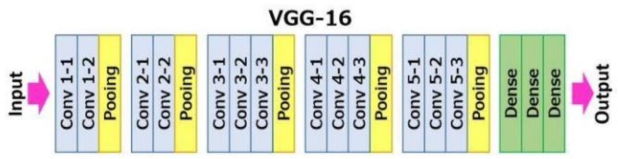
The Image Caption Generator model is the primary Artificial Intelligence model. The model will be built on a combination of three sub-models to generate captions from the input image.

- Feature Extraction Model
- Encoder Model
- Decoder Model

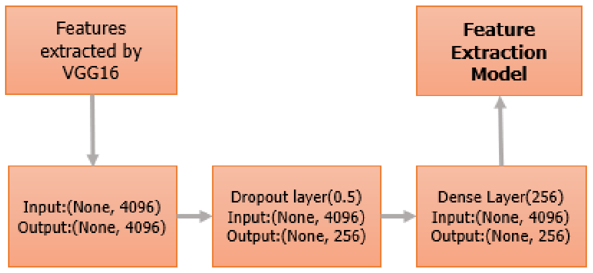
a) Feature Extraction Model

The primary function of this model is to extract features from the input image for the training. To efficiently extract features, the model uses a VGG16 architecture which is a combination of multiple 3x3 convolution layers and max pooling layers.

VGG16 is a model having 16 layers having weight in which there are 13 convolution layers, 5 Max pooling layers, and 3 Dense layers, summing up to 21 layers, but there are only 16 weight layers. The output of VGG16 model will be a vector size of 1*4096, which are used to represent the features of the images.



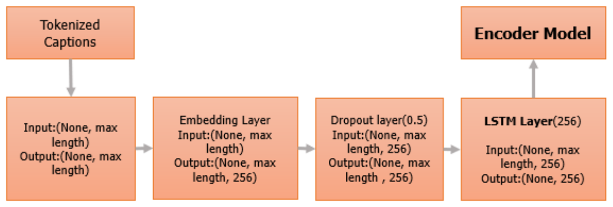
To reduce the overfitting problem, a dropout layer will be added to the model with a value range from 0.5 to 0.8. Finally, a dense layer will be used after the dropout layer with the 'ReLU' (Rectified Linear Units) activation function, input, and kernel with a bias. The output of the dense layer is the 256 sizes of the vectors. The final output will be the features extraction of the images. At last, the output of this model becomes the input of the Decoder model.


Features extracted by VGG16
Feature Extraction Model
Input:(None, 4096) Output:(None, 4096)
Dropout layer(0.5) Input:(None, 4096) Output:(None, 256)
Dense Layer(256) Input:(None, 4096) Output:(None, 256)

b) Encoder Model

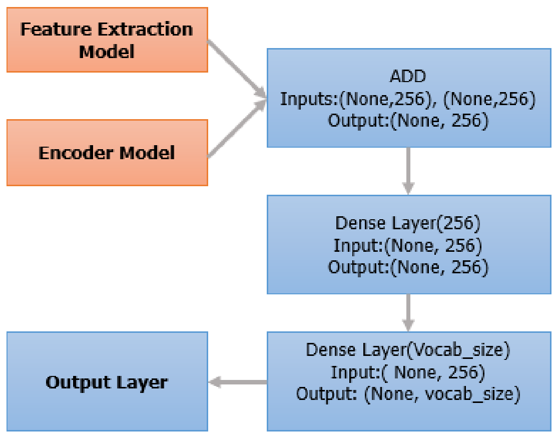
The primary responsibility of the encoder model is to preprocess the captions of each image while training. The system will first tokenize the captions of each image while training and convert each word into integers to process efficiently by the neural network. Then, tokenized captions are padded to equal lengths for all the sentences. Then, an embedding layer is attached to embed the tokenized captions into fixed dense vectors with output space of 256 by maximum length sentence from the training dataset.

Again, the dropout layer with a 0.5 value will be added to reduce the overfitting in the model. Finally, LSTM (Long Short-Term Memory) layer will be added to generate valid sentences or generate the word with the highest probability of occurrence after a specific word is encountered. The final layer will use ReLU as an activation function with 256 output space.


Tokenized Captions
Input:(None, max length) Output:(None, max length)
Embedding Layer Input:(None, max length) Output:(None, max length, 256)
Dropout layer(0.5) Input:(None, max length, 256) Output:(None, max length , 256)
LSTM Layer(256) Input:(None, max length, 256) Output:(None, 256)
Encoder Model

c) *Decoder Model*

The decoder model is the concatenates of both the feature extraction generator model and encoder model, and predicts the captions based on the input image as an output. Both the feature extraction generator model and the encoder model produce output vectors of dimension 256. Both these outputs will be concatenated and passed through a dense layer with ReLU as an activation function. Another dense layer will be used to decode the captions and all those words will be selected as the final output that has maximum probability by the SoftMax activation function.


Feature Extraction Model
Encoder Model
ADD Inputs:(None,256), (None,256) Output:(None, 256)
Dense Layer(256) Input:(None, 256) Output:(None, 256)
Dense Layer(Vocab_size) Input:(None, 256) Output: (None, vocab_size)
Output Layer

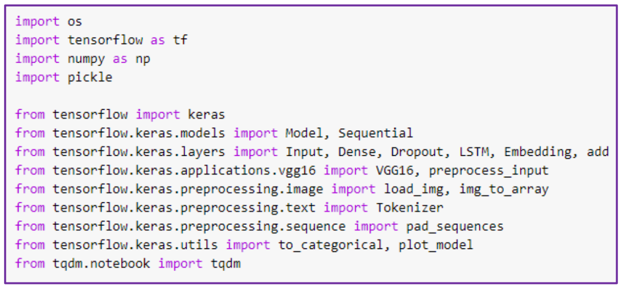
d) *Model Evaluation*

To evaluate the model, we will be using BLEU (Bilingual Evaluation Understudy) score for the correct captions or tags prediction. We will also try to use GRU (Gated Recurrent Unit) instead of LSTM in the encoder models and compare both outcomes for the final prediction. Also, we will try to test the Resnet CNN model for feature extraction for better performance.

VII. Implementation

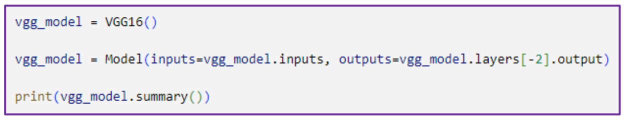
1. Product Tag Generator

a) *Import modules and libraries to be used in this project*

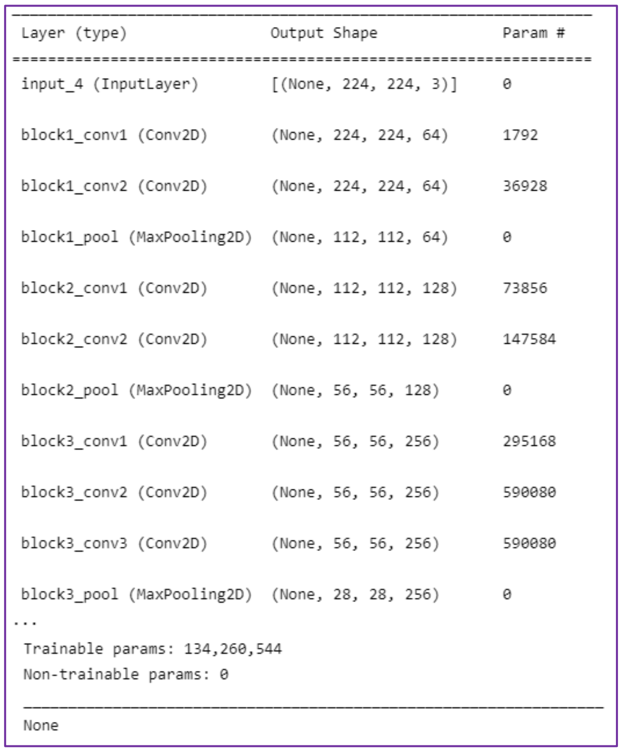


```
import os
import tensorflow as tf
import numpy as np
import pickle

from tensorflow import keras
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, LSTM, Embedding, add
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical, plot_model
from tqdm.notebook import tqdm
```

b) *Create and load VGG16 Model for the feature extraction*

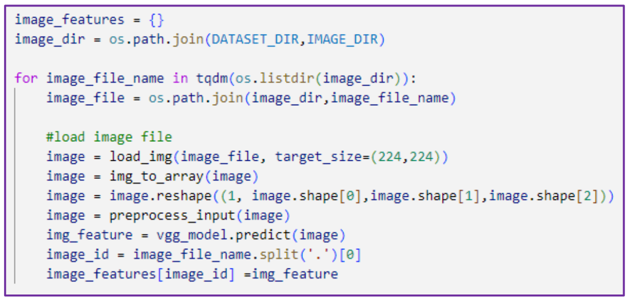


```
vgg_model = VGG16()

vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)

print(vgg_model.summary())
```



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (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
...
 Trainable params: 134,260,544
 Non-trainable params: 0
 _________________________________________________________________
 None
```

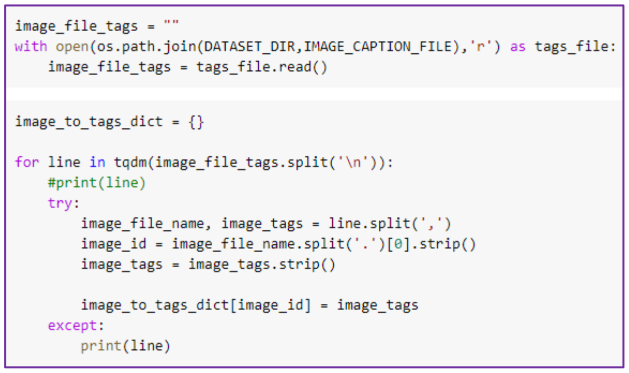
Create feature map dictionary that contains image file name as ID and its extracted features as a value.



```
image_features = {}
image_dir = os.path.join(DATASET_DIR,IMAGE_DIR)

for image_file_name in tqdm(os.listdir(image_dir)):
    image_file = os.path.join(image_dir,image_file_name)

    #load image file
    image = load_img(image_file, target_size=(224,224))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0],image.shape[1],image.shape[2]))
    image = preprocess_input(image)
    img_feature = vgg_model.predict(image)
    image_id = image_file_name.split('.')[0]
    image_features[image_id] =img_feature
```

c) *Proceeding caption data*

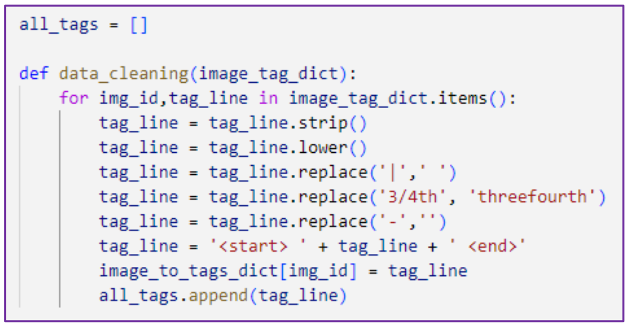
Create the mapping to map the image ids vs. the caption tags.



```
image_file_tags = ""
with open(os.path.join(DATASET_DIR,IMAGE_CAPTION_FILE),'r') as tags_file:
    image_file_tags = tags_file.read()

image_to_tags_dict = {}

for line in tqdm(image_file_tags.split('\n')):
    #print(line)
    try:
        image_file_name, image_tags = line.split(',')
        image_id = image_file_name.split('.')[0].strip()
        image_tags = image_tags.strip()

        image_to_tags_dict[image_id] = image_tags
    except:
        print(line)
```

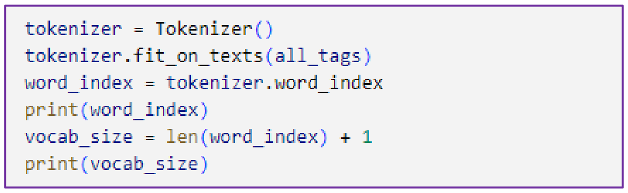
Processing captions text for the RNN or LSTM algorithm

- Convert to lower case
- Remove | from the captions
- Replace digits and special characters
- Mark "start" and "end" to recognize the string for the output



```
all_tags = []

def data_cleaning(image_tag_dict):
    for img_id,tag_line in image_tag_dict.items():
        tag_line = tag_line.strip()
        tag_line = tag_line.lower()
        tag_line = tag_line.replace('|',' ')
        tag_line = tag_line.replace('3/4th', 'threefourth')
        tag_line = tag_line.replace('-','')
        tag_line = '<start> ' + tag_line + ' <end>'
        image_to_tags_dict[img_id] = tag_line
        all_tags.append(tag_line)
```

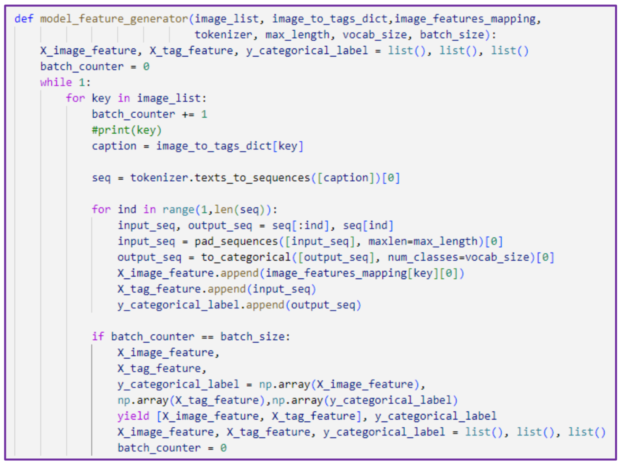
Tokenize all tags and find unique vocabulary size



```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_tags)
word_index = tokenizer.word_index
print(word_index)
vocab_size = len(word_index) + 1
print(vocab_size)
```

Generate features and target data in a batch based on token and extracted image features

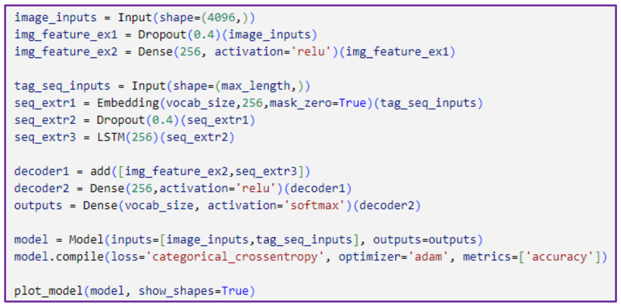
- Convert each caption according to tokenized sequence.
- Do padding based on maximum length.
- Convert label into categorical data.
- Return above processed data as a feature X and label Y.



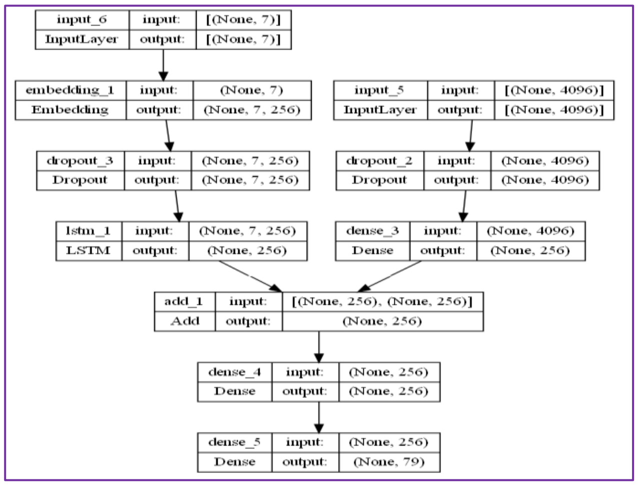
```
def model_feature_generator(image_list, image_to_tags_dict,image_features_mapping,
                            tokenizer, max_length, vocab_size, batch_size):
    X_image_feature, X_tag_feature, y_categorical_label = list(), list(), list()
    batch_counter = 0
    while 1:
        for key in image_list:
            batch_counter += 1
            #print(key)
            caption = image_to_tags_dict[key]

            seq = tokenizer.texts_to_sequences([caption])[0]

            for ind in range(1,len(seq)):
                input_seq, output_seq = seq[:ind], seq[ind]
                input_seq = pad_sequences([input_seq], maxlen=max_length)[0]
                output_seq = to_categorical([output_seq], num_classes=vocab_size)[0]
                X_image_feature.append(image_features_mapping[key][0])
                X_tag_feature.append(input_seq)
                y_categorical_label.append(output_seq)

            if batch_counter == batch_size:
                X_image_feature,
                X_tag_feature,
                y_categorical_label = np.array(X_image_feature),
                np.array(X_tag_feature),np.array(y_categorical_label)
                yield [X_image_feature, X_tag_feature], y_categorical_label
                X_image_feature, X_tag_feature, y_categorical_label = list(), list(), list()
                batch_counter = 0
```

d) *Create Decoder model*

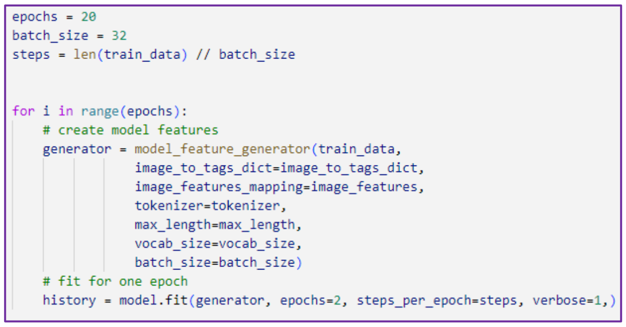
Create Decoder model by adding image features and tokenized encoder models



```
image_inputs = Input(shape=(4096,))
img_feature_ex1 = Dropout(0.4)(image_inputs)
img_feature_ex2 = Dense(256, activation='relu')(img_feature_ex1)

tag_seq_inputs = Input(shape=(max_length,))
seq_extr1 = Embedding(vocab_size,256,mask_zero=True)(tag_seq_inputs)
seq_extr2 = Dropout(0.4)(seq_extr1)
seq_extr3 = LSTM(256)(seq_extr2)

decoder1 = add([img_feature_ex2,seq_extr3])
decoder2 = Dense(256,activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[image_inputs,tag_seq_inputs], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

plot_model(model, show_shapes=True)
```


input_6 | InputLayer | input: [(None, 7)] | output: [(None, 7)]
embedding_1 | Embedding | input: (None, 7) | output: (None, 7, 256)
input_5 | InputLayer | input: [(None, 4096)] | output: [(None, 4096)]
dropout_3 | Dropout | input: (None, 7, 256) | output: (None, 7, 256)
dropout_2 | Dropout | input: (None, 4096) | output: (None, 4096)
lstm_1 | LSTM | input: (None, 7, 256) | output: (None, 256)
dense_3 | Dense | input: (None, 4096) | output: (None, 256)
add_1 | Add | input: [(None, 256), (None, 256)] | output: (None, 256)
dense_4 | Dense | input: (None, 256) | output: (None, 256)
dense_5 | Dense | input: (None, 256) | output: (None, 79)

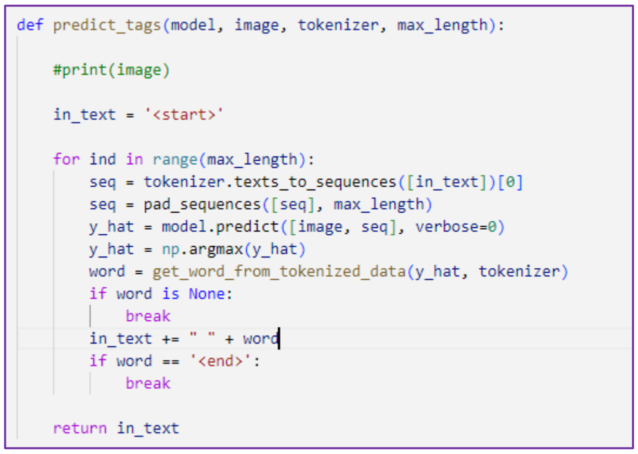
Train the model on batch



```
epochs = 20
batch_size = 32
steps = len(train_data) // batch_size


for i in range(epochs):
    # create model features
    generator = model_feature_generator(train_data,
                image_to_tags_dict=image_to_tags_dict,
                image_features_mapping=image_features,
                tokenizer=tokenizer,
                max_length=max_length,
                vocab_size=vocab_size,
                batch_size=batch_size)
    # fit for one epoch
    history = model.fit(generator, epochs=2, steps_per_epoch=steps, verbose=1,)
```

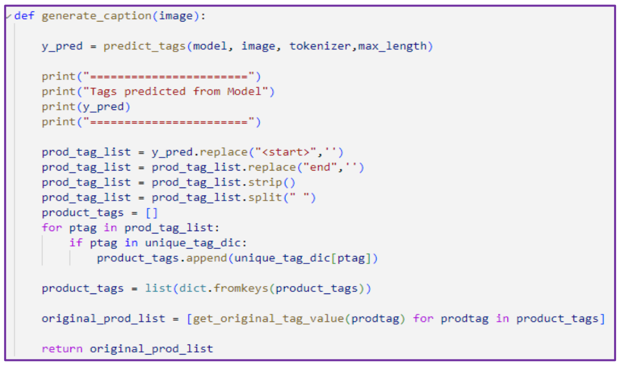
e) *Prediction*

Predict Tags or captions for the given image (image in the form of matrix)



```
def predict_tags(model, image, tokenizer, max_length):

    #print(image)

    in_text = '<start>'

    for ind in range(max_length):
        seq = tokenizer.texts_to_sequences([in_text])[0]
        seq = pad_sequences([seq], max_length)
        y_hat = model.predict([image, seq], verbose=0)
        y_hat = np.argmax(y_hat)
        word = get_word_from_tokenized_data(y_hat, tokenizer)
        if word is None:
            break
        in_text += " " + word
        if word == '<end>':
            break

    return in_text
```

Generate Captions for the given Image by prediction



```
def generate_caption(image):

    y_pred = predict_tags(model, image, tokenizer,max_length)

    print("=======================")
    print("Tags predicted from Model")
    print(y_pred)
    print("=======================")

    prod_tag_list = y_pred.replace("<start>",'')
    prod_tag_list = prod_tag_list.replace("end",'')
    prod_tag_list = prod_tag_list.strip()
    prod_tag_list = prod_tag_list.split(" ")
    product_tags = []
    for ptag in prod_tag_list:
        if ptag in unique_tag_dic:
            product_tags.append(unique_tag_dic[ptag])

    product_tags = list(dict.fromkeys(product_tags))

    original_prod_list = [get_original_tag_value(prodtag) for prodtag in product_tags]

    return original_prod_list
```

f) *Validate data using BLEU Score*

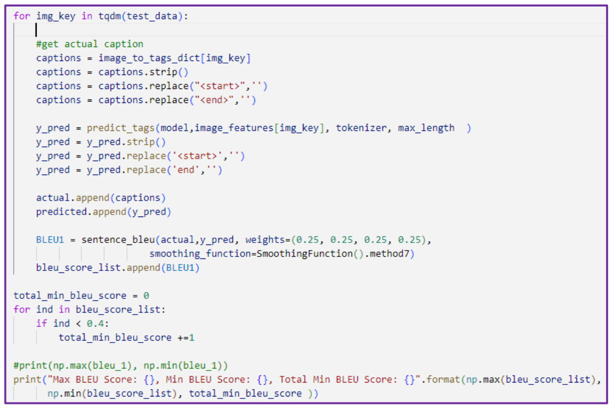
BLEU (Bilingual Evaluation Understudy) is a measurement of the difference between an automatic translation and human-created reference translations of the same source sentence.

We will be using sentence_bleu() method for the actual and predicted tags, and performing 4gm validation with each 0.25 value.

Low Score is less than 0.4, and High Score is greater than 0.4

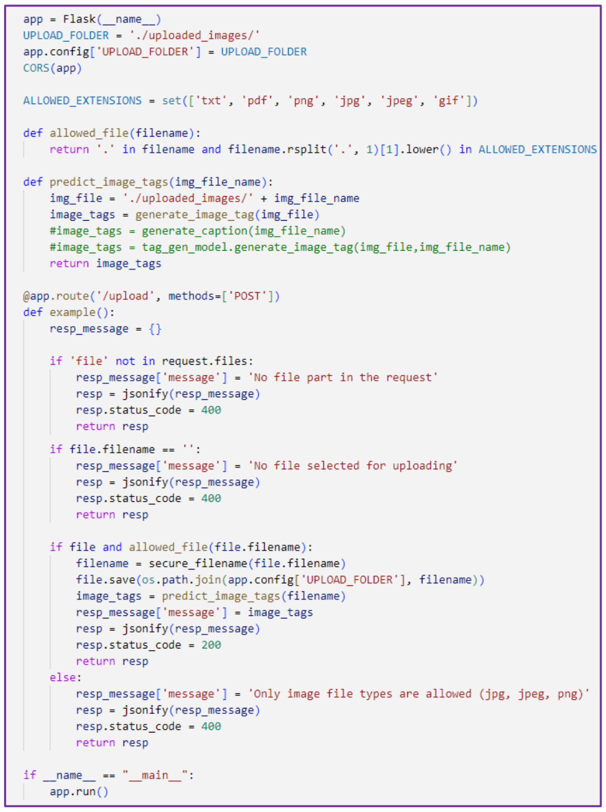
LOW SCORE < 0.4
GOOD SCORE > 0.4

Result: The model predicts only one record less than 0.4 score and performing well on test data.



```
for img_key in tqdm(test_data):

    #get actual caption
    captions = image_to_tags_dict[img_key]
    captions = captions.strip()
    captions = captions.replace("<start>",'')
    captions = captions.replace("<end>",'')

    y_pred = predict_tags(model,image_features[img_key], tokenizer, max_length  )
    y_pred = y_pred.strip()
    y_pred = y_pred.replace('<start>','')
    y_pred = y_pred.replace('end','')

    actual.append(captions)
    predicted.append(y_pred)

    BLEU1 = sentence_bleu(actual,y_pred, weights=(0.25, 0.25, 0.25, 0.25),
                          smoothing_function=SmoothingFunction().method7)
    bleu_score_list.append(BLEU1)

total_min_bleu_score = 0
for ind in bleu_score_list:
    if ind < 0.4:
        total_min_bleu_score +=1

#print(np.max(bleu_1), np.min(bleu_1))
print("Max BLEU Score: {}, Min BLEU Score: {}, Total Min BLEU Score: {}".format(np.max(bleu_score_list),
      np.min(bleu_score_list), total_min_bleu_score ))
```

2. REST API Layer



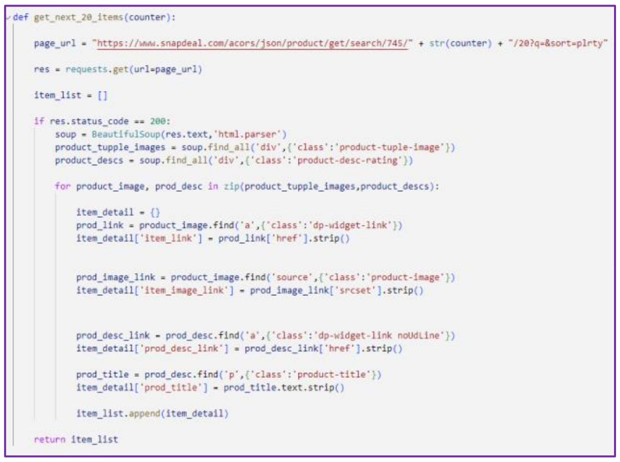
```
app = Flask(__name__)
UPLOAD_FOLDER = './uploaded_images/'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
CORS(app)

ALLOWED_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'])

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def predict_image_tags(img_file_name):
    img_file = './uploaded_images/' + img_file_name
    image_tags = generate_image_tag(img_file)
    #image_tags = generate_caption(img_file_name)
    #image_tags = tag_gen_model.generate_image_tag(img_file,img_file_name)
    return image_tags

@app.route('/upload', methods=['POST'])
def example():
    resp_message = {}

    if 'file' not in request.files:
        resp_message['message'] = 'No file part in the request'
        resp = jsonify(resp_message)
        resp.status_code = 400
        return resp
    if file.filename == '':
        resp_message['message'] = 'No file selected for uploading'
        resp = jsonify(resp_message)
        resp.status_code = 400
        return resp

    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        image_tags = predict_image_tags(filename)
        resp_message['message'] = image_tags
        resp = jsonify(resp_message)
        resp.status_code = 200
        return resp
    else:
        resp_message['message'] = 'Only image file types are allowed (jpg, jpeg, png)'
        resp = jsonify(resp_message)
        resp.status_code = 400
        return resp

if __name__ == "__main__":
    app.run()
```

3. Dataset Web Scraping Scripts

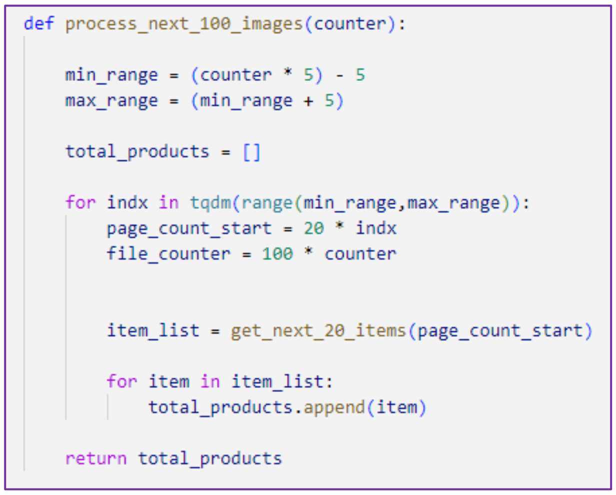
The Web scraping script is to download clothes images (particularly) women's Top along with some details such as type of sleeves, type of neck, type of top.

The script has two parts:

Part 1: Create multiple CSV files which contain image URLs and image captions. Each multiple CSV files then combine in one for preprocessing such as remove duplicate URLs.

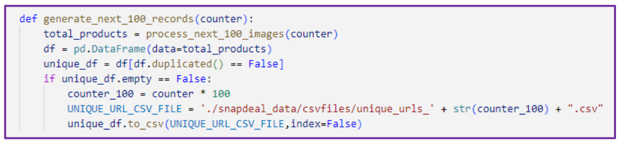


```
def get_next_20_items(counter):

    page_url = "https://www.snapdeal.com/acors/json/product/get/search/745/" + str(counter) + "/20?q=&sort=plrty"

    res = requests.get(url=page_url)

    item_list = []

    if res.status_code == 200:
        soup = BeautifulSoup(res.text,'html.parser')
        product_tupple_images = soup.find_all('div',{'class':'product-tuple-image'})
        product_descs = soup.find_all('div',{'class':'product-desc-rating'})

        for product_image, prod_desc in zip(product_tupple_images,product_descs):

            item_detail = {}
            prod_link = product_image.find('a',{'class':'dp-widget-link'})
            item_detail['item_link'] = prod_link['href'].strip()


            prod_image_link = product_image.find('source',{'class':'product-image'})
            item_detail['item_image_link'] = prod_image_link['srcset'].strip()


            prod_desc_link = prod_desc.find('a',{'class':'dp-widget-link noUdLine'})
            item_detail['prod_desc_link'] = prod_desc_link['href'].strip()

            prod_title = prod_desc.find('p',{'class':'product-title'})
            item_detail['prod_title'] = prod_title.text.strip()

            item_list.append(item_detail)

    return item_list
```



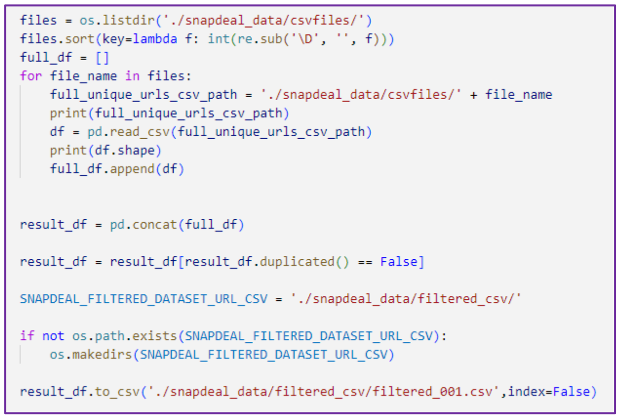
```
def process_next_100_images(counter):

    min_range = (counter * 5) - 5
    max_range = (min_range + 5)

    total_products = []

    for indx in tqdm(range(min_range,max_range)):
        page_count_start = 20 * indx
        file_counter = 100 * counter


        item_list = get_next_20_items(page_count_start)

        for item in item_list:
            total_products.append(item)

    return total_products
```

Generate in batch



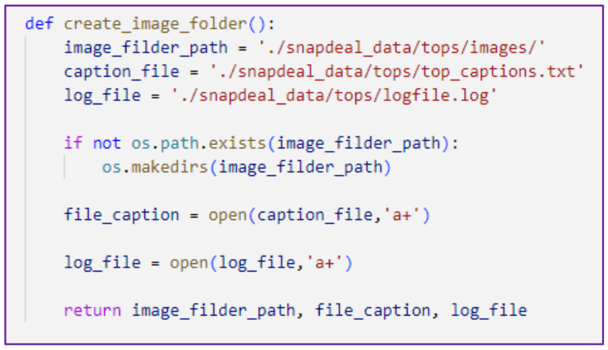
```
def generate_next_100_records(counter):
    total_products = process_next_100_images(counter)
    df = pd.DataFrame(data=total_products)
    unique_df = df[df.duplicated() == False]
    if unique_df.empty == False:
        counter_100 = counter * 100
        UNIQUE_URL_CSV_FILE = './snapdeal_data/csvfiles/unique_urls_' + str(counter_100) + ".csv"
        unique_df.to_csv(UNIQUE_URL_CSV_FILE,index=False)
```

Remove duplicated URLs



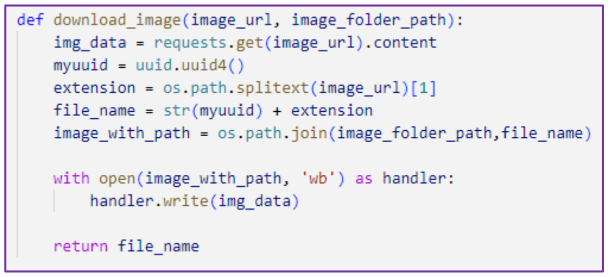
```
files = os.listdir('./snapdeal_data/csvfiles/')
files.sort(key=lambda f: int(re.sub('\D', '', f)))
full_df = []
for file_name in files:
    full_unique_urls_csv_path = './snapdeal_data/csvfiles/' + file_name
    print(full_unique_urls_csv_path)
    df = pd.read_csv(full_unique_urls_csv_path)
    print(df.shape)
    full_df.append(df)


result_df = pd.concat(full_df)

result_df = result_df[result_df.duplicated() == False]

SNAPDEAL_FILTERED_DATASET_URL_CSV = './snapdeal_data/filtered_csv/'

if not os.path.exists(SNAPDEAL_FILTERED_DATASET_URL_CSV):
    os.makedirs(SNAPDEAL_FILTERED_DATASET_URL_CSV)

result_df.to_csv('./snapdeal_data/filtered_csv/filtered_001.csv',index=False)
```

Part 2: Browse individual URLs from the CSV, then download and list all the image captions in text file

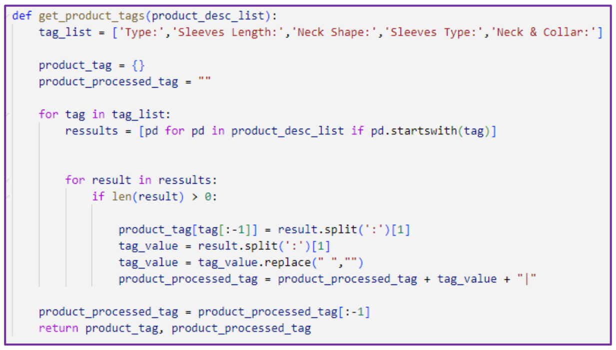


```
def create_image_folder():
    image_filder_path = './snapdeal_data/tops/images/'
    caption_file = './snapdeal_data/tops/top_captions.txt'
    log_file = './snapdeal_data/tops/logfile.log'

    if not os.path.exists(image_filder_path):
        os.makedirs(image_filder_path)

    file_caption = open(caption_file,'a+')

    log_file = open(log_file,'a+')

    return image_filder_path, file_caption, log_file
```

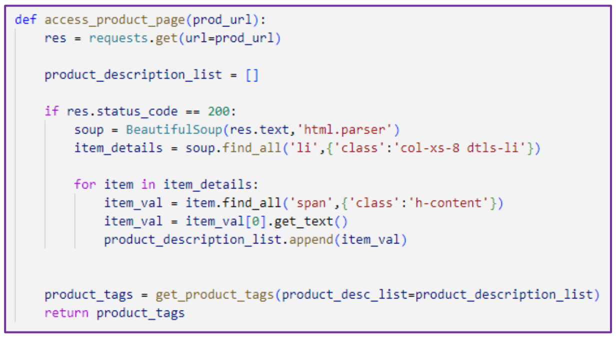
Download Each image from the unique URL csv file and generate captions or tags from the URL



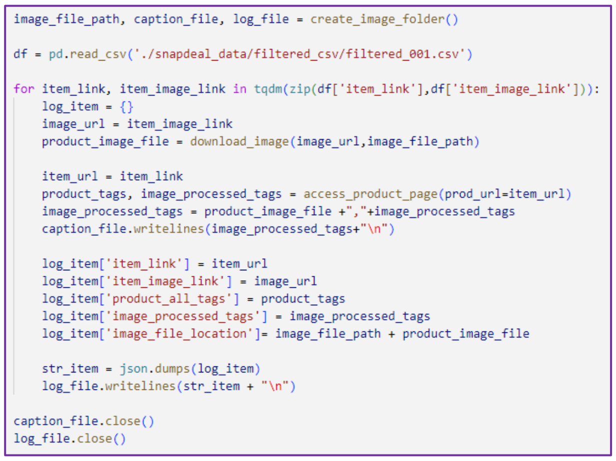
```python
def download_image(image_url, image_folder_path):
    img_data = requests.get(image_url).content
    myuuid = uuid.uuid4()
    extension = os.path.splitext(image_url)[1]
    file_name = str(myuuid) + extension
    image_with_path = os.path.join(image_folder_path,file_name)

    with open(image_with_path, 'wb') as handler:
        handler.write(img_data)

    return file_name
```



```python
def get_product_tags(product_desc_list):
    tag_list = ['Type:','Sleeves Length:','Neck Shape:','Sleeves Type:','Neck & Collar:']

    product_tag = {}
    product_processed_tag = ""

    for tag in tag_list:
        ressults = [pd for pd in product_desc_list if pd.startswith(tag)]


        for result in ressults:
            if len(result) > 0:

                product_tag[tag[:-1]] = result.split(':')[1]
                tag_value = result.split(':')[1]
                tag_value = tag_value.replace(" ","")
                product_processed_tag = product_processed_tag + tag_value + "|"

    product_processed_tag = product_processed_tag[:-1]
    return product_tag, product_processed_tag
```



```python
def access_product_page(prod_url):
    res = requests.get(url=prod_url)

    product_description_list = []

    if res.status_code == 200:
        soup = BeautifulSoup(res.text,'html.parser')
        item_details = soup.find_all('li',{'class':'col-xs-8 dtls-li'})

        for item in item_details:
            item_val = item.find_all('span',{'class':'h-content'})
            item_val = item_val[0].get_text()
            product_description_list.append(item_val)


    product_tags = get_product_tags(product_desc_list=product_description_list)
    return product_tags
```



```python
image_file_path, caption_file, log_file = create_image_folder()

df = pd.read_csv('./snapdeal_data/filtered_csv/filtered_001.csv')

for item_link, item_image_link in tqdm(zip(df['item_link'],df['item_image_link'])):
    log_item = {}
    image_url = item_image_link
    product_image_file = download_image(image_url,image_file_path)

    item_url = item_link
    product_tags, image_processed_tags = access_product_page(prod_url=item_url)
    image_processed_tags = product_image_file +","+image_processed_tags
    caption_file.writelines(image_processed_tags+"\n")

    log_item['item_link'] = item_url
    log_item['item_image_link'] = image_url
    log_item['product_all_tags'] = product_tags
    log_item['image_processed_tags'] = image_processed_tags
    log_item['image_file_location']= image_file_path + product_image_file

    str_item = json.dumps(log_item)
    log_file.writelines(str_item + "\n")

caption_file.close()
log_file.close()
```

VIII. Input Dataset & Tools

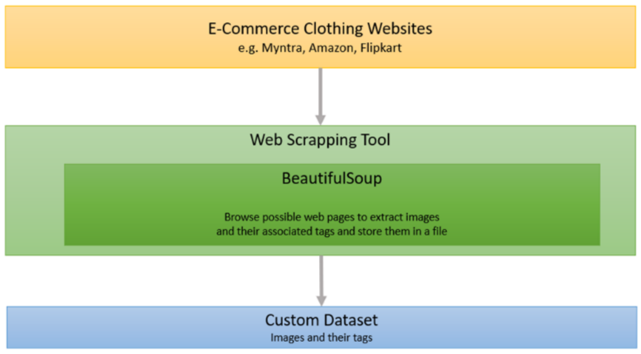
We are planning to use the following datasets to train the model.

Flickr8k dataset:

Initially, we will try to train our model on the publicly available Flickr8k dataset. The dataset contains approximately 8000 images with captions.

Custom e-Commerce Clothing dataset:

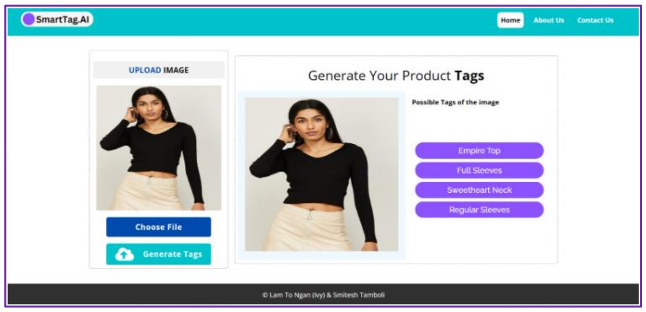
To apply in e-Commerce use case, we are also planning to create our own custom dataset that contains approximately 8000 specific kinds of clothing images with appropriate tags. We will use our image caption generation model on this dataset to predict the correct captions or tags for the given input image. To collect the images and captions for the training, we are planning to use the web scraping technique using BeautifulSoup module.

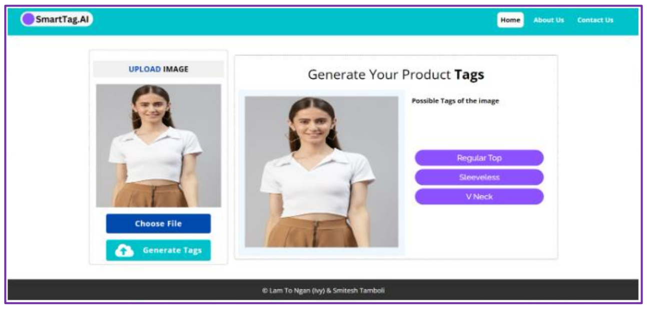

E-Commerce Clothing Websites
e.g. Myntra, Amazon, Flipkart
Web Scrapping Tool
BeautifulSoup
Browse possible web pages to extract images and their associated tags and store them in a file
Custom Dataset
Images and their tags

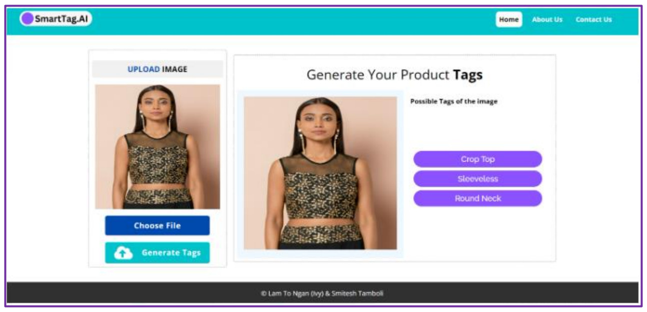
Toolsets:

- Artificial Intelligence Toolsets:
  - ✓ Programming Language: Python
  - ✓ Expected Algorithms: VGG16, ResNet, LSTM, GRU, Numpy, Pandas, NLTK (BLEU score)
  - ✓ Additional Modules: TensorFlow, Keras, Tqdm, Pickle, OS, BeautifulSoup, Flask
- Website or App Development:
  - ✓ Scripting and Programming Languages: HTML, CSS, Javascript, Android (Optionally for App)
- Deployment Technologies:
  - ✓ Deployment Tools: Docker and Heroku
  - ✓ Collaboration and Version Control: GitHub

IX. Project Outcome

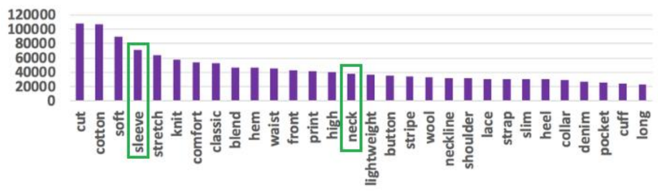






These are some of the results coming out from our project, which is a website where the users can click button "**Choose File**" to upload the image that they want to generate the product tags, then click button "**Generate Tags**" to get the auto-generated tags on the right-hand side associated to the input image...

There are many attributes related to clothes that we can consider generating more sophisticated captions for the clothes as shown in below figure.



However, with the limit time of our project with the limit of time, we are planning to focus on just a few attributes which is: **Type of top**, **type of sleeves** and **neck shape**.

X. Work Breakdown Structure (WBS)

This project is planned to be completed within 2 months with the WBS for two members in our group as below:

No.	Work items	PIC	
		Smitesh	Ivy
1	Project Asbtract	x	x
2	Project Proposal	x	x
3	Summary Research Paper	x	x
4	Proof of Concept (POC)	x	
5	Dataset Building	x	x
6	Coding	x	x
	6.1 User Interface	x	
	6.2 REST API Layer	x	
	6.3 Feature Extraction Model		x
	6.4 Encoder Model		x
	6.5 Decoder Model	x	
7	Testing	x	x
8	Project Report	x	x
9	Demo	x	x
10	Slides Deck		x

XI. Conclusion and Future Direction

This work presents a proposal to develop an AI application which can be used to auto-generate the caption for clothes images. This application will be useful for e-commerce industry to save time for writing captions for their products with the scalability.

Our system is using CNN and RNN classes to build 3 models: Feature Extraction Model, Encoder Model and Decoder Model to respectively extract features of the images, preprocess the captions and predict the caption for an input image.

This application can be extended in the future to auto-generate the captions with more attributes of the clothes and can be added products selling points to make the captions more sophisticated and more attractive to increase the impression and conversion rate on e-commerce platforms.

XII. References

"*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*" – Oct. 22 2019 by Aurélien Géron

"*Image Caption Generator*" – by Megha J Panicker, Vikas Upadhayay, Gunjan Sethi, Vrinda Mathur – Online – Available at: https://www.ijitee.org/wp-content/uploads/papers/v10i3/C83830110321.pdf

"*DigitalSreeni*" - YouTube Channel – Available at: https://www.youtube.com/@DigitalSreeni