Specification Document - Online Class Note Maker

Problem Description:

The COVID-19 has resulted in schools and colleges shut all across the world. As a result, education has changed dramatically, with the distinctive rise of e-learning, whereby teaching is undertaken remotely and on digital platforms.

These online classes have helped students during lockdown more than ever before. They get to learn new things and increase their knowledge day by day.

But taking notes in online classes can be a big challenge for students with learning and thinking differences. They may struggle with writing and organizing their notes while listening. Or they may have trouble keeping up with a teacher because of slow processing speed .

So here comes an Online class note maker which takes notes on students behalf and saves their time so that they can focus entirely on classes.

Solution:

The solution consists of generating summarized transcripts of the slides used by the professors.

1.Generating text

a. From Image

As our objective is to generate a summarized transcript so firstly we are required to obtain text embedded in images. For that purpose we are using **Pytesseract** or **Python-tesseract** which is an Optical Character Recognition (OCR) tool for python. It will read and recognize the text in images.

Mainly, 3 simple steps are involved here as shown below:-

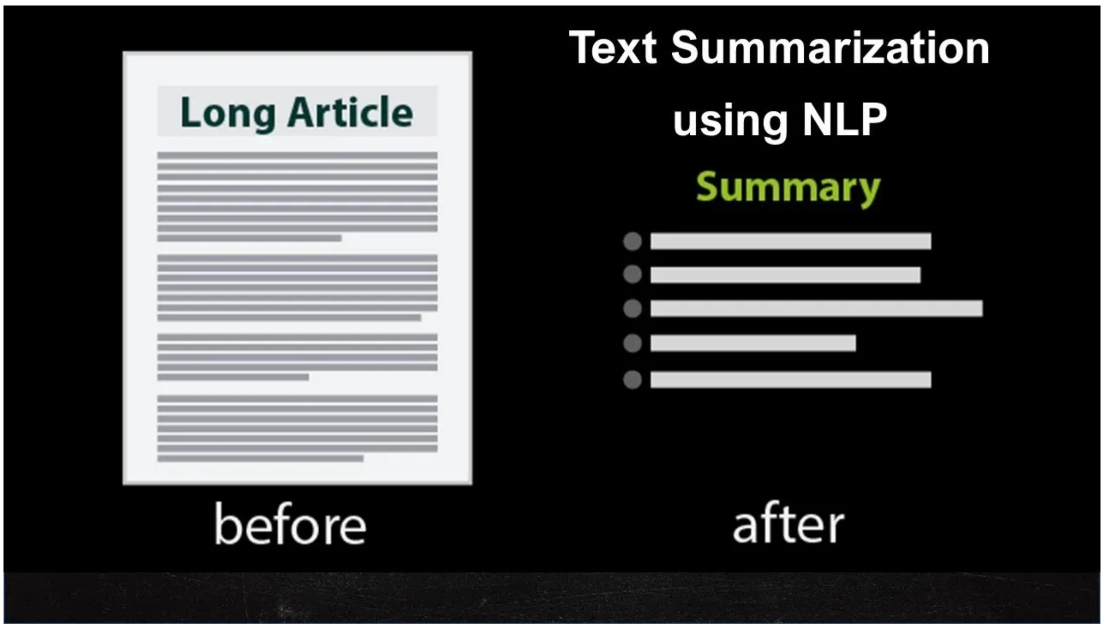
- Loading an Image saved from the computer . (Any Image with Text).
- Binarizing the Image (Converting Image to Binary).
- The image is passed through the OCR system.

After performing the above steps we get the text in the image in the form of a transcript which can be easily stored in a text file. But this method isn't very efficient in certain scenarios so we have directly extracted the text from the pdf provided by the professors.

b. From PDF

Another smart way of generating text is extracting it directly from pdf using an appropriate python library. So for this purpose, we have used the **pdfplumber** library of python.This library can serve different purposes while extracting text. If we want to extract text or tabular data from any document, this library can be much handy. It can access the files in PDF and has high rendering quality. So what this library does is that it takes the converted ppt-to-pdf file as an input. And then it goes through the pdf pages one by one and extracts all the text and returns it in the form of a string. Later this text can be summarized using the summarizer.

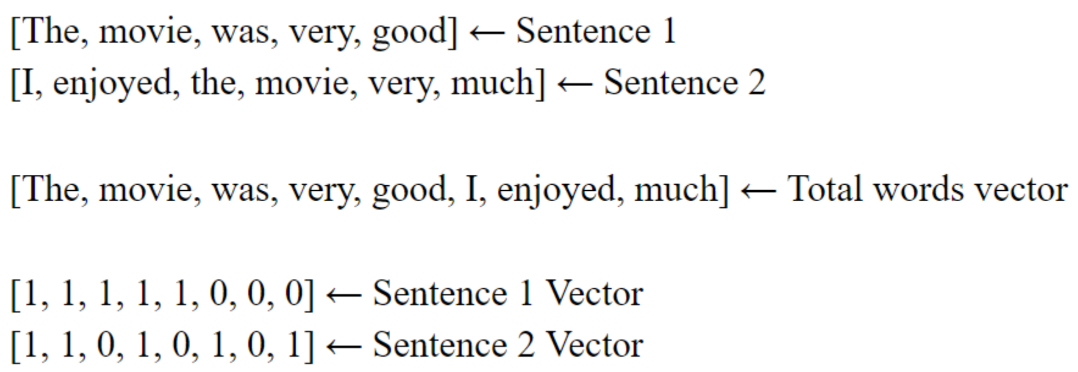
2.Summarization



The transcription generated from pdf needs to be summarized and get down to something concise. Here ***extractive text summarization*** is used, which can take the most important points and sentences directly from the transcription.The first step is to preprocess the text, making sure the algorithm can deal with it. This involves getting rid of the punctuation and capitalization, along with any stop words that don't help us with the meaning of the text. To do this the following steps are needed to be followed-

1. The first step is to tokenize the sentences because the text as a whole can't be processed . In this step we will reduce down the text into sentences. This can be performed using **sent_tokenize** from **nltk.tokenize.**

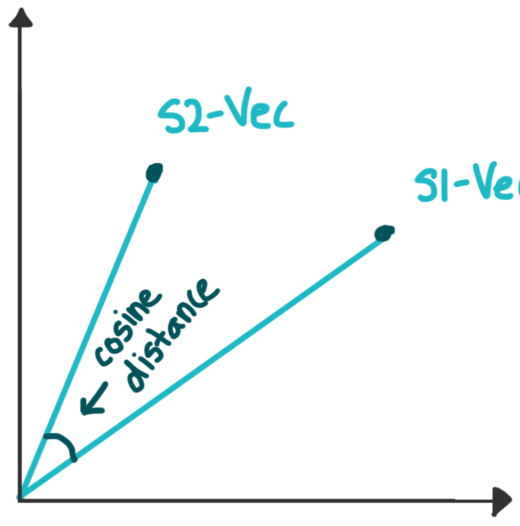
2. Punctuations (*the marks, such as full stop, comma, and brackets, used in writing to separate sentences and their elements and to clarify meaning*) in text are unnecessary while implementing text summarization and hence need to be removed. Punctuations can be removed by using **re.sub() in python.** In this, we replace all punctuation by empty string using a certain regex.
3. Stop words (commonly used words of a language − is, am, the, of, in, etc) are also unnecessary for implementing summarization and hence can easily be removed by using **stopwords** from **nltk.corpus.**
4. The next step includes the stemming of the words. This converts the words to their root words (eg.following->follow, running->run). It is an important step because if not performed then ‘ing’ form of a word and the root word will be considered as two different words .This task can be performed by using **PorterStemmer** from **nltk.stem**.
5. After preprocessing, vectors of sentences are created. Vectors for the constituent words in a sentence are fetched and then the mean/average of those vectors is taken to arrive at a consolidated vector for the sentence.


[The, movie, was, very, good] ← Sentence 1
[I, enjoyed, the, movie, very, much] ← Sentence 2

[The, movie, was, very, good, I, enjoyed, much] ← Total words vector

[1, 1, 1, 1, 1, 0, 0, 0] ← Sentence 1 Vector
[1, 1, 0, 1, 0, 1, 0, 1] ← Sentence 2 Vector

6. The next step is to find similarities between the sentences, so a similarity matrix is prepared. Here we have used Cosine Similarity to compute the similarity between a pair of sentences. Cosine similarity calculates the angle between the vectors, and the lower it is, the more similar the sentences are.



7. Before proceeding further, the similarity matrix is converted into a graph. The nodes of this graph will represent the sentences and the edges will represent the similarity scores between the sentences. On this graph, page rank summarization can be used to generate summary. So here we have used the **networkx** library of python to apply **text rank algorithm**. The task of converting similarity matrix to graph and applying **text rank algorithm**, both can be performed by **networkx** library. Text rank algorithm sorts the sentences in descending order of their cosine similarity score.

8. Finally, the top N sentences based on their rankings are extracted for summary generation. And we get a neat, concise and useful summary for our articles.

Alternate method for Summarisation:

Instead of applying the above method which includes a number of steps we can use **sumy** library. Sumy library provides several algorithms to implement Text summarization. So we need to import the desired algorithm rather than having to code it on our own.

Following algorithms are provided by Sumy library for summarization:

1. LexRank
2. Luhn
3. Latent Semantic Analysis, LSA
4. KL-Sum

Out of all the four algorithms, **Latent Semantic Analysis (LSA)** provides the best result. Latent Semantic Analysis is an unsupervised learning algorithm that can be used for extractive text summarization. It extracts semantically (on the basis of meaning of sentences) significant sentences by applying singular value decomposition (SVD) to the matrix of term-document frequency.

Image Detection

Images are also an important part of teaching as they depict information in a simple understandable way. So they need to be detected and then captioned as well to easily understand the concepts in a quick way.

So for this purpose, we have used the **PyMuPDF** (aka **“fitz”**) library of python, which is a lightweight PDF viewer. This library can access the files in PDF, XPS, etc and has high performance and high rendering quality. So what this library does is that it takes the converted ppt-to-pdf file as an input. And then it goes through the pdf pages one by one and extracts all the images in those pages in a location defined by us. Later these images can be captioned and we can get the context of the images.

Image Captioning

Image captioning, as clear from the name itself, is used to get the context of a particular image. In this project, it is really necessary as it provides information about all those images which are present in lecture slides or other literary means of teaching. So after summarising the text present in the slides and image detection, these images are captioned to get a better overview of the context of teaching. And then these captioned images are combined with the summarised text to produce the final result. Now in our project, deep learning concepts have been used for this purpose.

There are many open source datasets available for this problem, like Flickr 8k (containing 8k images), Flickr 30k (containing 30k images), MS COCO (containing around 200k images), etc. But for the purpose of this project, Flickr

8k dataset is used. Here’s the link to it. We have used this dataset because it gives good results for image captions despite containing less number of images. So, it is easier to train a model with this dataset. Training a model with large number of images (like coco dataset) may not be feasible on a system which is not a very high end PC/Laptop. Flickr 8k dataset contains 8000 images each with 5 captions. These images are divided as follows:

- Training Set — 6000 images
- Dev Set — 1000 images
- Test Set — 1000 images

Now, after finalising the dataset, the following steps are implemented for the purpose of image captioning:-

1. A dictionary named “descriptions” is created which contains the name of the image (without the .jpg extension) as keys and a list of the 5 captions for the corresponding image as values.
2. Then some basic cleaning is performed like lower-casing all the words (otherwise“hello” and “Hello” will be regarded as two separate words), removing special tokens (like ‘%’, ‘#’, etc.), eliminating words which contain numbers (like ‘hey199’, etc.). Along with this a vocabulary is also created which contains all the unique words present across all the 8000*5 (i.e. 40000) image captions (**corpus**) in the data set. The number of these unique words is found to be 8763 for this particular

dataset. This number is different for different datasets. This number comes down to 1652 as the words which have occurred less than 10 times in all of the image captions are removed.

3. In the dataset, there is a text file which contains the names of the images that belong to the training set. So these names are loaded into a list “train”. Also the descriptions of these 6000 images is loaded in a new python dictionary but with two tokens namely - ‘startseq’ and ‘endseq’ added at the start and end of every caption respectively.
4. After this, all the data is preprocessed in two steps. First, Images and then captions. So, every image is converted into a fixed sized vector which can then be fed as input to the neural network. For this purpose, **transfer learning** has been opted by using the InceptionV3 model (Convolutional Neural Network) created by Google Research. This model performs image classification on 1000 different classes of images. But our purpose here is not to classify the image but just get fixed-length informative vector for each image. Hence, the last softmax layer from the model (which performs classification of images) is removed and a 2048 length vector (bottleneck features) for every image is extracted and then saved on the disk using pickle file whose keys are image names and values are corresponding 2048 length feature vector. Similarly, test images are also preprocessed.

5. Now, captions are preprocessed. The prediction of the entire caption, given the image, does not happen at once. So the caption is predicted **word by word**. Thus, it is needed to encode each word into a fixed sized vector. First two Python Dictionaries namely “wordtoix” (word to index) and “ixtoword” (index to word) are created. Stating simply, every unique word in the vocabulary is represented by an integer (index). As seen above, we have 1652 unique words in the corpus and thus each word is represented by an integer index between 1 to 1652. These two Python dictionaries can be used as follows:

   wordtoix[‘abc’] -> returns index of the word ‘abc’

   ixtoword[k] -> returns the word whose index is ‘k’

   Also the maximum length of cation is calculated which is 34 in this particular case. This will be used in later steps.

6. Then the data is prepared in a manner which will be convenient to be given as input to the deep learning model. Image vector is the input and the caption is needed. But the way it is predicted is as follows:

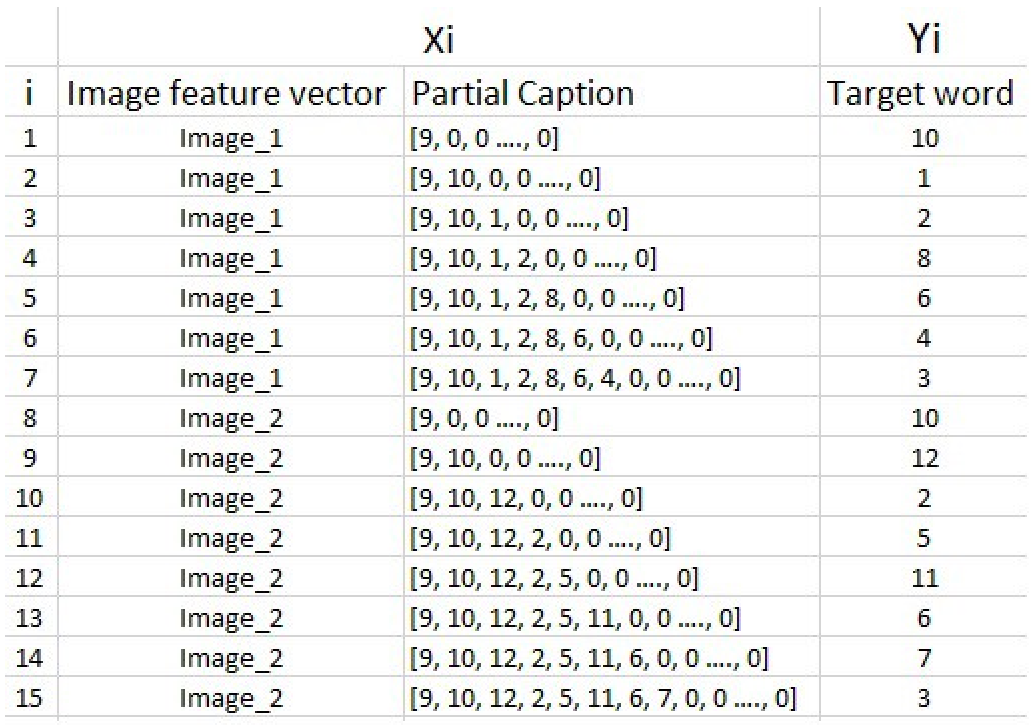
For example, take “startseq the black cat sat on grass endseq” as a caption for some image. For the first time, the image vector and the first word as input are provided and then the second word is predicted, i.e.:

Input = Image_1 + ‘startseq’; Output = ‘the’

Then image vector and the first two words as input are provided and the third word is predicted, i.e.:

Input = Image_1 + ‘startseq the’; Output = ‘cat’

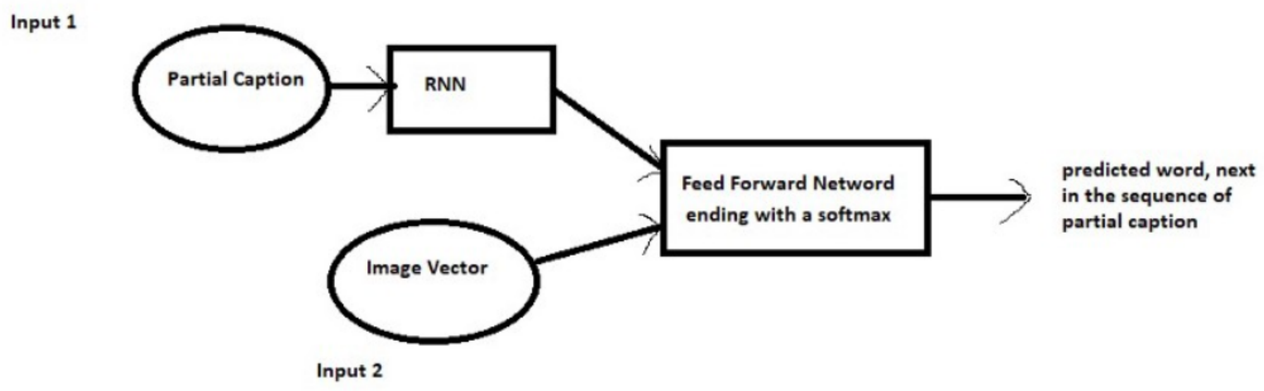
Now, Since **sequences** are being processed, a **Recurrent Neural Network** is employed to read these partial captions. But as shown above, actual words from captions are not passed, rather the indices corresponding to those words are passed in the model. Since **batch processing** is required, it is a need to make sure that each sequence is of **equal length**. Hence zero padding is opted at the end of each sequence. As the maximum length of a caption, which is 34, those many number of zeros which will lead to every sequence having a length of 34 are padded at the end of sequence. After this, a **generator functio**n in Python is used as main memory of more than around 3gb is required for this whole process, which is a pretty huge requirement and will make the system slow. Generator function is like an iterator which resumes the functionality from the point it left the last time it was called. So it does not need to store the entire dataset in the memory at once. Even if we have the current batch of points in the memory, it is sufficient for our purpose. The data matrix will look something like this after this step:- Here the indices: 9 signifies “startseq” and 3 signifies “endseq” which is the end of the caption.



	Xi		Yi
i	Image feature vector	Partial Caption	Target word
1	Image_1	[9, 0, 0 ...., 0]	10
2	Image_1	[9, 10, 0, 0 ...., 0]	1
3	Image_1	[9, 10, 1, 0, 0 ...., 0]	2
4	Image_1	[9, 10, 1, 2, 0, 0 ...., 0]	8
5	Image_1	[9, 10, 1, 2, 8, 0, 0 ...., 0]	6
6	Image_1	[9, 10, 1, 2, 8, 6, 0, 0 ...., 0]	4
7	Image_1	[9, 10, 1, 2, 8, 6, 4, 0, 0 ...., 0]	3
8	Image_2	[9, 0, 0 ...., 0]	10
9	Image_2	[9, 10, 0, 0 ...., 0]	12
10	Image_2	[9, 10, 12, 0, 0 ...., 0]	2
11	Image_2	[9, 10, 12, 2, 0, 0 ...., 0]	5
12	Image_2	[9, 10, 12, 2, 5, 0, 0 ...., 0]	11
13	Image_2	[9, 10, 12, 2, 5, 11, 0, 0 ...., 0]	6
14	Image_2	[9, 10, 12, 2, 5, 11, 6, 0, 0 ...., 0]	7
15	Image_2	[9, 10, 12, 2, 5, 11, 6, 7, 0, 0 ...., 0]	3

7. Actually the words or indices in the above step are mapped (embedded) to higher dimensional space through one of the word embedding techniques. So in this project, a pre-trained GLOVE Model is used to embed these indices to a 200-long vector. After this, an embedding matrix is created which is loaded into the model before training.

8. Here is the brief architecture which contains the high level sub-modules:


Input 1
Partial Caption
RNN
Feed Forward Netword ending with a softmax
predicted word, next in the sequence of partial caption
Image Vector
Input 2

High level architecture

9. Then, the model is accordingly defined and is compiled using the adam optimizer. Finally the weights of the model are updated through backpropagation algorithm and the model learns to output a word, given an image feature vector and a partial caption. So in summary:

   Input_1 -> Partial Caption

   Input_2 -> Image feature vector

   Output -> An appropriate word, next in the sequence of partial caption provided in the input_1 (or in probability terms it is said **conditioned** on image vector and the partial caption).

   The model is then trained for 30 epochs with the initial learning rate of 0.001 and 3 pictures per batch (batch size). However after 20 epochs, the learning rate is being reduced to 0.0001 and the model is trained on 6 pictures per batch. This is because the model is moving towards convergence, so we must lower the learning rate so that we take smaller steps towards the minima.

10. Now, this step concerns the inference of the model. There are many iterations in this final step of getting the caption from the model which has been provided with an input image. In the first iteration, the image vector and the ‘startseq’ as partial caption is provided as an input to the model. Then the word with the maximum probability is selected, given the feature vector and the partial caption. This is called as **Maximum Likelihood Estimation** (MLE) or **Greedy Search**. Then next iteration starts and it keeps going on until either of the below two conditions are met:-
    A. The model encounters an ‘**endseq**’ token which means this is the end of the caption.
    B. The model reaches a maximum **threshold** of the number of words generated by it.

Once the loop breaks, the generated caption is reported as the output of the model for the given image. So this model, without any rigorous hyper-parameter tuning does a decent job in generating captions for images.

There are several modifications which can be made to improve this solution like:

- Using a **larger** dataset like coco dataset.

- Doing more **hyper parameter tuning** (learning rate, batch size, number of layers, etc.).
- Using **Beam Search** instead of Greedy Search during Inference.

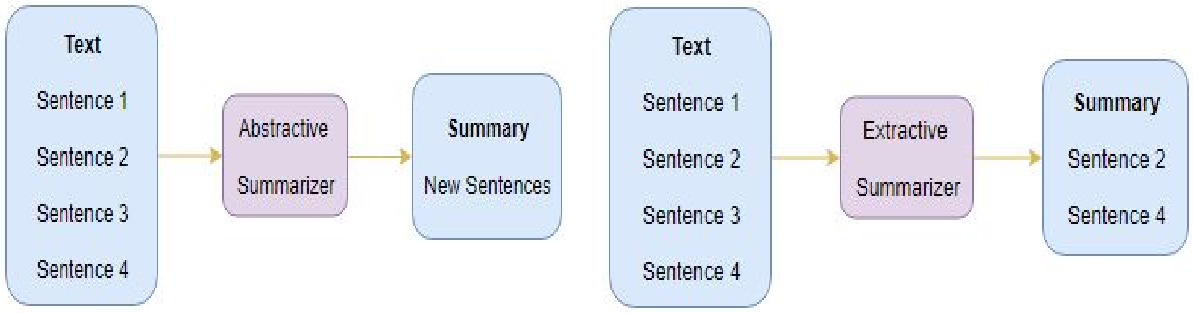
Future Goals

1. Making a GUI.

To make our project **user-friendly**. We wish to make a computer or a web application that would keep running in background while the lecture is going on and take multiple snapshots from the lecture as input and give us summarized text with captioned images as an output.

2. Use of Abstractive summarization in place of Extractive Summarization.

We have used Extractive summarization in our project in which relevant sentences in the text document are reproduced as a summary. **Extractive summarization** means identifying important sections of the text and generating a subset of the sentences from the original text; while **Abstractive summarization** reproduces important material in a new way after interpretation and examination of the text using advanced natural language techniques to generate a new shorter text that conveys the most critical information from the original one. We have planned to use Abstractive summarization, which is more advanced and interprets text documents more accurately.



3. Adding Speech to Text Conversion, then Summarization.

Our project only used the information present in the image, but the instructor or professor verbally delivers a large part of learning. Image or Slides (assuming as an input image) are generally summarized versions of what the instructor speaks, So it would not be helpful to summarize an already summarized version. So we wish to add a utility in which we will be taking speech input of a video altogether with text and then summarizing both. The proposed system performs a **Speech-to-Text summarization** as well as **Image-to-text summarization** (here image is referring to snapshots of the lecture slides) and generates much more informative summaries which can then be used as notes for that particular lecture.