CAPSTONE PROJECT:

IMAGE CAPTION

BY- PRIYANKA BISWAS

SHINAKSHI SANKHAYAN

GULFAIRUS KAIREDENOVA

SANDIPTA SUBIR KHARE

AGENDA

PROBLEM STATEMENT

Now days, if someone see the image and not able to recognise what image is showing. Then in that Scanerio Image Captioning would help the user to get the final output of the image.

If there is an user with medical conditions like blind and paralysis then they might able to use any technology for recognising the image. Our project Image Captioning with Voice recognition would help the user to get the final output.

OBJECTIVE

Image caption, automatically generating natural language descriptions according to the content observed in an image, is an important part of scene understanding, which combines the knowledge of computer vision and natural language processing. The application of image caption is extensive and significant, for example, the realization of human-computer interaction.

TOOLS & LIBRARIES

- ➢TOOLS USED:

1. Python

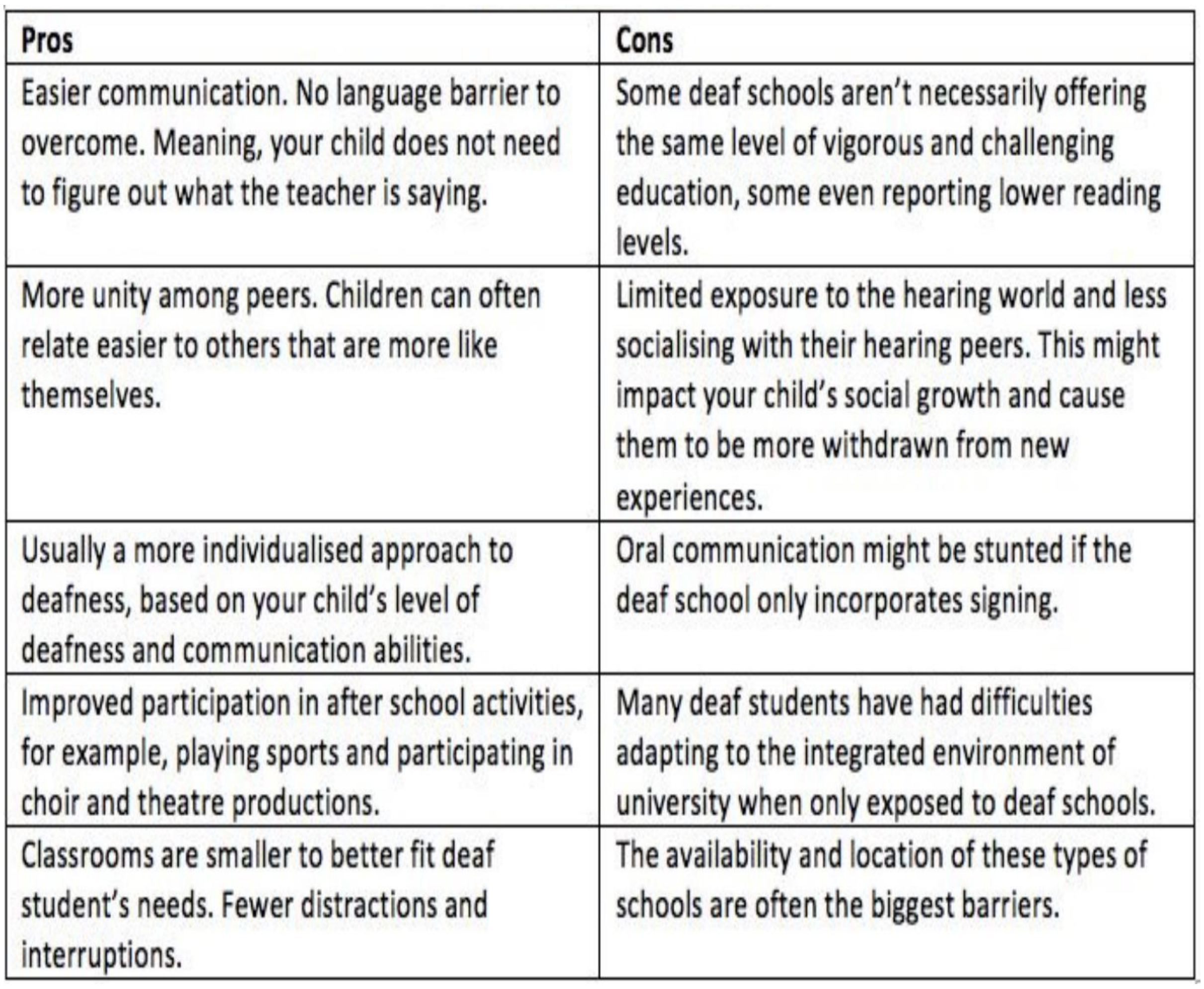
- ➢LIBRARIES:

1. Numpy
2. Pandas
3. Matplotlib
4. Keras

WHAT IS IMAGE CAPTION?

Image Captioning is the process of generating textual description of an image. It uses **Machine Learning**, **Deep Learning**, **Natural Language Processing** and **Computer Vision** to generate the captions. The dataset will be in the form [**image** → **captions**]. The dataset consists of input images and their corresponding output captions.

Pros and Cons of Image Captioning



Pros	Cons
Easier communication. No language barrier to overcome. Meaning, your child does not need to figure out what the teacher is saying.	Some deaf schools aren't necessarily offering the same level of vigorous and challenging education, some even reporting lower reading levels.
More unity among peers. Children can often relate easier to others that are more like themselves.	Limited exposure to the hearing world and less socialising with their hearing peers. This might impact your child's social growth and cause them to be more withdrawn from new experiences.
Usually a more individualised approach to deafness, based on your child's level of deafness and communication abilities.	Oral communication might be stunted if the deaf school only incorporates signing.
Improved participation in after school activities, for example, playing sports and participating in choir and theatre productions.	Many deaf students have had difficulties adapting to the integrated environment of university when only exposed to deaf schools.
Classrooms are smaller to better fit deaf student's needs. Fewer distractions and interruptions.	The availability and location of these types of schools are often the biggest barriers.

How to do Image Captioning Implementation

The task of image captioning can be divided into two modules logically:

- One is an **image based model** – which extracts the features and nuances out of our image, and
- The other is a **language based model** – which translates the features and objects given by our image based model to a natural sentence.

Techniques for Image Captioning

T he techniques which we may use for the image captioning can be broadly divided into two categories:

- ➢ Deep Learning based techniques and
- ➢ NLP based techniques.

Data Preparation

The **Flickr 8k dataset** have five different descriptions per image, that provide clear descriptions of the noticeable entities and events and are described by actual people. The dataset has 8000 images from Flickr and contains people and animals (mostly dogs) performing some action. We used ¾ of the data for training and ¼ for evaluation.

- lowercase all words
- remove punctuation like .,-<>()
- remove hanging ‘s’ and ‘a’
- remove numbers

Natural Processing Language

- Computers don’t understand English words, for computers, we will have to represent them with numbers. So, we have map each word of the vocabulary with a unique index value. Keras library provides us with the tokenizer function that we will use to create tokens from our vocabulary and save them to a “tokenizer's” pickle file.
- Our vocabulary contains 7577 words.

We calculate the maximum length of the descriptions. This is important for deciding the model structure parameters.

NLP Based Techniques

This technique combines Computer Vision and Natural Language Processing techniques to create a model that can describe contents of an image into natural language. This technique is useful for

- Captioning images on the internet, which helps visually impaired people using a screen reader to describe the contents of an image into words.
- Captioning a large database of images to make it easier to search images with description.

What is Tokenizer

Tokenization is a common task in Natural Language Processing (NLP). It’s a fundamental step in both traditional NLP methods like Count Vectorizer and Advanced Deep Learning-based architectures like **Transformers.**

Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or sub words. Hence, tokenization can be broadly classified into 3 types – word, character, and sub word (n-gram characters) tokenization.

For example, consider the sentence: “Never give up”.

The most common way of forming tokens is based on space. Assuming space as a delimiter, the tokenization of the sentence results in 3 tokens – Never-give-up. As each token is a word, it becomes an example of Word tokenization.

Similarly, tokens can be either characters or sub words. For example, let us consider “smarter”:

1. Character tokens: s-m-a-r-t-e-r
2. Sub word tokens: smart-er

Working of Tokenizer

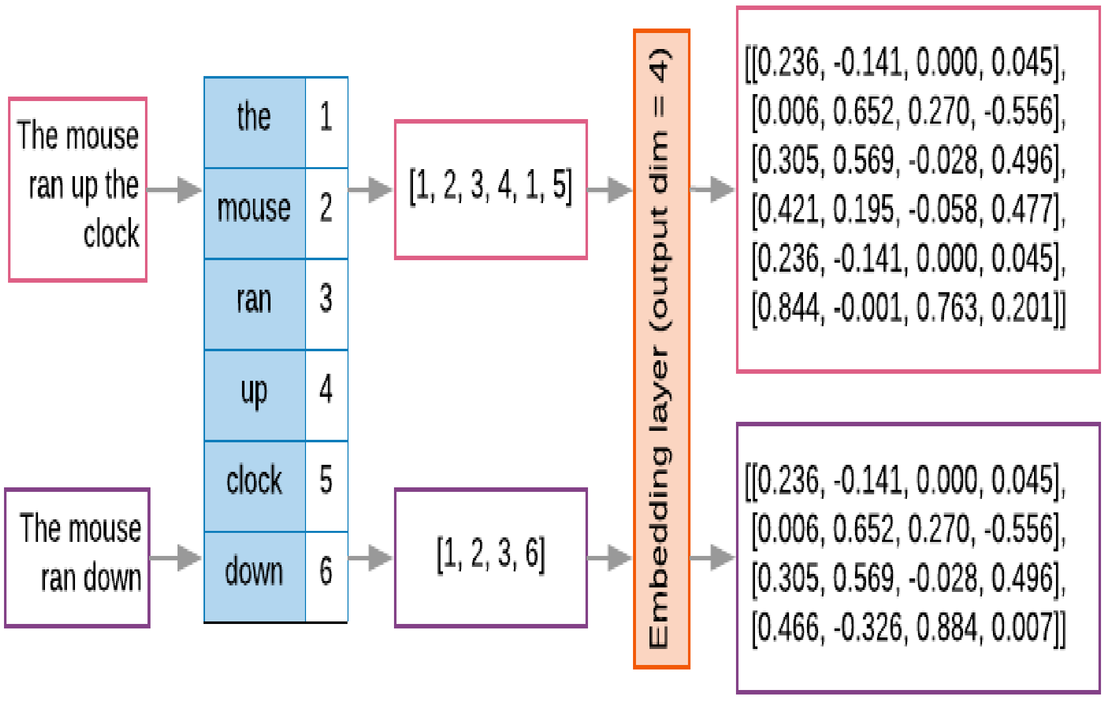
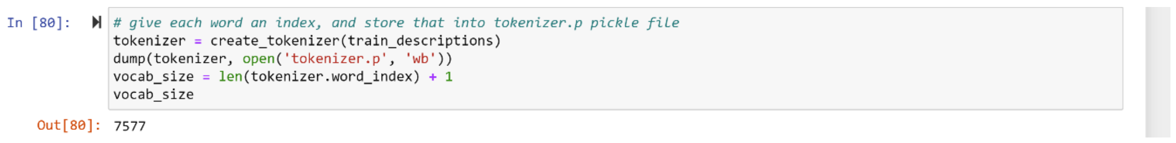


Fig 1.1

vocab_size:



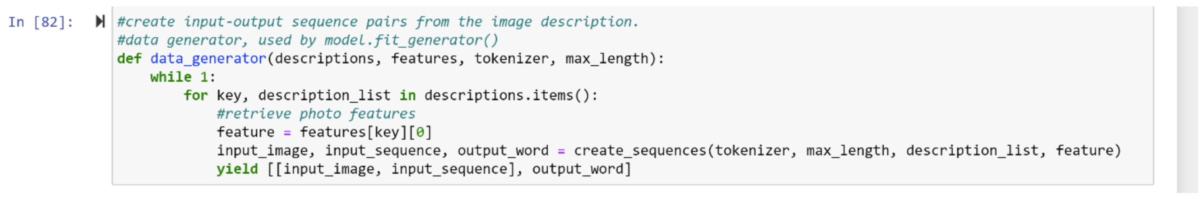
```
In [80]:  # give each word an index, and store that into tokenizer.p pickle file
          tokenizer = create_tokenizer(train_descriptions)
          dump(tokenizer, open('tokenizer.p', 'wb'))
          vocab_size = len(tokenizer.word_index) + 1
          vocab_size

Out[80]:  7577
```

Fig 1.2

Data Generator

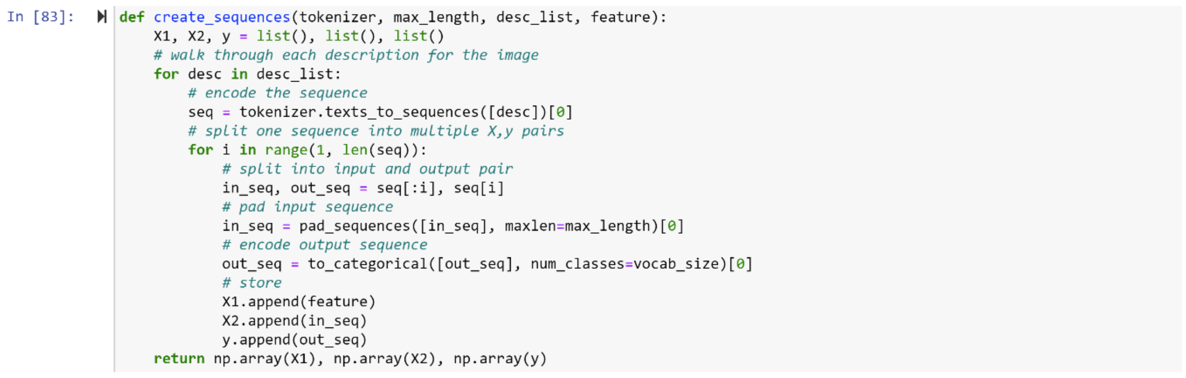
- We have train our model on 6000 images and each image will contain 2048 length feature vector and caption is also represented as numbers. This amount of data for 6000 images is not possible to hold into memory so we will be using a generator method that will yield batches.
- The generator will yield the input and output sequence.
- The input to our model is [x1, x2] and the output will be y, where x1 is the 2048 feature vector of that image, x2 is the input text sequence and y is the output text sequence that the model has to predict.

Create Data Generator:



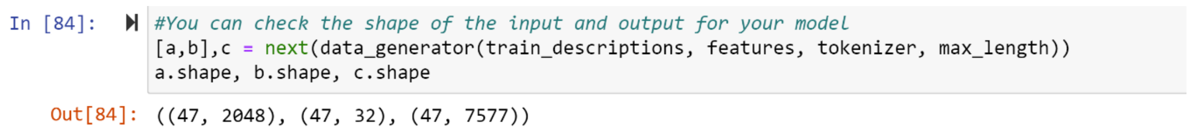
```
In [82]:  #create input-output sequence pairs from the image description.
          #data generator, used by model.fit_generator()
          def data_generator(descriptions, features, tokenizer, max_length):
              while 1:
                  for key, description_list in descriptions.items():
                      #retrieve photo features
                      feature = features[key][0]
                      input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, feature)
                      yield [[input_image, input_sequence], output_word]
```

Fig 1.3



```
In [83]:  def create_sequences(tokenizer, max_length, desc_list, feature):
              X1, X2, y = list(), list(), list()
              # walk through each description for the image
              for desc in desc_list:
                  # encode the sequence
                  seq = tokenizer.texts_to_sequences([desc])[0]
                  # split one sequence into multiple X,y pairs
                  for i in range(1, len(seq)):
                      # split into input and output pair
                      in_seq, out_seq = seq[:i], seq[i]
                      # pad input sequence
                      in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                      # encode output sequence
                      out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                      # store
                      X1.append(feature)
                      X2.append(in_seq)
                      y.append(out_seq)
              return np.array(X1), np.array(X2), np.array(y)
```

Fig 1.4



```
In [84]:  #You can check the shape of the input and output for your model
          [a,b],c = next(data_generator(train_descriptions, features, tokenizer, max_length))
          a.shape, b.shape, c.shape

Out[84]:  ((47, 2048), (47, 32), (47, 7577))
```

Fig 1.5

Caption Generation Process

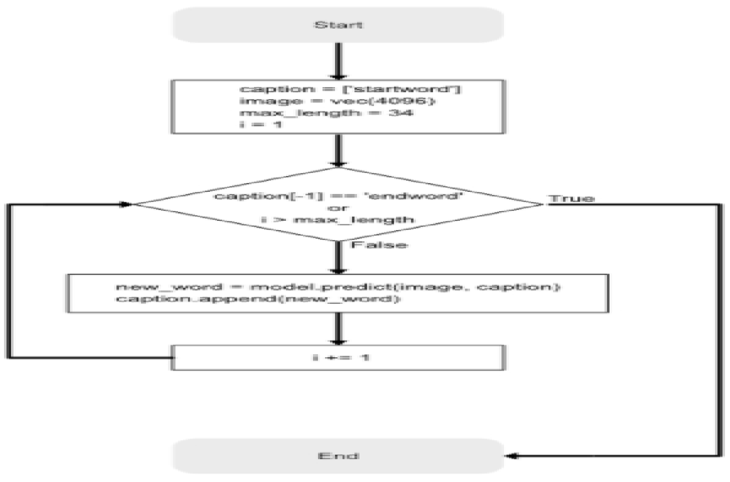


Fig 1.6

Deep Learning Techniques

- In deep learning based techniques, features are learned automatically from training data and they can handle a large and diverse set of images and videos. For example, Convolutional Neural Networks (CNN) are widely used for feature learning, and a classifier such as SoftMax is used for classification. CNN is generally followed by Recurrent Neural Networks (RNN) in order to generate captions.
- An embedding layer will handle the textual input, followed by the LSTM layer.
- Decoder – By merging the output from the above two layers, we will process by the dense layer to make the final prediction. The final layer will contain the number of nodes equal to our vocabulary size.

What is CNN (Convolution Neural Network)?

Convolutional Neural networks are specialized deep neural networks which can process the data that has input shape like a 2D matrix. Images are easily represented as a 2D matrix and CNN is very useful in working with images.

CNN is basically used for image classifications and identifying if an image is a bird, a plane or Superman, etc.

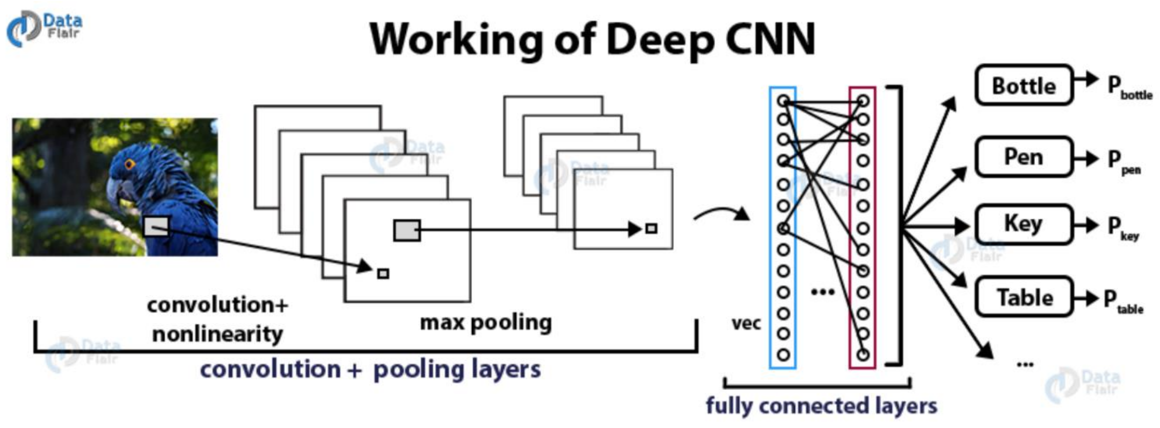


Fig 1.7

It scans images from left to right and top to bottom to pull out important features from the image and combines the feature to classify images. It can handle the images that have been translated, rotated, scaled and changes in perspective.

What is LSTM?

LSTM stands for **Long short term memory**, they are a type of RNN (**recurrent neural network**) which is well suited for sequence prediction problems. Based on the previous text, we can predict what the next word will be. It has proven itself effective from the traditional RNN by overcoming the limitations of RNN which had short term memory. LSTM can carry out relevant information throughout the processing of inputs and with a forget gate, it discards non-relevant information.
This is what an LSTM cell looks like –

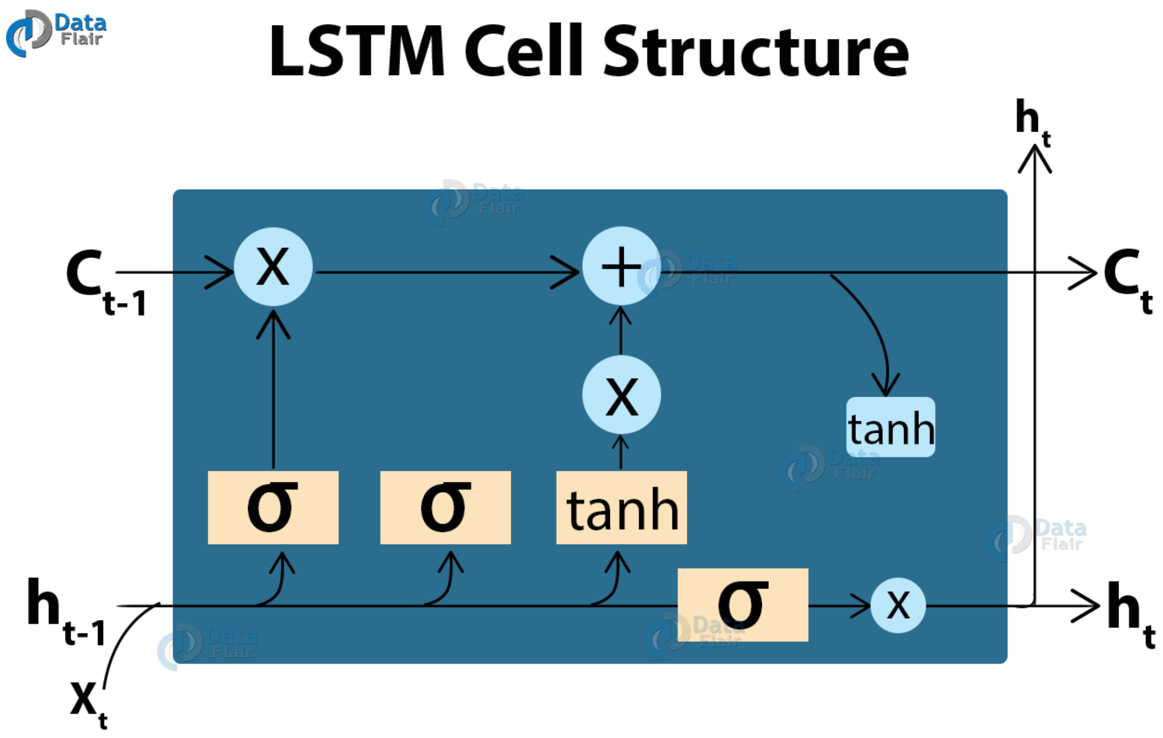


Fig 1.8

Defining CNN-RNN Model with LSTM:

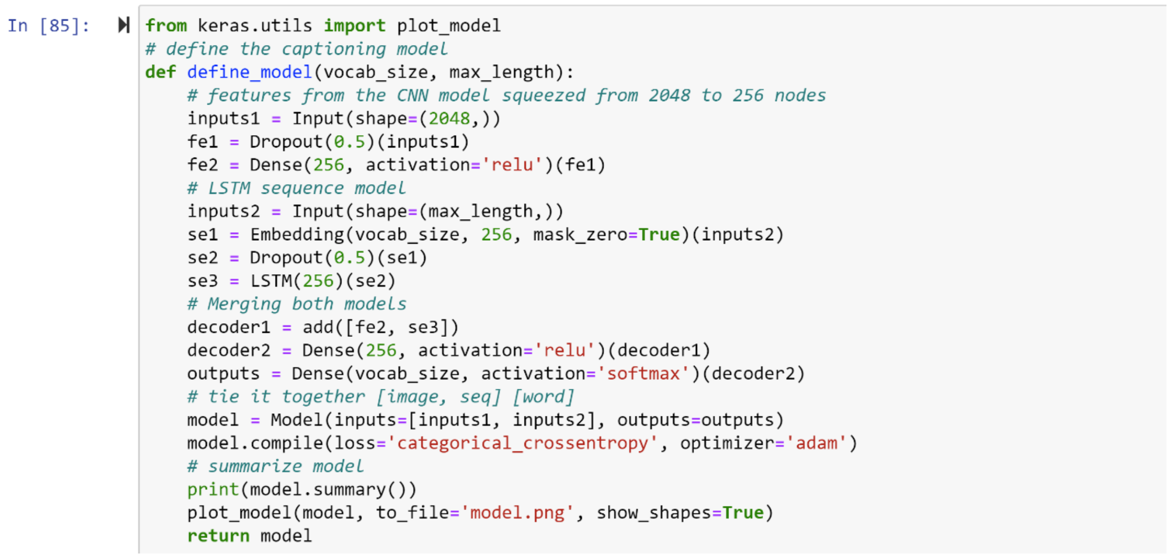

```
In [85]:
from keras.utils import plot_model
# define the captioning model
def define_model(vocab_size, max_length):
    # features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    # LSTM sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)
    # Merging both models
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    # summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)
    return model
```

Fig 1.9

Image Caption Generator Model

So, to make our image caption generator model, we will be merging these architectures. It is also called a CNN-RNN model.

- CNN is used for extracting features from the image. We will use the pre-trained model Xception.
- LSTM will use the information from CNN to help generate a description of the image.

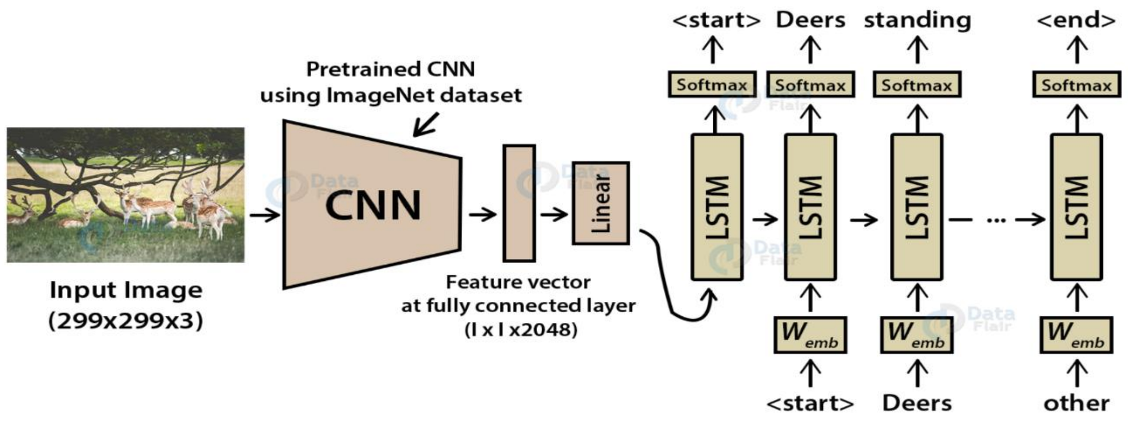
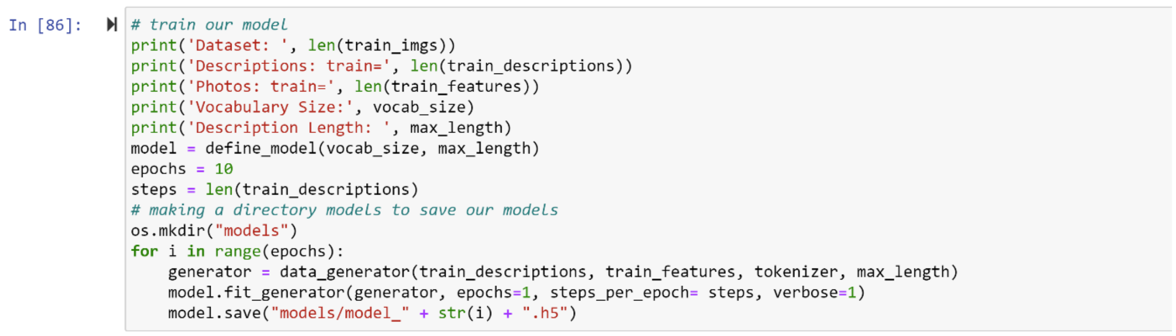


Fig 1.10

Training Model

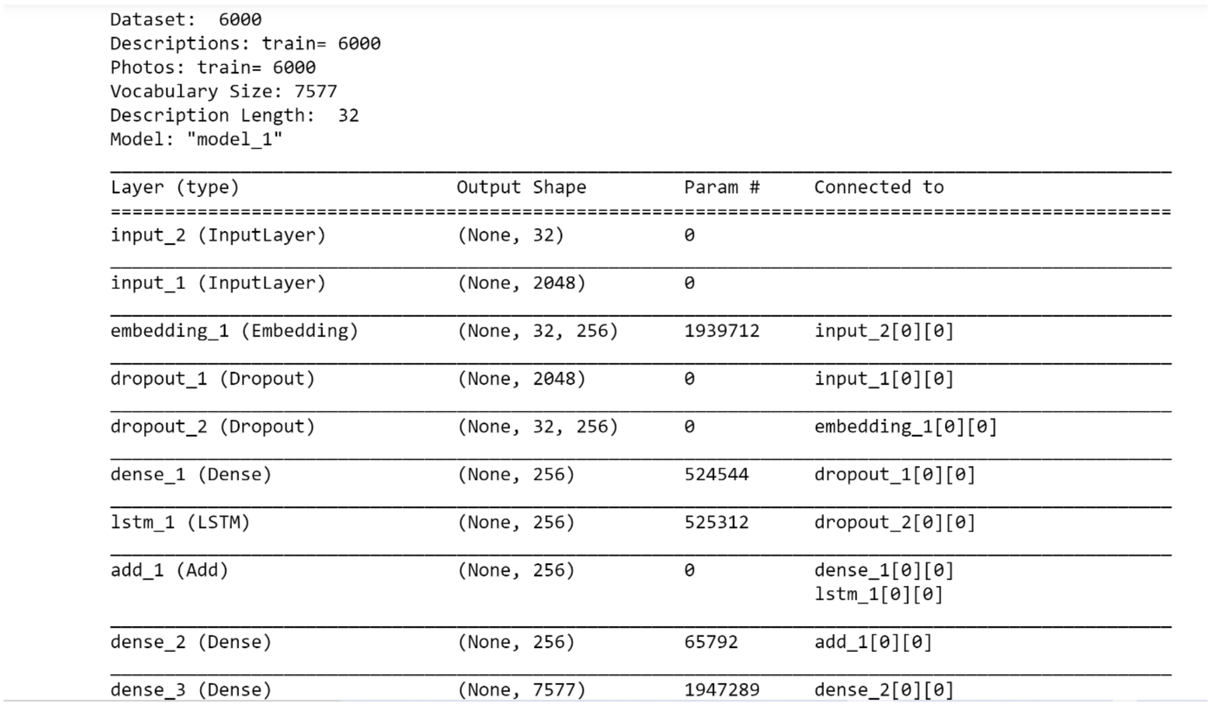
Training the model:



```
In [86]:  # train our model
          print('Dataset: ', len(train_imgs))
          print('Descriptions: train=', len(train_descriptions))
          print('Photos: train=', len(train_features))
          print('Vocabulary Size:', vocab_size)
          print('Description Length: ', max_length)
          model = define_model(vocab_size, max_length)
          epochs = 10
          steps = len(train_descriptions)
          # making a directory models to save our models
          os.mkdir("models")
          for i in range(epochs):
              generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
              model.fit_generator(generator, epochs=1, steps_per_epoch= steps, verbose=1)
              model.save("models/model_" + str(i) + ".h5")
```

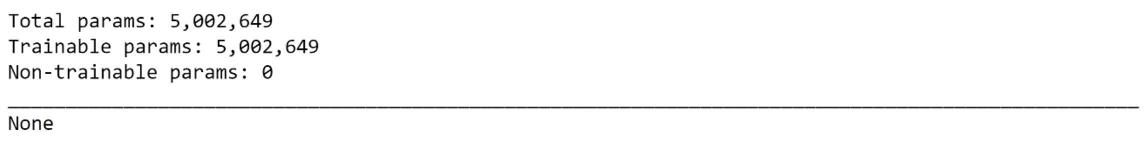
Fig 1.11

Output:



```
Dataset:  6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length:  32
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            (None, 32)           0
__________________________________________________________________________________________________
input_1 (InputLayer)            (None, 2048)         0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 32, 256)      1939712     input_2[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 2048)         0           input_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 32, 256)      0           embedding_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          524544      dropout_1[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 256)          525312      dropout_2[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 256)          0           dense_1[0][0]
                                                                 lstm_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          65792       add_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 7577)         1947289     dense_2[0][0]
```

Fig 1.12



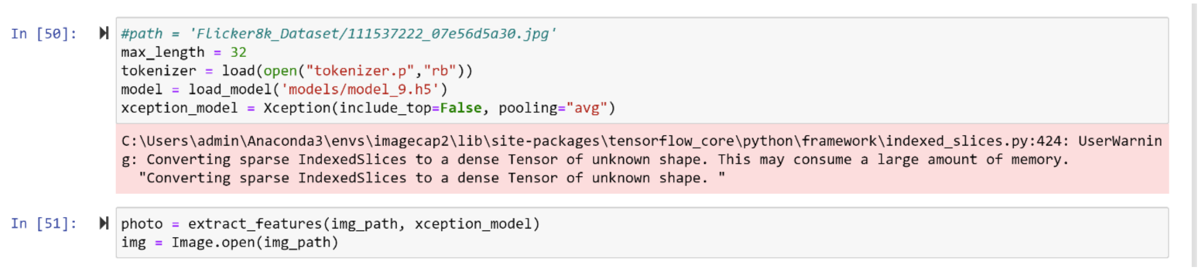
```
Total params: 5,002,649
Trainable params: 5,002,649
Non-trainable params: 0
__________________________________________________________________________________________________
None
```

Fig 1.13

Extracting Features

Extracting features from the image CNN is Xception Model.

Xception model is the pre-trained model that have been already trained on large datasets and extract the features from these models and use them for our tasks. Xception model has been trained on imagenet dataset that had 1000 different classes to classify. We can directly import this model from the keras.applications . Xception model takes 299*299*3 image size as input. We will remove the last classification layer and get the 2048 feature vector.



```
In [50]:  #path = 'Flicker8k_Dataset/111537222_07e56d5a30.jpg'
          max_length = 32
          tokenizer = load(open("tokenizer.p","rb"))
          model = load_model('models/model_9.h5')
          xception_model = Xception(include_top=False, pooling="avg")

          C:\Users\admin\Anaconda3\envs\imagecap2\lib\site-packages\tensorflow_core\python\framework\indexed_slices.py:424: UserWarnin
          g: Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.
            "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

In [51]:  photo = extract_features(img_path, xception_model)
          img = Image.open(img_path)
```

Fig 1.14

Layers in Model

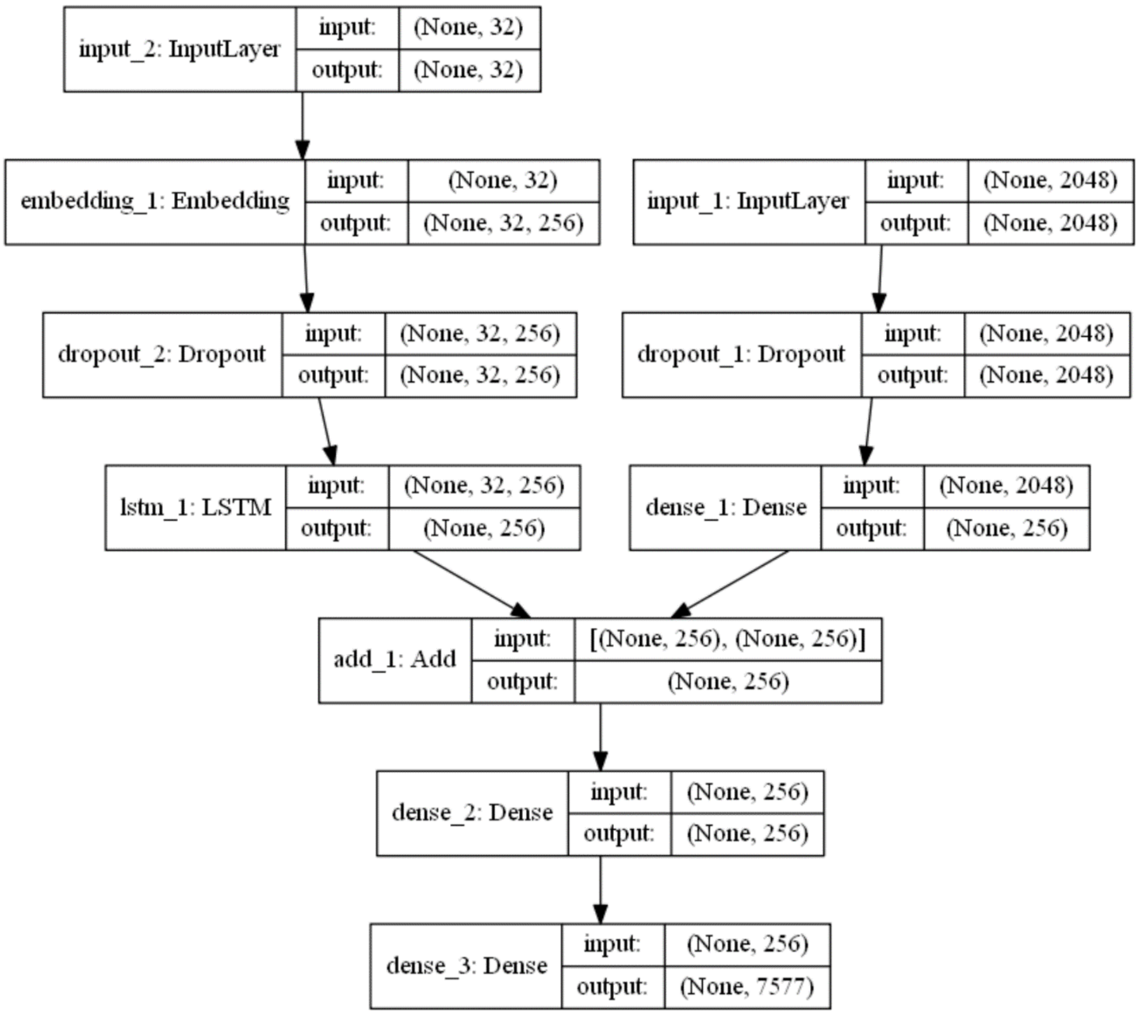
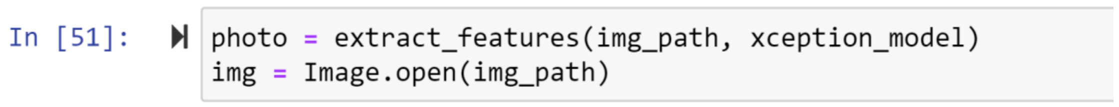


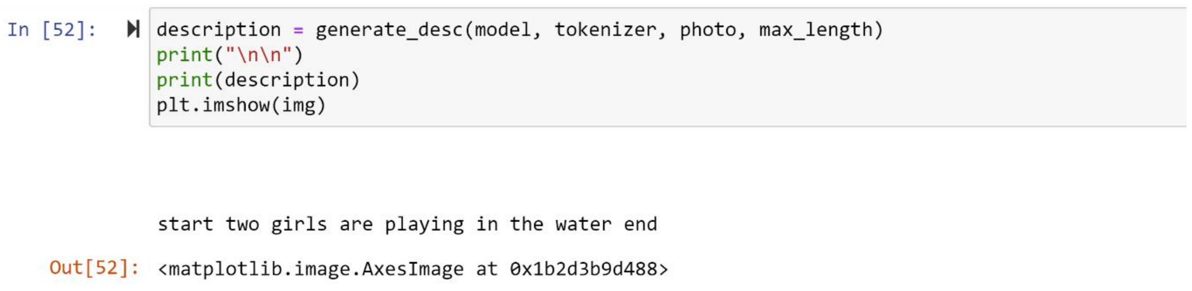
Fig 1.15

Testing the Model



```
In [51]:   photo = extract_features(img_path, xception_model)
           img = Image.open(img_path)
```

Fig 1.16



```
In [52]:   description = generate_desc(model, tokenizer, photo, max_length)
           print("\n\n")
           print(description)
           plt.imshow(img)


           start two girls are playing in the water end

Out[52]:   <matplotlib.image.AxesImage at 0x1b2d3b9d488>
```

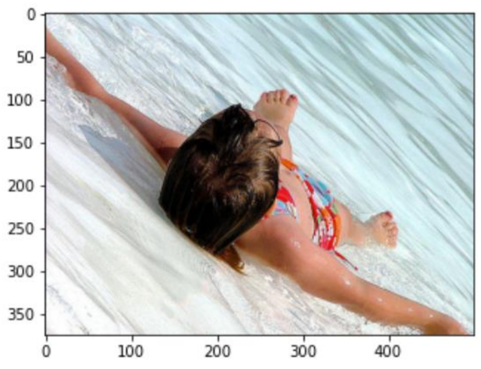
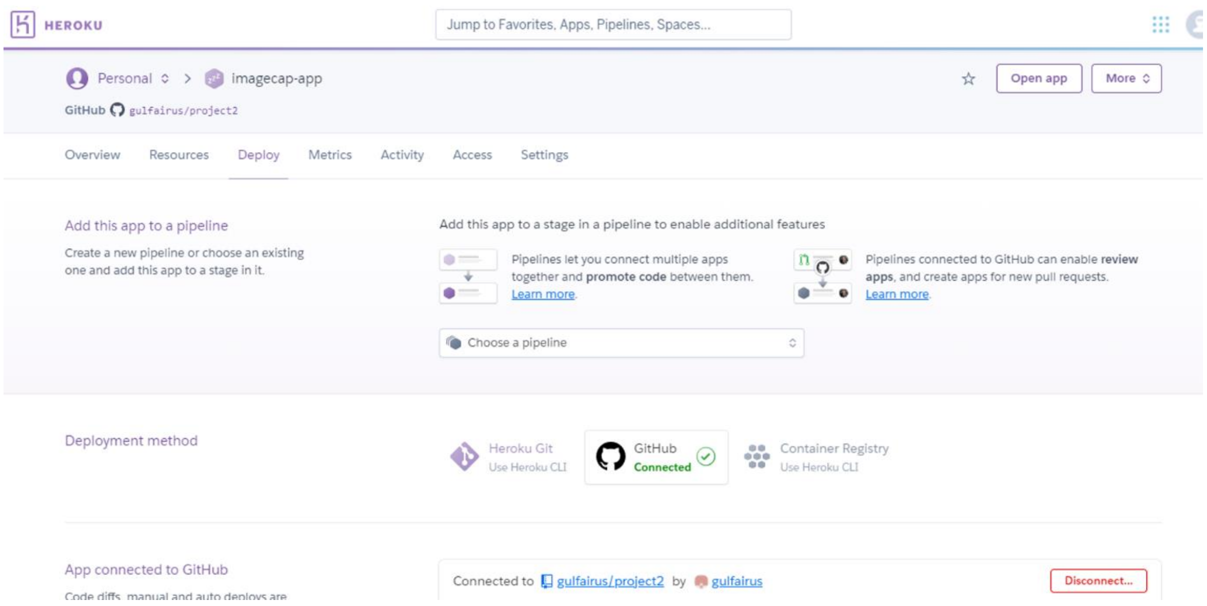


Fig 1.17

Deployment



Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them. Learn more.

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. Learn more.

Choose a pipeline

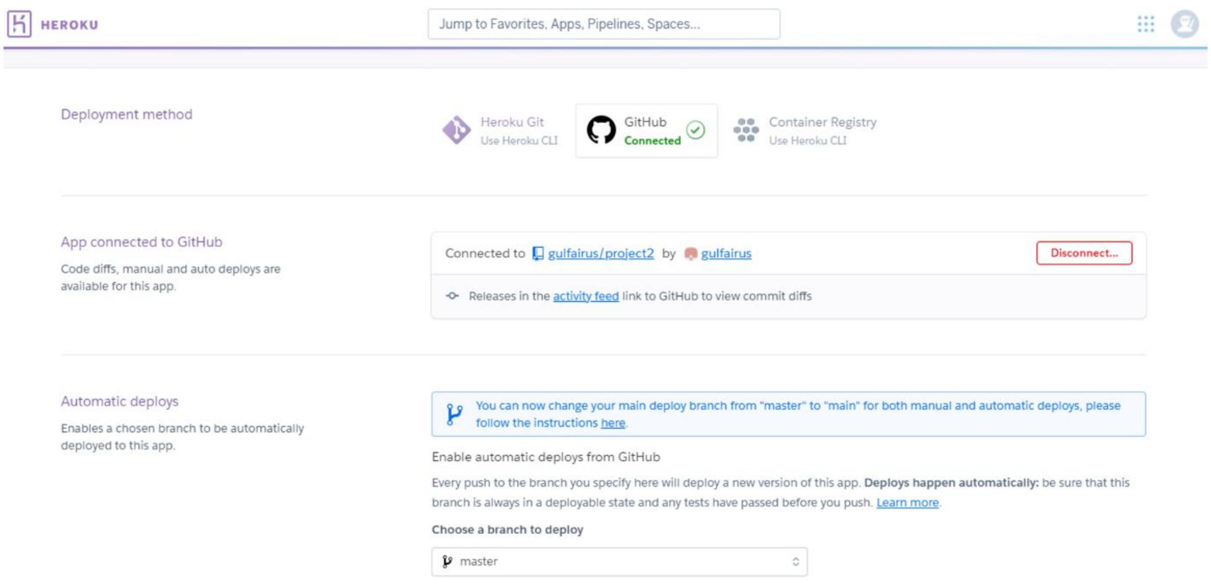
Deployment method

App connected to GitHub

Connected to gulfairus/project2 by gulfairus

Code diffs, manual and auto deploys are

Fig 1.18



Deployment method

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to gulfairus/project2 by gulfairus

Releases in the activity feed link to GitHub to view commit diffs

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions here.

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. Learn more.

Choose a branch to deploy

master

Fig 1.19

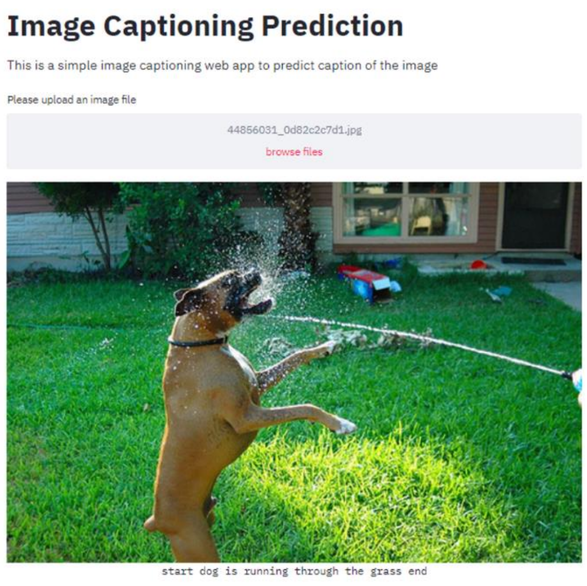


Image Captioning Prediction

This is a simple image captioning web app to predict caption of the image

Please upload an image file

44856031_0d82c2c7d1.jpg

browse files

```
start dog is running through the grass end
```

Fig 1.20

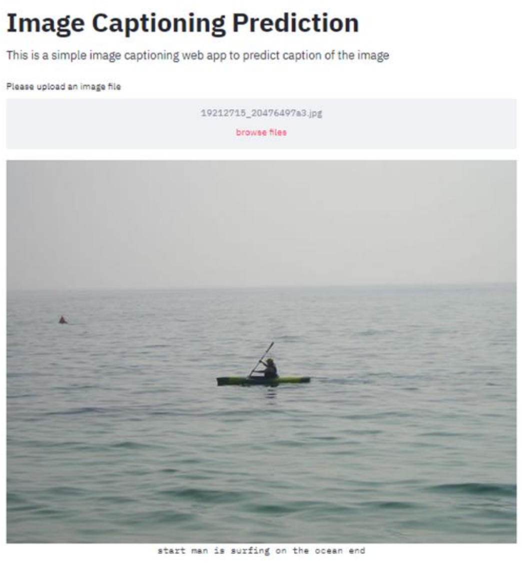


Image Captioning Prediction

This is a simple image captioning web app to predict caption of the image

Please upload an image file

19212715_20476497a3.jpg

browse files

```
start man is surfing on the ocean end
```

Fig 1.21

Reference

- https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/#:~:text=A%20Quick%20Rundown%20of%20Tokenization,-Tokenization%20is%20a&text=Tokenization%20is%20a%20way%20of,words%2C%20characters%2C%20or%20subwords.&text=As%20each%20token%20is%20a,be%20either%20characters%20or%20subwords.
- https://www.geeksforgeeks.org/nlp-how-tokenizing-text-sentence-words-works/
- https://www.analyticsvidhya.com/blog/2019/07/how-get-started-nlp-6-unique-ways-perform-tokenization/
- https://pypi.org/project/tokenizer/#:~:text=Tokenizer%20is%20a%20compact%20pure,%2C%20URL%2FURI%2C%20etc.
- https://www.tutorialspoint.com/python_text_processing/python_tokenization.htm