Architecture Document

Image Captioning

Revision No: 1.0

Last Date of Revision: 23/11/2023.

Document Version Control-

Date Issued	Version	Description	Author
23/11/2023	1.0	LLD – V- 1.0	Mahesh. A

Contents -

Abstract

By using the pre-trained of Image Captioning model user can get the auto caption for the uploaded image…

1 Introduction

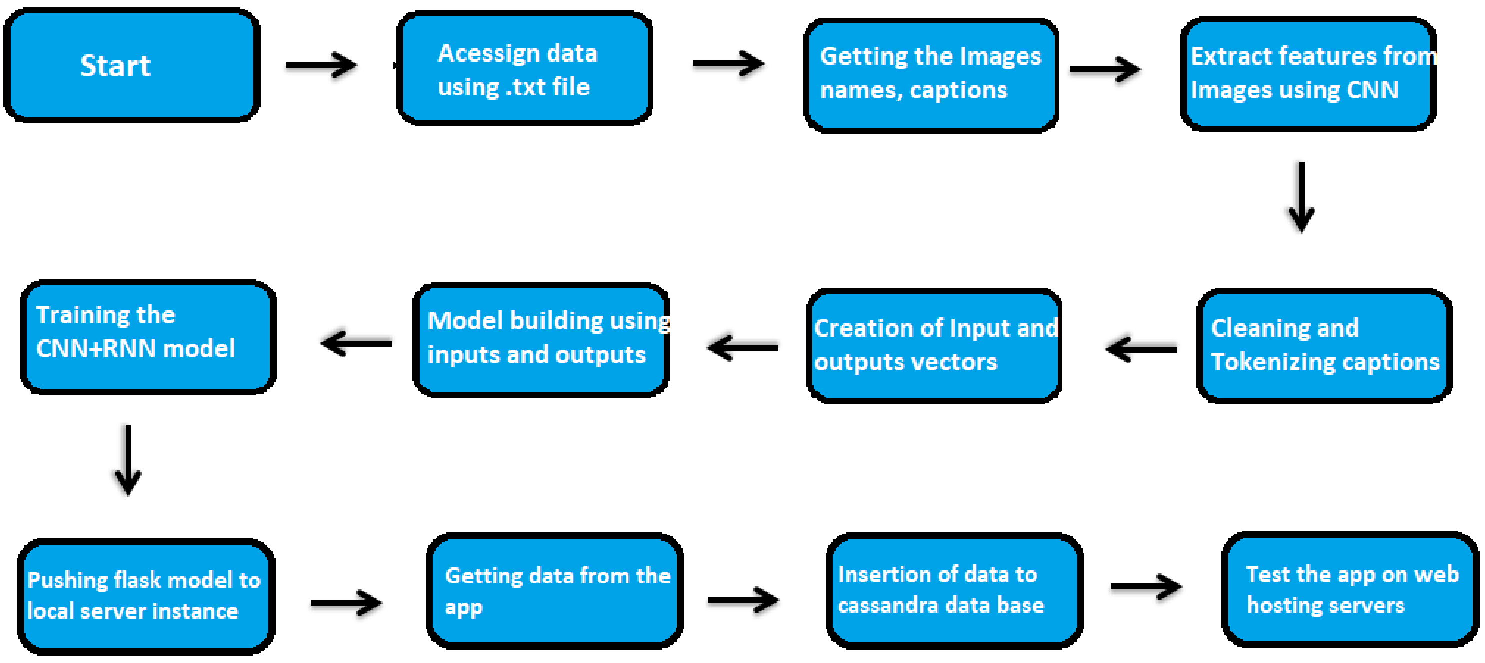
1.1. What is document required?

The goal of Architecture is to give the internal logical design of the actual programmed code for Image captioning model. Describes the captioned relations with images .It describes the modules so that the programmer can directly code the program from the document.

1.2. Scope

Architecture document is a component-level design process that follows a step-by-step process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

2 Architecture

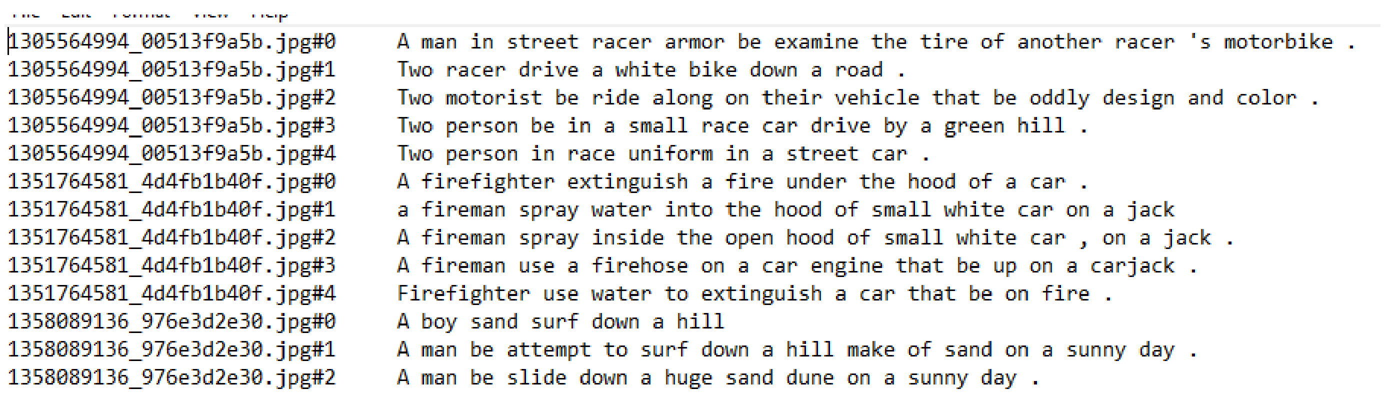

Start
Acessign data using .txt file
Getting the Images names, captions
Extract features from Images using CNN
Cleaning and Tokenizing captions
Creation of Input and outputs vectors
Model building using inputs and outputs
Training the CNN+RNN model
Pushing flask model to local server instance
Getting data from the app
Insertion of data to cassandra data base
Test the app on web hosting servers

3 Architecture Description

3.1 Data Accessing

We can access the data in the from the download link as available in the project.

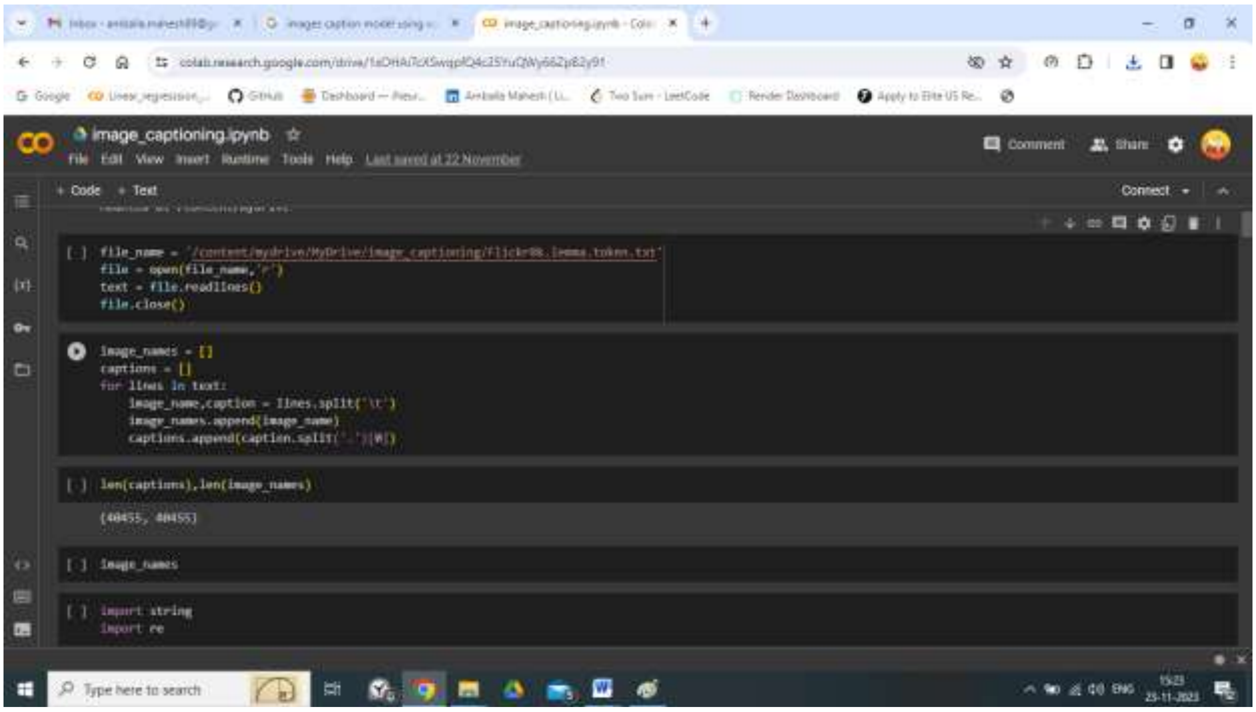
We load the data to the framework using the pandas read function.



```
1305564994_00513f9a5b.jpg#0    A man in street racer armor be examine the tire of another racer 's motorbike .
1305564994_00513f9a5b.jpg#1    Two racer drive a white bike down a road .
1305564994_00513f9a5b.jpg#2    Two motorist be ride along on their vehicle that be oddly design and color .
1305564994_00513f9a5b.jpg#3    Two person be in a small race car drive by a green hill .
1305564994_00513f9a5b.jpg#4    Two person in race uniform in a street car .
1351764581_4d4fb1b40f.jpg#0    A firefighter extinguish a fire under the hood of a car .
1351764581_4d4fb1b40f.jpg#1    a fireman spray water into the hood of small white car on a jack
1351764581_4d4fb1b40f.jpg#2    A fireman spray inside the open hood of small white car , on a jack .
1351764581_4d4fb1b40f.jpg#3    A fireman use a firehose on a car engine that be up on a carjack .
1351764581_4d4fb1b40f.jpg#4    Firefighter use water to extinguish a car that be on fire .
1358089136_976e3d2e30.jpg#0    A boy sand surf down a hill
1358089136_976e3d2e30.jpg#1    A man be attempt to surf down a hill make of sand on a sunny day .
1358089136_976e3d2e30.jpg#2    A man be slide down a huge sand dune on a sunny day .
```

3.2 Data Pre-Processing

By the usage of the different data manipulation techniques we will remove unwanted words by this reduce the dimensions and make all the features in numerical data type using Keras tokenizer to get the vector of caption and Keras Xception model to get features from images.



```
file_name = '/content/mydrive/MyDrive/image_captioning/Flickr8k.lemma.token.txt'
file = open(file_name,'r')
text = file.readlines()
file.close()
```

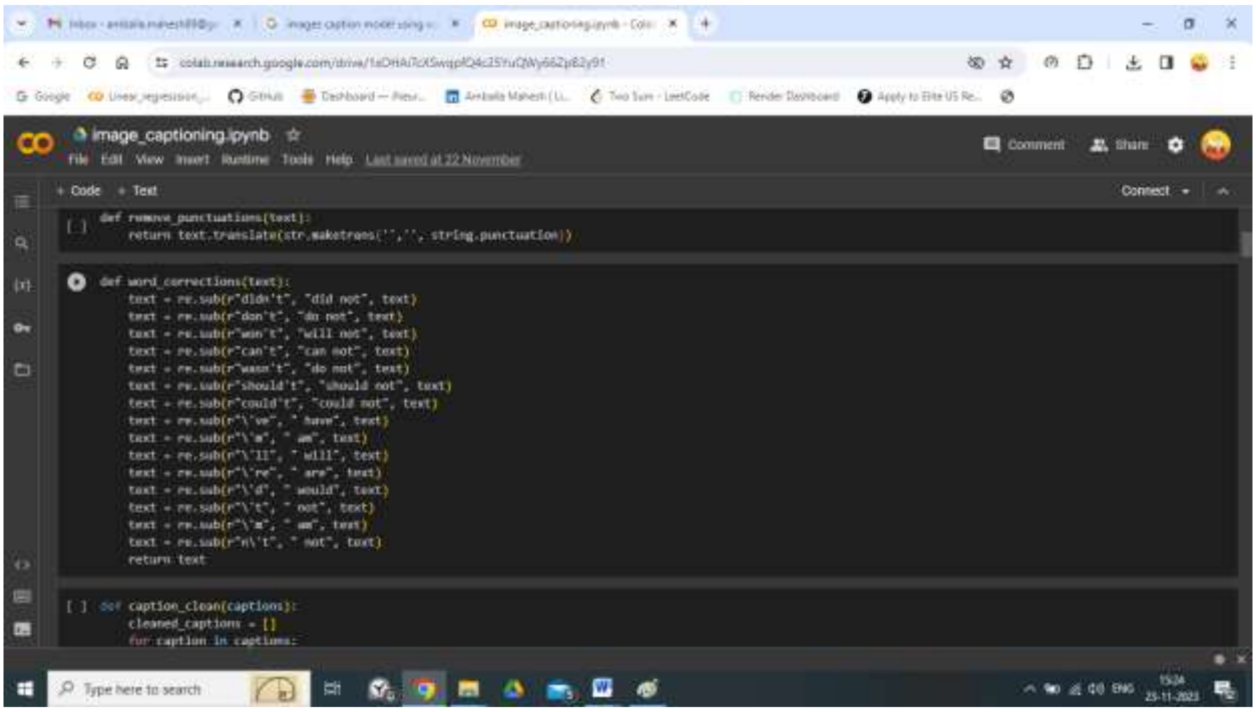
```
image_names = []
captions = []
for lines in text:
    image_name,caption = lines.split('\t')
    image_names.append(image_name)
    captions.append(caption.split('.')[0])
```

```
len(captions),len(image_names)
```

```
(40455, 40455)
```

```
image_names
```

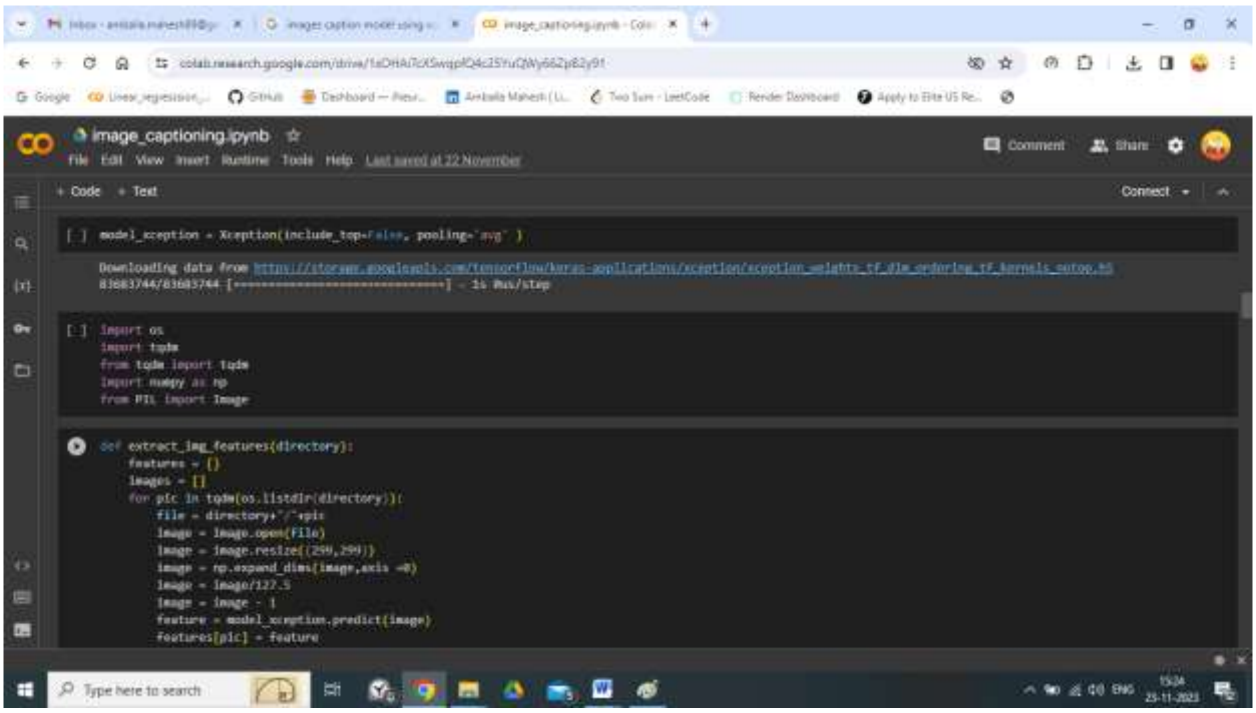
```
import string
import re
```



```
def remove_punctuations(text):
    return text.translate(str.maketrans('','', string.punctuation))
```

```
def word_corrections(text):
    text = re.sub(r"didn't", "did not", text)
    text = re.sub(r"don't", "do not", text)
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "can not", text)
    text = re.sub(r"wasn't", "do not", text)
    text = re.sub(r"should't", "should not", text)
    text = re.sub(r"could't", "could not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'m", " am", text)
    text = re.sub(r"n\'t", " not", text)
    return text
```

```
def caption_clean(captions):
    cleaned_captions = []
    for caption in captions:
```

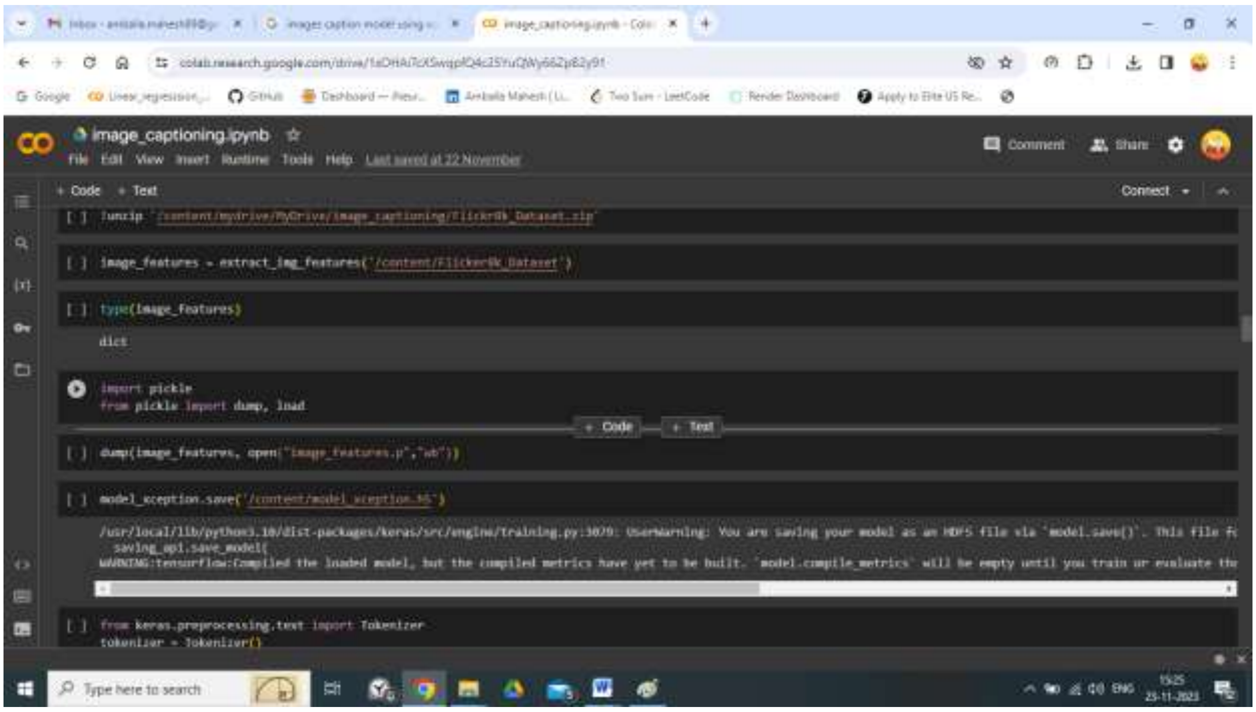


```
model_xception = Xception(include_top=False, pooling='avg' )
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 1s 0us/step
```

```
import os
import tqdm
from tqdm import tqdm
import numpy as np
from PIL import Image
```

```
def extract_img_features(directory):
    features = {}
    images = []
    for pic in tqdm(os.listdir(directory)):
        file = directory+"/"+pic
        image = Image.open(file)
        image = image.resize((299,299))
        image = np.expand_dims(image,axis =0)
        image = image/127.5
        image = image - 1
        feature = model_xception.predict(image)
        features[pic] = feature
```



```
!unzip '/content/mydrive/MyDrive/image_captioning/Flicker8k_Dataset.zip'
```

```
image_features = extract_img_features('/content/Flicker8k_Dataset')
```

```
type(image_features)
```

```
dict
```

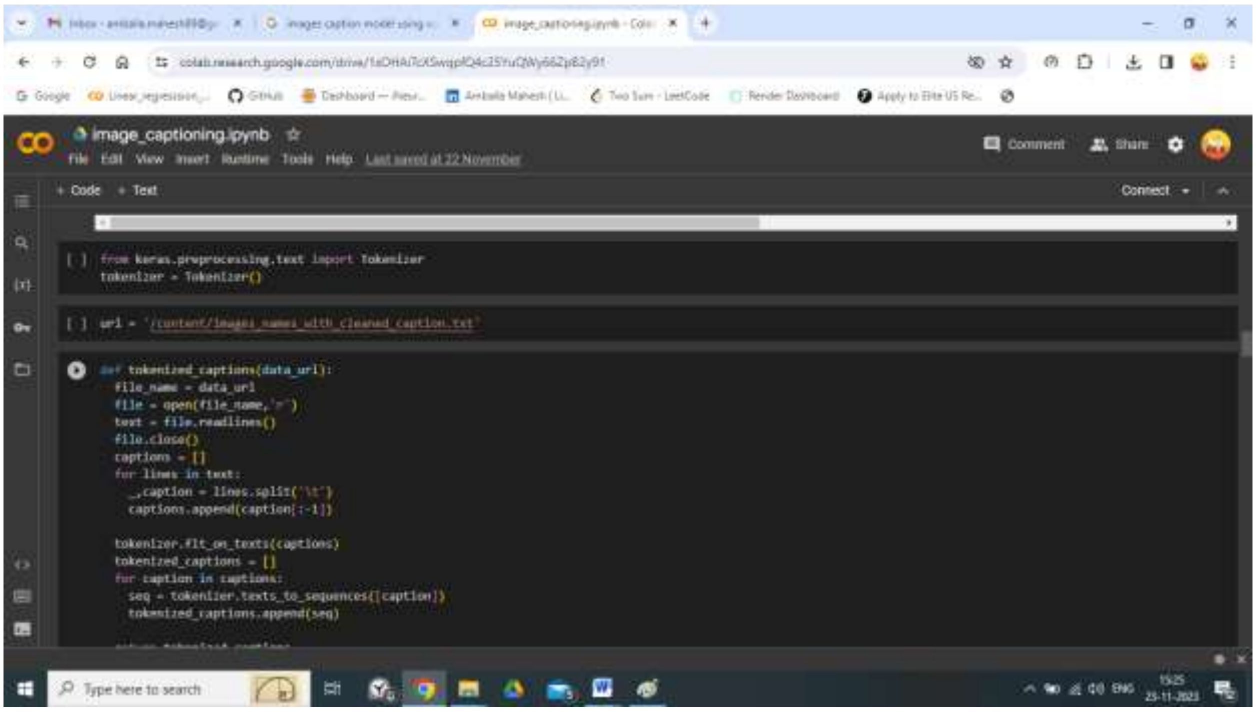
```
import pickle
from pickle import dump, load
```

```
dump(image_features, open("image_features.p","wb"))
```

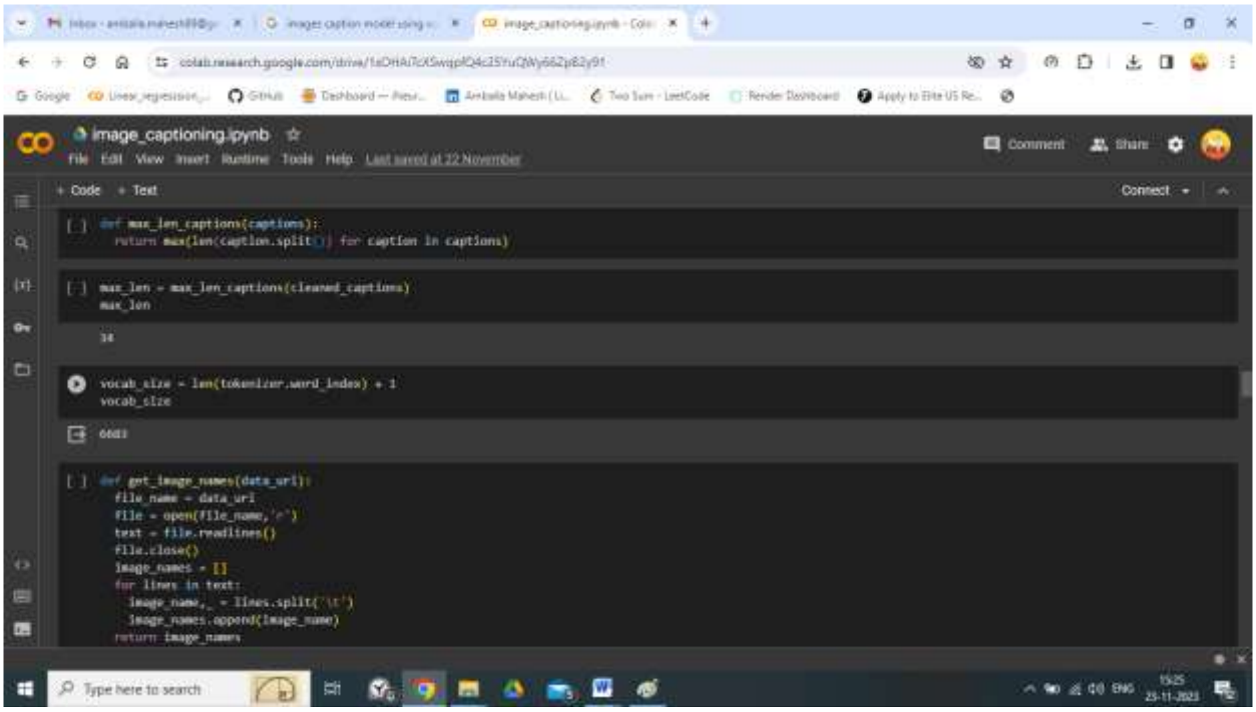
```
model_xception.save('/content/model_xception.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file fo
  saving_api.save_model(
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the
```

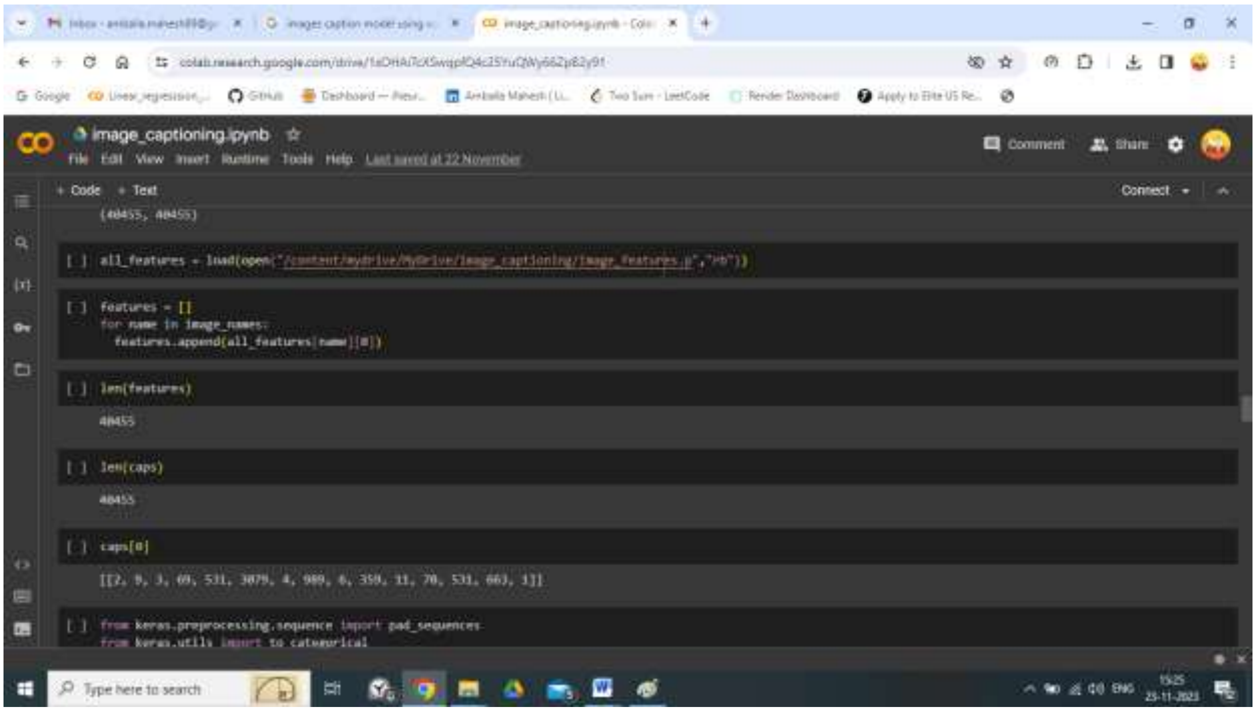
```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
```



```
[ ] from keras.preprocessing.text import Tokenizer
    tokenizer = Tokenizer()

[ ] url = '/content/images_names_with_cleaned_caption.txt'

    def tokenized_captions(data_url):
      file_name = data_url
      file = open(file_name,'r')
      text = file.readlines()
      file.close()
      captions = []
      for lines in text:
        _,caption = lines.split('\t')
        captions.append(caption[:-1])

      tokenizer.fit_on_texts(captions)
      tokenized_captions = []
      for caption in captions:
        seq = tokenizer.texts_to_sequences([caption])
        tokenized_captions.append(seq)
```



```
[ ] def max_len_captions(captions):
      return max(len(caption.split()) for caption in captions)

[ ] max_len = max_len_captions(cleaned_captions)
    max_len

    34

    vocab_size = len(tokenizer.word_index) + 1
    vocab_size

    6683

[ ] def get_image_names(data_url):
      file_name = data_url
      file = open(file_name,'r')
      text = file.readlines()
      file.close()
      image_names = []
      for lines in text:
        image_name,_ = lines.split('\t')
        image_names.append(image_name)
      return image_names
```


image_captioning.ipynb
File Edit View Insert Runtime Tools Help Last saved at 22 November
+ Code + Text
Comment Share
Connect

```
(40455, 40455)
```

```
all_features = load(open("/content/mydrive/MyDrive/image_captioning/image_features.p","rb"))
```

```
features = []
for name in image_names:
  features.append(all_features[name][0])
```

```
len(features)
```

```
40455
```

```
len(caps)
```

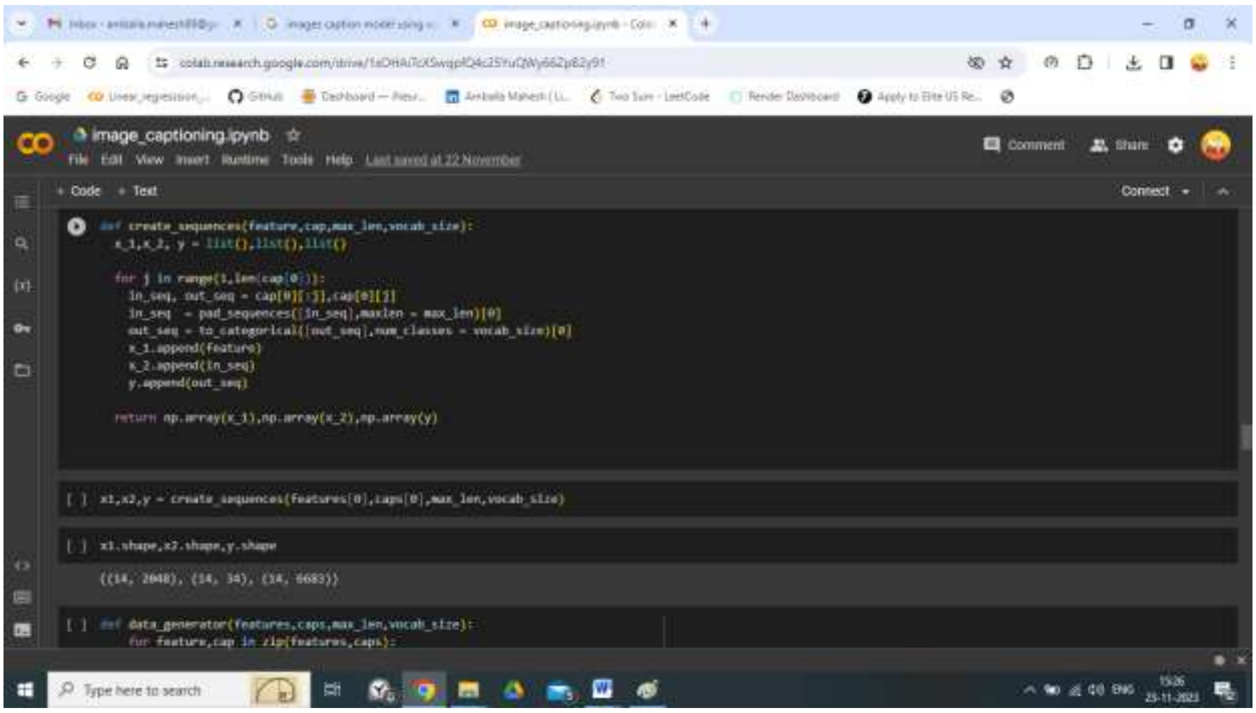
```
40455
```

```
caps[0]
```

```
[[2, 9, 3, 69, 531, 3079, 4, 989, 6, 359, 11, 70, 531, 663, 1]]
```

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

3.3 Creating Input and Output data



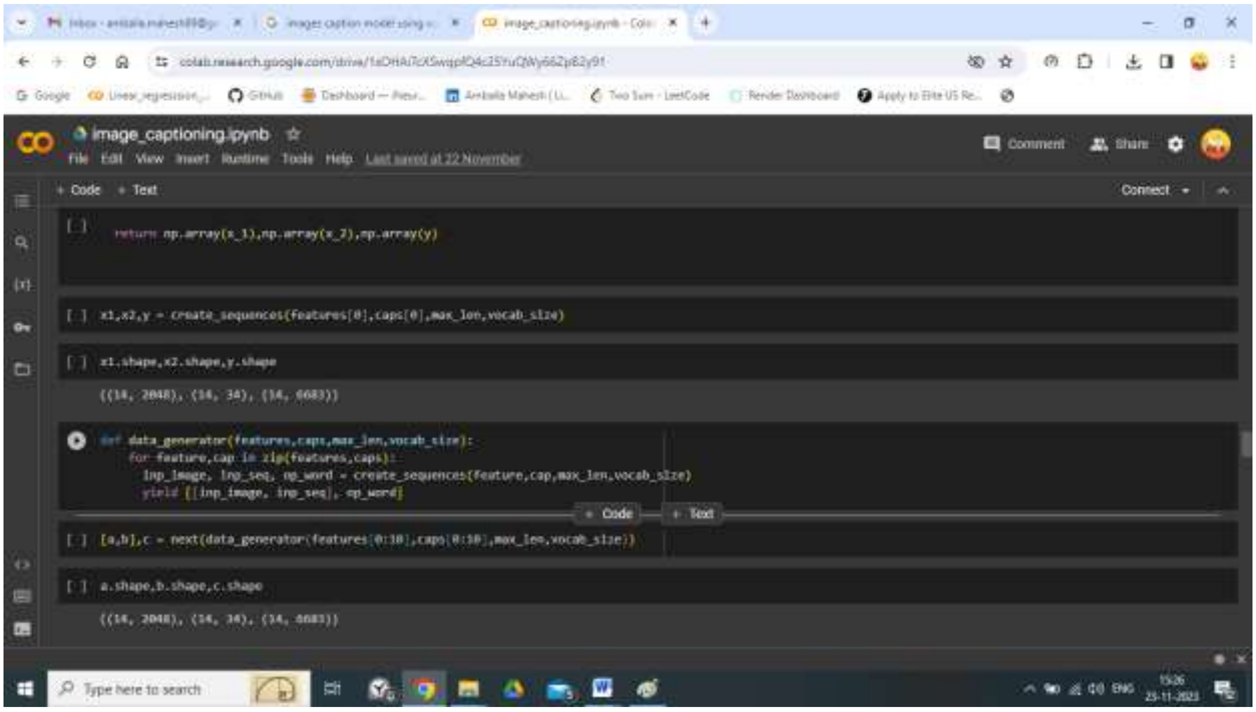
```
def create_sequences(feature,cap,max_len,vocab_size):
  x_1,x_2, y = list(),list(),list()

  for j in range(1,len(cap[0])):
    in_seq, out_seq = cap[0][:j],cap[0][j]
    in_seq  = pad_sequences([in_seq],maxlen = max_len)[0]
    out_seq = to_categorical([out_seq],num_classes = vocab_size)[0]
    x_1.append(feature)
    x_2.append(in_seq)
    y.append(out_seq)

  return np.array(x_1),np.array(x_2),np.array(y)
```

```
x1,x2,y = create_sequences(features[0],caps[0],max_len,vocab_size)
```

```
x1.shape,x2.shape,y.shape
```

```
((14, 2048), (14, 34), (14, 6683))
```

```
def data_generator(features,caps,max_len,vocab_size):
    for feature,cap in zip(features,caps):
```



```
  return np.array(x_1),np.array(x_2),np.array(y)
```

```
x1,x2,y = create_sequences(features[0],caps[0],max_len,vocab_size)
```

```
x1.shape,x2.shape,y.shape
```

```
((14, 2048), (14, 34), (14, 6683))
```

```
def data_generator(features,caps,max_len,vocab_size):
    for feature,cap in zip(features,caps):
      inp_image, inp_seq, op_word = create_sequences(feature,cap,max_len,vocab_size)
      yield [[inp_image, inp_seq], op_word]
```

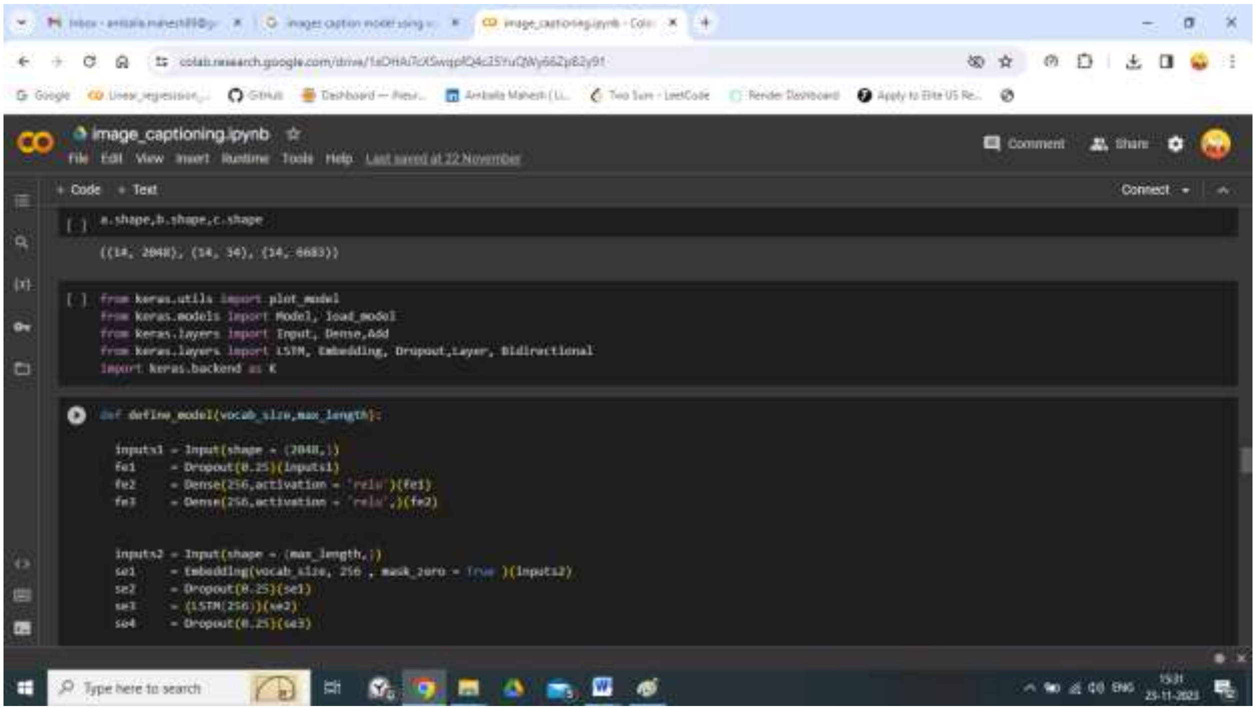
```
[a,b],c = next(data_generator(features[0:10],caps[0:10],max_len,vocab_size))
```

```
a.shape,b.shape,c.shape
```

```
((14, 2048), (14, 34), (14, 6683))
```

3.4 Model Building

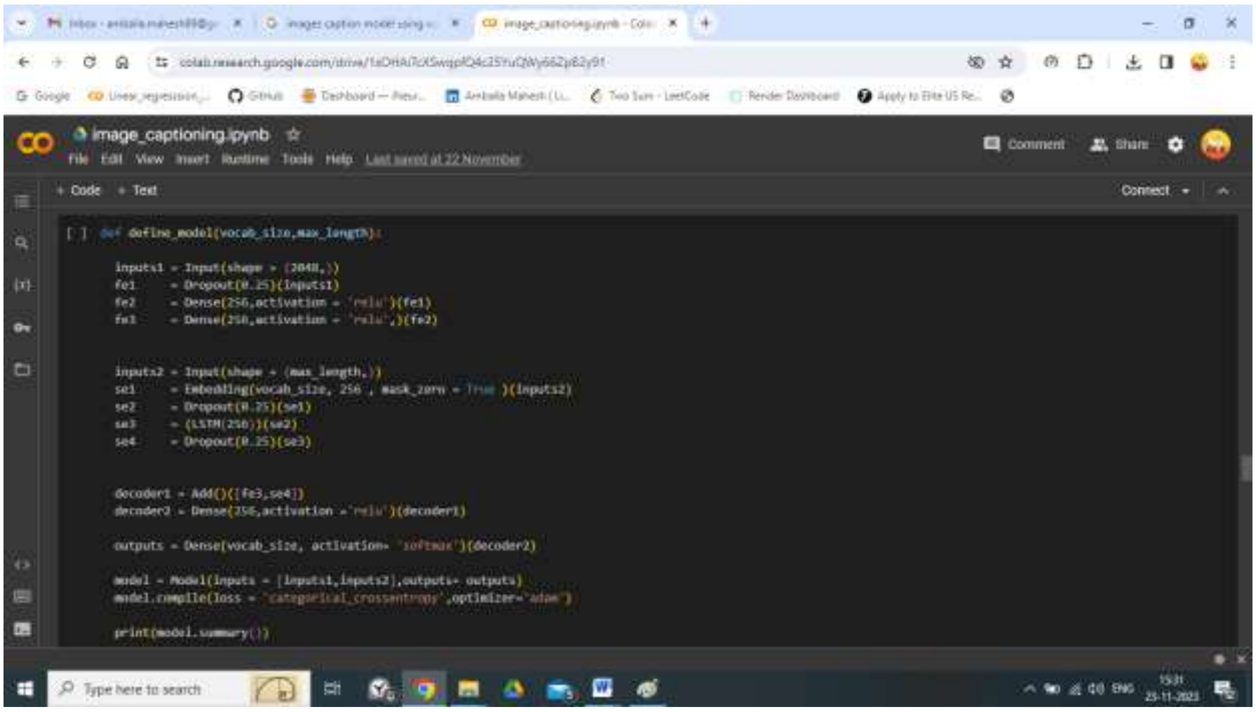
We create CNN + RNN base model using Keras as mentioned below.



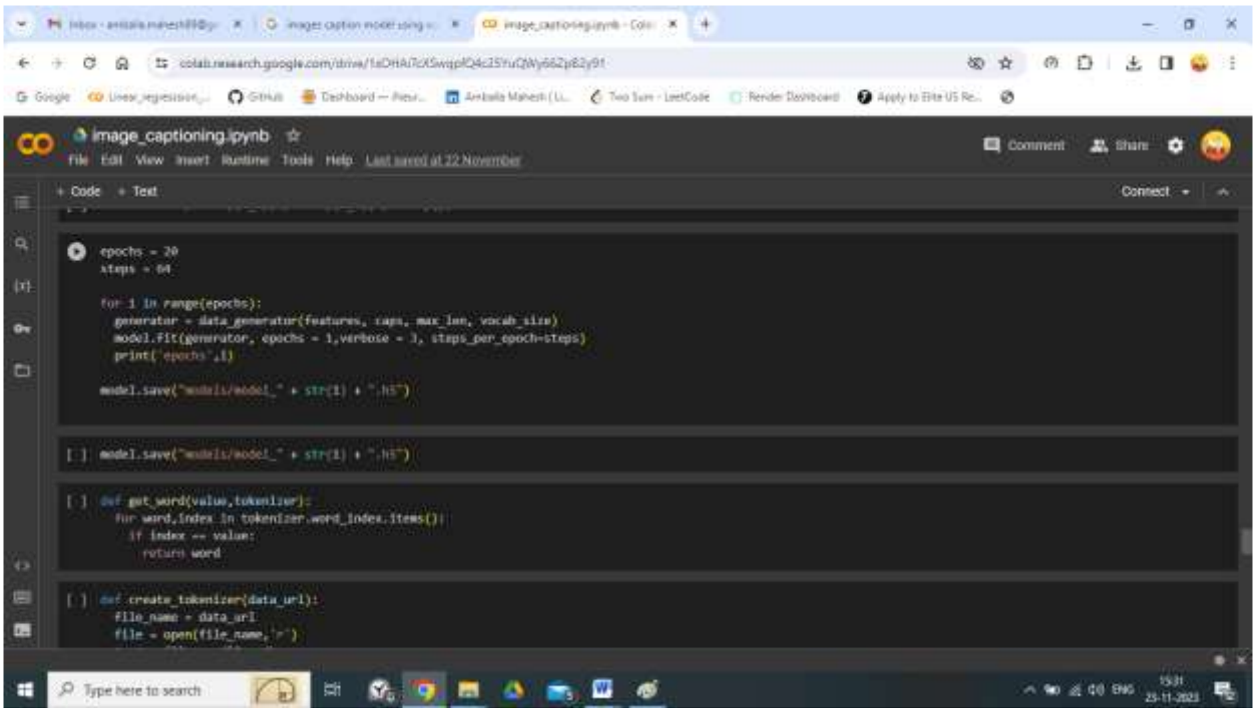
```
a.shape,b.shape,c.shape

((14, 2048), (14, 34), (14, 6683))
```

```
from keras.utils import plot_model
from keras.models import Model, load_model
from keras.layers import Input, Dense,Add
from keras.layers import LSTM, Embedding, Dropout,Layer, Bidirectional
import keras.backend as K
```

```
def define_model(vocab_size,max_length):

  inputs1 = Input(shape = (2048,))
  fe1     = Dropout(0.25)(inputs1)
  fe2     = Dense(256,activation = 'relu')(fe1)
  fe3     = Dense(256,activation = 'relu',)(fe2)


  inputs2 = Input(shape = (max_length,))
  se1     = Embedding(vocab_size, 256 , mask_zero = True )(inputs2)
  se2     = Dropout(0.25)(se1)
  se3     = (LSTM(256))(se2)
  se4     = Dropout(0.25)(se3)
```



```
def define_model(vocab_size,max_length):

  inputs1 = Input(shape = (2048,))
  fe1     = Dropout(0.25)(inputs1)
  fe2     = Dense(256,activation = 'relu')(fe1)
  fe3     = Dense(256,activation = 'relu',)(fe2)


  inputs2 = Input(shape = (max_length,))
  se1     = Embedding(vocab_size, 256 , mask_zero = True )(inputs2)
  se2     = Dropout(0.25)(se1)
  se3     = (LSTM(256))(se2)
  se4     = Dropout(0.25)(se3)


  decoder1 = Add()([fe3,se4])
  decoder2 = Dense(256,activation ='relu')(decoder1)

  outputs = Dense(vocab_size, activation= 'softmax')(decoder2)

  model = Model(inputs = [inputs1,inputs2],outputs= outputs)
  model.compile(loss = 'categorical_crossentropy',optimizer='adam')

  print(model.summary())
```



```
epochs = 20
steps = 64

for i in range(epochs):
  generator = data_generator(features, caps, max_len, vocab_size)
  model.fit(generator, epochs = 1,verbose = 1, steps_per_epoch=steps)
  print('epochs',i)

model.save("models/model_" + str(i) + ".h5")
```

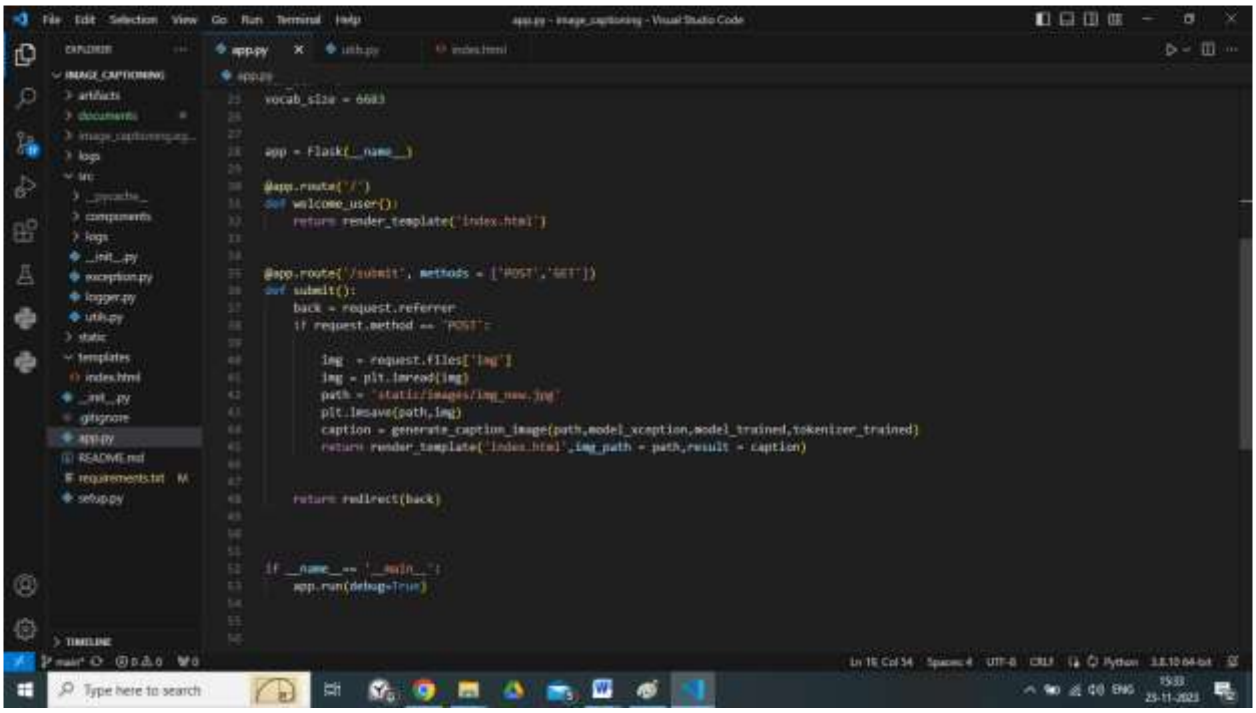
```
model.save("models/model_" + str(i) + ".h5")
```

```
def get_word(value,tokenizer):
  for word,index in tokenizer.word_index.items():
    if index == value:
      return word
```

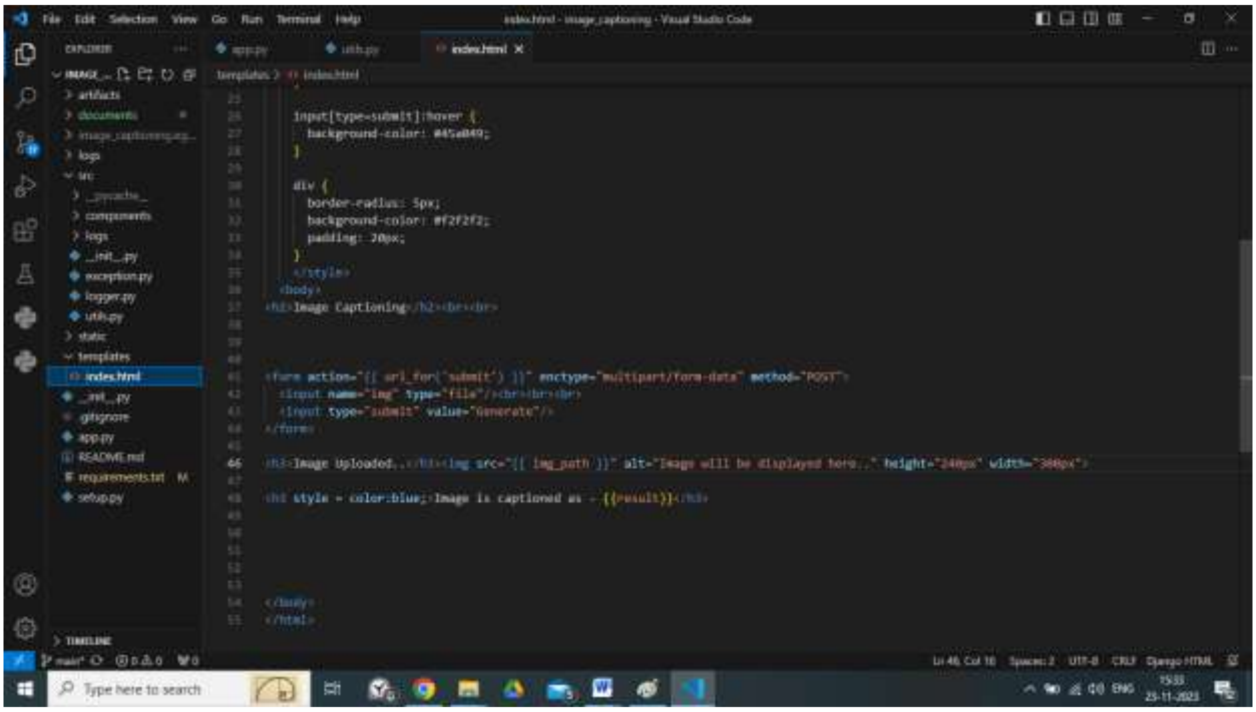
```
def create_tokenizer(data_url):
  file_name = data_url
  file = open(file_name,'r')
```

3.5 Create Front End User Module using flask

Once the model is created download and save the model and now we create GUI for front end user using the flask incorporated with HTML, CSS. Align and map the user data to the data base created. From user data create the data frame and load it to the model for the prediction the same prediction is send back to the user GUI and well saved in the data base (MySQL and Cassandra).



```python
vocab_size = 6683


app = Flask(__name__)

@app.route('/')
def welcome_user():
    return render_template('index.html')


@app.route('/submit', methods = ['POST','GET'])
def submit():
    back = request.referrer
    if request.method == 'POST':

        img  = request.files['img']
        img = plt.imread(img)
        path = 'static/images/img_new.jpg'
        plt.imsave(path,img)
        caption = generate_caption_image(path,model_xception,model_trained,tokenizer_trained)
        return render_template('index.html',img_path = path,result = caption)


    return redirect(back)



if __name__== '__main__':
    app.run(debug=True)
```



```html
    input[type=submit]:hover {
      background-color: #45a049;
    }

    div {
      border-radius: 5px;
      background-color: #f2f2f2;
      padding: 20px;
    }
    </style>
  <body>
<h2>Image Captioning</h2><br><br>



<form action="{{ url_for('submit') }}" enctype="multipart/form-data" method="POST">
  <input name="img" type="file"/><br><br><br>
  <input type="submit" value="Generate"/>
</form>

<h3>Image Uploaded...</h3><img src="{{ img_path }}" alt="Image will be displayed here..." height="240px" width="300px">

<h3 style = color:blue;>Image is captioned as - {{result}}</h3>





</body>
</html>
```

3.6 Testing the Model

- ✓ Verify whether the application is the loading on the local server instance.
- ✓ Verify whether the user can access the application.
- ✓ Verify the user can access the different fields for selection and can be visible
- ✓ Once the user selection the fields and made the submit
- ✓ Check the user can get the result or prediction.
- ✓ Once he gets the prediction.
- ✓ Check the data form the user and prediction from the model is loaded into the local MySQL and Cassandra Data base.
- ✓ Verify whether the application is the loading on the web service instance.
- ✓ Check the database and download the data…

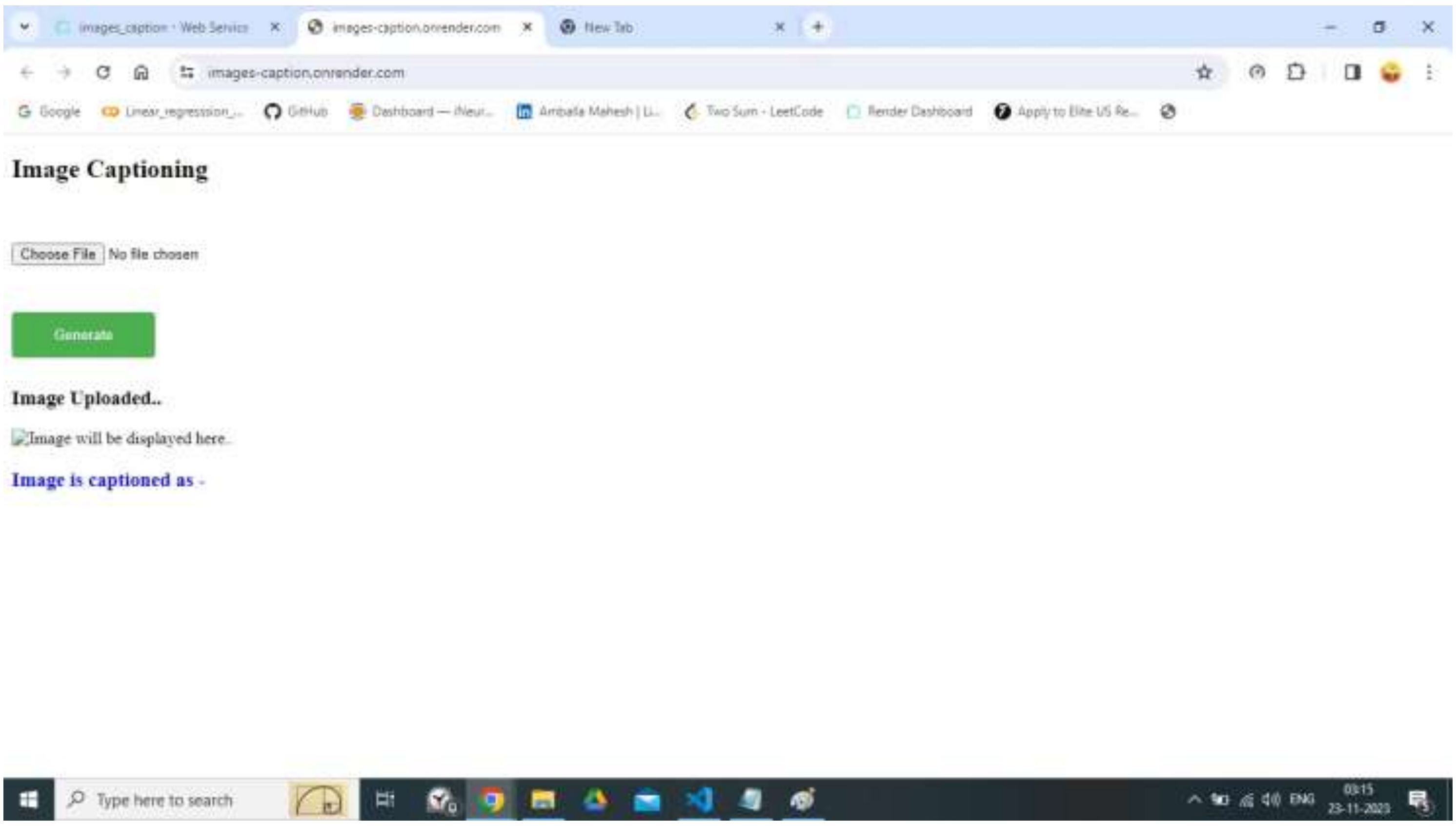

images_caption · Web Service
images-caption.onrender.com
New Tab
images-caption.onrender.com
Choose File
Generate
Type here to search

Image Captioning

No file chosen

Image Uploaded..

Image will be displayed here.

Image is captioned as -

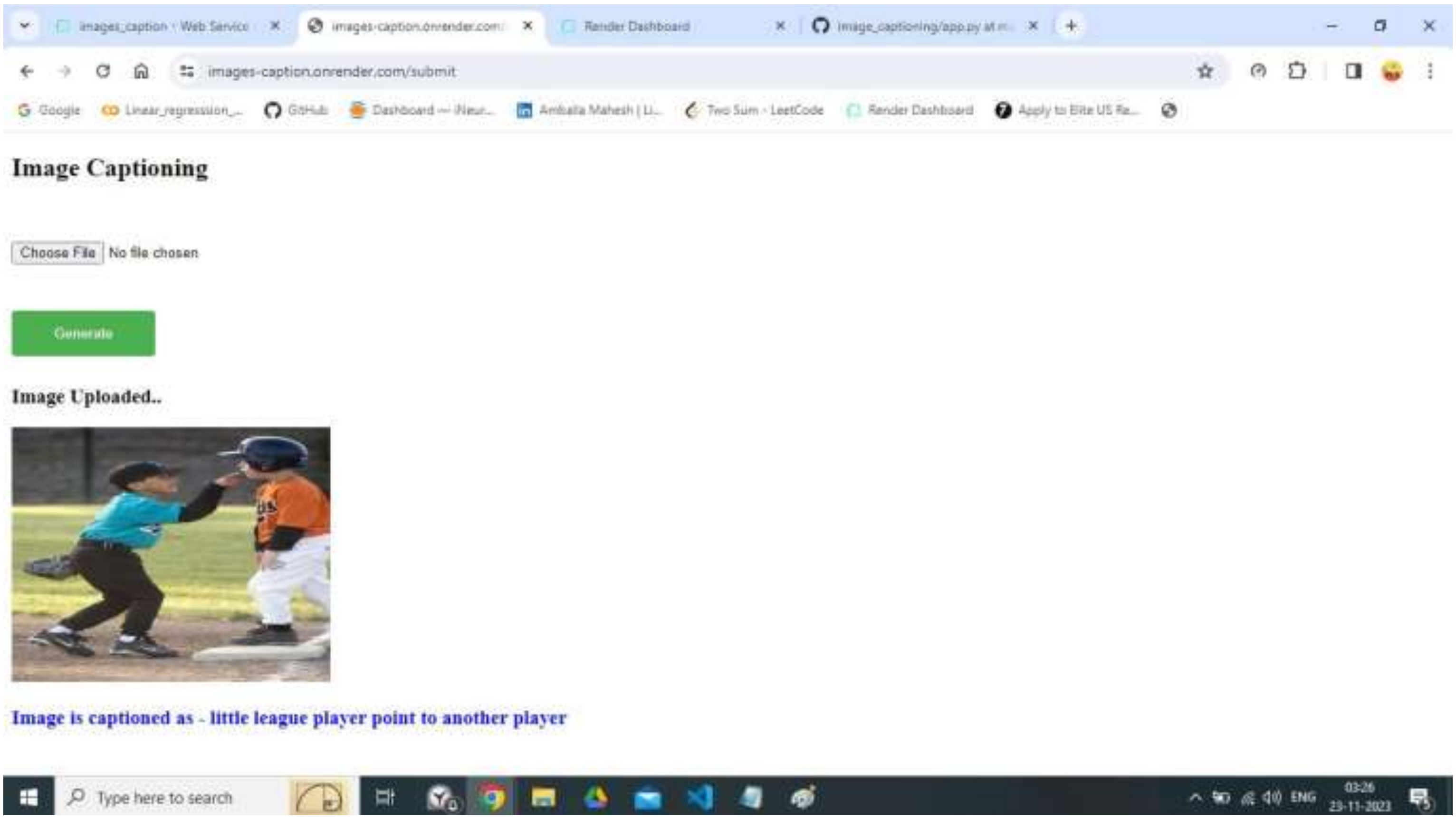


Image Captioning

Image Uploaded..

Image is captioned as - little league player point to another player

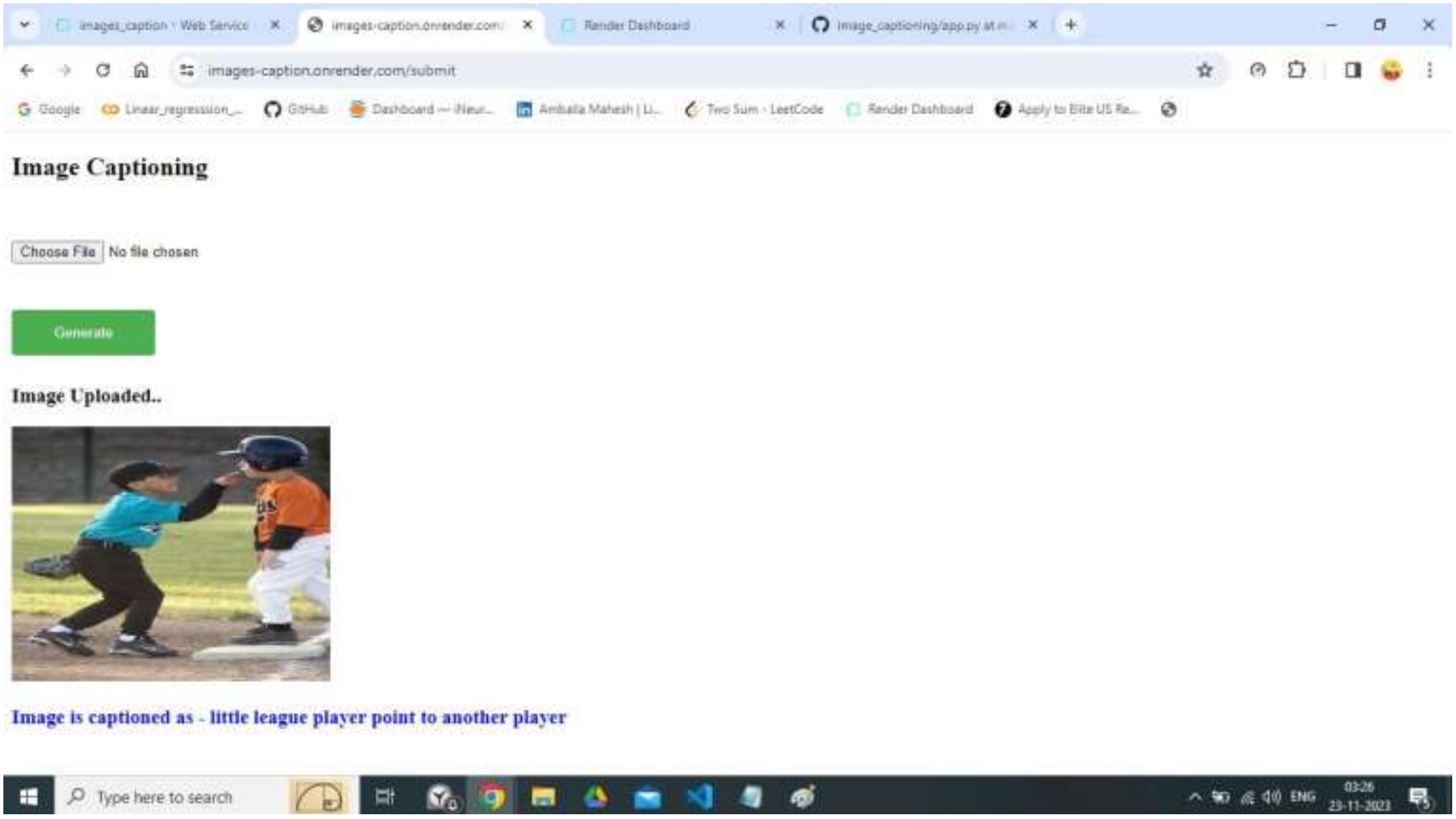
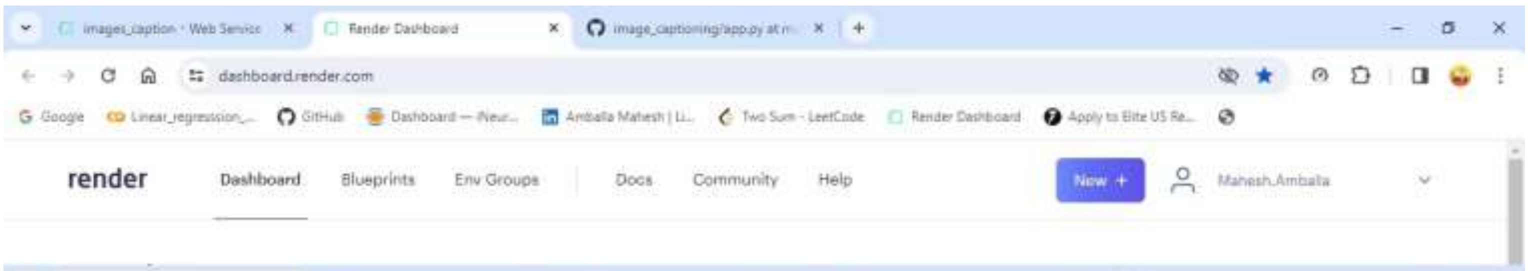
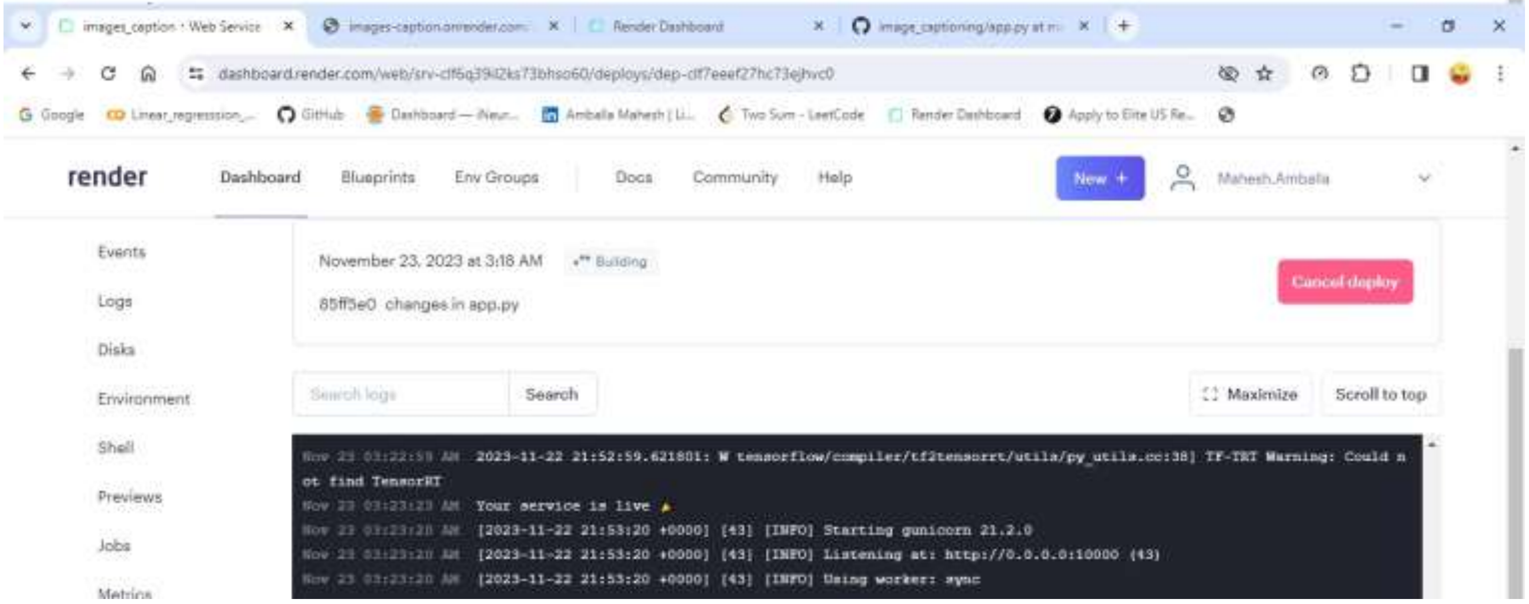


Image Captioning

Image Uploaded..

Image is captioned as - little league player point to another player

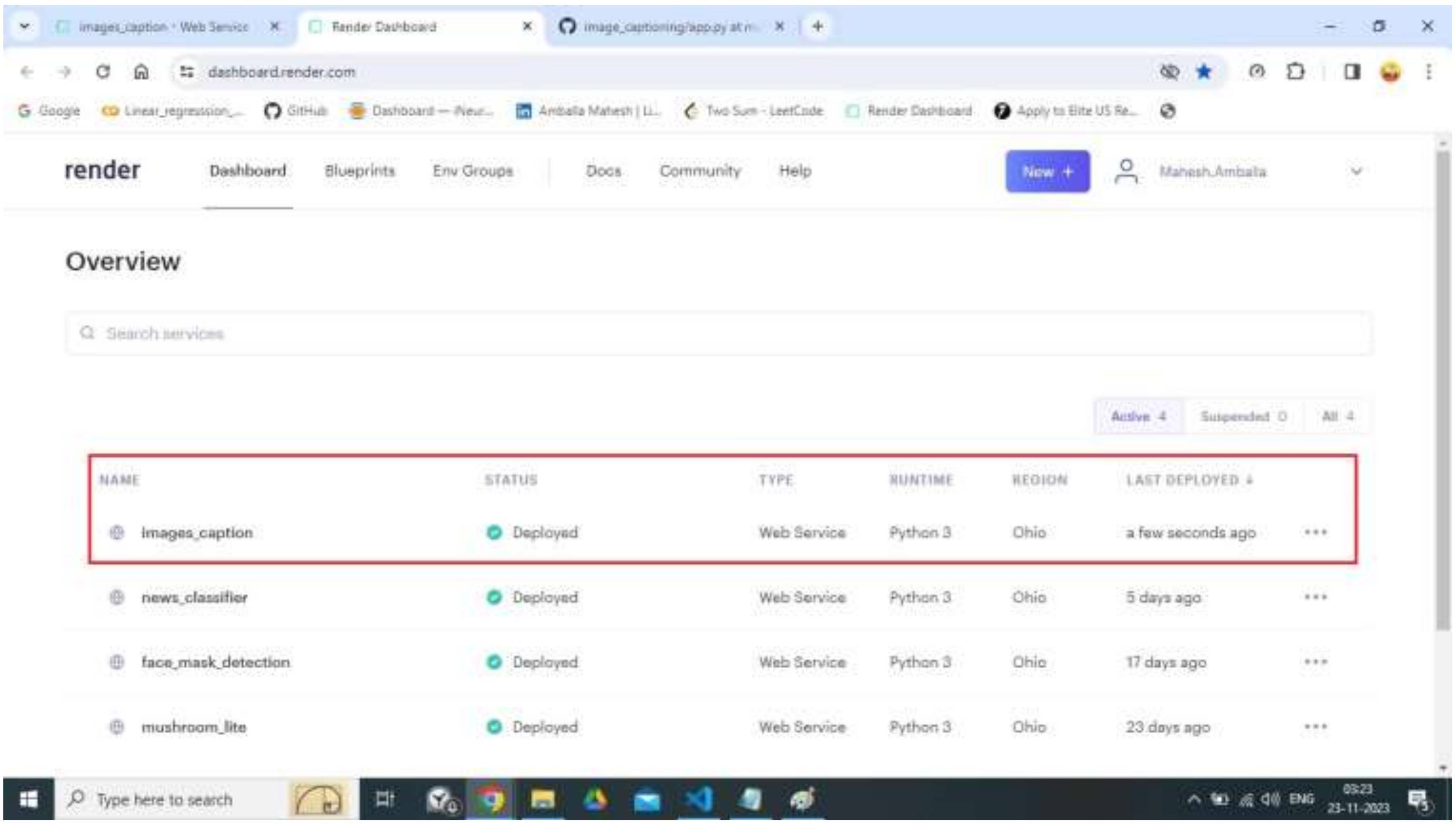




November 23, 2023 at 3:18 AM

85ff5e0 changes in app.py

```
Nov 23 03:22:59 AM  2023-11-22 21:52:59.621801: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could n
ot find TensorRT
Nov 23 03:23:23 AM  Your service is live 🎉
Nov 23 03:23:20 AM  [2023-11-22 21:53:20 +0000] [43] [INFO] Starting gunicorn 21.2.0
Nov 23 03:23:20 AM  [2023-11-22 21:53:20 +0000] [43] [INFO] Listening at: http://0.0.0.0:10000 (43)
Nov 23 03:23:20 AM  [2023-11-22 21:53:20 +0000] [43] [INFO] Using worker: sync
```



Overview

NAME	STATUS	TYPE	RUNTIME	REGION	LAST DEPLOYED ↓
images_caption	Deployed	Web Service	Python 3	Ohio	a few seconds ago
news_classifier	Deployed	Web Service	Python 3	Ohio	5 days ago
face_mask_detection	Deployed	Web Service	Python 3	Ohio	17 days ago
mushroom_lite	Deployed	Web Service	Python 3	Ohio	23 days ago

HOST WEB ADDRESS: https://image-captioning-iwhw.onrender.com

4. Key performance indicators (KPI)

- Time and work load reduction by using the flask model.
- Compare the accuracy of model using prediction and actual results.
- Check for the wrong predictions
- If found any wrong predictions again train the model with the new data along with previous data
- Retest the model unless the productions attain the good results.