NLP PROJECT REPORT

Answering Your “Deep” Questions

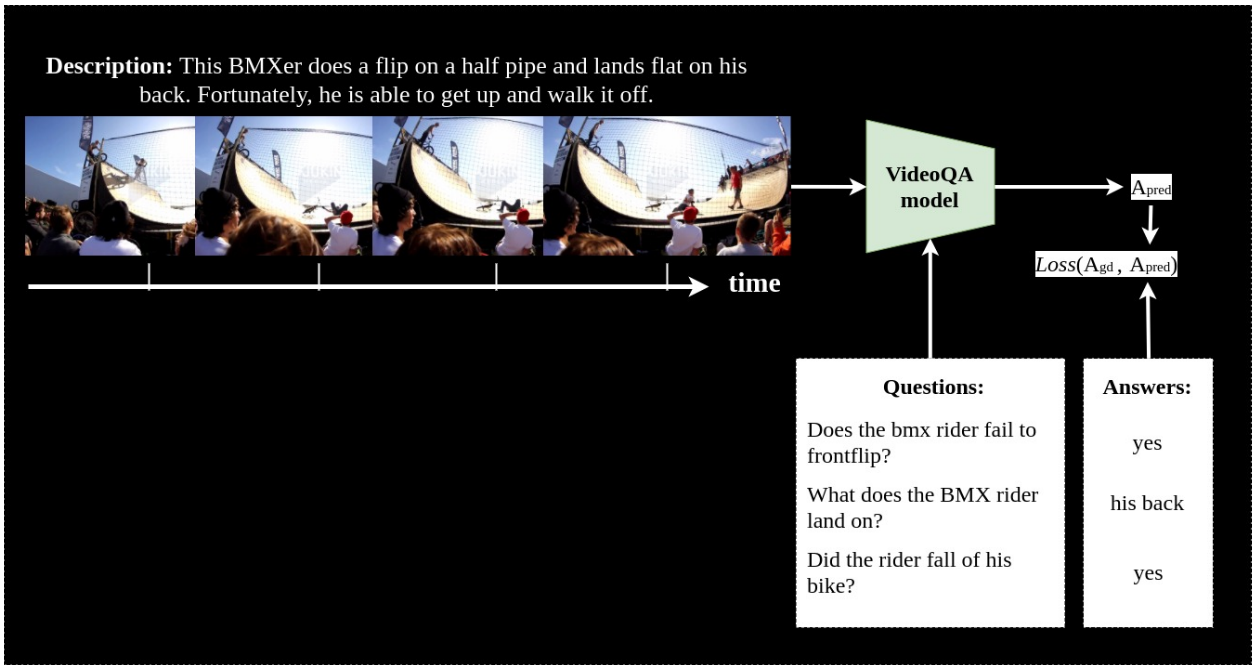
Names Removed on Purpose

Contents

1. Project Overview

The ultimate goal of most computer vision tasks is to enable a computer program to fully understand the content of an image/video as a human. However, we are still far away from achieving this goal. In order to make some progress, it is better to have a machine learning model which is able to answer any question related to an image/video. However, this task is undoubtedly challenging. In order to answer questions as a human, the model needs to not only fully understand visual observations in a much finer level but also generate an accurate answer for each question from a large answer space.

In this project, we are aiming at creating a system which can give appropriate answers to the corresponding questions after looking at a short video. In Deep Learning domain, the task we are targeting at is named as Video Question and Answering (VideoQA), and we name our system as "Answering Your 'Deep' Questions". The basic routine of VideoQA is shown as follows.


Description: This BMXer does a flip on a half pipe and lands flat on his back. Fortunately, he is able to get up and walk it off.
time
VideoQA model
A_{pred}
$Loss(A_{gd}, A_{pred})$
Questions:
Does the bmx rider fail to frontflip?
What does the BMX rider land on?
Did the rider fall of his bike?
Answers:
yes
his back
yes

Specifically, based on the above routine of VideoQA, our goal is to design the interior structure of 'VideoQA model' in the green box of the figure.

In our project, besides designing the above mentioned VideoQA model to complete the Video Question and Answering task, we need also do some necessary NLP analysis for the text part of the dataset to provide some experimental and theoretical basis for our model design. The NLP analysis in our project includes: a) Analyzing most common question type, b) Analyzing main contents of questions and c) Sentiment Analysis on the Videos.

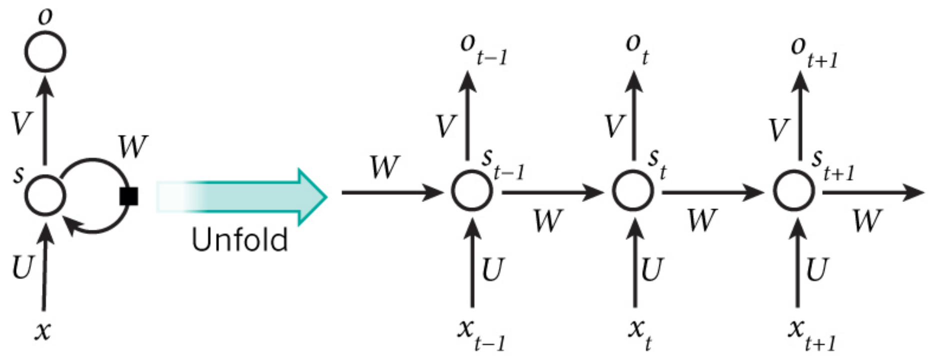
2. Video Question and Answering

Preliminaries

a. Long Short-Term Memory

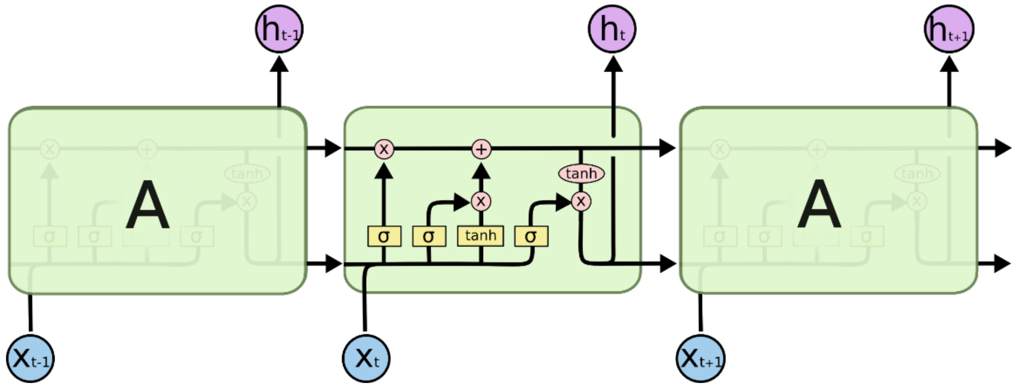
Long Short-Term Memory (LSTM) is based on its ancestor Recurrent Neural Network (RNN), whose main idea is to make use of sequential information. The 'recurrent' in RNN means that it performs the same task for every element of a sequence, with the output being depended on the previous computations, and the RNN cell has a ‘memory’ which captures information about what has been calculated so far.

In theory, the RNN can make use of information in arbitrarily long-time sequences, but in practice it is limited to looking back only a few steps. The basic structure of RNN and its unfolding version in time are shown as follows.



The reason why RNN can only do prediction for only a few steps and has difficulties learning long-range dependencies is because of the vanishing gradient problem, which means in the training process, the gradient values are shrinking exponentially fast, eventually vanishing completely after a few time steps. In order to overcome the vanishing gradient problem, the LSTM is designed based on RNN.

LSTM is explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. LSTM also has a chain like structure shown as follows.



The interior structure of a LSTM cell is shown in the above figure. There contain four neural network layers/gates in one LSTM cell, which are named as Input Gate i_t, Forget Gate f_t, Output Gate o_t and G Gate g_t. All of the four gates are determined by the current time step input x_t, and the last time step output h_{t-1}. The detailed operations inside the LSTM cell are shown as follows.

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} sigmoid \\ sigmoid \\ sigmoid \\ tanh \end{pmatrix} W^l \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \tag{1}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{2}$$

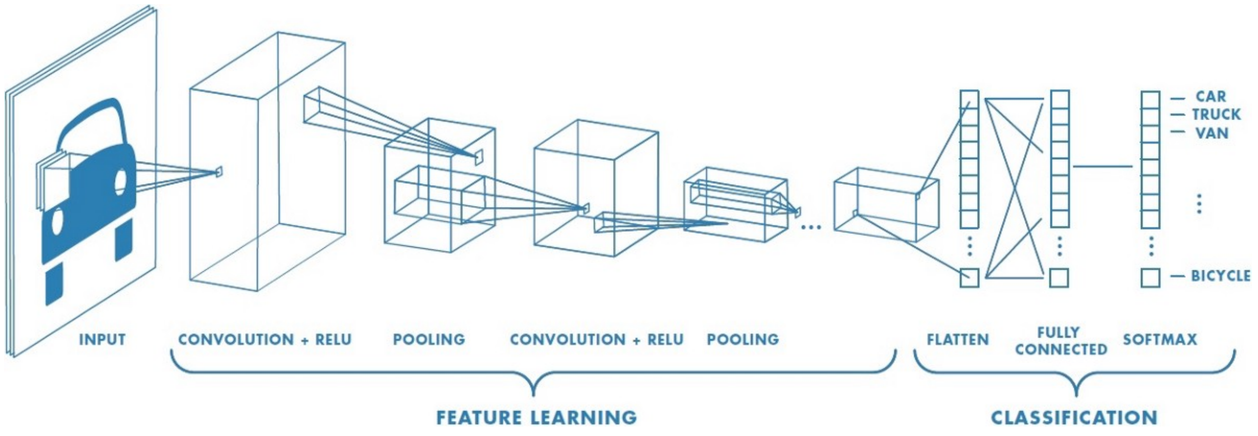
$$h_t = o_t \odot \tanh(c_t) \tag{3}$$

where W^l denotes the combination of weight matrixes for different gate, and $\odot$ denotes the element-wise product.

b. Convolutional Neural Network

Convolutional neural networks (CNN) is a class of deep, feed-forward (not recurrent) artificial neural networks that are applied to analyzing visual imagery.

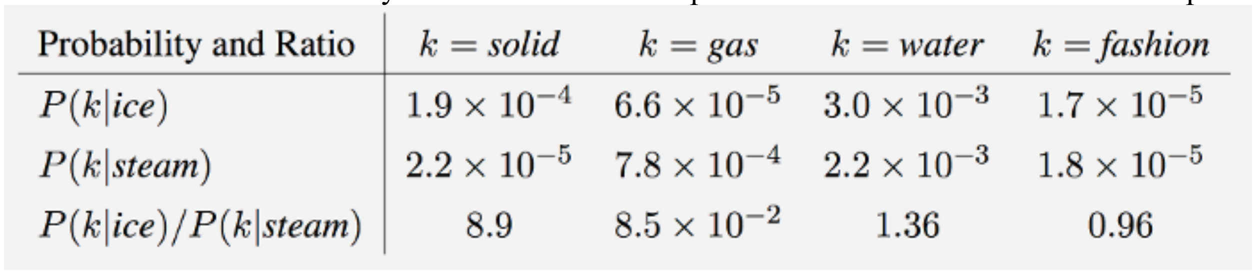
In general, Images are high-dimensional vectors so it would take a huge number of parameters to characterize the network. To address this problem, convolutional neural networks is proposed to reduce the number of parameters and adapt the network architecture specifically to vision tasks. Convolutional neural networks are usually composed by a set of layers which can be grouped by their functionalities. The basic architecture of CNN is shown as follows.



c. *Global Vectors for Word Representation*

Global Vectors for Word Representation (GloVe) is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning. For example, consider the co-occurrence probabilities for target words ice and steam with various probe words from the vocabulary. Here are some actual probabilities from a 6-billion-word corpus.

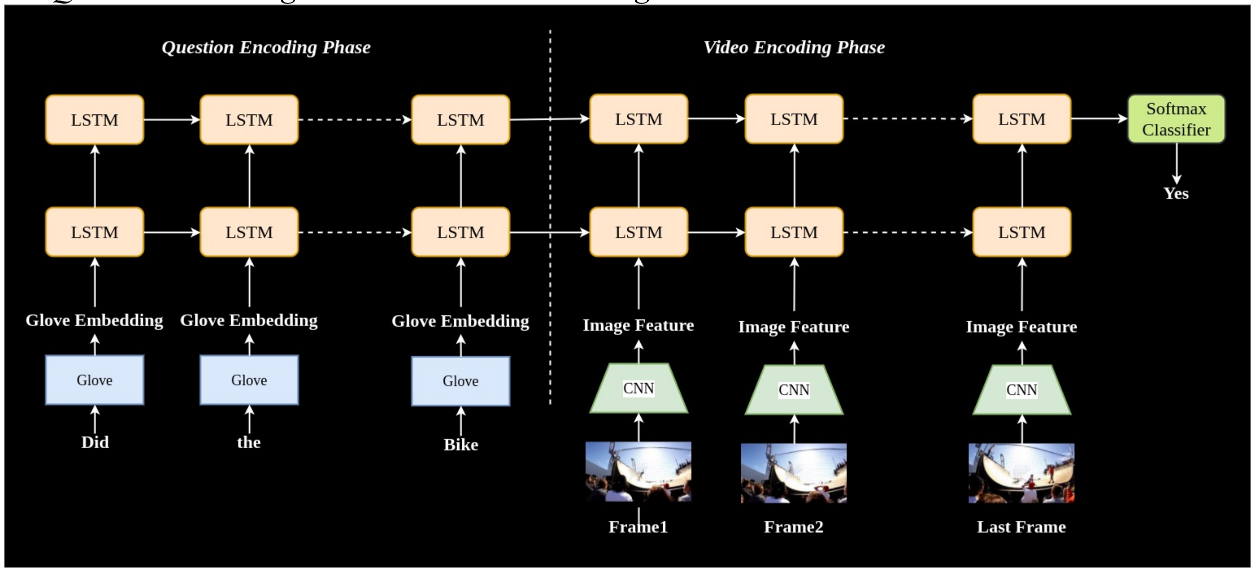


Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence.

Model

The VideoQA model designed by us is shown as follows, which including two phases, *Question Encoding Phase* and *Video Encoding Phase*.



According to the above figure, the working flow of the VideoQA model designed by us is shown as follows.

- During *Question Encoding Phase*, the GloVe embedding of each word in the question is fed into the first layer LSTM step by step until the last word 'Bike'.
- During the *Video Encoding Phase*, the image feature extracted by CNN of each frame is fed into the first layer LSTM step by step until the last frame of the video.
- After the encoding of question and video, the model will get the visual-textual representation, which will be fed into the Softmax Classifier.
- The Softmax Classifier will calculate the probability of each answer in the answer space and get the answer which has the largest probability as the predicted answer.

The basic architecture of our model is a stack of two-layer LSTM, the reason why we use the stack of two-layer LSTM rather than one layer LSTM is that the former one can better model the nonlinearity rather than the latter one.

Dataset

The dataset we use for training the designed VideoQA model is named VTW dataset, a commonly-used video dataset for language-level understanding, which is proposed by Zeng et al. in 2017.

The VTW dataset contains 14,100 videos for training, 2,000 videos for testing and 2000 videos for validation. The QA pairs for training and validation were automatically generated from user-curated descriptions using a commonly-used question generation method, while the QA pairs for testing were human-generated. Here shows some sample videos and question-answer pairs in the VTW dataset.

Automatically-generated QA pairs

Q: What struggled with its balance?
A: This kitten
Q: Did it fall off?
A: Yes
Q: Does cute kitten jump into man's hands?
A: No

Q: Who suddenly finds herself in a fight for her scarf?
A: The woman
Q: Who starts tug of war with tourists scarf?
A: Baby elephant

Human-generated QA pairs

Q: Does the guy do his backflip successfully?
A: No
Q: Where does the guy attempts back flips?
A: Beach
Q: Did the man attempt a backflip off the diving board?
A: No

Implementation and Experiments

In our experiments, we chose 146,700 QA pairs of 14,091 videos for training, and 3,600 QA pairs of 1,999 videos for testing. TensorFlow is used to implement the proposed model.

a. Video Preprocessing

In our implementation, we extracted 50 frames for each video in an evenly-paced manner, then resized and normalized each frame to zero mean and unit norm with a size of $224 \times 224 \times 3$. We extracted the visual feature of each frame using a widely- used CNN, the VGGNet pretrained on the ImageNet 2012 classification dataset and generated a 4,096-dimensional feature vector as its feature map.

b. Text Preprocessing

There are two types of text inputs: question and answer. The questions with length larger than 15 were trimmed to a maximum of 15 words, while the questions with length smaller than 15 were padded with zeros to the length of 15, and each word in the questions was represented as a 300D vector using GloVe.

For answers, we first calculated the occurrence frequency of each answer then retained the top-1000 frequent answers out of the 9,140 possible answers to create the answer space, then we represented each answer using a one-hot vector with length of 1000 based on the answer space.

Evaluation Metrics

For QA pairs whose answers are Yes/No, in order to penalize false-positive answers, we use:

$$ACC^{\dagger} = \frac{TP}{TP + FP + FN}$$

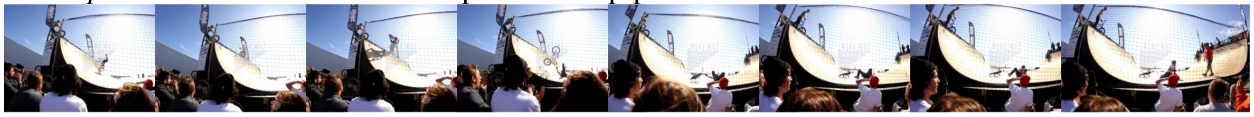
to evaluate both $Yes\ ACC^{\dagger}$ and $No\ ACC^{\dagger}$, where TP is true-positive, FP is false-positive and FN is false-negative. We also used the standard classification accuracy to evaluate the performance for answering the *Yes/No* questions.

For *Others* QA pairs whose questions are general 'what/how' kind of questions, we used both the standard classification accuracy and the widely-used metric *WUPS* to evaluate their performance. This is because the classification accuracy is too strict for the *Others* QA pairs, while *WUPS* can be considered as a relaxed version that deals with the word-level ambiguities. During the evaluation, we use the *WUPS* with threshold of 0.0 and 0.9 respectively.

Results and Analysis

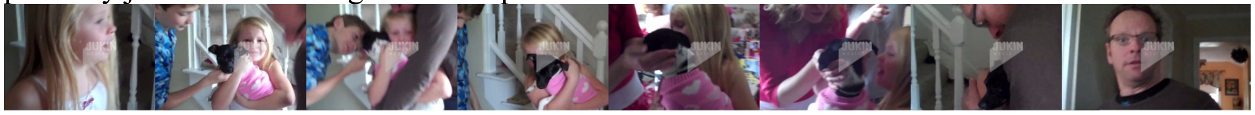
Here we show several examples from the testing results using our designed model.

Description: This BMXer does a flip on a half-pipe and lands flat on his back.



Question: Did the rider fall of his bike?
Answer: **Predicted Answer:** Yes; Ground Truth: Yes

Description: Sometimes the best present is not something an elf could make. On Christmas morning, this little girl was surprised by her dad's gift to her--a new puppy! The little dog was probably just as excited to get her as a present too.



Question: When does little girl gets puppy?
Answer: **Predicted Answer:** Christmas; Ground Truth: Christmas

Description: This little baby has the cutest face, but when his dad makes a weird noise, the little baby turns into grumpy cat!

Question: Who turns into grumpy cat?
Answer: **Predicted Answer:** baby; Ground Truth: baby

The results comparisons in terms of various accuracy-related metrics is shown as follows.

Models	Others WUPS 0.0(%)	Others WUPS 0.9(%)	Others Acc(%)	Yes Acc[†](%)	No Acc[†](%)	Yes/No Acc(%)
Skip-Thought	32.9	5.51	2.1	11.9	26.7	49.3
Extended-Soft Attention	51.4	15.5	8.4	36.4	28.8	52.4
Extended-Sequence to Sequence	48.7	14.2	7.3	34.5	25.8	49.5
Our Method	50.6	14.2	7.4	55.2	31.2	66.4

According to results shown in the above table, we can make the following observations.

- For the *Yes/No* questions, our model significantly out-performs all the other methods in terms of classification accuracies. *Extended-Soft Attention* turns out to perform the best among the three baseline methods. Our model achieves classification accuracies of 55.2%, 31.2% and 66.4% in terms of three metrics respectively, which represents 18.8%, 2.4% and 14% improvements over those of *Extended-Soft Attention*. It is worth mentioning that all the three baseline models achieve an accuracy of *Yes/No* QA pairs around 50%, which is the correct rate of random guessing; however, our model offers a better accuracy of 66.4%.

- For the *Others* questions, our model achieves similar performance with *Extended-Soft Attention* and *Extended-Sequence to Sequence* in terms of standard classification accuracy and *WUPS*.

3. Text Data Analysis

Preprocessing

The preprocessing of the data is done on the testing set of the Video Question Answering System. It contains the basic information of each visual in testing dataset records such as:

- Corresponding Question and Answer: Questions and Answers related to the Video
- Video_id: Unique ID of the Video
- Description: Description of the Video
- Title: Title of the Video

The dataset is taken from the Vision Science Lab @National Tsinghua University for the year of 2016. The data is in JavaScript Object Notation(JSON) format. The size of the dataset is 2000 JSON objects. Data is read using Pandas library in Python. The data preprocessing was done on the following attributes.

- Description
- Questions and Answers

Data cleansing and Data pre-processing comprise of following tasks:

- Reading data from the Test JSON file using Pandas and storing the data into a series object in python.
- Creating a data frame from the series object to create columns for Question & Answer, description, video_id, title and meta_id.
- At this point, each record in the data frame had multiple questions for the Q&A Column and hence our next task was to separate all the questions and create an independent list of questions.
- After performing the previous step, we got 7333 questions from 2000 records which explained that each video had around 3-4 questions on an average associated with it.
- Prior to this, we considered common type of questions which could be asked like 'What', 'Which', 'Did', 'Was' etc. and kept these categories of questions in a python list.
- After this I joined the list of questions to create a single string of questions separated by '?'.
- Now, using the sentence tokenizer, I segregated all the questions based on Question marks and Full stops to include the sentences for which question marks were missing initially.
- Data at this moment was ready for the exploratory data analysis as well as Sentimental Analysis.

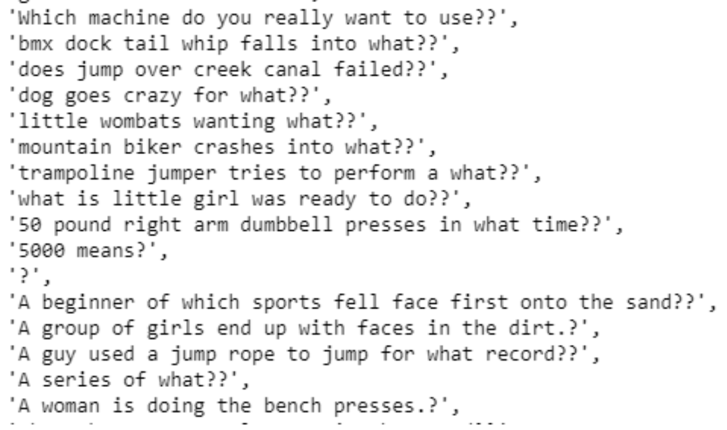
Analysis

a. *Most common questions type*

After extracting the information from the JSON file and pre-processing it, we retrieved the list of questions that were used in the training model. There were some questions which were not having question marks at the end, so the processing was done to insert the question mark at the very end of each question.

The questions were initially joined, but was tokenized using sentence tokenizer (sent_tokenize) from the NLTK library.

The list of questions thus generated are represented in the following way.

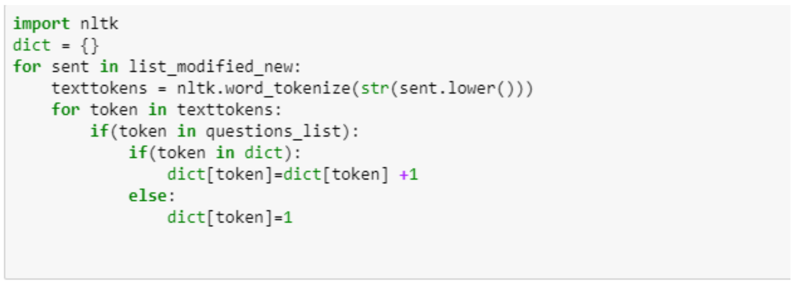


```
'Which machine do you really want to use??',
'bmx dock tail whip falls into what??',
'does jump over creek canal failed??',
'dog goes crazy for what??',
'little wombats wanting what??',
'mountain biker crashes into what??',
'trampoline jumper tries to perform a what??',
'what is little girl was ready to do??',
'50 pound right arm dumbbell presses in what time??',
'5000 means?',
'?',
'A beginner of which sports fell face first onto the sand??',
'A group of girls end up with faces in the dirt.?',
'A guy used a jump rope to jump for what record??',
'A series of what??',
'A woman is doing the bench presses.?',
```

The types of questions were mainly categorized into the following categories.

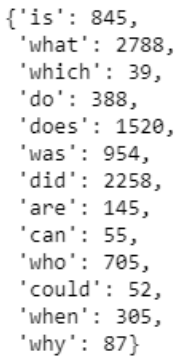
- Is?
- What?
- Which?
- Do?
- Does?
- Was?
- Did?
- Are?
- Can?
- Who?
- Could?
- When?
- Why?

To find the number of questions falling in each category, a loop was executed on each of the sentences, where each sentence was word tokenized and converted into lower case. The word was then matched with each of the words and if there is a match, the counter of that particular question is incremented by 1.



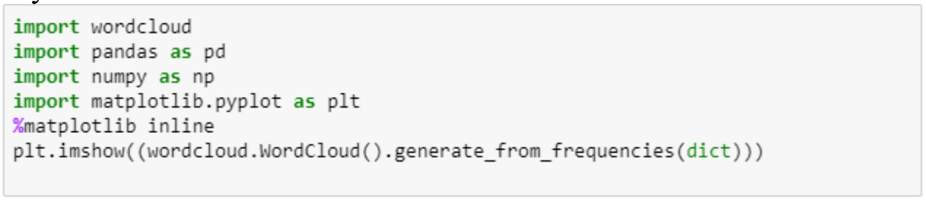
```
import nltk
dict = {}
for sent in list_modified_new:
    texttokens = nltk.word_tokenize(str(sent.lower()))
    for token in texttokens:
        if(token in questions_list):
            if(token in dict):
                dict[token]=dict[token] +1
            else:
                dict[token]=1
```

The above-mentioned function provided us with the following results.



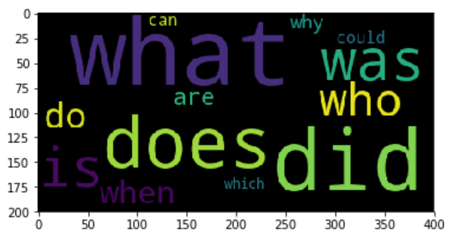
```
{'is': 845,
 'what': 2788,
 'which': 39,
 'do': 388,
 'does': 1520,
 'was': 954,
 'did': 2258,
 'are': 145,
 'can': 55,
 'who': 705,
 'could': 52,
 'when': 305,
 'why': 87}
```

This means that there are 2788 questions which focus on 'what' type of questions. For example, 'Dog Goes Crazy for What?', 'Mountain Biker crashes into what?' etc. On the basis of the frequencies mentioned above, a Word Cloud was generated using wordcloud library in Python. Below is the code for the same.



```
import wordcloud
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow((wordcloud.WordCloud().generate_from_frequencies(dict)))
```

The Word Cloud generated is as follows.

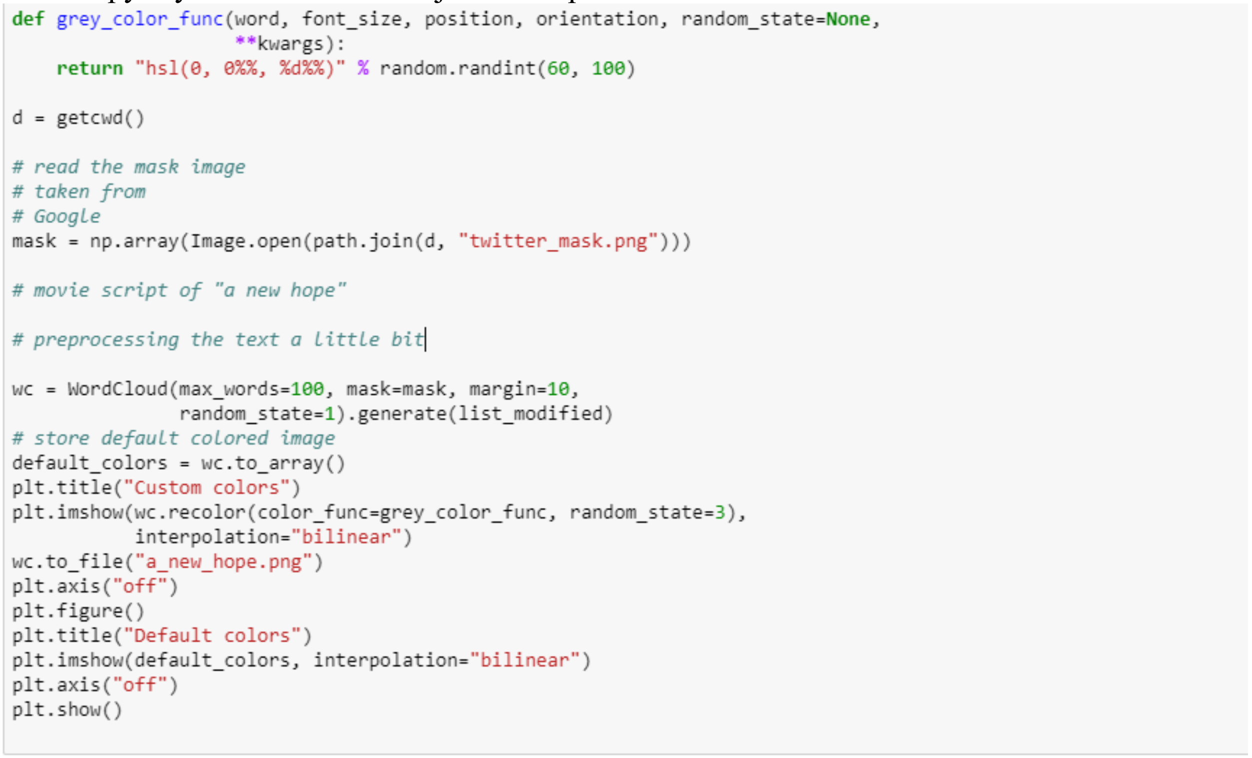


b. Analyzing the main content of questions

The motive to analyze the main content in the questions is to determine the subject in the questions. For example, in the question 'Are the babies playing basketball?', the subject is 'babies'. This determines that questions are based on these words.

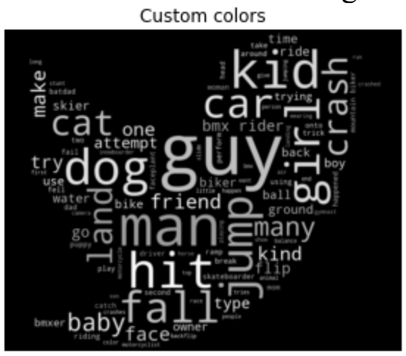
To analyze this, a word cloud was generated on the masked twitter bird image downloaded from Google search engine. Based on the frequencies of the words in the questions, top 100 words were extracted from the list and were plotted in the wordcloud.

From the wordcloud library, WordCloud and ImageColorGenerator were imported in python. We can color a word-cloud by using an image-based coloring strategy implemented in ImageColorGenerator. It uses the average color of the region occupied by the word in a source image. We can combine this with masking - pure-white will be interpreted as 'don't occupy' by the WordCloud object when passed as mask. Below is the code for the same.



```
def grey_color_func(word, font_size, position, orientation, random_state=None,
                    **kwargs):
    return "hsl(0, 0%%, %d%%)" % random.randint(60, 100)

d = getcwd()

# read the mask image
# taken from
# Google
mask = np.array(Image.open(path.join(d, "twitter_mask.png")))

# movie script of "a new hope"

# preprocessing the text a little bit

wc = WordCloud(max_words=100, mask=mask, margin=10,
               random_state=1).generate(list_modified)
# store default colored image
default_colors = wc.to_array()
plt.title("Custom colors")
plt.imshow(wc.recolor(color_func=grey_color_func, random_state=3),
           interpolation="bilinear")
wc.to_file("a_new_hope.png")
plt.axis("off")
plt.figure()
plt.title("Default colors")
plt.imshow(default_colors, interpolation="bilinear")
plt.axis("off")
plt.show()
```

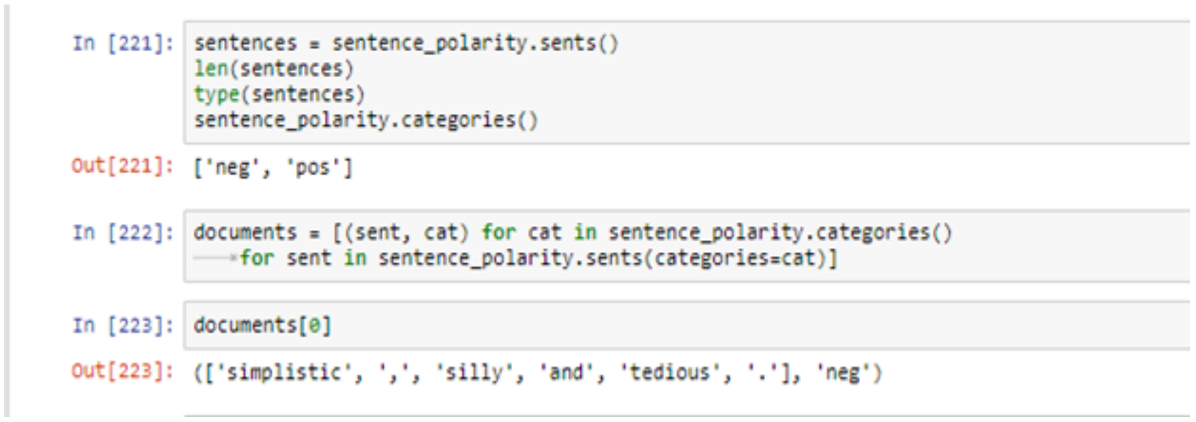
After executing the above-mentioned script, we were able to plot the most frequent 100 words in the Twitter masked image in the form of wordcloud.



The above generated wordcloud shows that most of the questions are concentrated towards words like 'guy', 'girl', 'dog' etc.

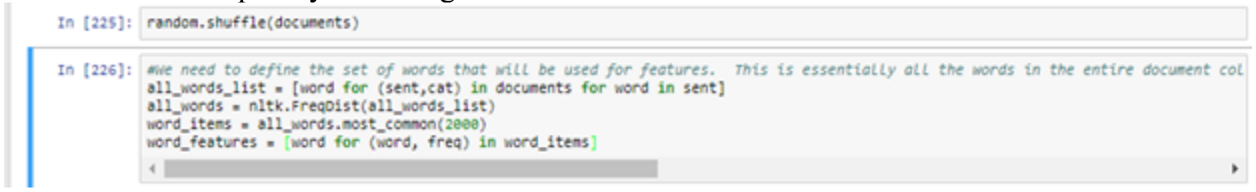
c. *Sentiment Analysis on the Videos*

Firstly, we have imported sentence polarity from nltk.corpus to get the list of positive and negative sentences to train and test the different models. The overall number of sentences are 10662. Each sentence is a document. The structure of a single document is shown as follows.



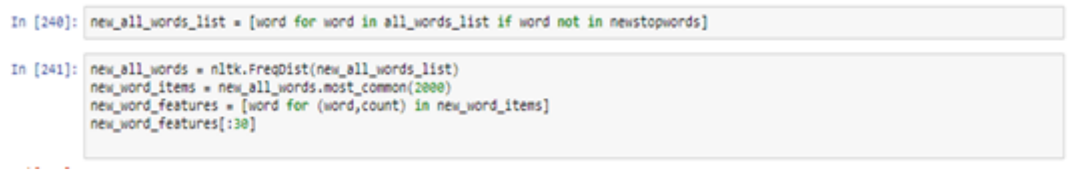
```
In [221]: sentences = sentence_polarity.sents()
          len(sentences)
          type(sentences)
          sentence_polarity.categories()

Out[221]: ['neg', 'pos']

In [222]: documents = [(sent, cat) for cat in sentence_polarity.categories()
                 for sent in sentence_polarity.sents(categories=cat)]

In [223]: documents[0]

Out[223]: (['simplistic', ',', 'silly', 'and', 'tedious', '.'], 'neg')
```

Then, we have randomly shuffled the documents. After that, we have picked up top 2000 most frequently occurring words from all the documents to train the different models.



```
In [225]: random.shuffle(documents)

In [226]: #We need to define the set of words that will be used for features.  This is essentially all the words in the entire document col
          all_words_list = [word for (sent,cat) in documents for word in sent]
          all_words = nltk.FreqDist(all_words_list)
          word_items = all_words.most_common(2000)
          word_features = [word for (word, freq) in word_items]
```

Creation of the best Model using different processing techniques have been done. The different processing techniques considered negation words and stop words to train the classifier. Through this technique, I went through the stop word list which we imported and removed the negation words from that based on the list we created earlier. Then, we took out all the stop words from the collection of all words and took the top 2000 common words as word features.

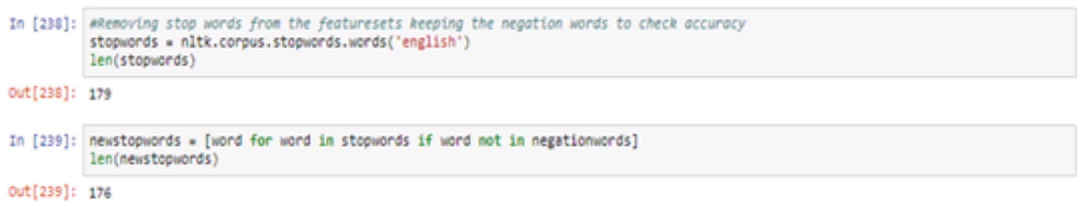


```
In [240]: new_all_words_list = [word for word in all_words_list if word not in newstopwords]

In [241]: new_all_words = nltk.FreqDist(new_all_words_list)
          new_word_items = new_all_words.most_common(2000)
          new_word_features = [word for (word,count) in new_word_items]
          new_word_features[:30]
```

We went through the document words in order adding the word features, but if the word followed a negation words, we changed the feature to negated word. The list of negation words taken is shown as follows.

- negationwords = ['no', 'not', 'never', 'none', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor', "couldn't", "wasn't", "didn't", "wouldn't", "shouldn't", "weren't", "don't", "doesn't", "haven't", "hasn't", "won't", "hadn't"]

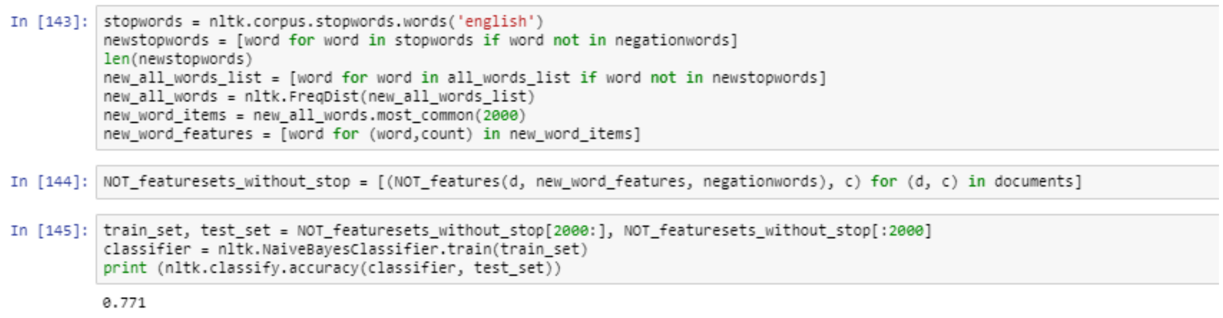
We reduced the stop words from 179 to 176 using the above-mentioned techniques.



```
In [238]: #Removing stop words from the featuresets keeping the negation words to check accuracy
          stopwords = nltk.corpus.stopwords.words('english')
          len(stopwords)

Out[238]: 179

In [239]: newstopwords = [word for word in stopwords if word not in negationwords]
          len(newstopwords)

Out[239]: 176
```

The feature label would be ‘contains (word)’ for each word in the word features set, and whenever the method created by us encounters a negation word, it adds the word to Not word feature.

After this, we have created the training and test sets, trained a Naïve Bayes classifier, and looked at the accuracy, and we have done 80/20 split of our approximately 10,000 documents.

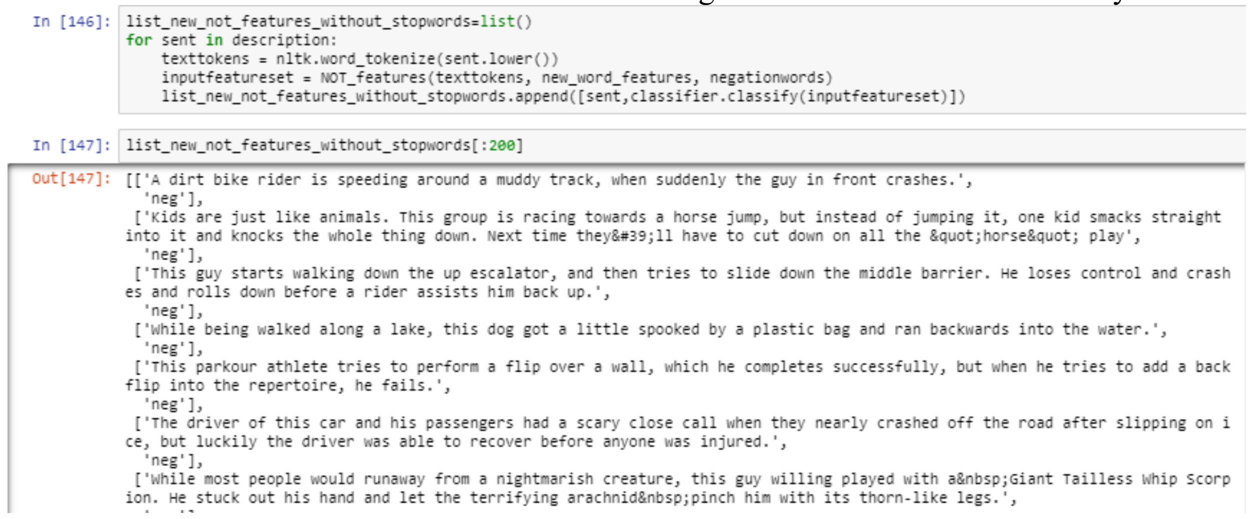


```
In [143]: stopwords = nltk.corpus.stopwords.words('english')
          newstopwords = [word for word in stopwords if word not in negationwords]
          len(newstopwords)
          new_all_words_list = [word for word in all_words_list if word not in newstopwords]
          new_all_words = nltk.FreqDist(new_all_words_list)
          new_word_items = new_all_words.most_common(2000)
          new_word_features = [word for (word,count) in new_word_items]

In [144]: NOT_featuresets_without_stop = [(NOT_features(d, new_word_features, negationwords), c) for (d, c) in documents]

In [145]: train_set, test_set = NOT_featuresets_without_stop[2000:], NOT_featuresets_without_stop[:2000]
          classifier = nltk.NaiveBayesClassifier.train(train_set)
          print (nltk.classify.accuracy(classifier, test_set))

          0.771
```

The accuracy achieved here is 77.1% and is better than the accuracy of any other models considered by us in previous labs.

After this, we used the best Model to classify the description of the videos as ‘positive’ or ‘negative’ using word tokenizer and the Naïve Bayes Classifier trained on the best Model.

It was observed that out of 2000 descriptions, 412 were positive and 1588 were negative which shows that most of the videos were about negative scenarios which is actually true.



```
In [146]: list_new_not_features_without_stopwords=list()
          for sent in description:
              texttokens = nltk.word_tokenize(sent.lower())
              inputfeatureset = NOT_features(texttokens, new_word_features, negationwords)
              list_new_not_features_without_stopwords.append([sent,classifier.classify(inputfeatureset)])

In [147]: list_new_not_features_without_stopwords[:200]

Out[147]: [['A dirt bike rider is speeding around a muddy track, when suddenly the guy in front crashes.',
            'neg'],
           ['Kids are just like animals. This group is racing towards a horse jump, but instead of jumping it, one kid smacks straight
          into it and knocks the whole thing down. Next time they&#39;ll have to cut down on all the &quot;horse&quot; play',
            'neg'],
           ['This guy starts walking down the up escalator, and then tries to slide down the middle barrier. He loses control and crash
          es and rolls down before a rider assists him back up.',
            'neg'],
           ['While being walked along a lake, this dog got a little spooked by a plastic bag and ran backwards into the water.',
            'neg'],
           ['This parkour athlete tries to perform a flip over a wall, which he completes successfully, but when he tries to add a back
          flip into the repertoire, he fails.',
            'neg'],
           ['The driver of this car and his passengers had a scary close call when they nearly crashed off the road after slipping on i
          ce, but luckily the driver was able to recover before anyone was injured.',
            'neg'],
           ['While most people would runaway from a nightmarish creature, this guy willing played with a&nbsp;Giant Tailless Whip Scorp
          ion. He stuck out his hand and let the terrifying arachnid&nbsp;pinch him with its thorn-like legs.',
```

4. Conclusions and Suggestions

In order to make computers understand video contents at human-level, we created a VideoQA system which can give appropriate answers to the corresponding questions after looking at a short video. The experimental results show that our model significantly outperforms all the other methods for *Yes/No* QA pairs, while achieves similar performance compared with two baseline methods for *Others* QA pairs.

Additionally, some necessary NLP analysis for the text part of the dataset has been done to provide experimental and theoretical basis for our model design.

To further improve the performance of VideoQA model, we could use some engineering techniques of Deep Learning such as the attention mechanism, or try some other feature extraction models instead of VGGNet, such as GoogleNet or ResNet. Considering the objective of the VideoQA is video, which contains motions besides appearances, we can also extract C3D features of each video using 3D-CNN, and feed such motion features into the model.

5. References

[1] Zeng K H, Chen T H, Chuang C Y, et al. Leveraging Video Descriptions to Learn Video Question Answering[C]. AAAI. 2017: 4334-4340.
[2] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.
[3] Pennington J, Socher R, Manning C. Glove: Global vectors for word representation[C]. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014: 1532-1543.
[4] Malinowski M, Rohrbach M, Fritz M. Ask your neurons: A neural-based approach to answering questions about images[C]. Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV). IEEE Computer Society, 2015: 1-9.
[5] Venugopalan S, Rohrbach M, Donahue J, et al. Sequence to sequence-video to text[C]. Proceedings of the IEEE international conference on computer vision. 2015: 4534-4542.
[6] Yao L, Torabi A, Cho K, et al. Describing videos by exploiting temporal structure[C]. Proceedings of the IEEE international conference on computer vision. 2015: 4507-4515.