3RDEYE

Image Recognition Application for Visually Impaired

Final Report

Group 3

Pavan Moganti, Shruthi Parameshwar, Mythri Gopaluni,
Gopika Gopi Krishnan, Santhana Krishnan

Georgia State University, J Mack Robinson College of Business

30 April 2019

Introduction

There is an interesting quote in the IBM training manual in 1991.

"For people without disabilities, technology makes things easier. For people with disabilities, technology makes things possible."

This quote both inspired us a team and drove us to come up with an idea for a product that could make things possible for people who may be challenged in some aspect of their life. Deloitte conducted a study which revealed that millennials believe that business should strive to build products that cater to social issues. It has become an inevitable part of societal development. Technology when put to best use, should yield great value to the society.

Keen to build a project that would aid the challenged, after some research the team found that, globally, it is estimated that approximately 1.3 billion people live with some form of vision impairment. This issue struck a chord and the team decided to work on building an application to aid the visually impaired to be cognizant of their surroundings through Artificial Intelligence.

When application is built, the application would be able to,

- Generate a caption or a short textual description for a given image.
- Translate the generated caption or short textual description into an audio format.
- Conveys the message from an image efficiently.

The Product

The visually impaired also have eyes, but they were not as fortunate as other are to be able to perceive their surroundings as most people around them. This product aims to be their 3RDEYE (Third Eye) to aid them with the closest experience of their surroundings by capturing an image from their mobile device and generating an audio that explains the image captured as accurately possible.

The product in itself may not sound so complicated for those of us are able to see and describe what we see. But, move this to the perspective of a computer and things get challenging and this where technology comes in.

Technology

A variety of technologies such as machine learning and artificial intelligence are required to make this product come to life. Some of the concepts that are involved are,

- Image recognition
- Image processing
- Natural Language Processing (NLP)
- Image Captioning
- Text-to-Speech

To implement the concepts, a large amount data would be required which undergo machine learning, so that the computer can make sense of these images. Some of these machine learning technologies include,

- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)
- Generative Adversarial Networks (GANs)

A platform is also required that can carry out the above-mentioned techniques and since they are complex the platform also needs to be equally powerful. Some of the needs for this project would be,

- Data storage
- Platform to run the machine learning models
- Powerful processors to process the data
- Cloud technologies to convert speech-to-text

All the above shall be discussed in further detail as part of this report along with results that are obtained and the challenges that were faced while executing some of these steps.

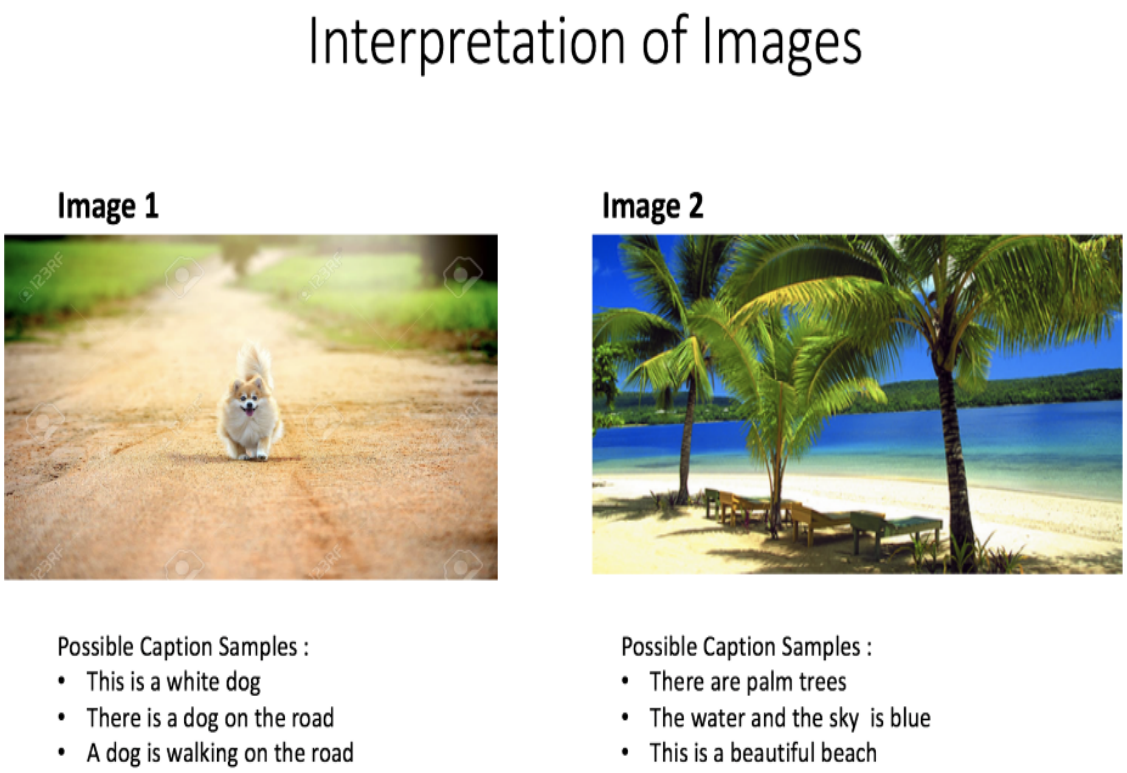
High level Architecture

Below is a graphical representation of the flow of data through an information system. It illustrates the flow of data in a process based on the inputs and outputs. Throughout our project there are various stages and processes through which the data will flow and thus it is imperative that there should be a well-defined path for the data to flow until the last stage to get the desired output.

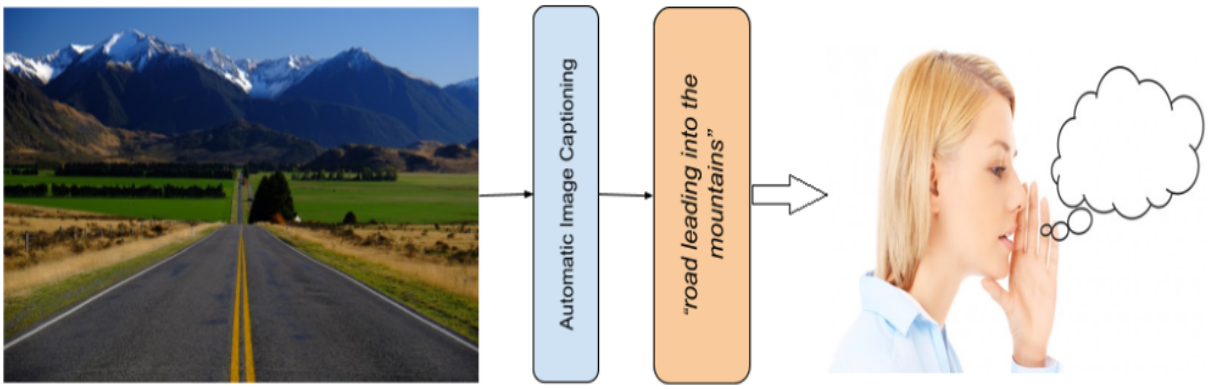
Along with the flow of data, the platforms and technologies used at different stages to handle and process the data is represented. These technologies will help with executing the tasks that required for the application run smoothly and provide the best results.

Image Recognition

An image can be described in several ways and as humans we tend to perceive and interpret things around us in a differently and thus, we can expect huge variance. Below is an example how an image can be understood in different ways.



We can see from the above two images that there is a variation in the interpretation of an image although finally they lead to similar conclusions. To understand this further, let us look at how this works.



A basic idea of our application can be seen from the above image. When a visual image is fed to our model it will automatically try to identify the objects and its connections with respect to its surroundings and then formulate a sentence and convert it to speech that will help the visually impaired people in understanding what that image represents.

Data Sources

The images in the dataset are in the jpg format which is the popular digital image format. Though image data can be stored on any database as it falls under the non-textual unstructured data format and the metadata contained in the image can then be used for various analytics.

In a real-life scenario, the application would capture an image on a mobile device and then have generate a caption and audio. But to accomplish this, a fixed set of data is required for training with a scope to add more data at the later stages into established model.

The most important aspect for our dataset is the variety of images, as in a real scenario, the image could range from people, things, animals, plants, vehicles and so on, for which relevant captions would focus on the object of interest along with the background information. To keep our research versatile, we have chosen Flickr as our primary source of data for sharp images was obtained from the University of Illinois available at the following link by filling out a form, https://forms.illinois.edu/sec/229675. A subset of VizWiz challenge data as the second source for blurred images.

1. Flickr dataset

The initial plan was to use the famous and large-scale open datasets called Flickr30k. The Flickr30k dataset has become a standard benchmark for sentence-based image description. This dataset would give 5 captions for every image which would increase the data for analysis and training. In total that would give us 31,000+ images and 158,000+ captions.

This large set of data requires a lot of processing power and cloud storage space which is something our team did not have access to and therefore, the dataset size had to be reduced. Fortunately, there was a smaller dataset that was available from the same source, the Flickr8k dataset. This dataset consisted of over 8,000 images again with 5 captions each, making it a caption set of 40,000+.

This dataset is designed for object detection, image captioning and segmentation. The images and captions focus on people involved in everyday activities and events. There are multiple objects in the image but in a caption usually the main subject and either one or two of the secondary subjects are included in the caption. This also helps satisfy the dynamic nature of images.

Below is a snapshot of the caption/ comment data obtained:

1000092795.jpg

- Two young guys with shaggy hair look at their hands while hanging out in the yard.
- Two young, White males are outside near many bushes.
- Two men in green shirts are standing in a yard.
- A man in a blue shirt standing in a garden.
- Two friends enjoy time spent together.

10002456.jpg

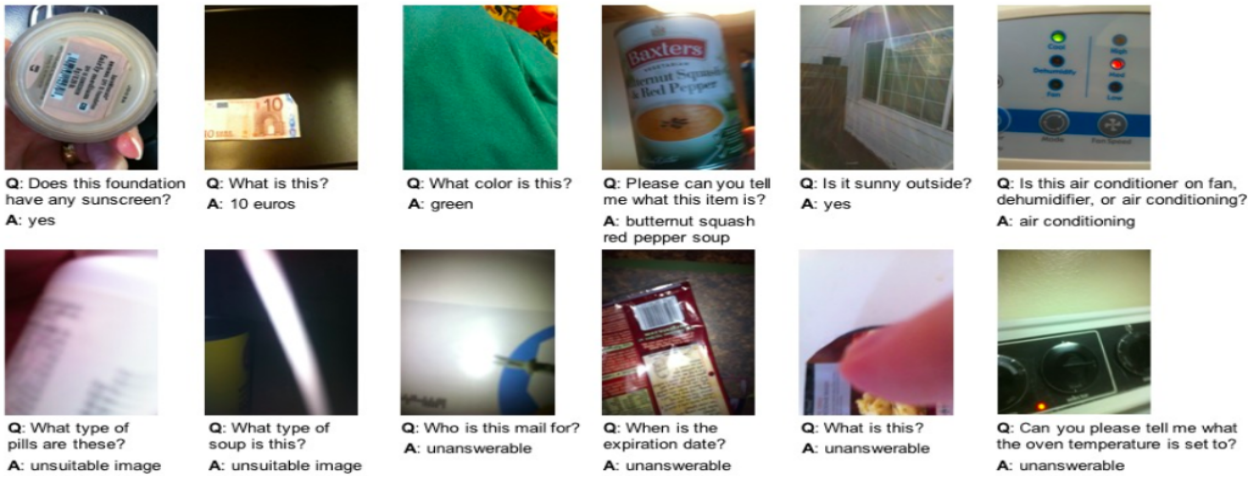
- Several men in hard hats are operating a giant pulley system.
- Workers look down from up above on a piece of equipment.
- Two men working on a machine wearing hard hats.
- Four men on top of a tall structure.
- Three men on a large rig.

1000268201.jpg

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

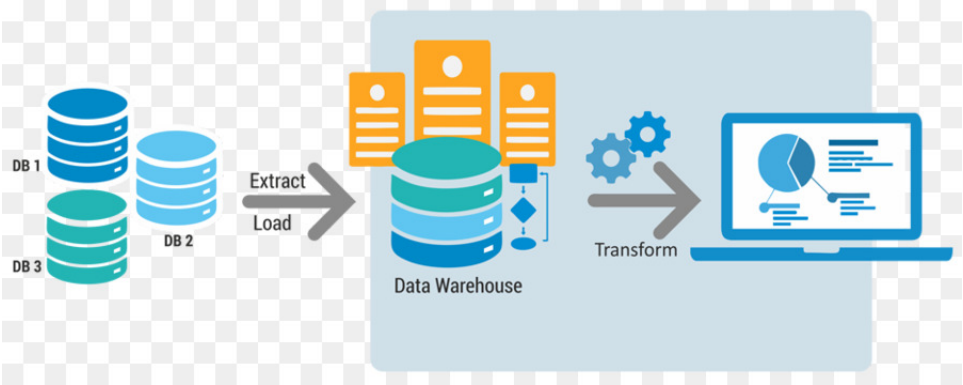
2. Viz-Wiz Challenge Dataset

The second source of data considered is the open source data from the Viz-Wiz challenge. The dataset was primarily designed for visual question and answer challenge, with pairs of images, corresponding question and answer pairs that help understand an image. For our product, we will be using the images from this dataset since they were taken by visually impaired people and the images are mostly blurred. This dataset will pose a challenge for our model to understand the images and provide accurate descriptions. Below is a snapshot of the dataset.



Data Storage and Processing

The meaningful information from the data needs to be derived for the data to be useful. Unstructured data such as images and text are major inputs in this project and since the volume is high, and Extract, Load and Transform (ELT) technique shall be adopted to handle our data. Here, the extracted data is first loaded into a data warehouse and then the required transformations such as normalization, filtering and sorting to clean the data are performed.



Data Extraction

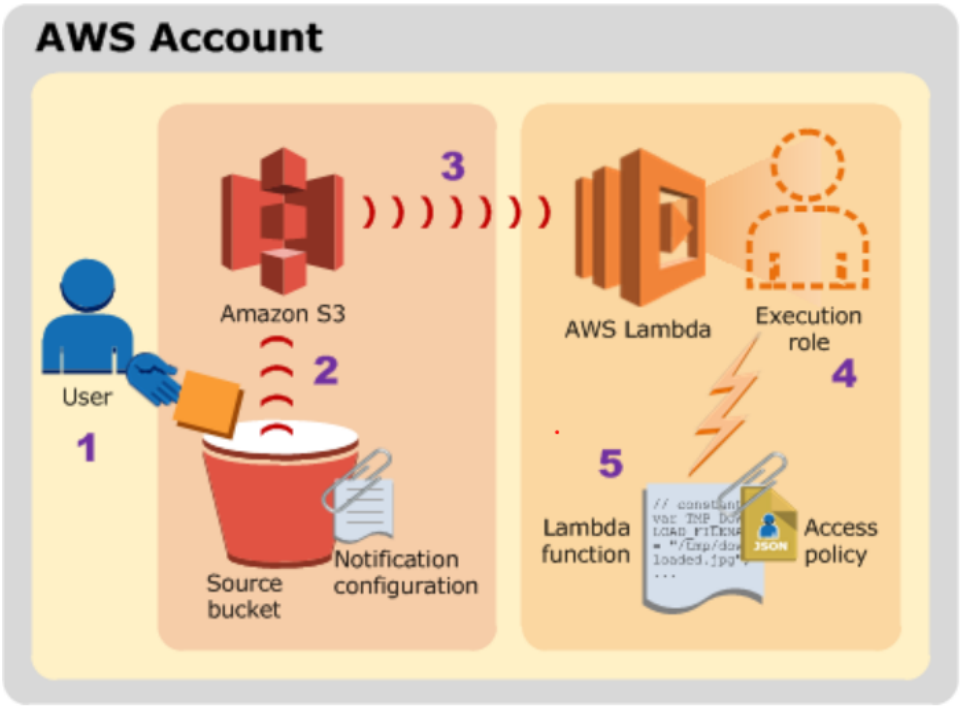
This is the first step in the ETL process where data is retrieved from various sources which could be in different formats. This source data is then converted to single format which makes the data suitable to be transformed.

The extraction process is to be conducted in two phases, Initial extraction and Incremental Extraction. The primary source of our data is as mentioned earlier the Flickr8k data set which has over 8000 images and 5 captions for each image. This will be our source of data (Images and Captions) for **Initial extraction** to build our model. The actual data will be real time inputs from mobile devices of visually impaired persons which will account for the **Incremental extraction**. This incremental extraction will take place when the image from the client (mobile) device is sent to the backend server which then updates the existing data by adding the new image taken.

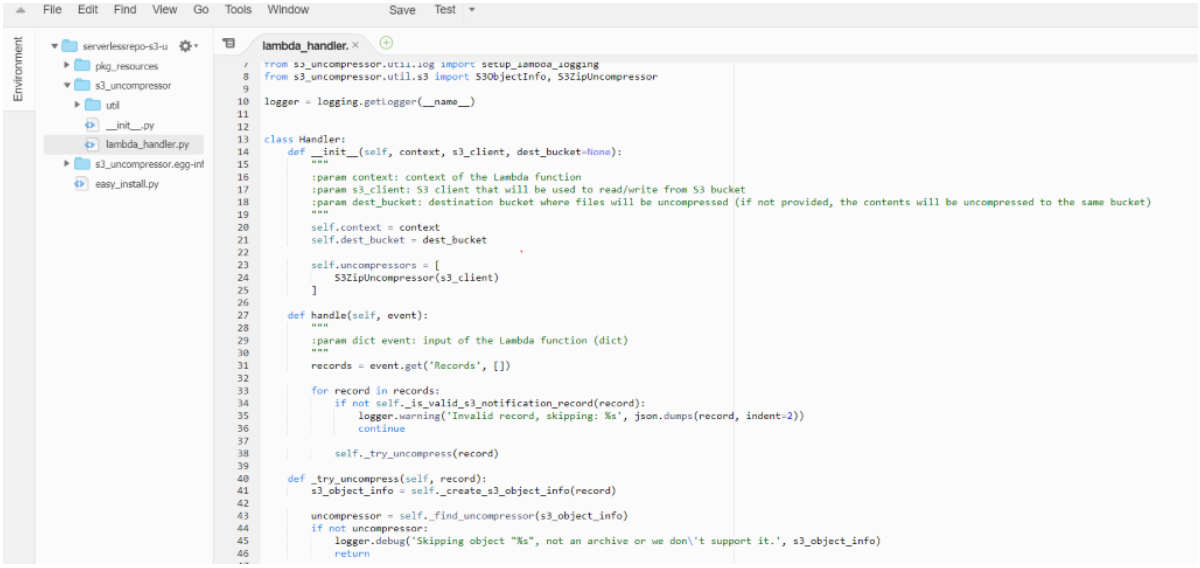
Data Storage and Load

AWS CLI and Lambda

The next step in the process is Loading. The data will be loaded to the database before transforming as the volume of data is large. An AWS S3 (Simple Storage Service) bucket is used to store the data source that has been extracted. To upload the data to S3, AWS CLI was used to leverage the parallel processing which helps with faster upload. Since the size of the data is large and is in ZIP format, a Lambda function was built to extract the data from the zip file and store the individual files in an accessible format by the Databricks platform.



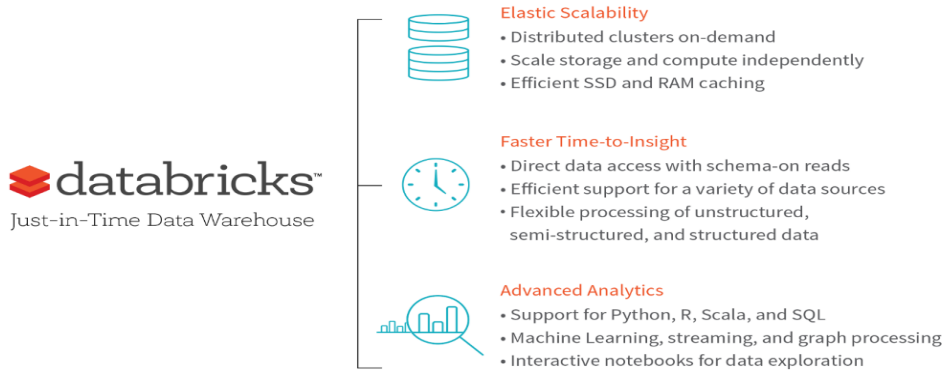
This function will be triggered the moment the zip file reaches S3 Bucket. Lambda will automatically unzip the file and places the data back to S3. Below is the snapshot of the Lambda function that was written.



Databricks

The implementation of the project involves Deep Learning models applied on large datasets. The local machines will not be able to process the data at this scale. Hence, Databricks platform will be used to ease data storage and enhance processing performance. Databricks provides an environment that makes it easy to build, train, and deploy deep learning (DL) models at scale. Many deep learning libraries are available in Databricks Runtime ML, a machine learning runtime that provides a ready-to-go environment for machine learning and data science.

Some of them that will be useful for our implementation are TensorFlow, Keras and Pytorch. Graphics processing units (GPUs) can accelerate deep learning tasks.



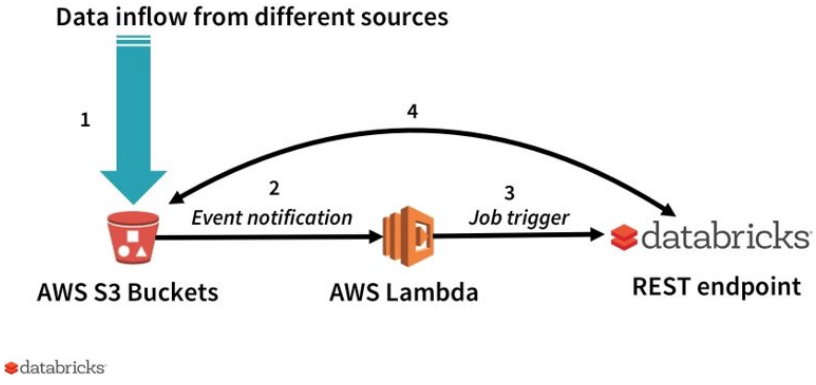
AWS S3

The AWS S3 bucket shall be mounted through DBFS for storing the data in cloud as AWS S3 offers more storage capacity and provides high availability. The mount is a pointer that points to an S3 address. There are two ways to establish access to S3; (a) IAM roles and (b) access keys. S3 can be easily mounted with DBFS using the mount command given below.

```
dbutils.fs.mount(s"s3a://$AccessKey:$EncodedSecretKey@$AwsBucketName", s"/mnt/$MountName")
```

Once the S3 bucket is mounted the data can be accessed as if they are stored in the local machine and the following command can be used to easily unmount an S3 bucket.

```
dbutils.fs.unmount("/mnt/MOUNT_NAME")
```



Using AWS S3 on Databricks provides the following advantages:

1. Security
   It is important to make sure the data is secure, since the visually impaired person could click images that can reveal their whereabouts, or the images may contain personally identifiable information. IAM roles shall be used to specify which cluster can access which buckets, since keys can show up in logs and table metadata and are therefore fundamentally insecure.

2. Schema
   While creating a data frame we can infer the schema or explicitly set a pre-defined schema if we have one. Once we have specified the file metadata, we can create a Data Frame.
   df = spark.read.format(dbutils.widgets.get("file_type")).option("inferSchema", "true").load(dbutils.widgets.get("file_location"))

3. Scalability
   With cross-AZ replication that automatically replicates across different data centers, S3's availability and durability is far superior than traditional file systems such as HDFS. Amazon claims 99.99% durability and 99.99% availability.

4. Performance
   Performance is validated by testing the scalability and the reliability of hardware, software and network. Even though File systems support strong throughput (~3GB/node local read) on a per node basis compared to S3(~600MB/s read), cheap storage facility on S3 will make it 2X better compared to EFS/HDFS.
   https://databricks.com/blog/2017/05/31/top-5-reasons-for-choosing-s3-over-hdfs.html

Data Transformation

In this step, we transform the data into usable format. The data is processed such that they follow the properties of database like Atomicity, Consistency, Integrity and Durability. We load the data through the Databricks DBFS where the data is staged for transformations. Databricks allows creation of tables for staging the data on DBFS.

We will perform cleaning and transformation on the data from database. There are two inputs in our dataset that is image and text descriptions. Cleansing of the data is an important part of preparing the data before it can be used for modeling and prediction. Data that is clean and devoid of any noise causing elements is extremely important for modeling results to be accurate. We have two data cleansing tasks one pertaining to the images and the other text. In the data we have, every image has five textual comments, each of which is formatted in sentence case. Hence, it is important for us to have clean images and clean text for analytics. We are listing below the steps of data cleansing that we think are necessary for our data.

Transformations on text data

Below is the text file as available from the data source, which provides 5 captions per image which the number present along with each image to identify the sequence of captions.

```
▸ (1) Spark Jobs
1000092795.jpg_0         Two young guys with shaggy hair look at their hands while hanging out in the yard .
1000092795.jpg_1        " Two young , White males are outside near many bushes ."
1000092795.jpg_2         Two men in green shirts are standing in a yard .
1000092795.jpg_3         A man in a blue shirt standing in a garden .
1000092795.jpg_4         Two friends enjoy time spent together .
10002456.jpg_0   Several men in hard hats are operating a giant pulley system .
10002456.jpg_1   Workers look down from up above on a piece of equipment .
10002456.jpg_2   Two men working on a machine wearing hard hats .
10002456.jpg_3   Four men on top of a tall structure .
10002456.jpg_4   Three men on a large rig .
```

Some of the steps involved for text transformations:

- Converting to lower case.
    - For ease in processing all comments shall be converted to lower case.
    - This needs to be done, otherwise “child” and “Child will be considered as different words which can affect the model performance.

- Removal of Special Characters.
    - Characters like ‘#’, ‘$’, ‘&’ shall be removed from the comments column as they are not required for generating captions.
- Alphanumeric Characters.
    - There are certain words like ‘bird201’ which are alphanumeric. We shall exclude these while processing the data.

After Cleansing the data

- Storing Unique words.
    - We need to identify and store the unique words from the entire set of 40,000+ captions for the 8000+ images that we have.
    - This will act as the vocabulary of words for the processing and prediction model.
- Word occurrence greater than 10.
    - Being a prediction model, it would not make sense to have all the words in the vocabulary as there would be many words with very few occurrences.
    - This will affect the performance of the model making it fragile to outliers, hence making more mistakes.
    - To avoid this issue, we shall consider only the words that have occurred at least 10 times.

Implementation with code:

1. Firstly, the data is read into a dictionary to form key value pairs of image and text, where ‘image.jpg’ is read as key with 5 captions per image as values.

{'5412034666': ['Two men in plaid shirts and orange hard hats are standing on a hillside looking at a recently sawed down tree .', 'Two men with orange safety hats are inspecting the stump of a recently cut down tree .', 'Two men wearing orange hard hats in the woods looking at a cut down tree .', 'Two men in a forest cutting down a tree .', 'Two men a chopping lumber .'], '2774012125': ['" Children playing on a sandy playground , and one of them has a bike ."', 'Some South American children walk

2. Tokenize the words and perform the a few operations such as Remove punctuation from each token, hanging 's' and 'a' and tokens with numbers in them before storing them as strings.

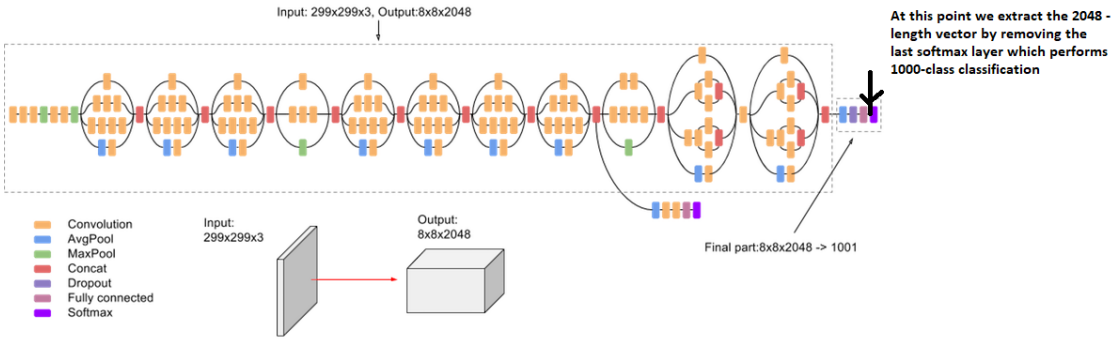
```
descriptions['1000092795']
```

```
Out[37]:
['two young guys with shaggy hair look at their hands while hanging out in the yard',
 'two young white males are outside near many bushes',
 'two men in green shirts are standing in yard',
 'man in blue shirt standing in garden',
 'two friends enjoy time spent together']
```

3. The text data is parsed such that vocabulary of words are formed out of the loaded descriptions. There are around 8735 unique words in the captions among the 40,000+ captions.

Transformations with code implemented on images

1. The images and the input variable (X) in this case and are to be converted to a fixed size vector so that they can be fed into the Convolutional Neural Network, in this case the InceptionV3 model developed by Google. The purpose in this process is not to classify the image but get a fixed length vector for each image. The process of conversion is represented in the image below.



The code block below has the code to carry out the process mentioned above.

Cmd 16

```
def preprocess(image_path):
    # Convert all the images to size 299x299 as expected by the inception v3 model
    img = image.load_img(image_path, target_size=(299, 299))
    # Convert PIL image to numpy array of 3-dimensions
    x = image.img_to_array(img)
    # Add one more dimension
    x = np.expand_dims(x, axis=0)
    # preprocess the images using preprocess_input() from inception module
    x = preprocess_input(x)
    return x
```

2. The captions shall be the output variable (Y) as the model is going to predict the caption. But the caption prediction doesn't happen all at once, instead happens word by word. To carry this out a few items are required.
    a. Two python dictionaries – one for word-to-index and one for index-to-word
    b. A parameter to calculate the maximum length of a caption.

Visualization

Visual representation of the contents of the text data is one of the most effective way to explore the hidden information. By extracting the information from the text data at different level of detail we can easily communicate with the viewer on what exactly a given piece of text is trying to convey.

The dataset we have chosen is composed of Images and their textual descriptions. We have performed analysis and Visualization on the textual descriptions. Analysis on the textual description of images gives an understanding of the images and the entities they represent. We have employed the techniques below to understand the trends in the text to see the most popular words and other word trends to give us a deeper understanding of what the captions are mostly composed of.

Analysis of Textual Data

Reading the text data for captions and storing it as a data frame.

```
In [3]: df=pd.read_csv(r"C:\Users\gopika\Downloads\Flickr30k.csv", delimiter=',', encoding="utf-8-sig")

In [4]: df.head()
```

Out[4]:

	image_name	comment
0	1000092795.jpg_0	Two young guys with shaggy hair look at their...
1	1000092795.jpg_1	Two young , White males are outside near many...
2	1000092795.jpg_2	Two men in green shirts are standing in a yard .
3	1000092795.jpg_3	A man in a blue shirt standing in a garden .
4	1000092795.jpg_4	Two friends enjoy time spent together .

Data Pre-Processing

Steps Performed:

1. Null values for text column (if any) are removed.
2. Full stops and other punctuation marks are removed.

```
In [10]: #text data pre-processing
df = df[~df['comment'].isnull()]
def preprocess(comment):
    comment = comment.str.replace("(<br/>)", "")
    comment = comment.str.replace('(<a).*(>).*(</a>)', '')
    comment = comment.str.replace('(&amp)', '')
    comment = comment.str.replace('(&gt)', '')
    comment = comment.str.replace('(&lt)', '')
    comment = comment.str.replace('(\xa0)', ' ')
    return comment
df['comment'] = preprocess(df['comment'])
```

3. Added two columns named comment length that stores the length of each comment and word count that stores the number of words in each comment using the lambda function for visualization purpose.

```
In [11]: #adding two columns called comment length and word count
         df['comment_len'] = df['comment'].astype(str).apply(len)
         df['word_count'] = df['comment'].apply(lambda x: len(str(x).split()))
```

```
In [16]: df[['comment_len']]
```

Out[16]:

	comment_len
0	84
1	55
2	49
3	45
4	40
5	63
6	58
7	49
8	38
9	27
10	73

```
In [17]: df[['word_count']]
```

Out[17]:

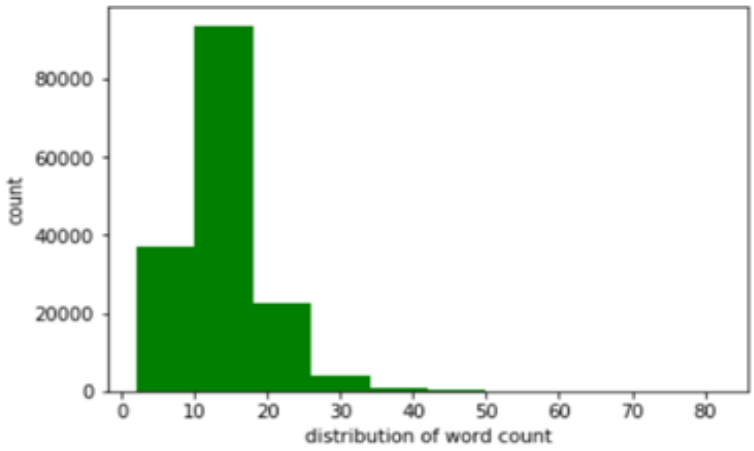
	word_count
0	17
1	11
2	11
3	11
4	7
5	12
6	12
7	10
8	9
9	7
10	18

1. Word Frequency

Word frequency graph shows the distribution of word count of each caption in the textual description of images. A simple histogram is plotted to find out the distribution of word count. It can be observed that most of the captions (~8800) has a word count in the range of 10 to 20. A

few captions have a word count of 30 and above. This analysis gives an idea of how descriptive an image can be represented through captions.

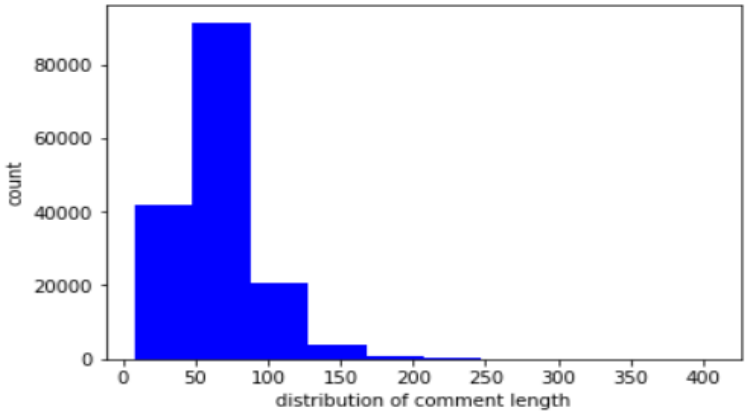
```
In [19]: _=plt.hist(df['word_count'],color='g')
         _=plt.xlabel('distribution of word count')
         _=plt.ylabel('count')
         plt.show()
```



2. Word Length

Word Length graph shows the distribution of the word length of each caption in the textual description of images. A simple histogram is plotted to find out the distribution of word length. It can be observed that most of the captions (~8800) has a word count in the range of 10 to 20.A few captions have a word count of 30 and above. This analysis again represents the extent of information depicted in images.

```
In [22]: import matplotlib.pyplot as plt
         _=plt.hist(df['comment_len'],color='b')
         _=plt.xlabel('distribution of comment length')
         _=plt.ylabel('count')
         plt.show()
```



3. Most common words

An analysis to find out the most frequent words used in the whole corpus of textual captions is done. This is implemented with the help of a user defined function. The top 20 most common words are then displayed.

In [38]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [40]:
```
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
```

In [42]:
```
common_words = get_top_n_words(df['comment'], 20)
for word, freq in common_words:
    print(word, freq)
```

```
in 83520
the 62984
on 45686
and 44299
man 42626
is 41117
of 38844
with 36208
woman 22213
two 21680
are 20196
to 17627
people 17338
at 16262
an 15884
wearing 15714
shirt 14342
white 13290
young 13219
black 12391
```

As it can be observed from the output that most of the high frequency words are stop words like in, the, and, etc. with less relevance. Thus, we further did an analysis to find out the most common words after eliminating the stop words from the analysis. Now we have got some relevant pieces of words that truly represents the images in the dataset.

Finding the most common words after eliminating stop words like and, the, at etc.

In [14]:
```
# Finding the most common words after removing the stop words
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer(stop_words = 'english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words = get_top_n_words(df['comment'], 20)
for word, freq in common_words:
    print(word, freq)
```

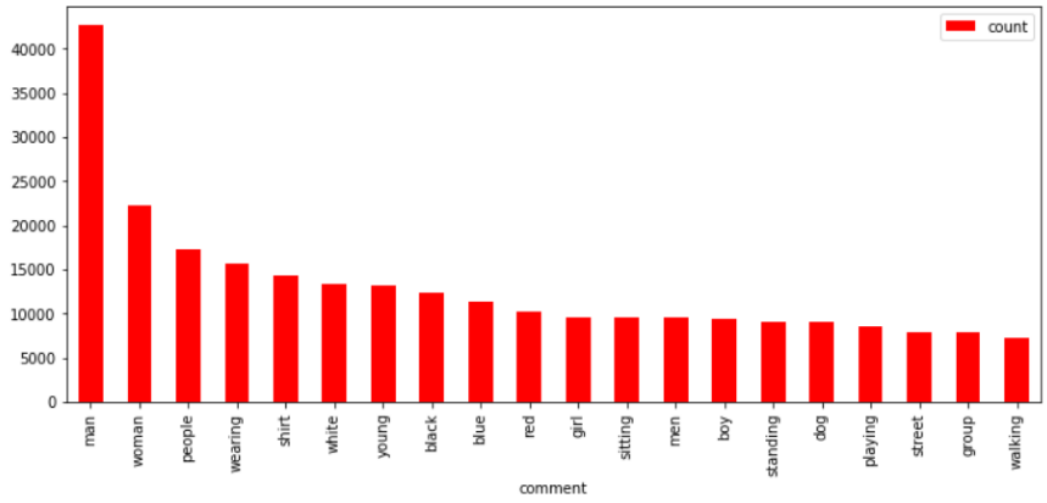
```
man 42626
woman 22213
people 17338
wearing 15714
shirt 14342
white 13290
young 13219
black 12391
blue 11390
red 10278
girl 9656
sitting 9620
men 9499
boy 9430
standing 9114
dog 9092
playing 8622
street 8012
group 7853
walking 7339
```

A graph is then plotted to get the visual representation of the most common words. From the graph it can be observed that most of the images have a common entity called man.

Distribution of the most common words after removing the stop words

In [81]:
```
import matplotlib.pyplot as plt
%matplotlib inline
df1 = pd.DataFrame(common_words, columns = ['comment' , 'count'])
df1.plot.bar(x='comment',y='count',color='r',figsize=(12,5))
```

Out[81]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d762e7780>
```



4. Top Bigrams after removing the stop words

A bigram model (N = 2) predicts the occurrence of a word based on its previous word. The word groups provide more benefits than only one word when explaining the meaning of a text. Here we have found the most common bigrams using the traditional bag of words approach in the text captions after eliminating the stop words.

Fetching the most common bigrams after eliminating the stop words

In [16]:
```
# Top Bigrams after removing the stop words
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2, 2), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words = get_top_n_bigram(df['comment'], 20)
for word, freq in common_words:
    print(word, freq)
df2 = pd.DataFrame(common_words, columns = ['comment' , 'count'])
```

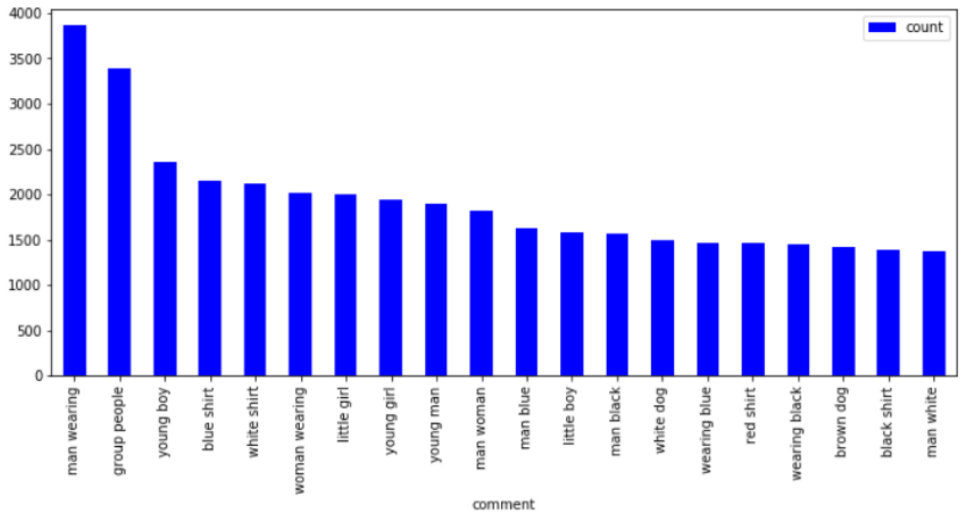
```
man wearing 3858
group people 3386
young boy 2360
blue shirt 2158
white shirt 2128
woman wearing 2020
little girl 1999
young girl 1938
young man 1899
man woman 1830
man blue 1627
little boy 1590
```

Bigram Visualization

A graph is then plotted to get the visual representation of the bigrams. From the plot it can be observed that most of the images have “man wearing” as the caption.

In [17]:
```
# Distribution of bigrams after removing the stop words
df2 = pd.DataFrame(common_words, columns = ['comment' , 'count'])
df2.plot.bar(x='comment',y='count',color='b',figsize=(12,5))
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x2192f0e24a8>



5. Top Trigrams after removing the stop words

A trigram model (N = 3) predicts the occurrence of a word based on its previous two words. Here we have found the most common trigrams using the traditional bag of words approach in the text captions after eliminating the stop words.

```
In [24]: # Top Trigrams after removing the stop words
def get_top_n_trigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(3, 3), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words = get_top_n_trigram(df['comment'], 20)
for word, freq in common_words:
    print(word, freq)
df3 = pd.DataFrame(common_words, columns = ['comment' , 'count'])
```

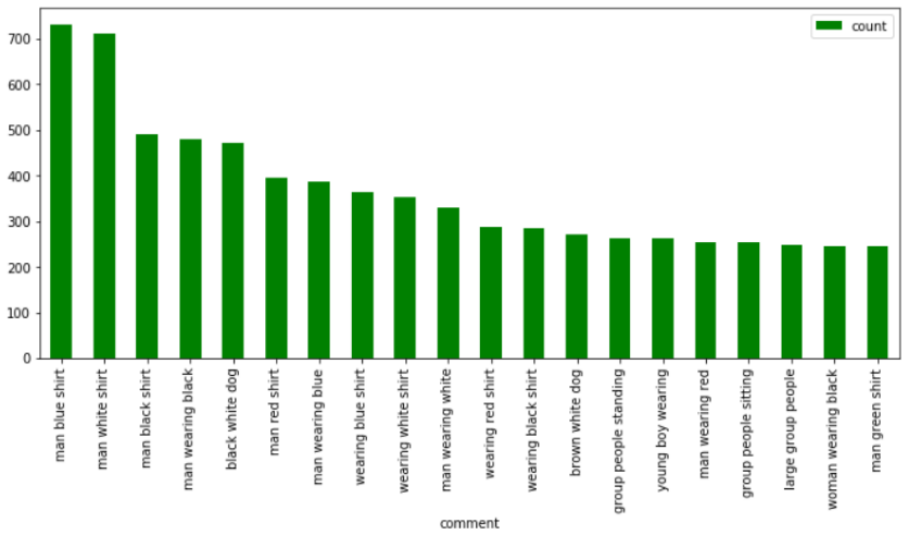
```
man blue shirt 730
man white shirt 711
man black shirt 491
man wearing black 479
black white dog 471
man red shirt 395
man wearing blue 388
wearing blue shirt 364
wearing white shirt 352
man wearing white 330
wearing red shirt 287
wearing black shirt 285
brown white dog 271
group people standing 264
young boy wearing 262
man wearing red 255
```

Trigram Visualization

A graph is then plotted to get the visual representation of the trigrams. From the plot it can be observed that most of the images have “man wearing blue shirt” as the caption.

```
In [26]: # Distribution of trigrams after removing the stop words
df3 = pd.DataFrame(common_words, columns = ['comment' , 'count'])
df3.plot.bar(x='comment',y='count',color='g',figsize=(12,5))
```

```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x21934578c18>
```



6. Word-Cloud

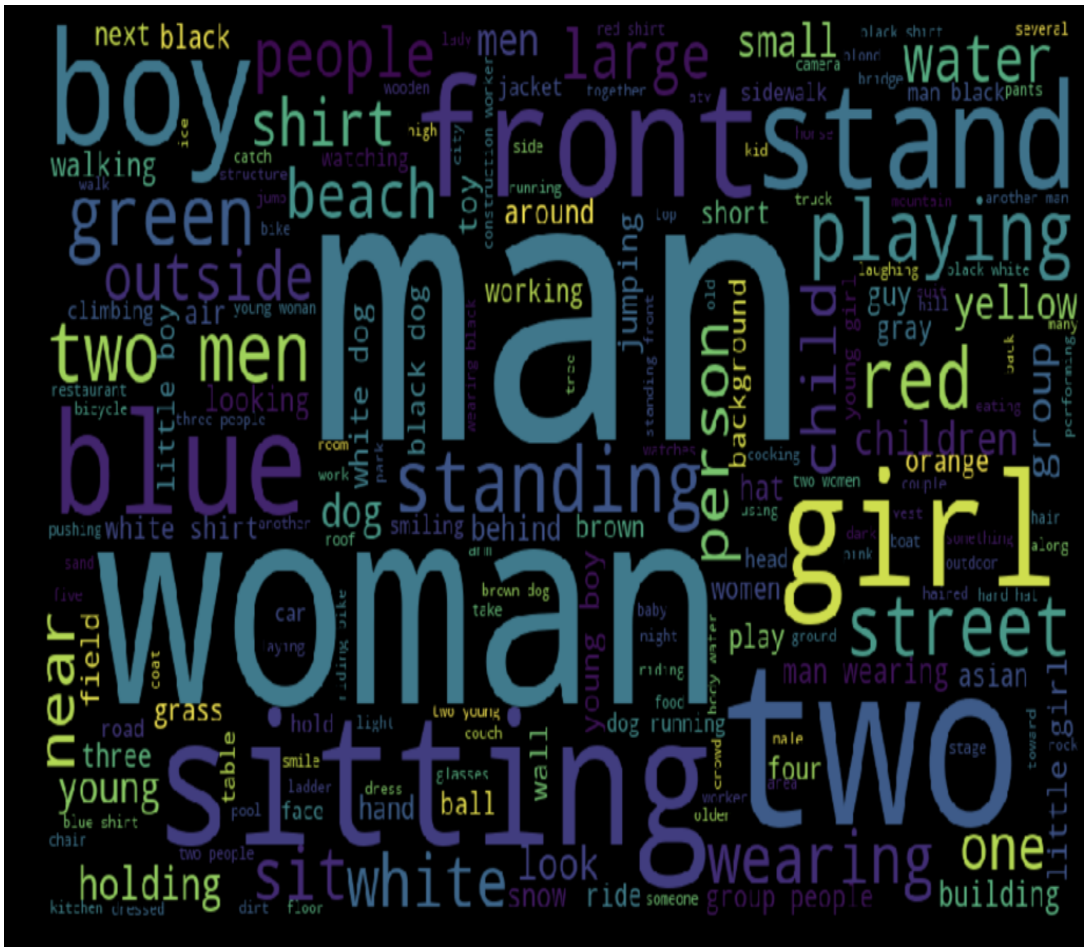
A word-cloud is used to depict tags arranged in space that vary in size, color, and position based on tag frequency, categorization, or significance. The most frequently occurring words are bigger in size. As we can see from the plot, words like man, woman, sitting, boy, front, standing, sitting seem to be the important tags for this data. We can probably say that the dataset describes scenes that involve people performing a variety of actions.

Code:

```
df = pd.read_csv("/Users/mythri/Downloads/Flickr30k.csv", header=0)
cloud_words = ' '
stopwords = set(STOPWORDS)
```

```
# word cloud
#include standard stop words
cloud_words = ' '
stopwords = set(STOPWORDS)

cloud_words=' '
for row in df.loc[:,"comment"]:
    # typecasting each value in the df to string
    row = str(row)


    # splitting the values into tokens
    tokens = row.split()

    # Converting the tokens to lower case
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()


    for words in tokens:
        cloud_words = cloud_words + words + ' '

    wordcloud = WordCloud(width = 1000, height = 1000,
                background_color ='black',
                stopwords = stopwords,
                min_font_size = 12,mask=image_cloud).generate(cloud_words)
```

```
plt.figure(figsize = (12, 12), facecolor ="black")
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



7. Sentiment Analysis

Sentiment analysis is mining of text which identifies and extracts subjective information in the given text and helps understand if the text has more of a positive, negative or neutral tone. We have used the nltk library and the VADER's sentiment analyzer which is a simplified form of sentiment analysis. For each caption, we have a corresponding score for positive, negative and neutral components. We have also plotted a histogram to see how the positive, negative and neutral scores (0 to 1) are varying across different captions in the dataset.

Analysis

In [146]:

```
df = pd.read_csv("/Users/mythri/Downloads/flickr30k.csv", header=0,encoding="UTF-8")


def sentiment_analysis(text):


    sentiment_analysis = SentimentIntensityAnalyzer()
    score =  sentiment_analysis.polarity_scores(text)
    return score



sentiment_analysis_results = [sentiment_analysis(row) for row in df.loc[:,"comment"]]
sentiment_analysis_results_df = pd.DataFrame(sentiment_analysis_results)
comment_df = pd.DataFrame(df, columns = ["comment"])
analysis_df = comment_df.join(sentiment_analysis_results_df)


analysis_df.head(10)
```

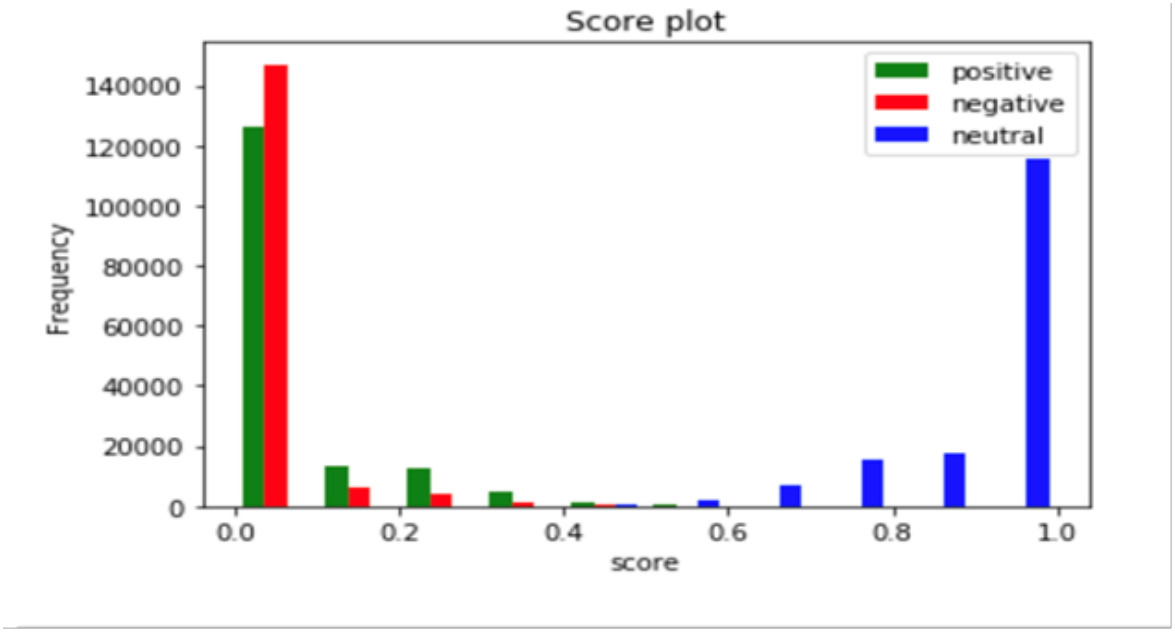
Out[146]:

	comment	compound	neg	neu	pos
0	Two young guys with shaggy hair look at their...	0.0000	0.000	1.000	0.000
1	Two young , White males are outside near many...	0.0000	0.000	1.000	0.000
2	Two men in green shirts are standing in a yard .	0.0000	0.000	1.000	0.000
3	A man in a blue shirt standing in a garden .	0.0000	0.000	1.000	0.000
4	Two friends enjoy time spent together .	0.7430	0.000	0.388	0.612
5	Several men in hard hats are operating a gian...	-0.1027	0.135	0.865	0.000
6	Workers look down from up above on a piece of...	0.0000	0.000	1.000	0.000
7	Two men working on a machine wearing hard hats .	-0.1027	0.167	0.833	0.000
8	Four men on top of a tall structure .	0.2023	0.000	0.769	0.231
9	Three men on a large rig .	-0.1280	0.273	0.727	0.000

Plot

In [86]:

```
legend = ['positive', 'negative','neutral']
pos = analysis_df['pos']
neg = analysis_df['neg']
neu = analysis_df['neu']

plt.hist([pos, neg,neu], color=['green', 'red','blue'])
plt.legend(legend)
plt.xlabel("score")
plt.ylabel("Frequency")
plt.title('Score plot')

plt.show()
```



Generative Adversarial Networks (GANs)

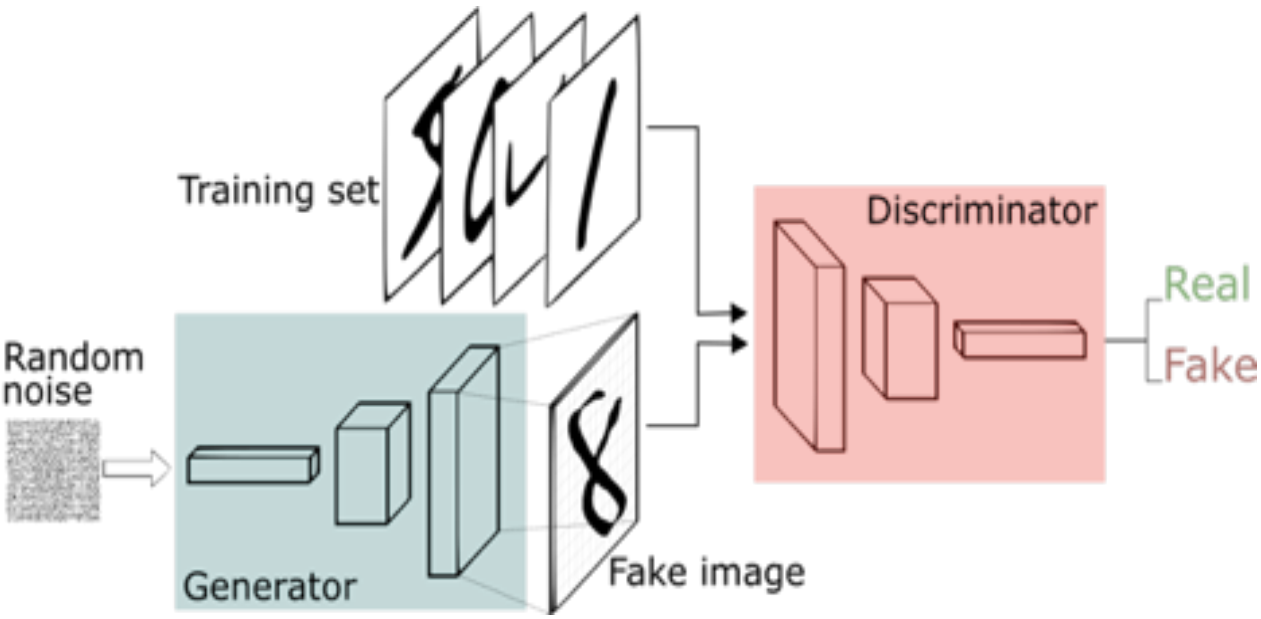
We proposed to use GANs for this project to remove blur from images. Motion blur is caused due to a relative motion between the camera and the object.

We planned to use the Vizwiz dataset which contains the images captured by visually impaired people. These images have a motion blur because of which processing them to obtain a caption will be a difficult task.

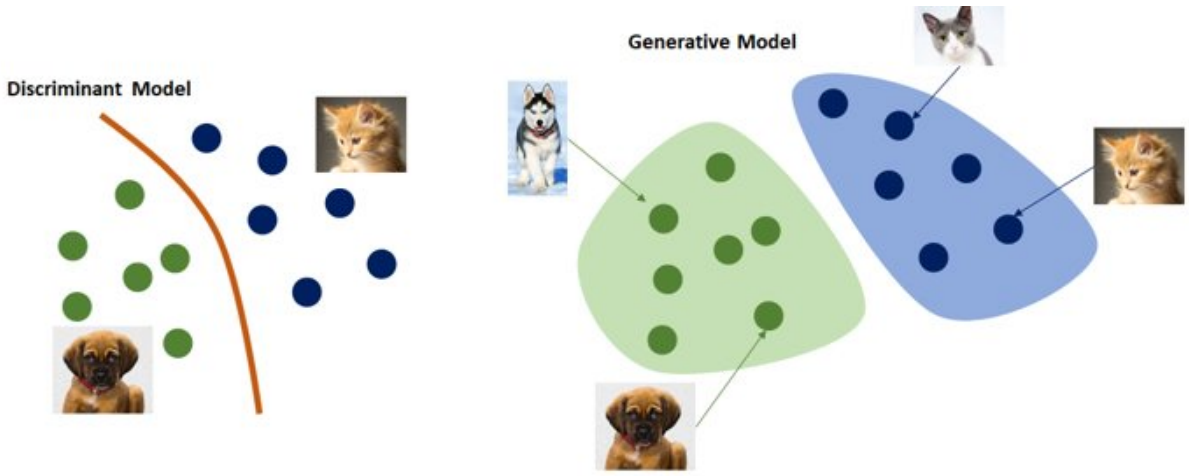
We started to explore GANs and attempted to understand the techniques through various literature sources. After our study we concluded that GANs requires a good dataset with training images that are blurred and have a corresponding sharp image. The model needs good training with plenty of images to be able to read new images and de blur them. Hence, we used the GoPro dataset (9 GB) that contains a set of these images. We employed the technique of GANs on this data. (https://drive.google.com/file/d/1H0PIXvJH4c40pk7ou6nAwoxuR4Qh_Sa2/view)

General Working of GANs

- Generative adversarial networks (GANs) consists of two deep neural networks pitting one against the other. This gives it the name adversarial.
- GANs can mimic any distribution of data. It can reproduce images, speech, music and images.
- GANS consists of two neural networks.
    - One neural network is the generator that generates new data instances.
    - The other network is the discriminator which evaluates them for authenticity, if this instance belongs to the original data or not.
- These networks are based on Generative and Discriminative Algorithms.

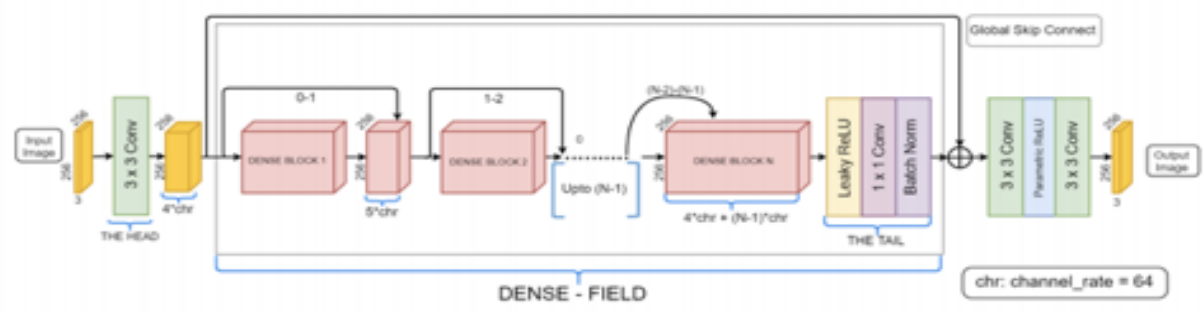


Generative Algorithms	Discriminative Algorithms
• Model the distribution of individual classes in a given dataset	• Discriminative models learn the boundary between classes
• Assuming a label, they try to predict the features based on the label.	• Based on the features, they predict a label or category



The above picture describes the discriminative and generative model with images of cats and dogs.

Model Architecture



- This model consists of densely connected generator and discriminator.
- Generator recycles features spanning across multiple receptive scales to generate an image that confuses the discriminator
- Discriminator has to analyze different patches in each image and identifiy the distribution from which each of its input images is coming from.

Generator

All the components below work together to create fake images.

Head

- Channel rate is the constant number of activation channels that are output by each convolutional layer with a value of 64**.**
- There is a simple 3 × 3 convolutional layer which convolves over the raw input image and outputs 4×channel-rate (256) feature activations.
- This provides sufficient activation to trigger densely connected stack of layers.

```
Cmd 8

1 channel_rate = 64
2 image_shape = (256, 256, 3)
3 patch_shape = (channel_rate, channel_rate, 3)

Command took 0.01 seconds -- by mgopaluni1@student.gsu.edu at 5/3/2019, 1:39:47 PM on gans_cluster
```

Dense Field

- This section consists of N number of convolutional ‘blocks’ placed sequentially.
- The blocks have their outputs fully connected with the output of the layer ahead of them.

Tail

- The tail helps in increasing the number of features to map.

Global Skip Connection

- The output from the head of the network with the output of the tail is concatenated to improve the generator performance.
- We pass knowledge from lower level to the upper layers through dense connection which enables the network to learn faster and perform deblurring much more efficiently.

```
Cmd 9

def generator_model():
    inputs = Input(shape=(None, None, 3))
    h = Convolution2D(filters=4 * channel_rate, kernel_size=(3, 3), padding='same')(inputs)
    # Dense blocks
    d_1 = dense_block(inputs=h)
    x = concatenate([h, d_1])
    d_2 = dense_block(inputs=x, dilation_factor=(1, 1))
    x = concatenate([x, d_2])
    d_3 = dense_block(inputs=x)
    x = concatenate([x, d_3])
    d_4 = dense_block(inputs=x, dilation_factor=(2, 2))
    x = concatenate([x, d_4])
    d_5 = dense_block(inputs=x)
    x = concatenate([x, d_5])
    d_6 = dense_block(inputs=x, dilation_factor=(3, 3))
    x = concatenate([x, d_6])
    d_7 = dense_block(inputs=x)
    x = concatenate([x, d_7])
    d_8 = dense_block(inputs=x, dilation_factor=(2, 2))
    x = concatenate([x, d_8])
    d_9 = dense_block(inputs=x)
    x = concatenate([x, d_9])
    d_10 = dense_block(inputs=x, dilation_factor=(1, 1))
    # Tail
    x = LeakyReLU(alpha=0.2)(d_10)
    x = Convolution2D(filters=4 * channel_rate, kernel_size=(1, 1), padding='same')(x)
    x = BatchNormalization()(x)
    # Global Skip Connection
    x = concatenate([h, x])
    x = Convolution2D(filters=channel_rate, kernel_size=(3, 3), padding='same')(x)
    x = LeakyReLU(alpha=0.2)(x)
    #get the output image
    outputs = Convolution2D(filters=3, kernel_size=(3, 3), padding='same', activation='tanh')(x)
    model = Model(inputs=inputs, outputs=outputs, name='Generator')
    return model

Command took 0.01 seconds -- by mgopaluni1@student.gsu.edu at 5/3/2019, 1:39:55 PM on unknown cluster
```

Discriminator

- Discriminator guides the statistics that the generator employs to create images
- A Markovian discriminator with 10 convolutional layers is employed.

```
def discriminator_model():

    inputs = Input(shape=patch_shape)
    x = Convolution2D(filters=channel_rate, kernel_size=(3, 3), strides=(2, 2), padding="same")(inputs)
    x = BatchNormalization()(x)
    x = LeakyReLU(alpha=0.2)(x)

    x = Convolution2D(filters=2 * channel_rate, kernel_size=(3, 3), strides=(2, 2), padding="same")(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(alpha=0.2)(x)

    x = Convolution2D(filters=4 * channel_rate, kernel_size=(3, 3), strides=(2, 2), padding="same")(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(alpha=0.2)(x)

    x = Convolution2D(filters=4 * channel_rate, kernel_size=(3, 3), strides=(2, 2), padding="same")(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(alpha=0.2)(x)

    x = Flatten()(x)
    outputs = Dense(units=1, activation='sigmoid')(x)
    model = Model(inputs=inputs, outputs=outputs, name='PatchGAN')

    inputs = Input(shape=image_shape)

    list_row_idx = [(i * channel_rate, (i + 1) * channel_rate) for i in
                    range(int(image_shape[0] / patch_shape[0]))]
    list_col_idx = [(i * channel_rate, (i + 1) * channel_rate) for i in
                    range(int(image_shape[1] / patch_shape[1]))]

    list_patch = []
    for row_idx in list_row_idx:
        for col_idx in list_col_idx:
            x_patch = Lambda(lambda z: z[:, row_idx[0]:row_idx[1], col_idx[0]:col_idx[1], :])(inputs)
            list_patch.append(x_patch)

    x = [model(patch) for patch in list_patch]
    outputs = Average()(x)
    model = Model(inputs=inputs, outputs=outputs, name='Discriminator')
    return model
```

Loss Functions

- The losses are defined by l1 or l2 loss between the ground truth and the rectified image as the chief objective function.
- For adversarial framework this loss is pooled with the adversarial loss which measures how well the generator is performing with respect to fooling the discriminator.

L1 loss alone creates extremely smooth images and thus they look dull and the blur still remains. In adversarial loss the color distribution is better, however it does not have a good idea of the structure of the image and the discriminator is judging the generator performance based on the output image and is not considering the blurred input. These are limitations that do not generate the perfect image we are looking for hence, we have to make use of perceptual loss.

Perceptual Loss

We must add structural knowledge into the generator to counter the patch-wise judgement of the Markovian discriminator. We consider perceptual loss which is the Euclidean difference between deep convolutional activations of the ground truth and generated latent image.

Cmd 17

```
def l1_loss(y_true, y_pred):
    return K.mean(K.abs(y_pred - y_true))
```

Command took 0.01 seconds -- by mgopaluni1@student.gsu.edu at 5/3/2019, 1:41:06 PM on gans_cluster

Cmd 18

```
def perceptual_loss(y_true, y_pred):
    vgg = VGG16(include_top=False, weights='imagenet', input_shape=image_shape)
    loss_model = Model(inputs=vgg.input, outputs=vgg.get_layer('block3_conv3').output)
    loss_model.trainable = False
    return K.mean(K.square(loss_model(y_true) - loss_model(y_pred)))
```

Command took 0.01 seconds -- by mgopaluni1@student.gsu.edu at 5/3/2019, 1:41:12 PM on gans_cluster

Cmd 19

```
def generator_loss(y_true, y_pred):
    return K_1 * perceptual_loss(y_true, y_pred) + K_2 * l1_loss(y_true, y_pred)
```

Command took 0.01 seconds -- by mgopaluni1@student.gsu.edu at 5/3/2019, 1:41:17 PM on gans_cluster

Cmd 20

```
def adversarial_loss(y_true, y_pred):
    return -K.log(y_pred)
```

Command took 0.01 seconds -- by mgopaluni1@student.gsu.edu at 5/3/2019, 1:41:21 PM on gans_cluster

Images into the Adversarial Network

We used a blurred image and a corresponding deblurred image for the training set. Along with this a corresponding pair of images created by the generator was fed into the discriminator. After processing we got an output that contained the images we fed and the deblurred image.

Cmd 12

```
def generator_containing_discriminator(generator, discriminator):
    inputs = Input(shape=image_shape)
    generated_image = generator(inputs)
    outputs = discriminator(generated_image)
    model = Model(inputs=inputs, outputs=outputs)
    return model
```

Command took 0.01 seconds -- by mgopaluni1@student.gsu.edu at 5/3/2019, 1:40:11 PM on gans_cluster

Cmd 26

```
if __name__ == '__main__':
    train(batch_size=1, epoch_num=5)
    test(2)
```

```
epoch:  1 / 5
batches:  3
index  0
batch 1 d_loss : 0.689947
/databricks/python/lib/python3.6/site-packages/keras/engine/training.py:490: UserWarning: Discrepancy between trainable weights
calling `model.compile` after ?
  'Discrepancy between trainable weights and collected trainable'
batch 1 d_on_g_loss : 0.494236
batch 1 g_loss : 24063.669922
index  1
inside interim
batch 2 d_loss : 0.167773
batch 2 d_on_g_loss : 0.133070
batch 2 g_loss : 20547.886719
index  2
inside interim
batch 3 d_loss : 0.173458
```

Results

We have used 25 training images and 4 test images. Issues in upload and processing time lead to uploading few images. only We have tried to run the experiment with different number of epochs. One epoch consists of one full training cycle on the training set.

Time consumed:

- 1 epoch: 1.5- 2 hours
- 5 epochs: 6 hours.

Below are the images that we obtained through deblurring.

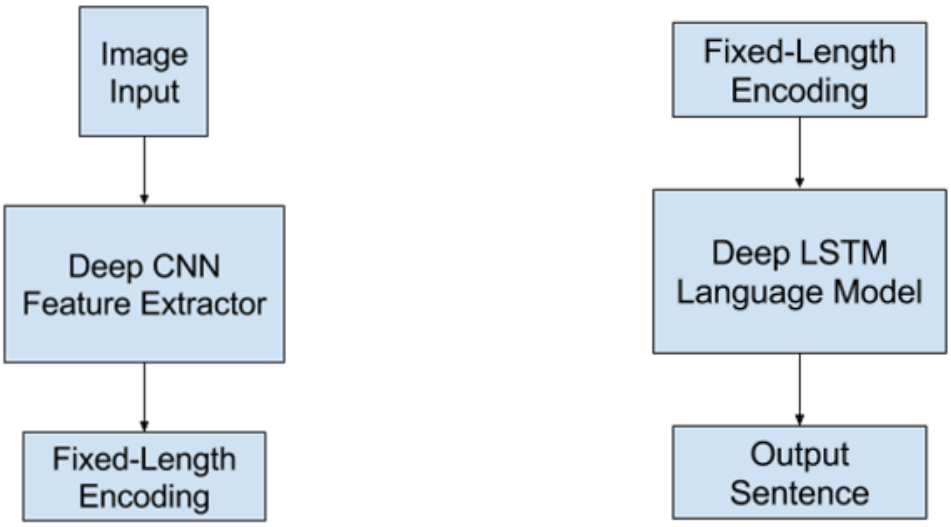
The first image is a result of running one epoch while the second is a result of running 5 epochs. Also, we can see that the first image has more components than the second image. This makes the second image relatively easily to process and produce better results. Upon increasing the number, the time taken was very long and the clusters were getting terminated. Also, we were facing “out of memory “issue. Hence, we were unable to get images that were deblurred. Given the limitations, we need to find a robust and cost-effective platform to be able to get the desired output-sharp/deblurred images.

Implementation & Experience

As mentioned in the introduction, defining a purpose for the project was extremely important in order to be able to set the goals and objectives right. With a social cause on mind, the project 3RDEYE was designed carefully considering the pros and cons for each technology that would be used. It was only when we started to implement it and integrate the technologies, we were paused multiple times to research around the possible solutions.

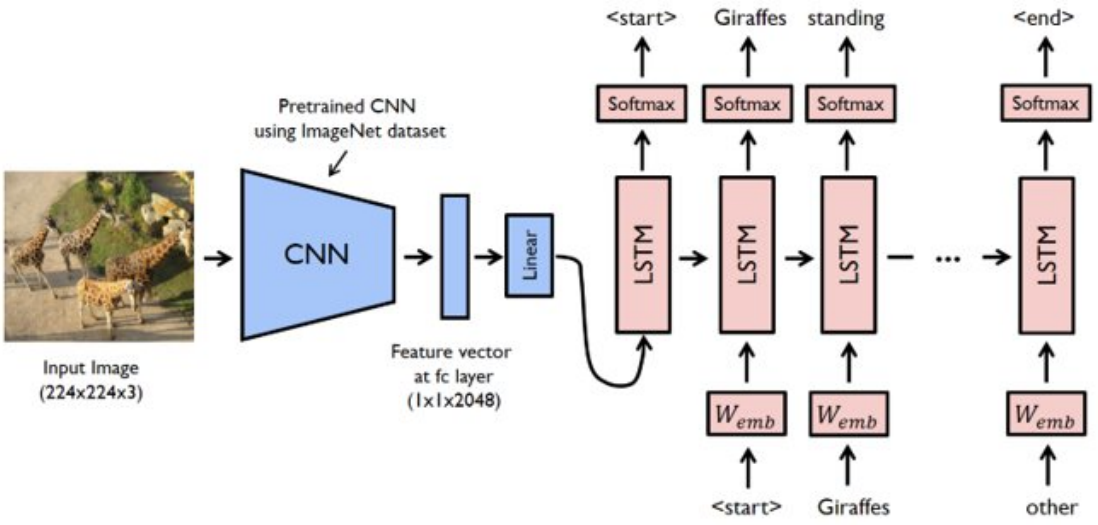
The project having to aid visually impaired, it was important to consider various possibilities in terms of inputs. Having blurred images along with clear ones was included in this phase. Further, the dataset had to be large for training our model well and to suffice this we decided to mount that data on AWS S3 bucket. It was a learning for some of us in the team on how the S3 is mounted, how the Lambda is used to unzip the files. These being some of the basic tasks on AWS S3, we also learnt how to integrate Databricks Enterprise Edition with AWS S3 bucket. Here, we faced multiple challenges in terms of the cost and integration. Databricks Enterprise Edition costs $0.07/DBU (Data Engineering Light) to $0.40/DBU (Data Analytics) where DBU is a unit of processing capability per hour, billed on a per-second usage. We started with a trial version on the enterprise edition and went ahead to integrate S3 bucket. This task required for us to mount a new bucket and follow procedures on integration from the link https://docs.databricks.com/administration-guide/cloud-configurations/aws/iam-roles.html

Once the integration was successful, we were able to access our data on S3 and pull them on DBFS (Database File Systems) for further data transformations and analytics. Multiple transformations and cleaning procedures were done on the dataset to get it to a usable format as listed above. To perform image recognition on the data, we have used ImagenetV3 pretrained model that gives us the vectorized features of each image. The vector output was streamlined with tokenized and cleansed text descriptions of each image into a LSTM model.



The output at this step was given to Keras deep learning model which is then trained using various epoch, batch size and activation functions. One of the challenges we faced at this stage was that the model took hours to process. The reason was computation time, with the cluster that was used having 8 Cores and no computation boost, our model ran for over 3 hours. This kept us from further improving the model. Hence, we had to buy a cluster and EC2 instance that would provide computational boost.

We requested on AWS for a c2.xlarge processor and used it for processing our models. The model took about 2 hours with this instance and cost about 0.20/hour. We tuned the hyper parameters to get a better accuracy on the train dataset. The results are were a mix of perfect descriptions on the images and a few incorrect ones. This is because the model was trained on 8K images which is too less for it to learn well. It had a better chance at detecting animals and gave breed names right, but when there were people involved the model was fuzzy.



As in the last image below, with a greater number of people involved the accuracy on the prediction of description reduced. Increasing the dataset and training the model with variety of images and increasing the cluster computation capabilities would aid in improvising the results.

Below, we can see the different code snippets for how we have built our model and the different elements used in order to complete the modelling. The first code snippet, is a data generator, which is a function that is called before the model is to be fit and this function prepares the data to be fed into the fit_generator().

```
#CIS8395 model.fit_generator()
def data_generator(descriptions, photos, wordtoix, max_length, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    # loop for ever over images
    while 1:
        for key, desc_list in descriptions.items():
            n+=1
            photo = photos[key+'.jpg']
            for desc in desc_list:
                seq = [wordtoix[word] for word in desc.split(' ') if word in wordtoix]
                for i in range(1, len(seq)):
                    in_seq, out_seq = seq[:i], seq[i]
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(out_seq)
            if n==num_photos_per_batch:
                yield [[array(X1), array(X2)], array(y)]
                X1, X2, y = list(), list(), list()
                n=0
```

This code below declares the specification for the model we are building with the data that we have. These include data like the vocabulary size, dense value, the activation required for decoder and output of the neural network.

```
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)

model.summary()
model.layers[2]
```

Next, the epoch values is defined as 10, which for certain data considered it run through 10 iterations and a batch size of 6. After this the function to run and fit the model is executed.

```
epochs = 10
number_pics_per_bath = 6
steps = len(train_descriptions)//number_pics_per_bath
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, wordtoix, max_length, number_pics_per_bath)
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
model.save_weights('model_4.h5')
model.load_weights('model_4.h5')
```

Now, that the model has been fit and run. The next step is to test out the model using some images from our test set to see the results captions produced by the model. The code below uses a method of searching called Greedy Search to build the caption, which basically uses the probability of word matching the images and building a caption based on that.

```
def greedySearch(photo):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = [wordtoix[w] for w in in_text.split() if w in wordtoix]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = ixtoword[yhat]
        in_text += ' ' + word
        if word == 'endseq':
            break
    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final
```

```
#CIS8395 Test code
z=0
z+=1
pic = list(encoding_test.keys())[z]
image = encoding_test[pic].reshape((1,2048))
x=plt.imread(images+pic)
plt.imshow(x)
plt.show()
print(pic)
print("Greedy:",greedySearch(image))
```

Results

We have attached the caption results obtained for some images as can be seen below.

```
[[0.6506623  0.9936032  0.5315549  ... 1.0775828  0.81474996 0.18483704]]
1096395242_fc69f0ae5a.jpg
Greedy: young boy wearing red shirt is playing with toy
Command took 0.09 seconds -- by vmoganti1@student.gsu.edu at 4/25/2019, 6:35:21 PM on Cluster1
```

Cmd 53

```
1 # Below path contains all the images
2 images = '/mnt/My_New_Mount/Flickr_8k/'
3
4 from pyspark.ml.image import ImageSchema
5 image_df = ImageSchema.readImages(images+pic)
6 display(image_df)
```

▸ (1) Spark Jobs
▸ image_df: pyspark.sql.dataframe.DataFrame = [image: struct]

image

```
[[0.2551504  0.1605703  0.11077098 ... 0.10488173 0.62740225 0.083
1107246521_d16a476380.jpg
Greedy: black dog is running through the grass
```

Command took 0.07 seconds -- by vmoganti1@student.gsu.edu at 4/26/2019, 12:52:

Cmd 53

```
# Below path contains all the images
images = '/mnt/My_New_Mount/Flickr_8k/'

from pyspark.ml.image import ImageSchema
image_df = ImageSchema.readImages(images+pic)
display(image_df)
```

▸ (1) Spark Jobs

▸ image_df: pyspark.sql.dataframe.DataFrame = [image: struct]

image


```
[[0.19112732 0.2743543  0.15996975 ... 0.37788364 0.44252807 0.5963851 ]]
136552115_6dc3e7231c.jpg
Greedy: man on bike riding down dirt hill
```

Command took 0.07 seconds -- by vmoganti1@student.gsu.edu at 4/26/2019, 1:07:17 AM on Cluster1

Cmd 57

```
images = '/mnt/My_New_Mount/Flickr_8k/'
image_df = ImageSchema.readImages(images+pic)
display(image_df)
```

▸ (1) Spark Jobs

▸ image_df: pyspark.sql.dataframe.DataFrame = [image: struct]

image


```
1174629344_a2e1a2bdbf.jpg
Greedy: woman in black walking down the street
```

Command took 0.07 seconds -- by vmoganti1@student.gsu.edu at 4/26/2019, 12:56:56 AM on Cluster1

Cmd 59

```
images = '/mnt/My_New_Mount/Flickr_8k/'
image_df = ImageSchema.readImages(images+pic)
display(image_df)
```

▸ (1) Spark Jobs

▸ image_df: pyspark.sql.dataframe.DataFrame = [image: struct]

image


Show image preview

Text-to-Speech – AWS Polly

The final step at transforming text descriptions to audio was implemented using AWS Polly. Each description generated by the model will be stored on the S3 bucket. This will be read using a piece of code written in Python to access and feed to Polly services which provides the audio and also stores it on the S3. Finally, the application would send out this onto the device of the visually impaired individual and played loud and clear. This again was a great learning, for the emerging technologies have made the whole process much simpler to implement or integrate.

Below is the snapshot of the code written in Python to get the text file of the caption generated by the model saved in S3. Use the text to send it to AWS Polly to convert it to speech and then save the file back in S3 so that it can be retrieved for further use.

```
import boto3
from pygame import mixer
import os
#list_open = open('C:/Users/sasid/Desktop/demo.txt',"r")

#reading file from s3
import pandas as pd
s3=boto3.client('s3')
s3.download_file('image-captioning-project','demo.txt','demo_2')
list_open = open('C:/Users/gopika/PycharmProjects/untitled/demo_2',"r")

polly=boto3.client('polly')
text_to_speech=polly.synthesize_speech(Text=list_open.read(),
                        OutputFormat='mp3',
                        VoiceId='Aditi')
print(text_to_speech)

with open('pollyoutput.mp3','wb')as f:
    f.write(text_to_speech['AudioStream'].read())
    f.close()

# Saving to s3
s3 = boto3.resource('s3')
s3.Bucket('image-captioning-project').upload_file('C:/Users/gopika//PycharmProjects//untitled/pollyoutput.mp3','pollyoutput.mp3')
```

Conclusion

With a social cause on mind, the project 3RDEYE was designed carefully considering the pros and cons for each technology that would be used. We have implemented and integrated all the phases of the architecture of our project.

For the purpose of this project the ELT approach is chosen to handle the large volume of unstructured data involved such as images and text. AWS S3 was chosen as the storage platform due to many advantages such as security, schema, performance and scalability. As the data can be accessed from the S3 bucket, it can now be transformed so that the data is in a format that is suitable to fit on different model and obtain predictions. Some of the transformation involved, storing the images and their caption in a dictionary where the image name is the key and the captions its value, tokenizing work and performing operation to remove punctuations and other special characters, forming a vocabulary of word from the caption description of the images, converting the images to fixed size vectors so that they can be sent through Inception V3 CNN model, etc. The data for this project was therefore successfully extracted, loaded and transformed so that the data is now ready to be fit to a model and trained to then obtain the predicted caption. We have included the visualizations to analyze our textual descriptions. From the analysis we find that most captions are neutral in tone. We can extend the visualizations to more captions as they are available.

In the interest of this cause, we hope to extend our work by adding more data to our training and make the predictions more accurate. We also hope to work on GANs and overcome the challenges we faced and be able to produce the desired output. We hope to build a product that is robust and is easy to use. We shall explore different platforms to deploy the application.

References

- Avinoam, R. (2019, February 19). ETL vs ELT: The Difference is in the How. Retrieved April 21, 2019, from https://blog.panoply.io/etl-vs-elt-the-difference-is-in-the-how
- Databricks (2019). Amazon S3. Retrieved April 23, 2019, from https://docs.databricks.com/spark/latest/data-sources/aws/amazon-s3.html
- Amazon Web Services. (2019). What Is AWS Lambda? Retrieved April 23, 2019, from https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- Amazon Web Services. (2019). AWS CLI Command Reference. Retrieved April 23, 2019, from https://docs.aws.amazon.com/cli/latest/reference/lambda/index.html
- Dong, X., Zhu, L., Zhang, D., Yang, Y., & Wu, F. (2018). Fast Parameter Adaptation for Few-shot Image Captioning and Visual Question Answering. 2018 ACM Multimedia Conference on Multimedia Conference - MM 18.
- Hossain, M. Z., Sohel, F., Shiratuddin, M. F., & Laga, H. (2019). A Comprehensive Survey of Deep Learning for Image Captioning. ACM Computing Surveys,51(6), 1-36.
- Chen, C., Mu, S., Xiao, W., Ye, Z., Wu, L., & Qi, J. (2018). Improving Image Captioning with Conditional Generative Adversarial Nets. (4). Retrieved March 31, 2019, from https://ui.adsabs.harvard.edu/abs/2018arXiv180507112C/abstract.
- Soh, M. (2018). Learning CNN-LSTM Architectures for Image Caption Generation. Department of Computer Science, Stanford University. Retrieved April 2, 2019, from https://cs224d.stanford.edu/reports/msoh.pdf.
- https://docs.databricks.com/index.html
- https://www.solvexia.com/blog/white-paper/10-data-sourcing-best-practices-for-reporting/
- https://aclweb.org/anthology/P18-1238
- https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment
- Deep Generative Filter for Motion Deblurring https://arxiv.org/abs/1709.03481
- https://medium.com/mlreview/multi-modal-methods-image-captioning-from-translation-to-attention-895b6444256e
- https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f
- https://skymind.ai/wiki/generative-adversarial-network-gan
- https://www.analyticsvidhya.com/blog/2019/04/top-5-interesting-applications-gans-deep-learning/