Video Captioning Web Application With Ability To Read Out The Video Captions

A project report submitted in partial fulfilment of the requirements for the degree of

Bachelor of Engineering
in
Computer Engineering

by

Dion Trevor Castellino (Roll No. 8592)
Joshua Godinho (Roll No. 8607)
Umang Kavedia (Roll No. 8611)

Under the guidance of

Prof. Merly Thomas

DEPARTMENT OF COMPUTER ENGINEERING

Fr. Conceicao Rodrigues College of Engineering, Bandra (W),Mumbai – 400050

University of Mumbai2021-22

This work is dedicated to my family.

I am very thankful for their motivation and support.

Internal Approval Sheet

CERTIFICATE

This is to certify that the project entitled "**Video Captioning Web Application With Ability To Read Out The Video Captions**"is a bonafide work of **Dion Trevor Castellino (8592), Joshua Godinho (8607)** and **Umang Kavedia (8611)** submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the degree of **Bachelor in Computer Engineering.**

(Name and Sign)
Supervisor/Guide

(Name and Sign)
Head of Department

(Name and Sign)
Principal

Approval Sheet

Project Report Approval

This project report entitled by **Video Captioning Web Application With Ability To Read Out The Video Captions** by **Dion Trevor Castellino (8592), Joshua Godinho (8607)** and **Umang Kavedia (8611)** is approved for the degree of Bachelor of Engineering in Computer Engineering.

Examiner 1.————————

Examiner 2.———————--

Date:

Place:

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Instituteand can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Dion Trevor Castellino (Roll No. 8592) (sign)_____________

Joshua Godinho (Roll No. 8607) (sign)_____________

Umang Kavedia (Roll No. 8611) (sign)_____________

Date:

Place:

Abstract

Real-world videos often have complex dynamics; and methods for generating open-domain video descriptions should be sensitive to temporal structure and allow both input (sequence of frames) and output (sequence of words) of variable length. Additionally, access to the descriptions of such videos is quite limited. To approach this problem, we employ an end-to-end sequence-to-sequence model to generate captions for videos. For this we exploit recurrent neural networks, specifically LSTMs, which have demonstrated state-of-the-art performance in image caption generation. After being trained on video-sentence pairs and learning to associate a sequence of video frames to a sequence of words in order to generate a description of the event in the video clip our model naturally is able to learn the temporal structure of the sequence of frames as well as the sequence model of the generated sentences, i.e. a language model. Further, we put the algorithm to good use by combining it with a text to speech module and deploying it as a web application.

Acknowledgments

We have great pleasure in presenting the report on "**Video Captioning Web Application With Ability To Read Out The Video Captions**". I take this opportunity to express my sincere thanks towards the guide **Prof. Merly Thomas**, C.R.C.E, Bandra (W), Mumbai, for providing the technical guidelines, and the suggestions regarding the line of this work. We enjoyed discussing the work progress withhim during our visits to department.

We thank Prof. Dr. Brijmohan S. Daga, Head of Computer Department, Principal and the management of C.R.C.E., Mumbai for encouragement and providing necessary infrastructure for pursuing the project.

We also thank all non-teaching staff for their valuable support, to complete our project.

Dion Trevor Castellino (Roll No. 8592)

Joshua Godinho (Roll No. 8607)

Umang Kavedia (Roll No. 8611)

Date:

Table of Contents

List of Figures

Chapter 1

Introduction

Current video sharing platforms contain a large amount of video material. The ability to generate descriptions of this content would be highly valuable for many tasks, such as content-based retrieval or recommendation. Moreover, they would enable visually-impaired people to consume video material and improve their quality of life. This kind of video descriptions are usually provided as natural language sentences or captions in a compact and intuitive format which most importantly, can be digested by humans.

The task is usually formulated as a sequence-to-sequence (video to caption) task. Therefore, the progress in the field is significantly influenced by advances in machine translation. Hence, many models rely on an encoder-decoder architecture which consists of two recurrent neural networks (RNNs).

Considering the natural co-occurrence of videos in every aspect of our day-to-day use of technology, recent advances in deep learning have successfully achieved video captioning. Yet, most of the existing works of video captioning require very high computing power or niche resources like Google Colaboratory. In this work, we address this issue by introducing a web application that employs a sequence-to-sequence captioning model combined with a text-to-speech module which 'speaks' the captions in a legible manner. Our captioning module is inspired by the S2VT architecture while the web application is deployed with the help of flask.

Chapter 2

Literature Review

In Literature review, various references of the existing projects are taken into considerations which are similar to our current project**.**

Tagging-Clustering Approach

Early work on video captioning considered tagging videos with metadata [1] and clustering captions and videos [2] for retrieval tasks. Several previous methods for generating sentence descriptions [3] used a two-stage pipeline that first identifies the semantic content and then generates a sentence based on a template. This typically involved training individual classifiers to identify candidate objects, actions and scenes. They then use a probabilistic graphical model to combine the visual confidences with a language model in order to estimate the most likely content in the video, which is then used to generate a sentence. While this simplified the problem by detaching content generation and surface realization, it requires selecting a set of relevant objects and actions to recognize. Moreover, a template-based approach to sentence generation is insufficient to model the richness of language used in human descriptions. In contrast, our approach avoids the separation of content identification and sentence generation by learning to directly map videos to full human-provided sentences, learning a language model simultaneously conditioned on visual features.

Our model builds on the image caption generation model in [4]. The first step is to generate a fixed length vector representation of an image by extracting features from a CNN. The next step learns to decode this vector into a sequence of words composing the description of the image. While any RNN can be used in principle to decode the sequence, the resulting long-term dependencies can lead to inferior performance. To mitigate this issue, LSTM models have been exploited as sequence decoders, as they are more suited to learning long-range dependencies. In addition, since we are using variable-length video as input, we use LSTMs as sequence-to-sequence transducers, following the language translation models of [5].

LSTM Approach

In [6], LSTMs are used to generate video descriptions by pooling the representations of individual frames. Their technique extracts CNN features for frames in the video and then mean-pools the results to get a single feature vector representing the entire video. They then use an LSTM as a sequence decoder to generate a description based on this vector. A major shortcoming of this approach is that this representation completely ignores the ordering of the video frames and fails to exploit any temporal information. The approach in [4] also generates video descriptions using an LSTM; however, they employ a version of the two-step approach that uses CRFs to obtain semantic tuples of activity, object, tool, and location and then use an LSTM to translate this tuple into a sentence. Moreover, the model in [4] is applied to the limited domain of cooking videos while ours is aimed at generating descriptions for videos "in the wild". Contemporaneous with our work, the approach in [7] also addresses the limitations of [6] in two ways. First, they employ a 3-D convnet model that incorporates spatiotemporal motion features. To obtain the features, they assume videos are of fixed volume (width, height, time). They extract dense trajectory features (HoG, HoF, MBH) [8] over non-overlapping cuboids and concatenate these to form the input. The 3-D convnet is pre-trained on video datasets for action recognition. Second, they include an attention mechanism that learns to weight the frame features nonuniformly conditioned on the previous word input(s) rather than uniformly weighting features from all frames as in [6]. The 3-D convnet alone provides limited performance improvement, but in conjunction with the attention model it notably improves performance. We propose a simpler approach to using temporal information by using an LSTM to encode the sequence of video frames into a distributed vector representation that is sufficient to generate a sentential description. Therefore, our direct sequence to sequence model does not require an explicit attention mechanism.

Another recent project [9] uses LSTMs to predict the future frame sequence from an encoding of the previous frames. Their model is more similar to the language translation model in [5], which uses one LSTM to encode the input text into a fixed representation, and another LSTM to decode it into a different language. In contrast, we employ a single LSTM that learns both encoding and decoding based on the inputs it is provided. This allows the LSTM to share weights between encoding and decoding.

Chapter 3

Problem Statement

To improve and employ a video-captioning algorithm to analyze videos and generate captions which can be ‘read out’ with the help of a text-to-speech module and a web application.

Objective

Our objective is to work on a video-captioning algorithm to analyze videos and generate captions without requiring a generous amount of computing power. We then intend to provide these captions to a text-to-speech module which delivers these captions. This is then deployed as a web application.

We implemented a sequence-to-sequence architecture for the video-captioning mechanism which is then validated with loss and accuracy metrics.

Applications

Visually Impaired people can benefit a lot from this application as it describes frames of the video. This application can further be used in wearables where after further development of this system, it will be able to basically “speak” in real-time about the surroundings of the individual using text to speech API. Apart from this, the video-captioning module can be used in search algorithms for better content retrieval and recommendation systems.

Chapter 4

Proposed System

Existing Systems

1. End-to-End Dense Video Captioning with Masked Transformer (Sentences act as weak labels: contiguous sequences of words correspond to some particular (unknown) location in videos.)
2. TVT: Two-View Transformer Network for Video Captioning (Works fine only with videos within dataset.)
3. Video Captioning with Sparse Boundary-Aware Transformer (Very complex and time taking strategy, which is very difficult and costlier to deploy in real world.)

Dataset

For the purpose of this study, we have used the **MSVD** data set by Microsoft. This data set contains 1450 short YouTube clips that have been manually labelled for training and 100 videos for testing. Each video has been assigned a unique ID and each ID has about 15–20 captions.

The Microsoft Video description corpus, is a collection of Youtube clips collected on Mechanical Turk by requesting workers to pick short clips depicting a single activity. The videos were then used to elicit single sentence descriptions from annotators. The original corpus has multi-lingual descriptions, in this work we use only the English descriptions.

The dataset contains **training_data** and **testing_data** folders. Each of the folders contain a **video** sub folder which contains the videos that will be used for training as well as testing. These folders also contain **feat** sub folder which is short for features. The feat folders contain the features of the video. There is also a **training_label** and **testing_label** json files. These json files contain the captions for each ID.

```
{'caption': ['A man slicing butter into a bowl.',
  'A man cut up butter into a pan.',
  'A man cutting butter into a mixing bowl.',
  'A man is chopping butter into a container.',
  'A man is cutting a butter.',
  'A man is cutting pieces of butter into a mixing bowl.',
  'A man is cutting slices of butter into a mixing bowl.',
  'A man is slicing butter into an electric mixer.',
  'A man is slicing butter.',
  'A man is slicing some butter pieces and putting it into a steel bowl.',
  'A man puts butter into a mixing bowl.',
  'A person is putting butter chunks in the food processor.',
  'An individual cuts food and drops it into a mixer.',
  'Butter is being put into a bowl.',
  'Butter is being put into a mixer.',
  'Pieces of butter is added in the stand mixer.',
  'Someone is shown putting a small square object into a mixing bowl.',
  'The man cut butter into a bowl.',
  'The man is cutting butter.',
  'A person is adding butter into a pot.'],
 'id': 'IBgsLmDcL78 80 84.avi'},
```


Fig 4.1: Json file containing captions concerning videos

Cleaning and Preprocessing Captions

We will load all the captions and pair them with their video IDs. The train_list contains a pair of captions and it's video ID. We add the <bos> and <eos> tokens before and after each caption respectively.

- **<bos>** denotes the beginning of the sentence hence the model knows to start predicting from here and

- **<eos>** denotes the end of the statement, this is where the model knows to stop the prediction.

```
['<bos> A woman goes under a horse. <eos>', 'xBePrplM40A_6_18.avi']
["<bos> A horse defecated on a woman's head. <eos>", 'xBePrplM40A_6_18.avi']
["<bos> A horse is defecating on a woman's head. <eos>", 'xBePrplM40A_6_18.avi']
['<bos> A horse poops on a woman. <eos>', 'xBePrplM40A_6_18.avi']
['<bos> A horse poops on a woman. <eos>', 'xBePrplM40A_6_18.avi']
['<bos> A woman goes underneath a horse. <eos>', 'xBePrplM40A_6_18.avi']
['<bos> A woman is crawling under a horse. <eos>', 'xBePrplM40A_6_18.avi']
['<bos> A woman is walking between horse legs. <eos>', 'xBePrplM40A_6_18.avi']
['<bos> A man slicing butter into a bowl. <eos>', 'IBgsLmDcL78_80_84.avi']
['<bos> A man cut up butter into a pan. <eos>', 'IBgsLmDcL78_80_84.avi']
['<bos> A man cutting butter into a mixing bowl. <eos>', 'IBgsLmDcL78_80_84.avi']
['<bos> A man is chopping butter into a container. <eos>', 'IBgsLmDcL78_80_84.avi']
['<bos> A man is cutting a butter. <eos>', 'IBgsLmDcL78_80_84.avi']
['<bos> A man is slicing butter. <eos>', 'IBgsLmDcL78_80_84.avi']
['<bos> A man puts butter into a mixing bowl. <eos>', 'IBgsLmDcL78_80_84.avi']
['<bos> Butter is being put into a bowl. <eos>', 'IBgsLmDcL78_80_84.avi']
['<bos> Butter is being put into a mixer. <eos>', 'IBgsLmDcL78_80_84.avi']
```

Fig 4.2: Json file after prepocessing

The train_list is split into training and validation. The training_list contains 85% of the data and the rest is present in the validation_list.

The vocab_list contains only the captions from the training_list because we only use the words in the training data to tokenize. After tokenizing we pad the captions so that all the sentences are of the same length. Here, we have padded all of them to be of 10 words. We only use captions where the number of words is between 6 and 10.

This is because the caption with the maximum number of words in all of the data set it has 39 words but for most captions the number of words are between 6 and 10. If we do not filter out some of the captions we will have to pad them all to the maximum length of the captions, here in our case 39. Now if most sentences are of 10 words and we will have to pad them to double its length this would lead to a lot of padding. These highly padded sentences will be used for training which will lead to the model predicting mostly padded tokens. Since padding basically means adding white spaces so most of the sentences predicted by the model will just contain more blank spaces less words leading to incomplete sentences.

We only used the top 1500 words as the vocabulary for the captions. Any of the captions we see generated has to be a part of the 1500 words. Even though the number of unique words are way more than 1500 most of the words appear very few only 1, 2 or 3 times making the vocabulary prone to outliers. Hence to keep it avoid weird caption predictions we only use the top 1500 most occurring words.

Working of Sequence to Sequence Model

Mostly for problems related to text generation, the preferred model is an encoder-decoder architecture. Here in our problem statement since the text has to be generated, we also use this sequence-to-sequence architecture. A sequence to sequence model aims to map a fixed-length input with a fixed-length output where the length of the input and output may differ.

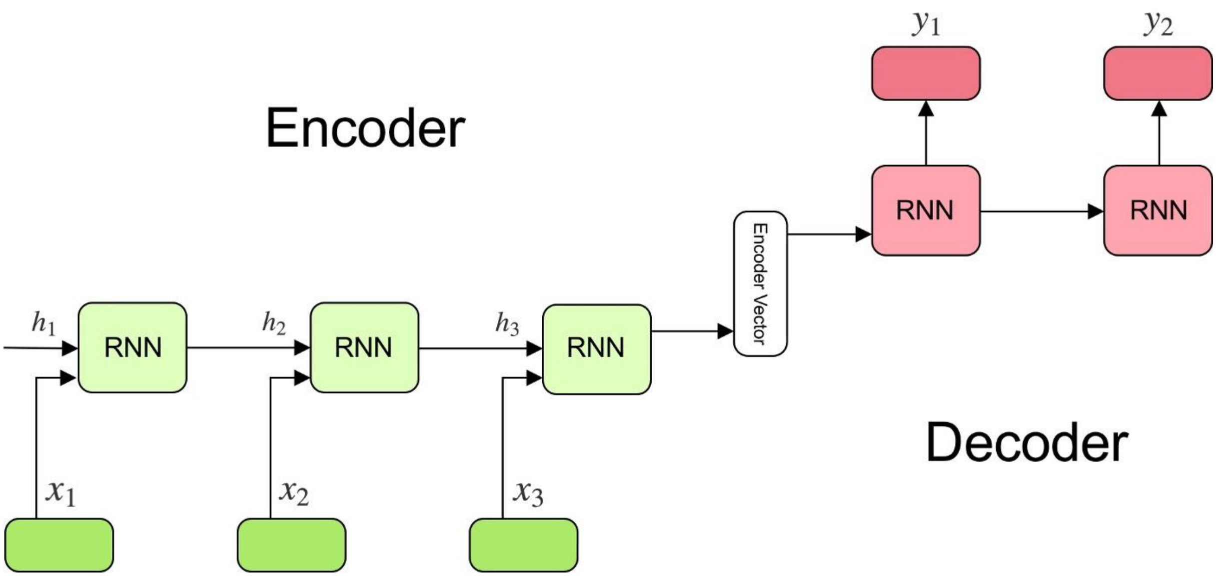


Fig 4.3: Encoder-Decoder Sequence to Sequence Model

The model consists of 3 parts: encoder, intermediate (encoder) vector and decoder.

Encoder

- A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward.
- In question-answering problem, the input sequence is a collection of all words from the question. Each word is represented as *x_i* where *i* is the order of that word.
-

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

Fig 4.4: Formula to compute the hidden states *h_i*

This simple formula represents the result of an ordinary recurrent neural network. We apply the appropriate weights to the previous hidden state *h_(t-1)* and the input vector *x_t*.

Encoder Vector

- This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above.
- This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
- It acts as the initial hidden state of the decoder part of the model.

Decoder

- A stack of several recurrent units where each predicts an output *y_t* at a time step *t*.
- Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state.
- In the question-answering problem, the output sequence is a collection of all words from the answer. Each word is represented as *y_i* where *i* is the order of that word.
-

$$h_t = f(W^{(hh)} h_{t-1})$$

Fig 4.5: Formula to compute any hidden state *h_i*

We are just using the previous hidden state to compute the next one.

-

$$y_t = softmax(W^S h_t)$$

Fig 4.6: Formula to compute the output *y_t* at time step *t*

We calculate the outputs using the hidden state at the current time step together with the respective weight W(S). Softmax is used to create a probability vector which will help us determine the final output (e.g. word in the question-answering problem).

Model Implementation

In this architecture, the final state of the encoder cell always acts as the initial state of the decoder cell. In our problem we will use the encoder to input the video features and the decoder will be fed the captions.

A video is basically a sequence of images. For anything related to sequence we always prefer using RNNs or LSTMs. In our case we will use an LSTM. We use LSTMs in the decoder as well because the decoder will generate captions which are basically a sequence of words.

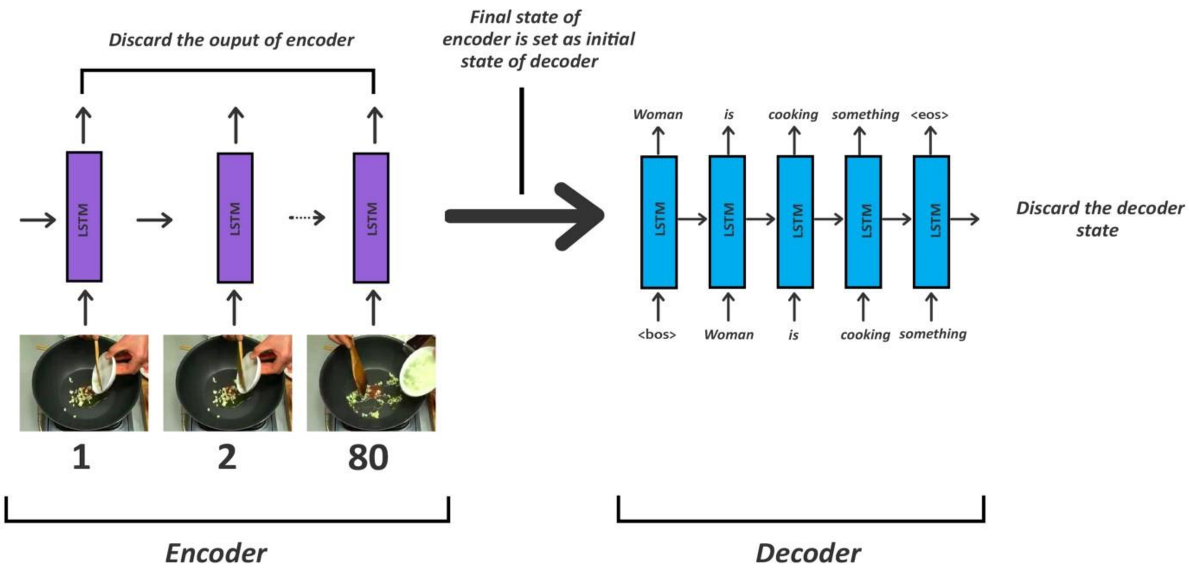


Fig 4.7: Training model

Here in the figure the features of the first frame are fed into the 1st LSTM cell of the encoder. This is followed by the features of the second frame and this goes on till the 80th frame. For this problem we are interested only in the final state of the encoder so all the other outputs from the encoder are discarded. Now the final state of the encoder LSTM acts as the initial state for the decoder LSTM. Here in the first decoder LSTM <bos> acts as input to start the sentence. Each and every word of the caption from the training data is fed one by one until <eos>.

So for the example above, if the actual caption is woman is cooking something the decoder starts with <bos> in the first decoder LSTM. In the next cell the next word from the actual caption woman is fed followed by is cooking something. This ends with <eos> token.

The time steps for the encoder is the number of LSTM cells we will use for the encoder which is equal to 80. Encoder tokens is the number of features from video which is 4096 in our case. Time steps for decoder is the number of LSTM cells for decoder which is 10 and number of tokens is the length of vocabulary which is 1500.

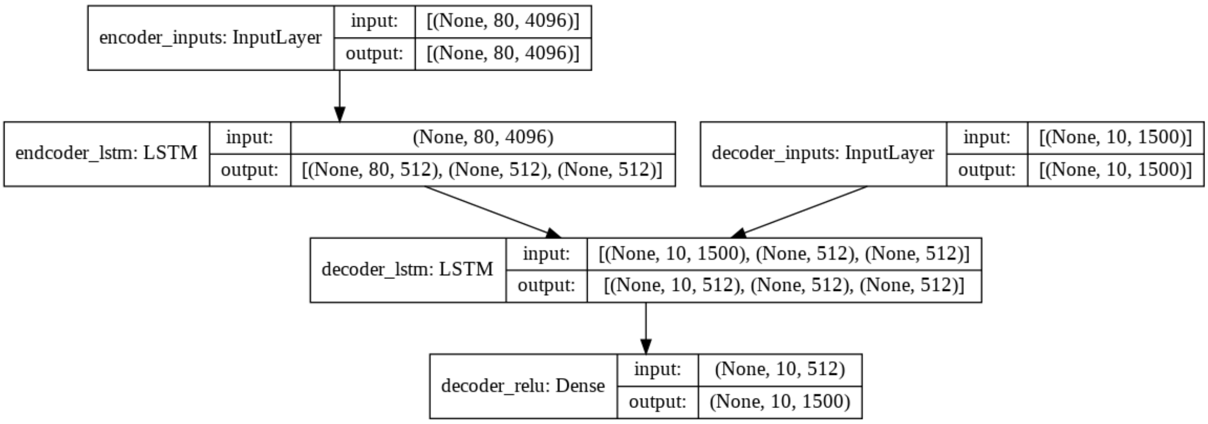


Fig 4.8: Training architecture

Loading The Dataset

Loading data into the model is also a very important part of the training. The number of training data points is around 14k which will definitely cause RAM memory issues. To avoid such a problem, we use a data generator with a batch size of 320. Since there are two inputs for training. we convert it into a list and then feed the two together as encoder input which contains the features of the video as input and decoder input which are the captions that have been tokenized and padded converted to categorical features with 1500 labels as that is the length of vocabulary we will use as the number of decoder tokens. We use the yield statement to return the output. Yield statements are used to create generators. Here we use a custom generator because we have two inputs. We load all the features in the form of a dictionary so that it takes less time loading the same arrays again and again.

Model For Inference

Unlike most neural networks the training and testing models are different for encoder-decoder. We do not save the whole model as is after training. We save the encoder model and the decoder part differently. Now let us look into the inference model.

First we will use the encoder model. Features from all the 80 frames are passed into the model. This part of the model is the same as it was for training. The encoder model gives us the predictions. Here again we are interested in the final output state so all the other outputs from the encoder will be discarded. The final state of encoder is fed into the decoder as it's initial state along with the <bos> token so that the decoder predicts the next word.

There are two ways to generate captions (beam search and greedy search) but for faster results in real time prediction we have used greedy search. Greedy search selects the most likely word at each step in the output sequence. Beam search algorithm selects multiple alternatives for an input sequence at each timestep based on conditional probability.

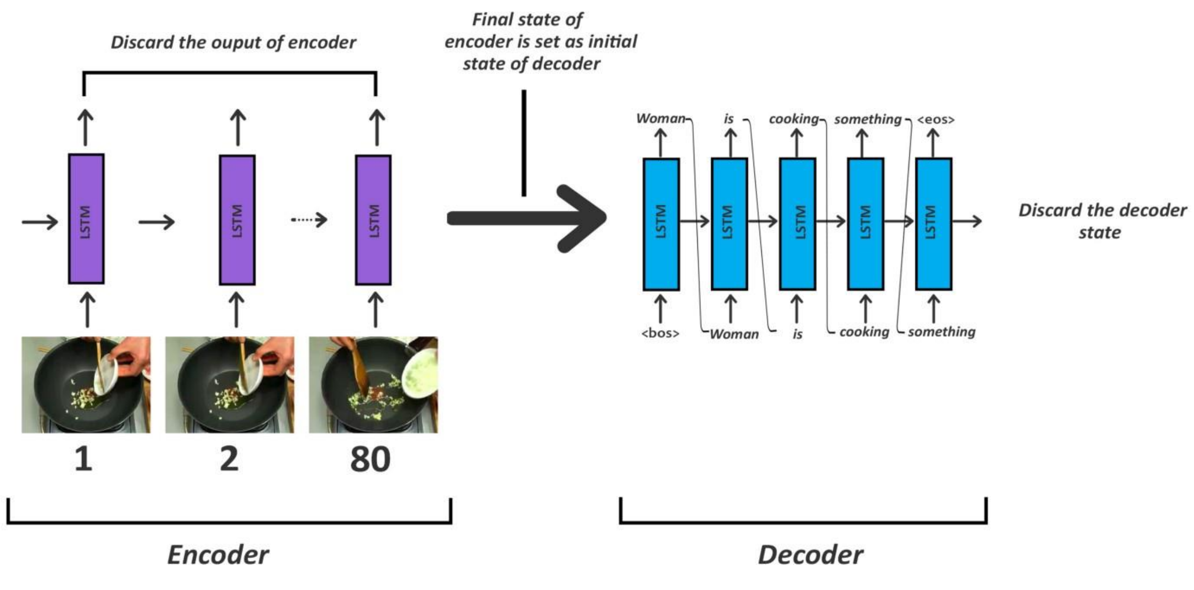


Fig 4.9: Inference model

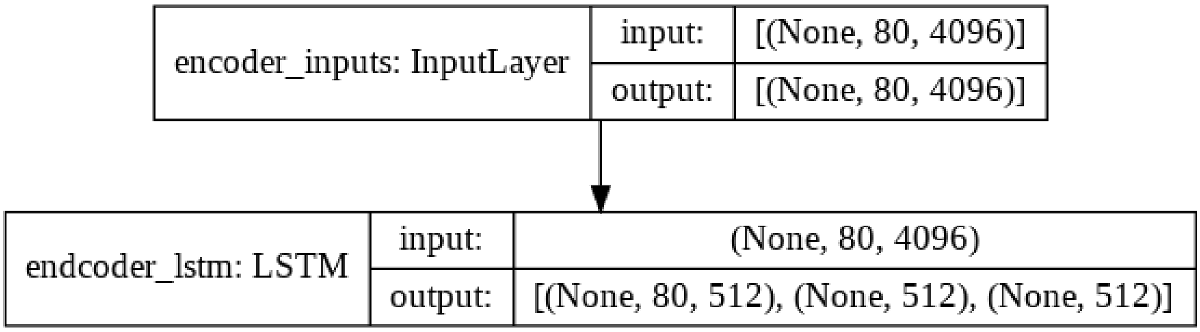


Fig 4.10: Inference Architecture – Encoder Model

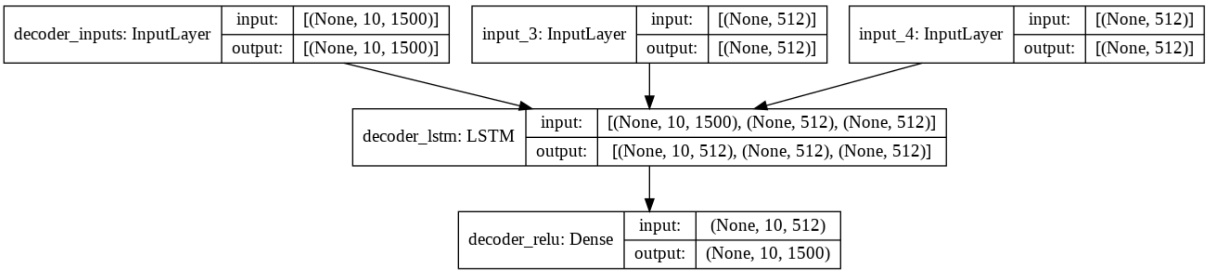


Fig 4.11: Inference Architecture – Decoder Model

Now if the model is trained properly as you can see above it should predict woman as the token. In training the next input is always the next word in the captions. Since we have no captions here the next word is the output from the previous LSTM cell. The output woman is then fed into the next cell along with the state of the previous cell. This goes on to predict the next word is. This goes on until the model predicts <eos>. We will no longer need any more predictions because the sentence is complete.

Design and Methodology

After preprocessing the data, training the model and saving the model weights, we started building a web application in Flask that can accept videos from the user and generate captions on new data in real-time. Later on we use a text to speech API to output those captions in audio format using Pyttsx3

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

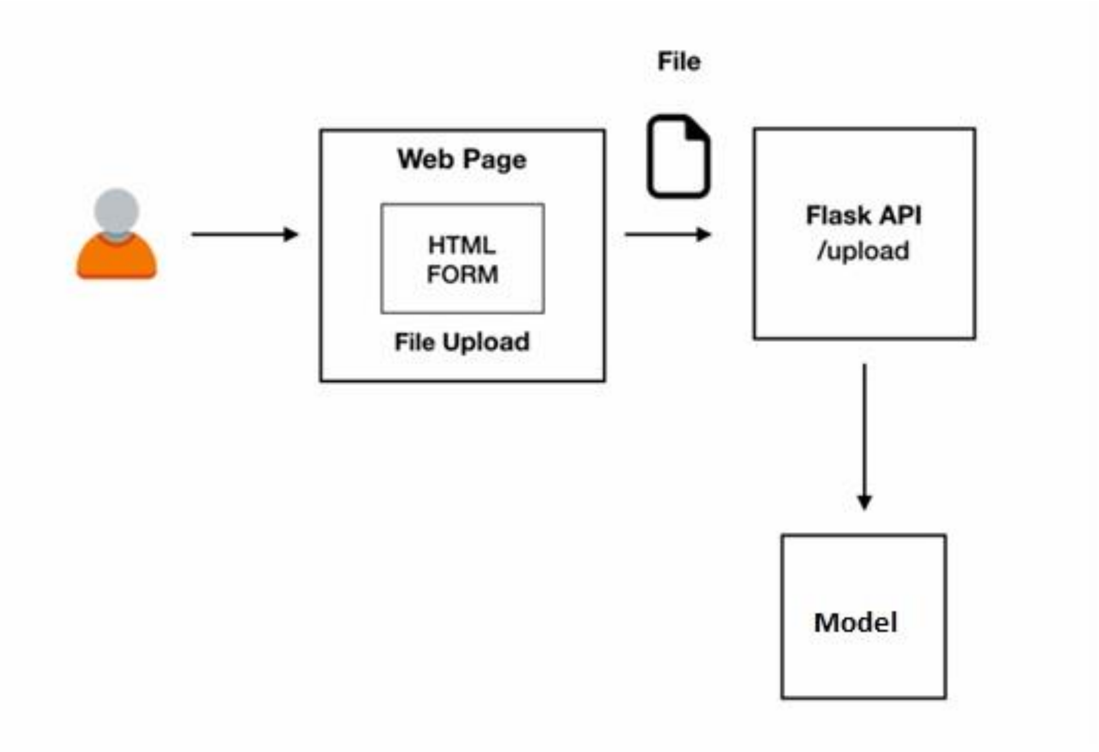


Fig 4.12: Flask Architecture

Component of Flask that we used is Werkzeug. It is a utility library for the Python programming language, in other words a toolkit for Web Server Gateway Interface (WSGI). Werkzeug can realize software objects for request, response, and utility functions. It can be used to build a custom software framework on top of it and supports Python 2.7 and 3.5 and later.

Features of Flask:

- Development server and debugger
- Integrated support for unit testing
- RESTful request dispatching
- Uses Jinja templating
- Support for secure cookies (client side sessions)
- 100% WSGI 1.0 compliant
- Unicode-based
- Complete documentation
- Google App Engine compatibility
- Extensions available to extend functionality

To read out the captions (audio format) generated by the model, we used pyttsx3. pyttsx3 is a text-to-speech conversion library in Python. Unlike alternative libraries, it works offline and is compatible with both Python 2 and 3. An application invokes the pyttsx3.init() factory function to get a reference to a pyttsx3. Engine instance. it is a very easy to use tool which converts the entered text into speech. The pyttsx3 module supports two voices first is female and the second is male which is provided by “sapi5” for windows. It supports three TTS engines:

- *sapi5* – SAPI5 on Windows
- *nsss* – NSSpeechSynthesizer on Mac OS X
- *espeak* – eSpeak on every other platform

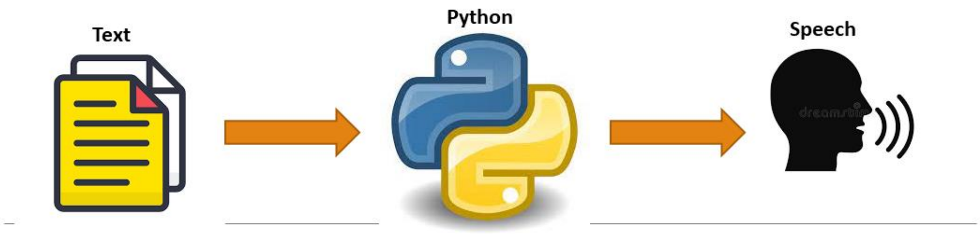


Fig 4.13: Text to Speech using Python

Features of pyttsx3:

- Fully offline text to speech conversion
- Choose among different voices installed in your system
- Control speed/rate of speech
- Tweak Volume
- Save the speech audio as a file
- Simple, powerful, & intuitive API

The client uploads a video through our HTML form which is embedded in Flask. The video is then sent to predict_realtime python file which calls upon functions VideoDescription and video_to_text which consists of our sequence-to-sequence architecture. The config.yml file links our models and dataset to provide us with an output that is the caption to the video uploaded. The caption is then sent to our web application along with the video displayed. Text to speech API converts the text caption into mp3 format audible file, which is played simultaneously with the video being displayed and the caption being printed.

Chapter 5

System Requirements

Hardware Requirements

Currently, we have only used Kaggle (Nvidia Tesla P100) and Google Collaboratory (Nvidia K80) free GPUs access to train our models and build our project. However, for faster results we would require a machine with higher specifications and a GPU with longer access durations (more than 12 hours).

Software Requirements

- **Kaggle**
  Kaggle offers a no-setup, customizable, Jupyter Notebooks environment. Access free GPUs and a huge repository of community-published data & code.

- **Google Colaboratory**
  It was also used for writing code for the project as it allows you to write and execute Python in your browser, with zero configuration and most importantly gives free access to GPUs.

- **Tensorflow**
  TensorFlow is an end-to-end open-source platform and has a comprehensive, flexible ecosystem of tools, libraries and community resources that allows you to easily build and deploy ML-powered applications. It was used to build our project.

- **Keras**
  Keras is a deep learning API written in Python, running on top of the TensorFlow platform.

Chapter 6

Conclusion and Future Development

Conclusion

This project proposed a novel approach to video description. In contrast to related work, we construct descriptions using a sequence to sequence model, where frames are first read sequentially and then words are generated sequentially. This allows us to handle variable-length input and output while simultaneously modeling temporal structure. Our model achieves state-of-the-art performance on the MSVD dataset. We understand that training data should be semantically similar to testing data. For example, if we train the model on videos of animals and test it on different activities it is bound to give bad results.

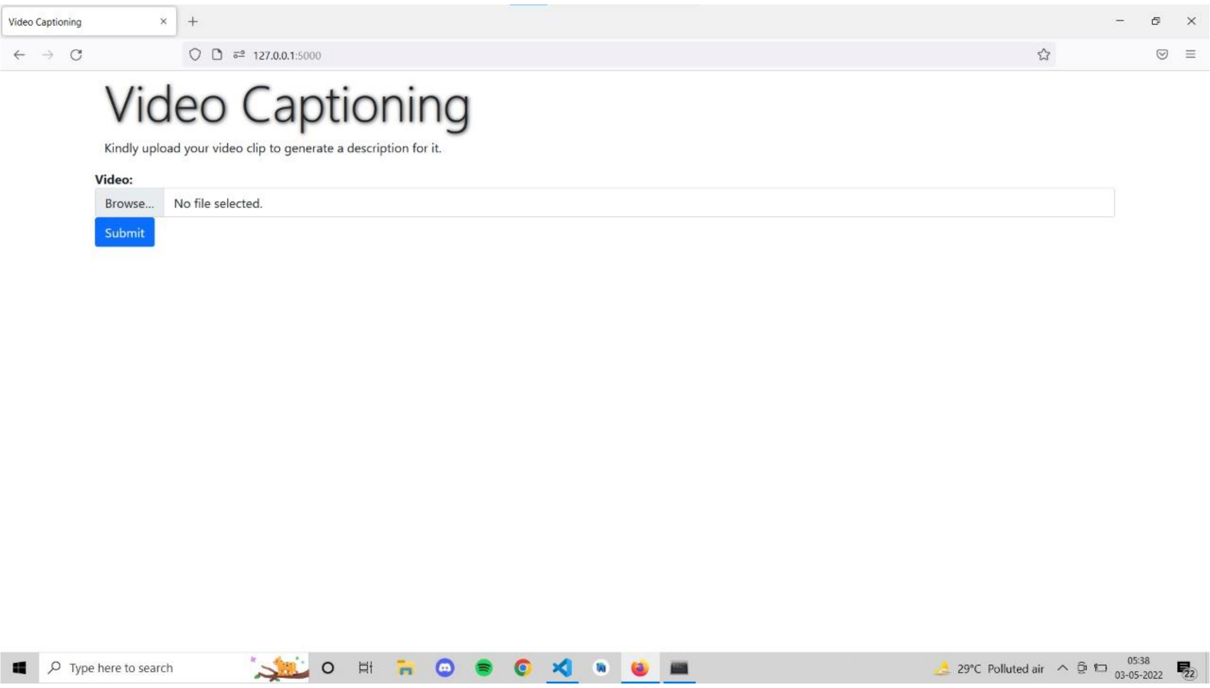
Future Work

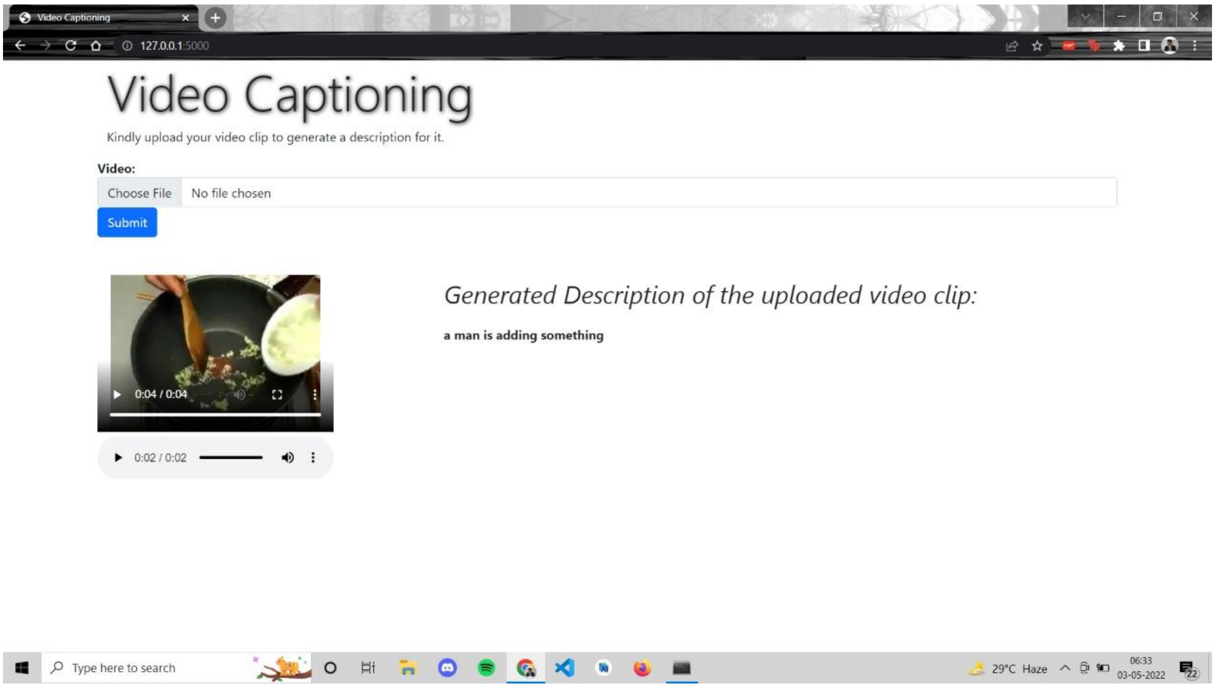
- Using other pretrained models to extract features specially ones made for understanding videos like I3D
- Adding attention blocks and pretrained embedding like glove so that the model understands sentences better
- Right now the model uses only 80 frames improvements need to be made so that it can work even for longer videos
- Improve the UI to make it more attractive and deploy it on some platform

Chapter 7

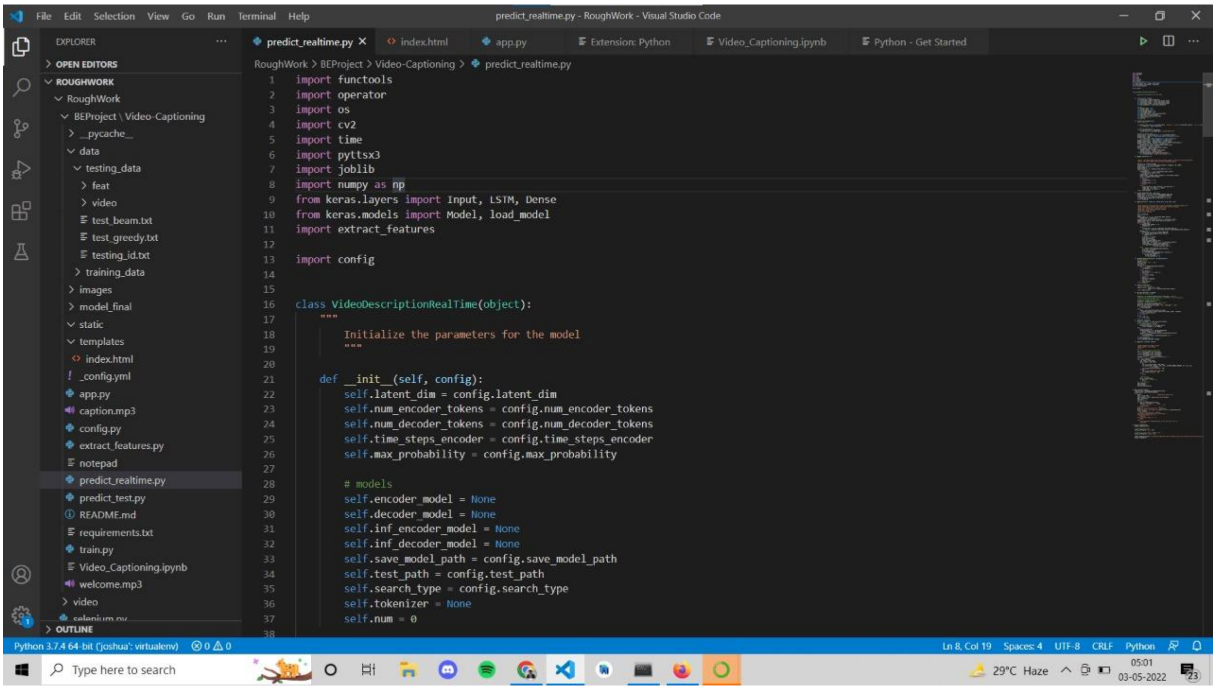
Output and Implementation

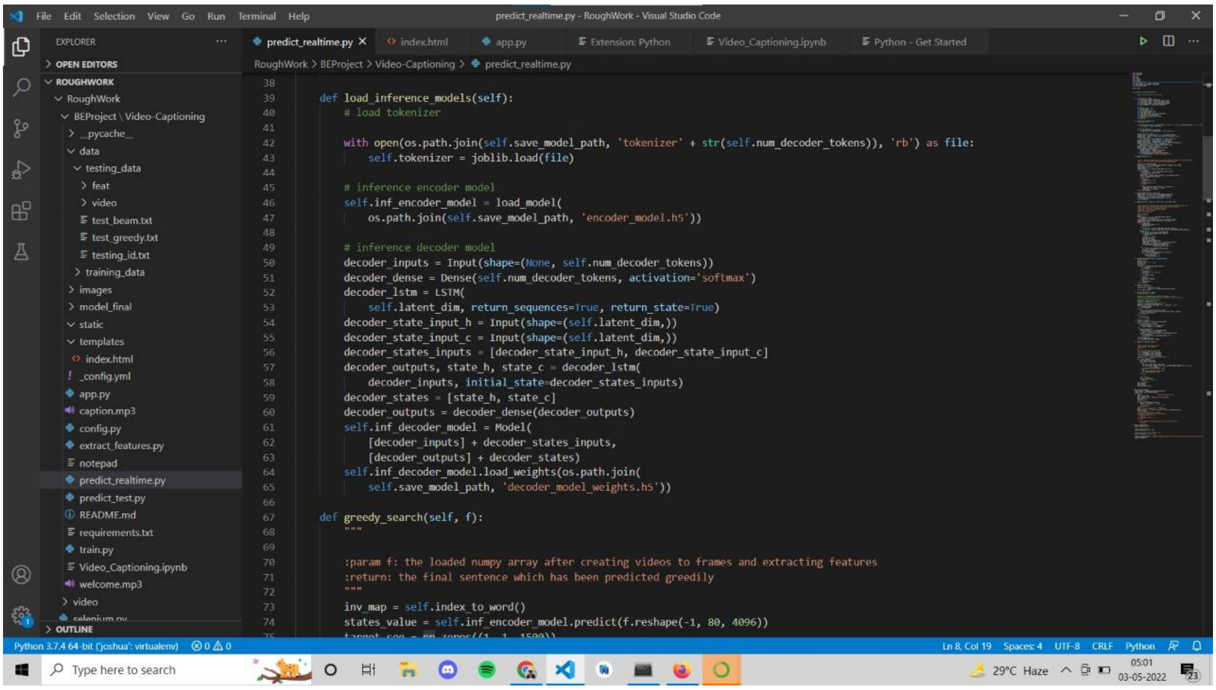
Output

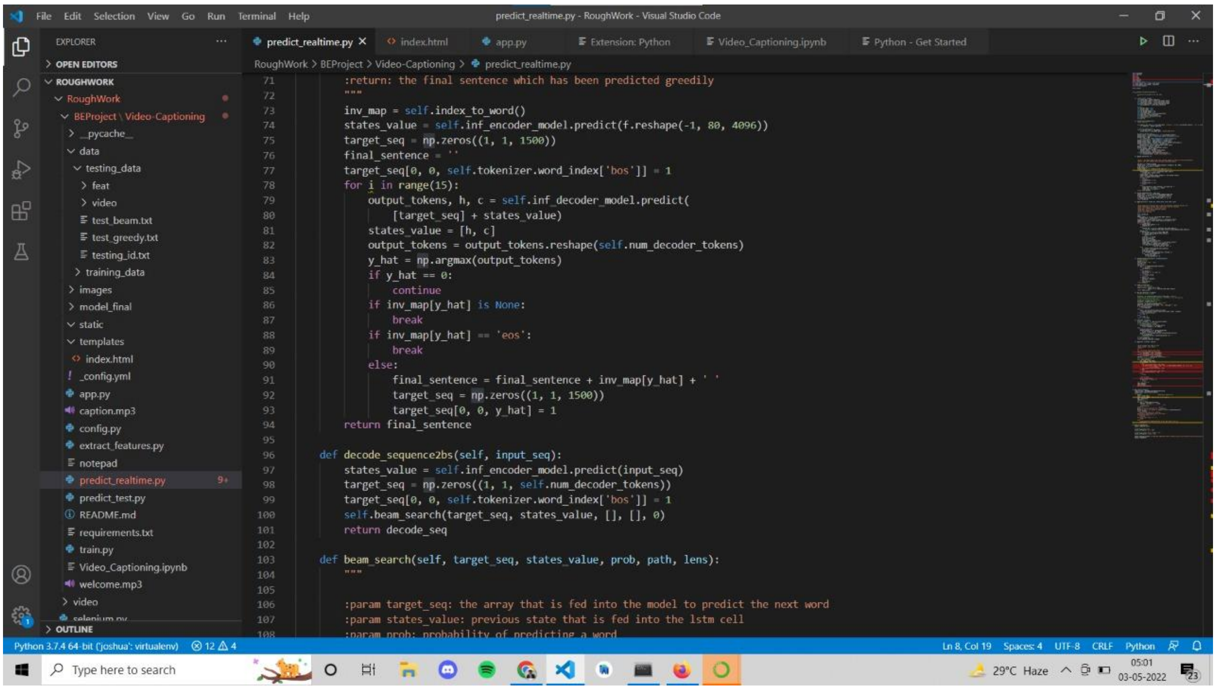


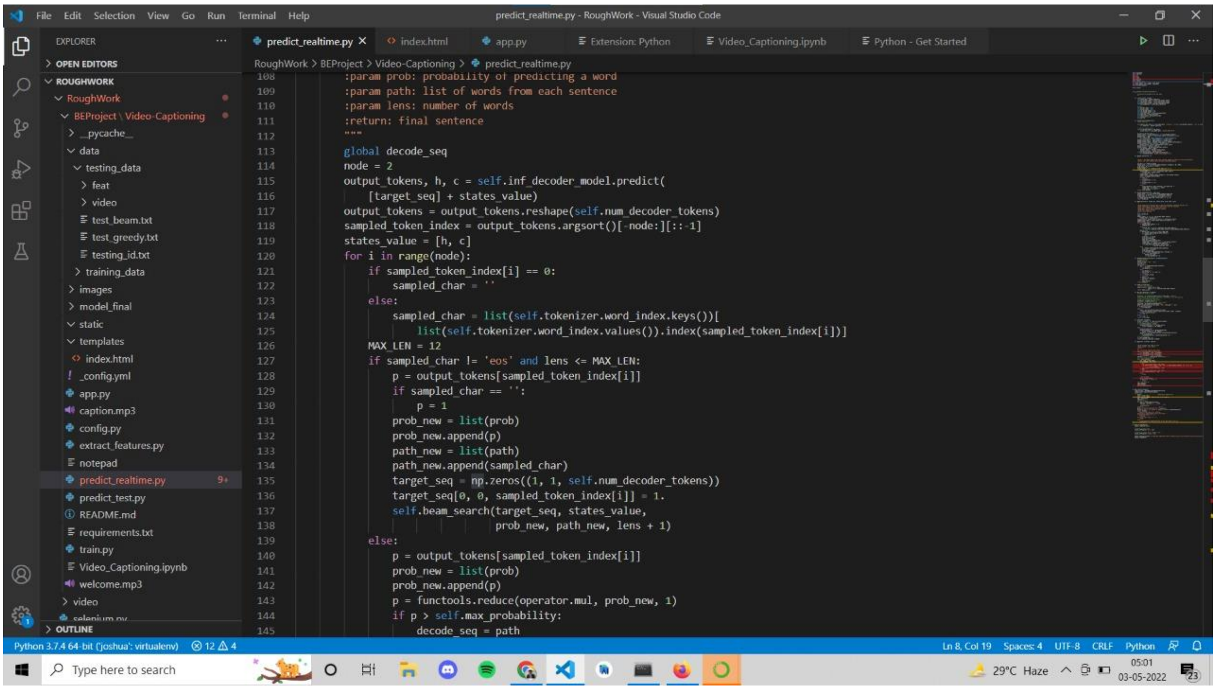


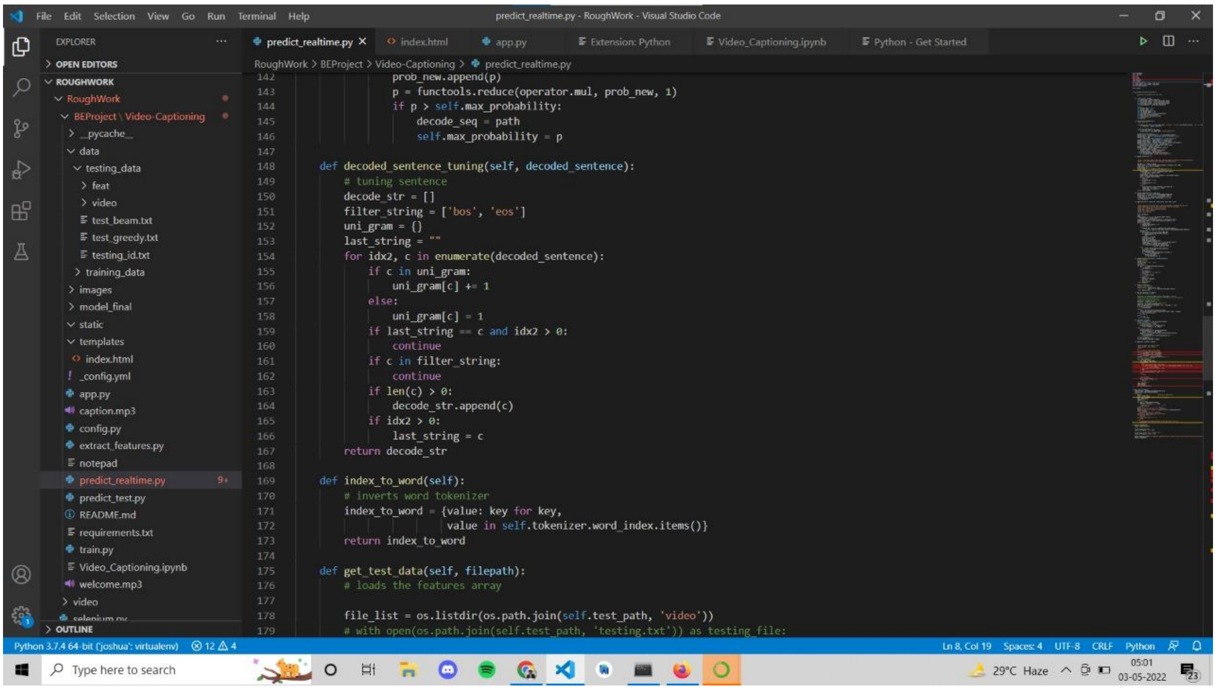
Implementation

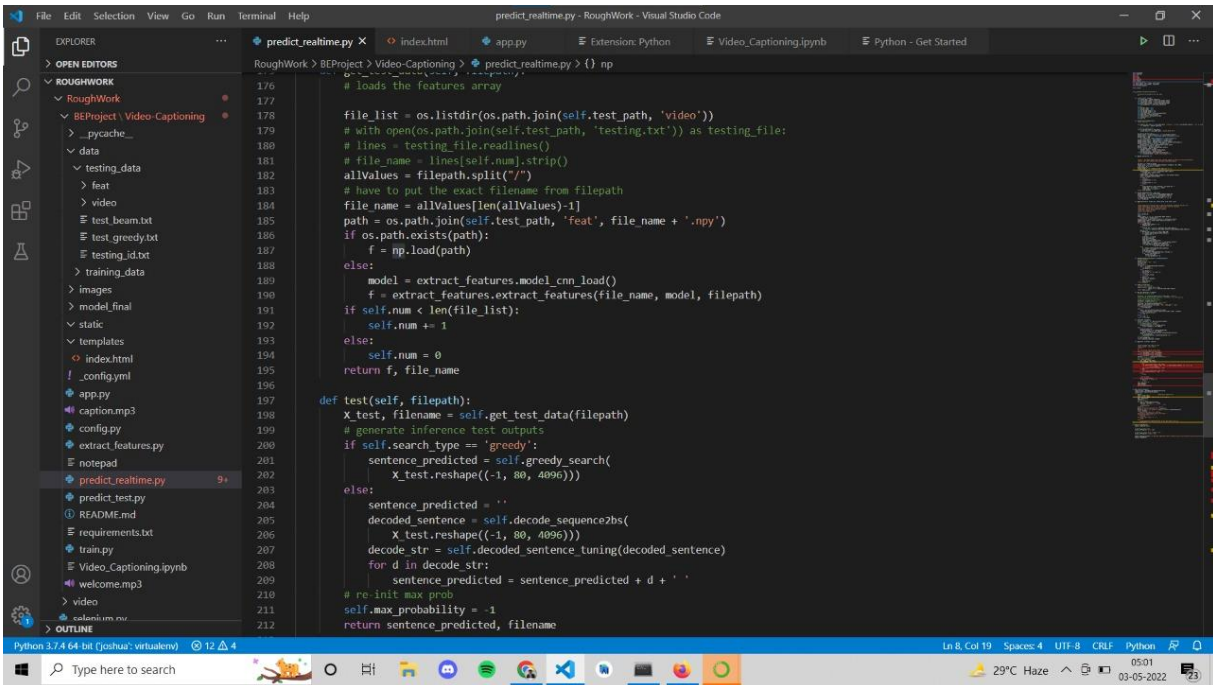


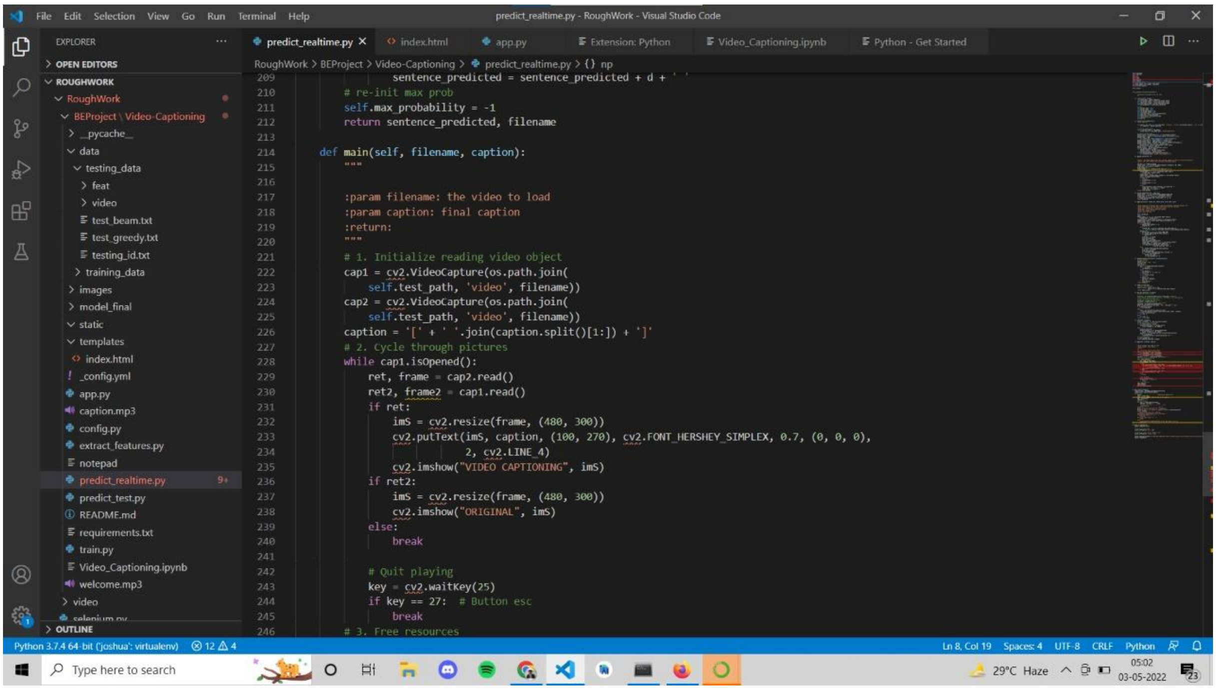


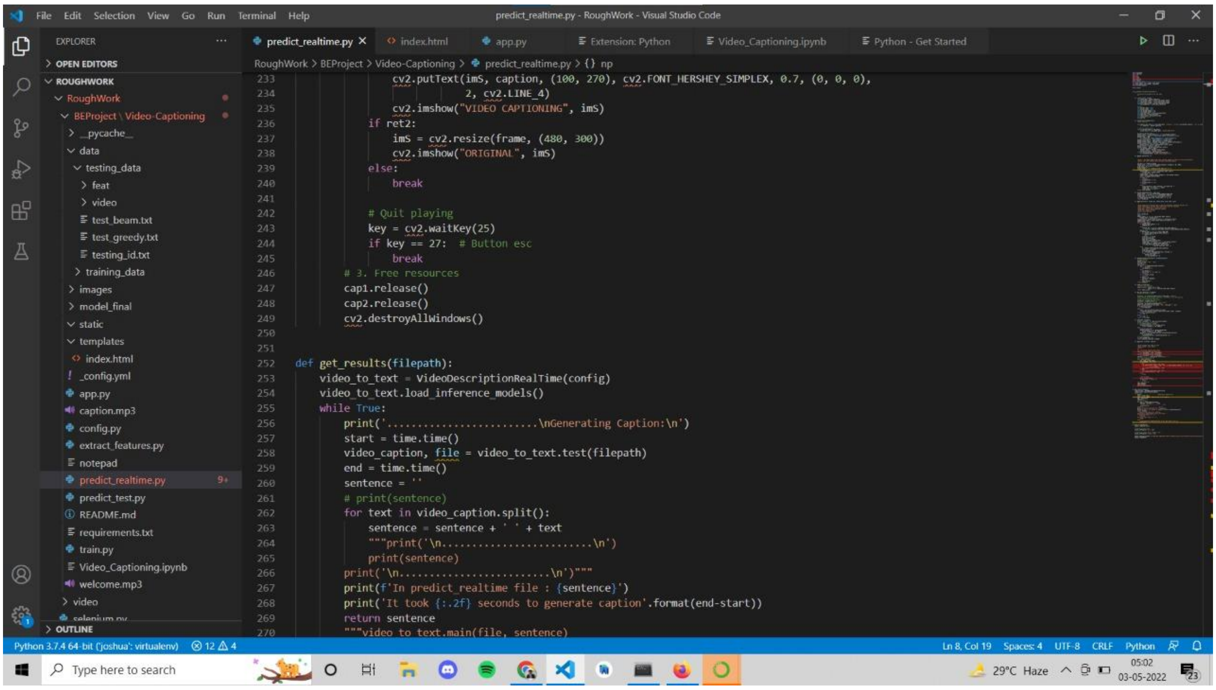

predict_realtime.py - RoughWork - Visual Studio Code
RoughWork > BEProject > Video-Captioning > predict_realtime.py
:return: the final sentence which has been predicted greedily
"""
inv_map = self.index_to_word()
states_value = self.inf_encoder_model.predict(f.reshape(-1, 80, 4096))
target_seq = np.zeros((1, 1, 1500))
final_sentence = ''
target_seq[0, 0, self.tokenizer.word_index['bos']] = 1
for i in range(15):
output_tokens, h, c = self.inf_decoder_model.predict(
[target_seq] + states_value)
states_value = [h, c]
output_tokens = output_tokens.reshape(self.num_decoder_tokens)
y_hat = np.argmax(output_tokens)
if y_hat == 0:
continue
if inv_map[y_hat] is None:
break
if inv_map[y_hat] == 'eos':
break
else:
final_sentence = final_sentence + inv_map[y_hat] + ' '
target_seq = np.zeros((1, 1, 1500))
target_seq[0, 0, y_hat] = 1
return final_sentence
def decode_sequence2bs(self, input_seq):
states_value = self.inf_encoder_model.predict(input_seq)
target_seq = np.zeros((1, 1, self.num_decoder_tokens))
target_seq[0, 0, self.tokenizer.word_index['bos']] = 1
self.beam_search(target_seq, states_value, [], [], 0)
return decode_seq
def beam_search(self, target_seq, states_value, prob, path, lens):
"""
:param target_seq: the array that is fed into the model to predict the next word
:param states_value: previous state that is fed into the lstm cell

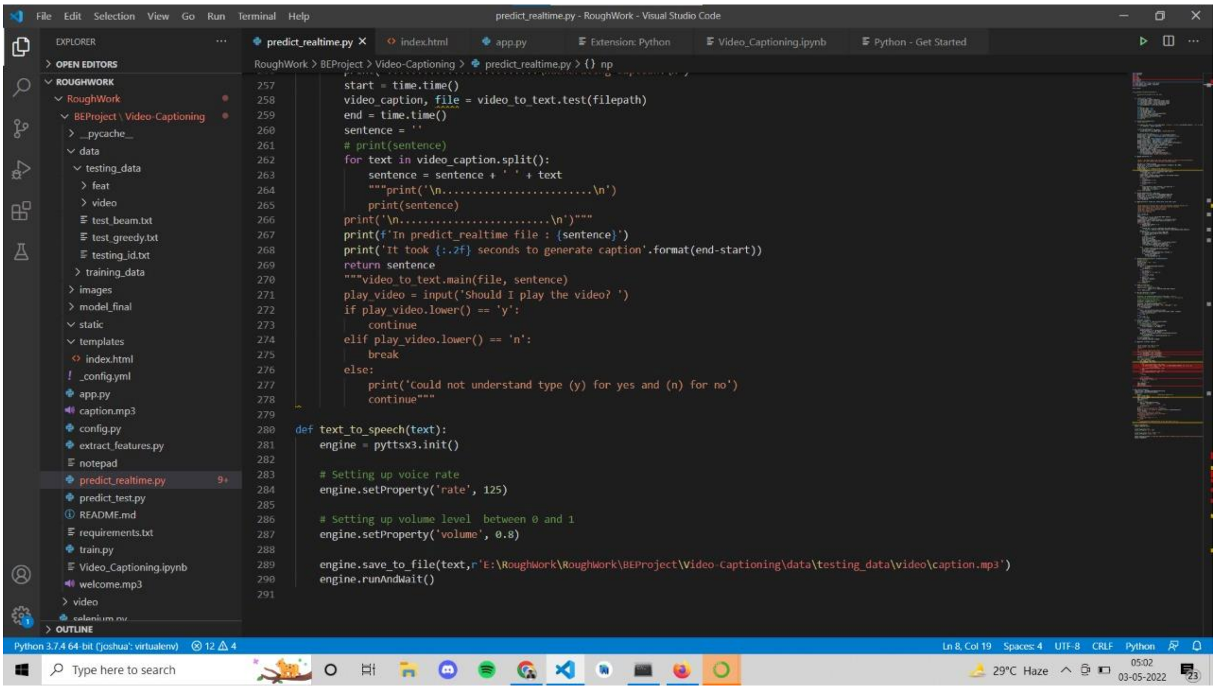

predict_realtime.py - RoughWork - Visual Studio Code
RoughWork > BEProject > Video-Captioning > predict_realtime.py
:param prob: probability of predicting a word
:param path: list of words from each sentence
:param lens: number of words
:return: final sentence
"""
global decode_seq
node = 2
output_tokens, h, c = self.inf_decoder_model.predict(
[target_seq] + states_value)
output_tokens = output_tokens.reshape(self.num_decoder_tokens)
sampled_token_index = output_tokens.argsort()[-node:][::-1]
states_value = [h, c]
for i in range(node):
if sampled_token_index[i] == 0:
sampled_char = ''
else:
sampled_char = list(self.tokenizer.word_index.keys())[
list(self.tokenizer.word_index.values()).index(sampled_token_index[i])]
MAX_LEN = 12
if sampled_char != 'eos' and lens <= MAX_LEN:
p = output_tokens[sampled_token_index[i]]
if sampled_char == '':
p = 1
prob_new = list(prob)
prob_new.append(p)
path_new = list(path)
path_new.append(sampled_char)
target_seq = np.zeros((1, 1, self.num_decoder_tokens))
target_seq[0, 0, sampled_token_index[i]] = 1.
self.beam_search(target_seq, states_value,
prob_new, path_new, lens + 1)
else:
p = output_tokens[sampled_token_index[i]]
prob_new = list(prob)
prob_new.append(p)
p = functools.reduce(operator.mul, prob_new, 1)
if p > self.max_probability:
decode_seq = path

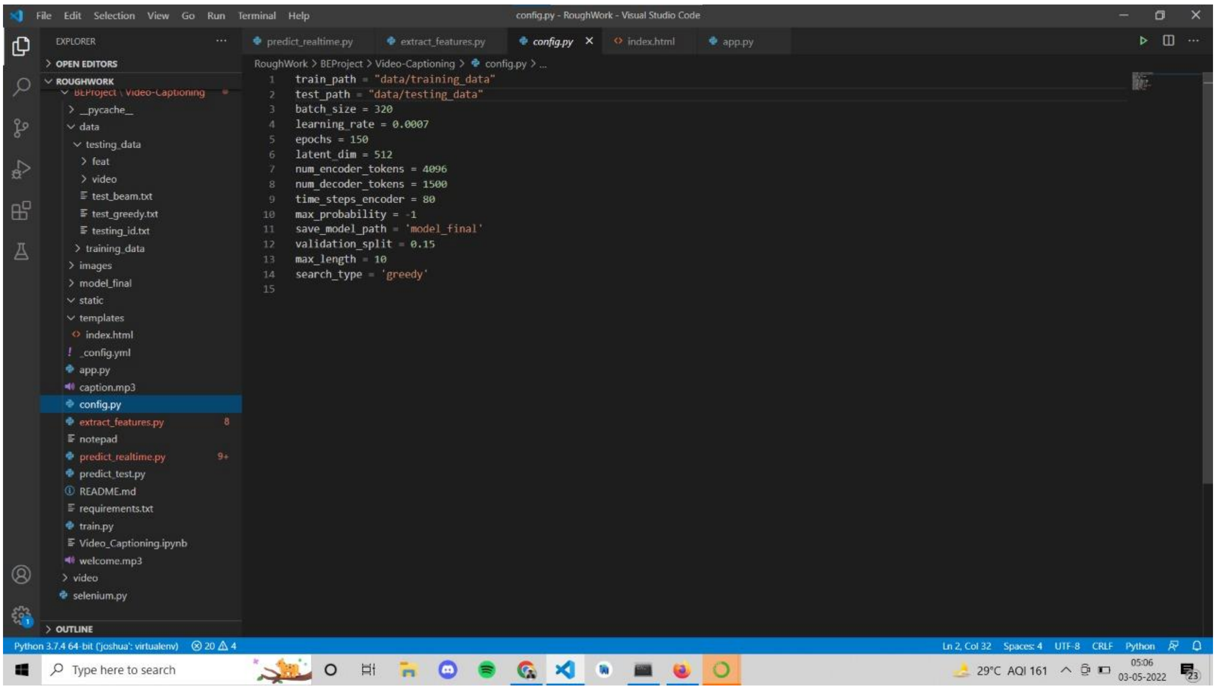

predict_realtime.py - RoughWork - Visual Studio Code
RoughWork > BEProject > Video-Captioning > predict_realtime.py
prob_new.append(p)
p = functools.reduce(operator.mul, prob_new, 1)
if p > self.max_probability:
decode_seq = path
self.max_probability = p
def decoded_sentence_tuning(self, decoded_sentence):
# tuning sentence
decode_str = []
filter_string = ['bos', 'eos']
uni_gram = {}
last_string = ""
for idx2, c in enumerate(decoded_sentence):
if c in uni_gram:
uni_gram[c] += 1
else:
uni_gram[c] = 1
if last_string == c and idx2 > 0:
continue
if c in filter_string:
continue
if len(c) > 0:
decode_str.append(c)
if idx2 > 0:
last_string = c
return decode_str
def index_to_word(self):
# inverts word tokenizer
index_to_word = {value: key for key,
value in self.tokenizer.word_index.items()}
return index_to_word
def get_test_data(self, filepath):
# loads the features array
file_list = os.listdir(os.path.join(self.test_path, 'video'))
# with open(os.path.join(self.test_path, 'testing.txt')) as testing_file:

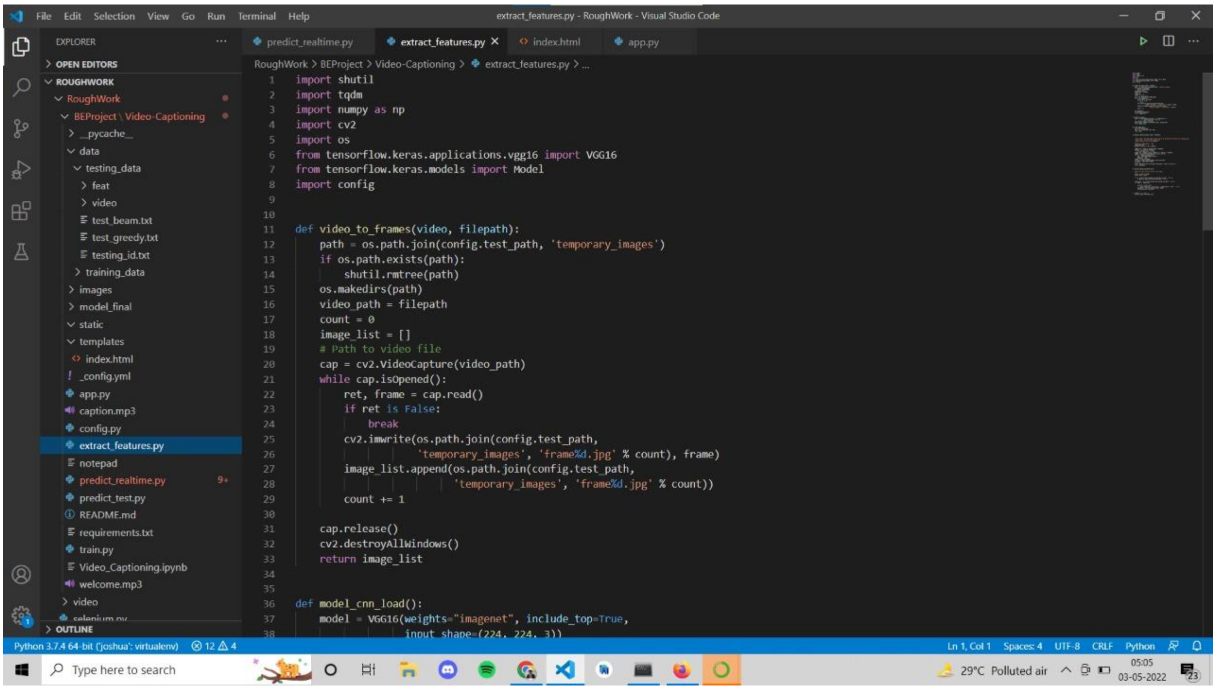

predict_realtime.py - RoughWork - Visual Studio Code
RoughWork > BEProject > Video-Captioning > predict_realtime.py > {} np
# loads the features array
file_list = os.listdir(os.path.join(self.test_path, 'video'))
# with open(os.path.join(self.test_path, 'testing.txt')) as testing_file:
# lines = testing_file.readlines()
# file_name = lines[self.num].strip()
allValues = filepath.split("/")
# have to put the exact filename from filepath
file_name = allValues[len(allValues)-1]
path = os.path.join(self.test_path, 'feat', file_name + '.npy')
if os.path.exists(path):
f = np.load(path)
else:
model = extract_features.model_cnn_load()
f = extract_features.extract_features(file_name, model, filepath)
if self.num < len(file_list):
self.num += 1
else:
self.num = 0
return f, file_name
def test(self, filepath):
X_test, filename = self.get_test_data(filepath)
# generate inference test outputs
if self.search_type == 'greedy':
sentence_predicted = self.greedy_search(
X_test.reshape((-1, 80, 4096)))
else:
sentence_predicted = ''
decoded_sentence = self.decode_sequence2bs(
X_test.reshape((-1, 80, 4096)))
decode_str = self.decoded_sentence_tuning(decoded_sentence)
for d in decode_str:
sentence_predicted = sentence_predicted + d + ' '
# re-init max prob
self.max_probability = -1
return sentence_predicted, filename

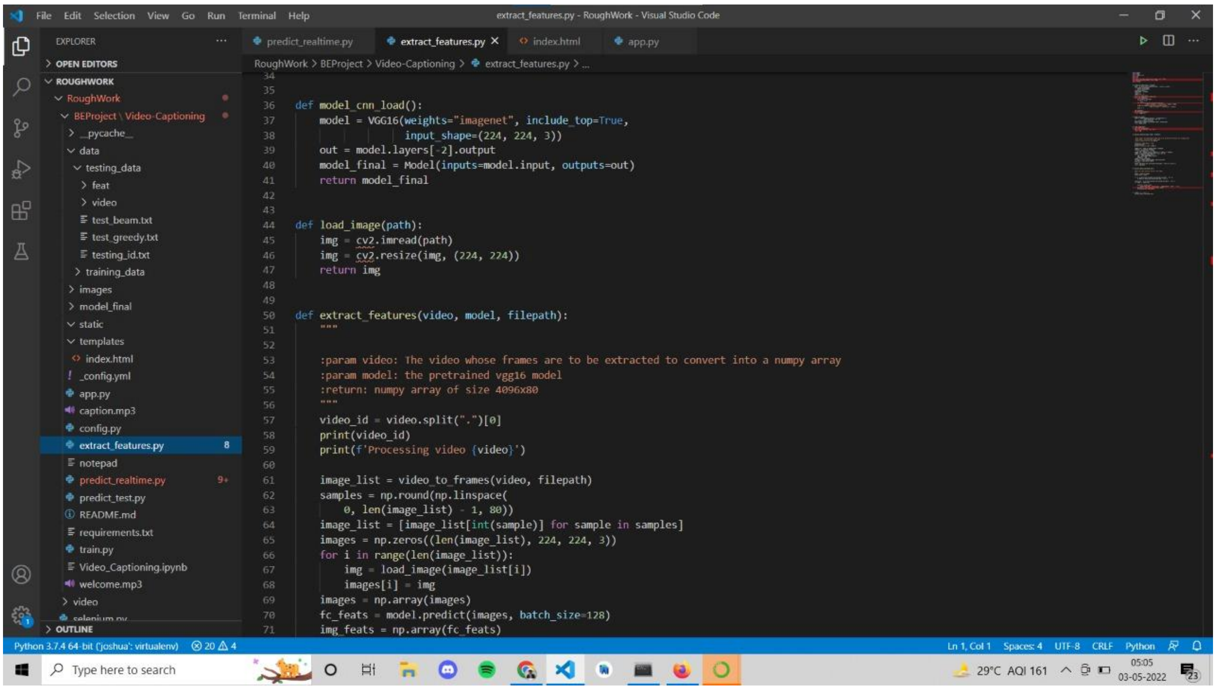

predict_realtime.py - RoughWork - Visual Studio Code
sentence_predicted = sentence_predicted + d + ' '
# re-init max prob
self.max_probability = -1
return sentence_predicted, filename
def main(self, filename, caption):
"""
:param filename: the video to load
:param caption: final caption
:return:
"""
# 1. Initialize reading video object
cap1 = cv2.VideoCapture(os.path.join(
self.test_path, 'video', filename))
cap2 = cv2.VideoCapture(os.path.join(
self.test_path, 'video', filename))
caption = '[' + ' '.join(caption.split()[1:]) + ']'
# 2. Cycle through pictures
while cap1.isOpened():
ret, frame = cap2.read()
ret2, frame2 = cap1.read()
if ret:
imS = cv2.resize(frame, (480, 300))
cv2.putText(imS, caption, (100, 270), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0),
2, cv2.LINE_4)
cv2.imshow("VIDEO CAPTIONING", imS)
if ret2:
imS = cv2.resize(frame, (480, 300))
cv2.imshow("ORIGINAL", imS)
else:
break
# Quit playing
key = cv2.waitKey(25)
if key == 27: # Button esc
break
3. Free resources

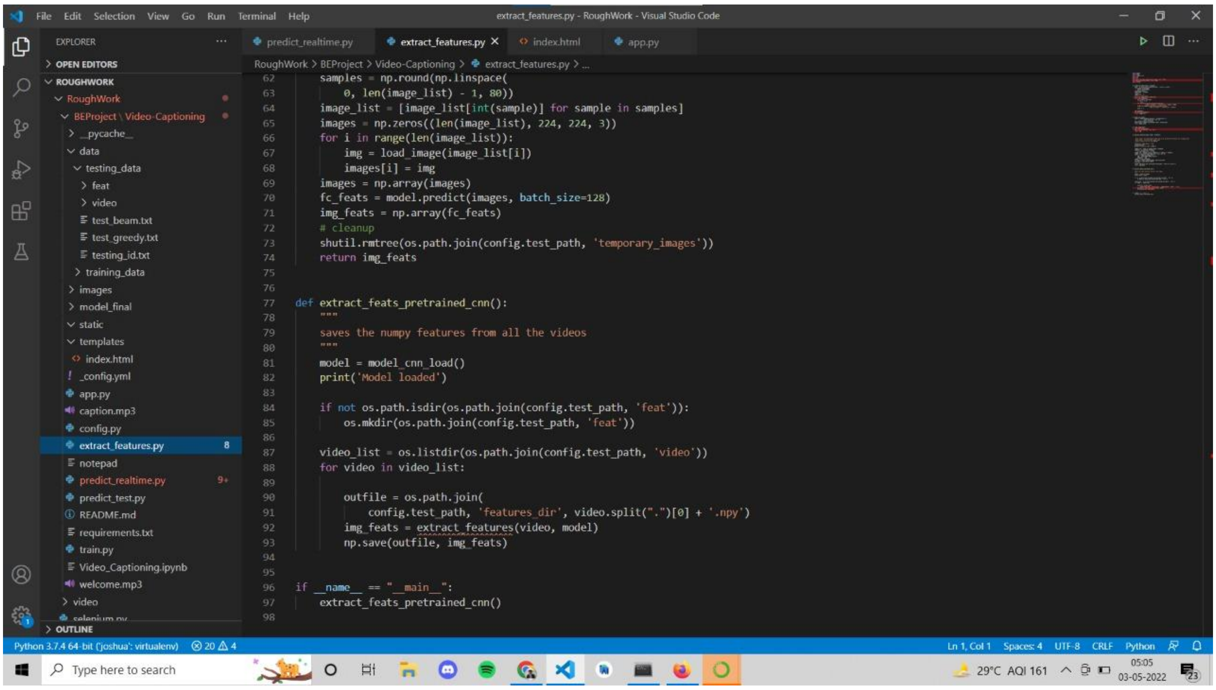

predict_realtime.py - RoughWork - Visual Studio Code
cv2.putText(imS, caption, (100, 270), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0),
2, cv2.LINE_4)
cv2.imshow("VIDEO CAPTIONING", imS)
if ret2:
imS = cv2.resize(frame, (480, 300))
cv2.imshow("ORIGINAL", imS)
else:
break
# Quit playing
key = cv2.waitKey(25)
if key == 27: # Button esc
break
# 3. Free resources
cap1.release()
cap2.release()
cv2.destroyAllWindows()
def get_results(filepath):
video_to_text = VideoDescriptionRealTime(config)
video_to_text.load_inference_models()
while True:
print('........................\nGenerating Caption:\n')
start = time.time()
video_caption, file = video_to_text.test(filepath)
end = time.time()
sentence = ''
# print(sentence)
for text in video_caption.split():
sentence = sentence + ' ' + text
"""print('\n........................\n')
print(sentence)
print('\n........................\n')"""
print(f'In predict_realtime file : {sentence}')
print('It took {:.2f} seconds to generate caption'.format(end-start))
return sentence
"""video_to_text.main(file, sentence)

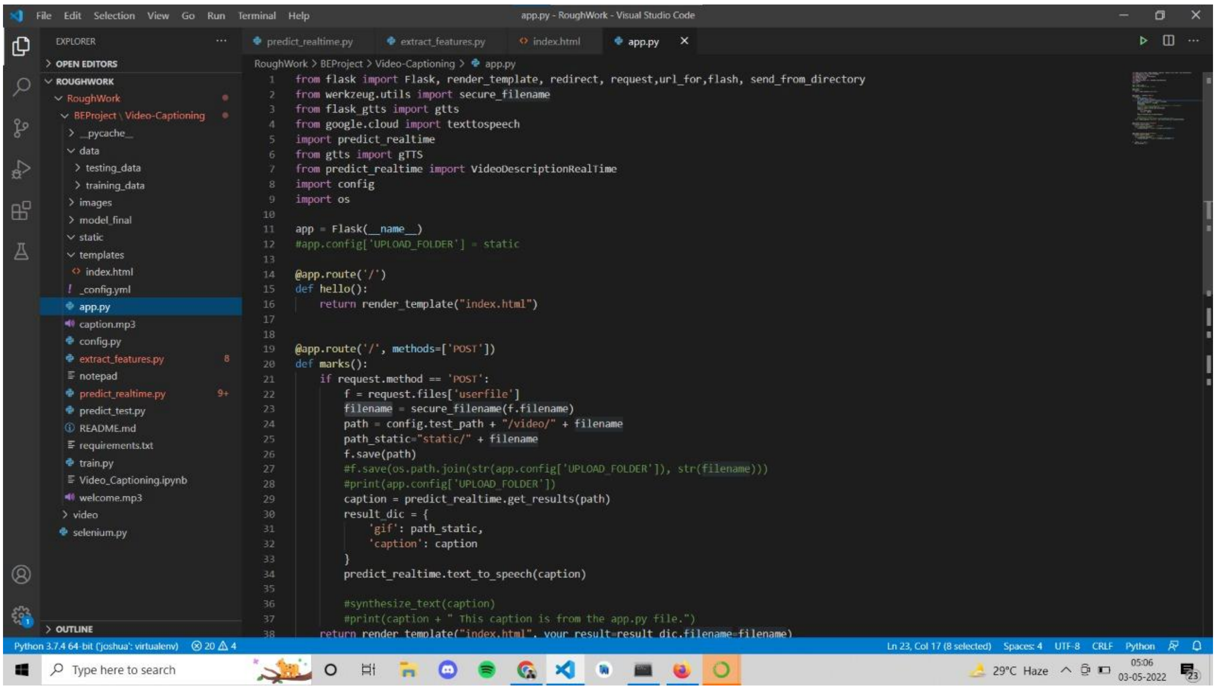

predict_realtime.py - RoughWork - Visual Studio Code
start = time.time()
video_caption, file = video_to_text.test(filepath)
end = time.time()
sentence = ''
# print(sentence)
for text in video_caption.split():
sentence = sentence + ' ' + text
"""print('\n........................\n')
print(sentence)
print('\n........................\n')"""
print(f'In predict_realtime file : {sentence}')
print('It took {:.2f} seconds to generate caption'.format(end-start))
return sentence
"""video_to_text.main(file, sentence)
play_video = input('Should I play the video? ')
if play_video.lower() == 'y':
continue
elif play_video.lower() == 'n':
break
else:
print('Could not understand type (y) for yes and (n) for no')
continue"""
def text_to_speech(text):
engine = pyttsx3.init()
# Setting up voice rate
engine.setProperty('rate', 125)
# Setting up volume level between 0 and 1
engine.setProperty('volume', 0.8)
engine.save_to_file(text,r'E:\RoughWork\RoughWork\BEProject\Video-Captioning\data\testing_data\video\caption.mp3')
engine.runAndWait()

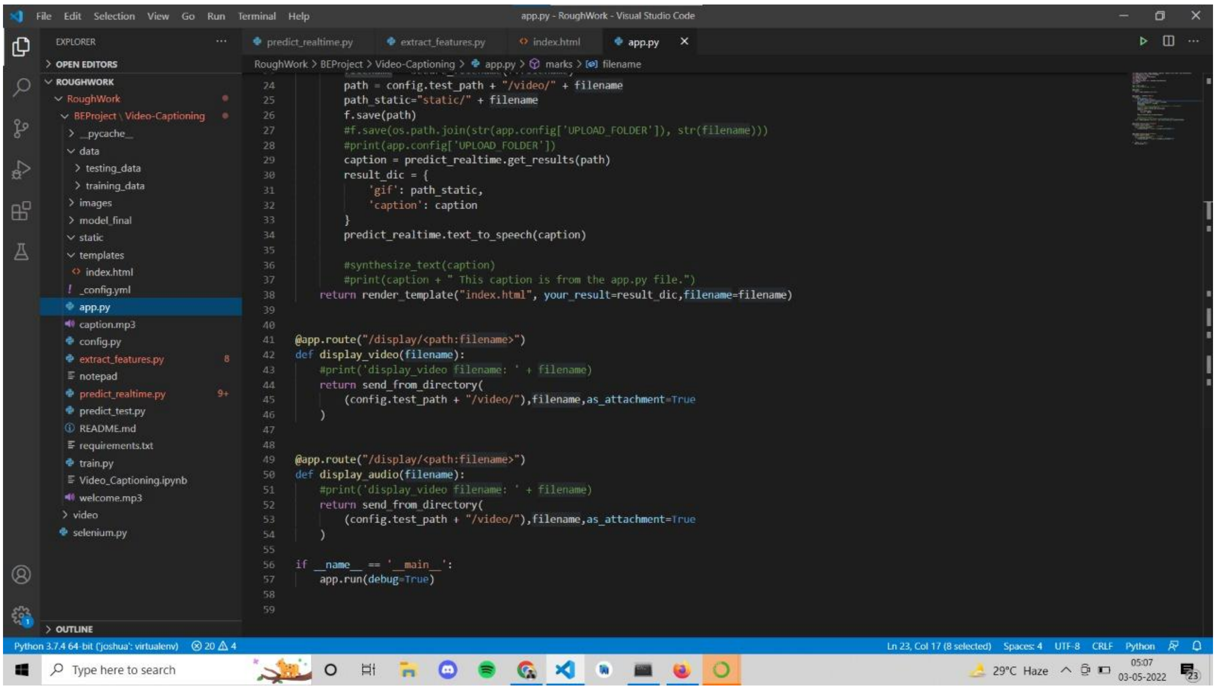

config.py - RoughWork - Visual Studio Code
train_path = "data/training_data"
test_path = "data/testing_data"
batch_size = 320
learning_rate = 0.0007
epochs = 150
latent_dim = 512
num_encoder_tokens = 4096
num_decoder_tokens = 1500
time_steps_encoder = 80
max_probability = -1
save_model_path = 'model_final'
validation_split = 0.15
max_length = 10
search_type = 'greedy'

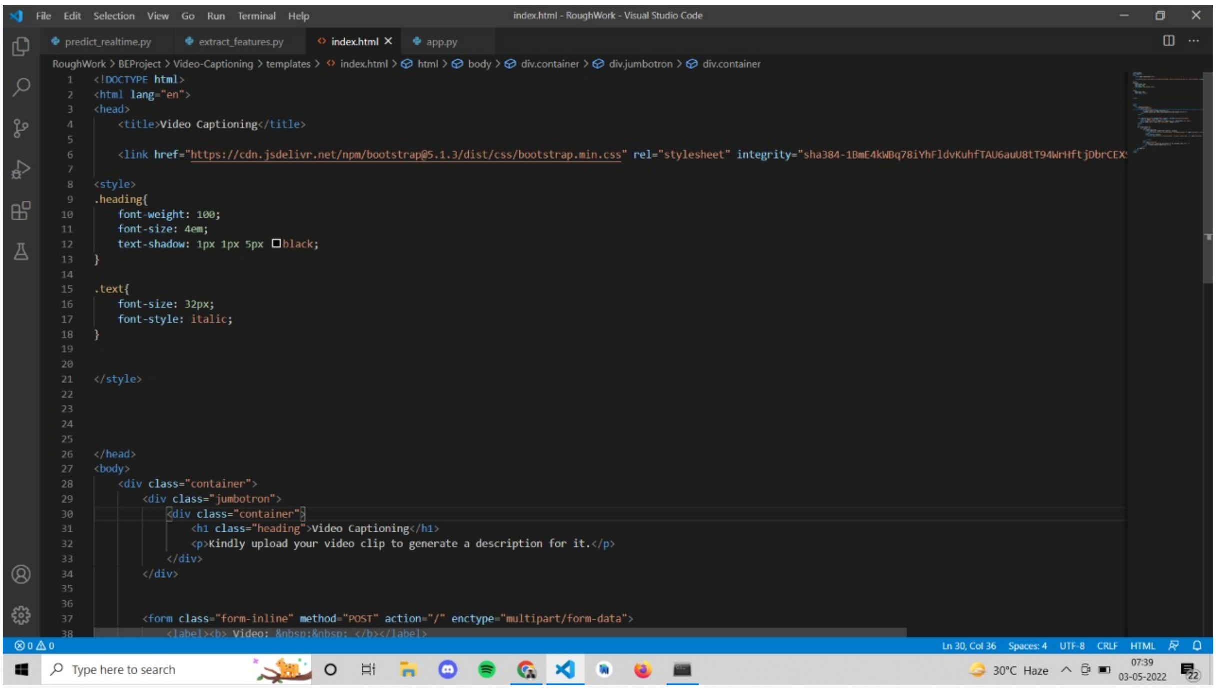

extract_features.py - RoughWork - Visual Studio Code
import shutil
import tqdm
import numpy as np
import cv2
import os
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Model
import config
def video_to_frames(video, filepath):
path = os.path.join(config.test_path, 'temporary_images')
if os.path.exists(path):
shutil.rmtree(path)
os.makedirs(path)
video_path = filepath
count = 0
image_list = []
# Path to video file
cap = cv2.VideoCapture(video_path)
while cap.isOpened():
ret, frame = cap.read()
if ret is False:
break
cv2.imwrite(os.path.join(config.test_path,
'temporary_images', 'frame%d.jpg' % count), frame)
image_list.append(os.path.join(config.test_path,
'temporary_images', 'frame%d.jpg' % count))
count += 1
cap.release()
cv2.destroyAllWindows()
return image_list
def model_cnn_load():
model = VGG16(weights="imagenet", include_top=True,

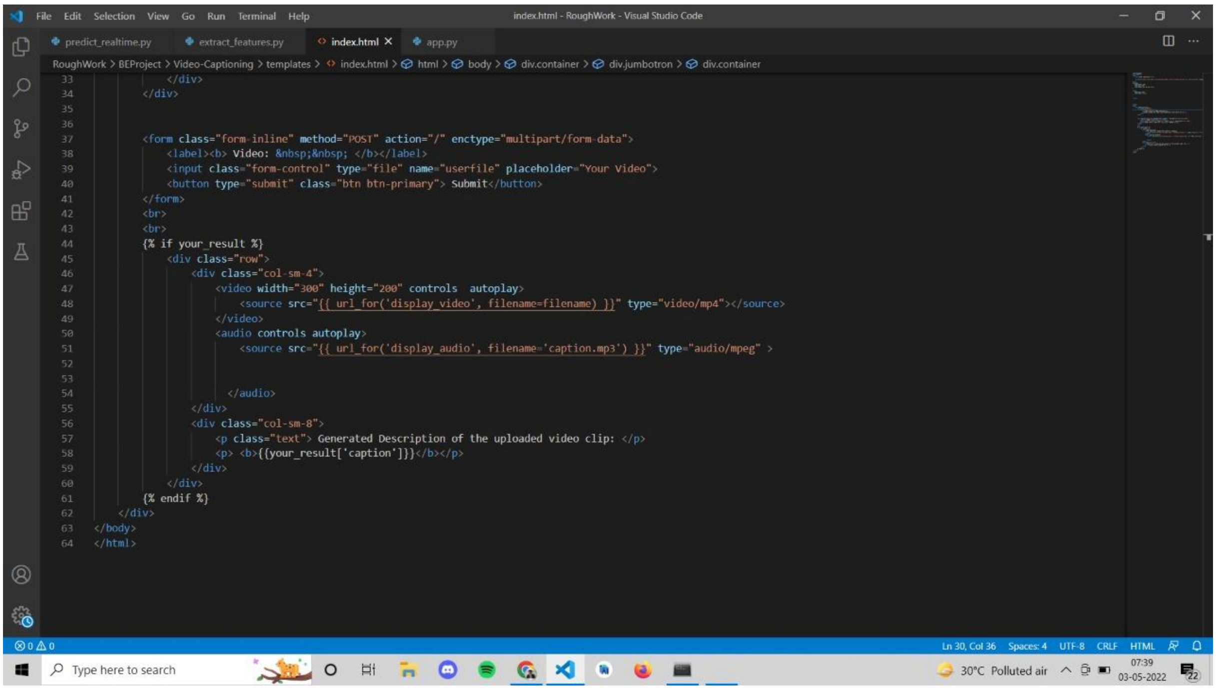

extract_features.py - RoughWork - Visual Studio Code
def model_cnn_load():
model = VGG16(weights="imagenet", include_top=True,
input_shape=(224, 224, 3))
out = model.layers[-2].output
model_final = Model(inputs=model.input, outputs=out)
return model_final
def load_image(path):
img = cv2.imread(path)
img = cv2.resize(img, (224, 224))
return img
def extract_features(video, model, filepath):
"""
:param video: The video whose frames are to be extracted to convert into a numpy array
:param model: the pretrained vgg16 model
:return: numpy array of size 4096x80
"""
video_id = video.split(".")[0]
print(video_id)
print(f'Processing video {video}')
image_list = video_to_frames(video, filepath)
samples = np.round(np.linspace(
0, len(image_list) - 1, 80))
image_list = [image_list[int(sample)] for sample in samples]
images = np.zeros((len(image_list), 224, 224, 3))
for i in range(len(image_list)):
img = load_image(image_list[i])
images[i] = img
images = np.array(images)
fc_feats = model.predict(images, batch_size=128)
img_feats = np.array(fc_feats)


```python
    samples = np.round(np.linspace(
        0, len(image_list) - 1, 80))
    image_list = [image_list[int(sample)] for sample in samples]
    images = np.zeros((len(image_list), 224, 224, 3))
    for i in range(len(image_list)):
        img = load_image(image_list[i])
        images[i] = img
    images = np.array(images)
    fc_feats = model.predict(images, batch_size=128)
    img_feats = np.array(fc_feats)
    # cleanup
    shutil.rmtree(os.path.join(config.test_path, 'temporary_images'))
    return img_feats


def extract_feats_pretrained_cnn():
    """
    saves the numpy features from all the videos
    """
    model = model_cnn_load()
    print('Model loaded')

    if not os.path.isdir(os.path.join(config.test_path, 'feat')):
        os.mkdir(os.path.join(config.test_path, 'feat'))

    video_list = os.listdir(os.path.join(config.test_path, 'video'))
    for video in video_list:

        outfile = os.path.join(
            config.test_path, 'features_dir', video.split(".")[0] + '.npy')
        img_feats = extract_features(video, model)
        np.save(outfile, img_feats)


if __name__ == "__main__":
    extract_feats_pretrained_cnn()
```

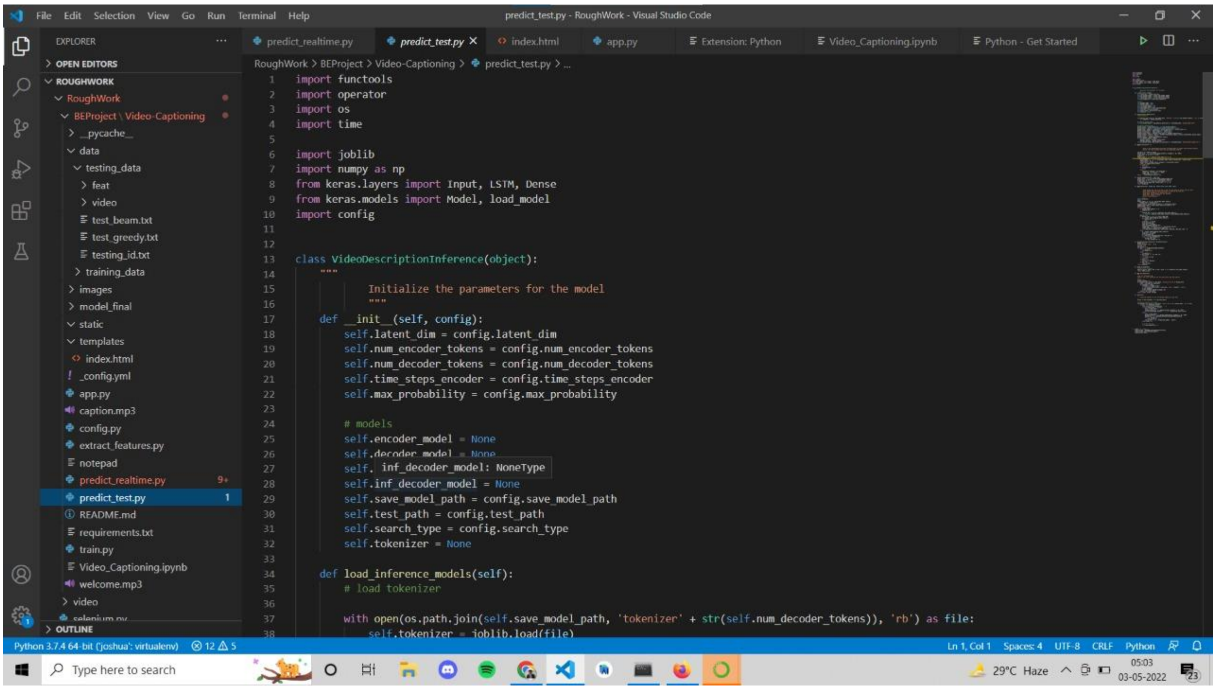

```python
from flask import Flask, render_template, redirect, request,url_for,flash, send_from_directory
from werkzeug.utils import secure_filename
from flask_gtts import gtts
from google.cloud import texttospeech
import predict_realtime
from gtts import gTTS
from predict_realtime import VideoDescriptionRealTime
import config
import os

app = Flask(__name__)
#app.config['UPLOAD_FOLDER'] = static

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


@app.route('/', methods=['POST'])
def marks():
    if request.method == 'POST':
        f = request.files['userfile']
        filename = secure_filename(f.filename)
        path = config.test_path + "/video/" + filename
        path_static="static/" + filename
        f.save(path)
        #f.save(os.path.join(str(app.config['UPLOAD_FOLDER']), str(filename)))
        #print(app.config['UPLOAD_FOLDER'])
        caption = predict_realtime.get_results(path)
        result_dic = {
            'gif': path_static,
            'caption': caption
        }
        predict_realtime.text_to_speech(caption)

        #synthesize_text(caption)
        #print(caption + " This caption is from the app.py file.")
    return render_template("index.html", your_result=result_dic,filename=filename)
```

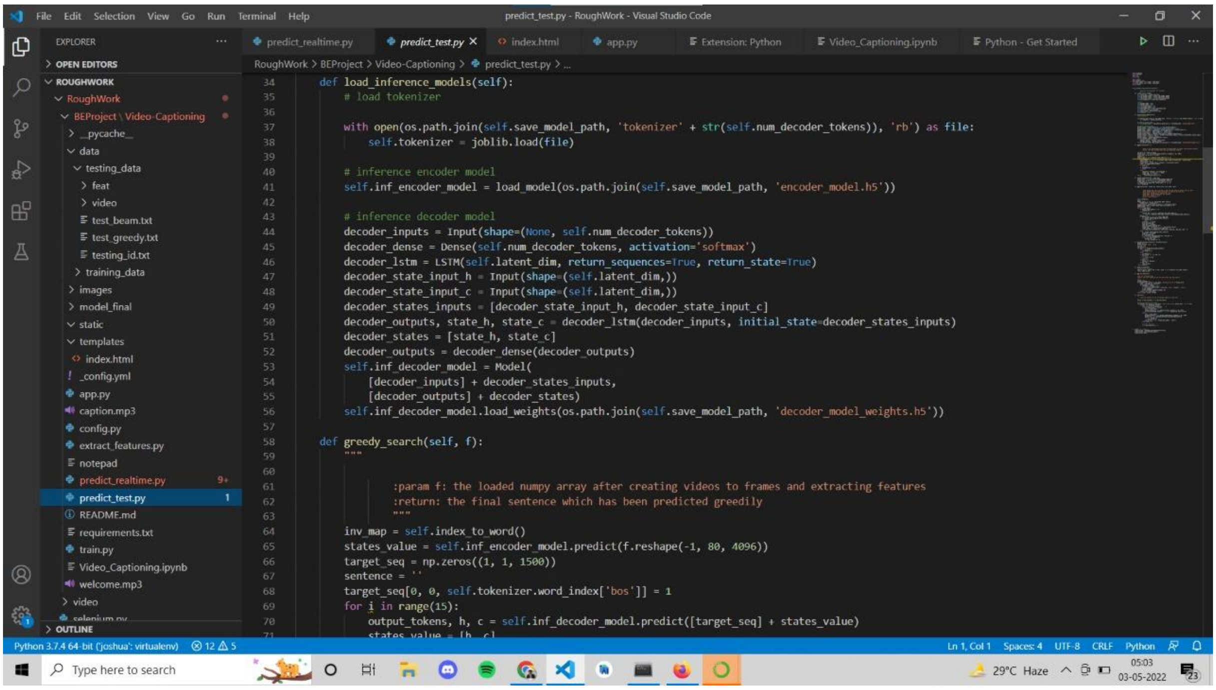

app.py - RoughWork - Visual Studio Code
predict_realtime.py
extract_features.py
index.html
app.py
RoughWork > BEProject > Video-Captioning > app.py > marks > filename
path = config.test_path + "/video/" + filename
path_static="static/" + filename
f.save(path)
#f.save(os.path.join(str(app.config['UPLOAD_FOLDER']), str(filename)))
#print(app.config['UPLOAD_FOLDER'])
caption = predict_realtime.get_results(path)
result_dic = {
'gif': path_static,
'caption': caption
}
predict_realtime.text_to_speech(caption)
#synthesize_text(caption)
#print(caption + " This caption is from the app.py file.")
return render_template("index.html", your_result=result_dic,filename=filename)
@app.route("/display/<path:filename>")
def display_video(filename):
#print('display_video filename: ' + filename)
return send_from_directory(
(config.test_path + "/video/"),filename,as_attachment=True
)
@app.route("/display/<path:filename>")
def display_audio(filename):
#print('display_video filename: ' + filename)
return send_from_directory(
(config.test_path + "/video/"),filename,as_attachment=True
)
if __name__ == '__main__':
app.run(debug=True)


index.html - RoughWork - Visual Studio Code
RoughWork > BEProject > Video-Captioning > templates > index.html > html > body > div.container > div.jumbotron > div.container
<!DOCTYPE html>
<html lang="en">
<head>
<title>Video Captioning</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEX
<style>
.heading{
font-weight: 100;
font-size: 4em;
text-shadow: 1px 1px 5px black;
}
.text{
font-size: 32px;
font-style: italic;
}
</style>
</head>
<body>
<div class="container">
<div class="jumbotron">
<div class="container">
<h1 class="heading">Video Captioning</h1>
<p>Kindly upload your video clip to generate a description for it.</p>
</div>
</div>
<form class="form-inline" method="POST" action="/" enctype="multipart/form-data">


index.html - RoughWork - Visual Studio Code
predict_realtime.py
extract_features.py
index.html
app.py
RoughWork > BEProject > Video-Captioning > templates > index.html > html > body > div.container > div.jumbotron > div.container
</div>
</div>
<form class="form-inline" method="POST" action="/" enctype="multipart/form-data">
<label> Video: </label>
<input class="form-control" type="file" name="userfile" placeholder="Your Video">
<button type="submit" class="btn btn-primary"> Submit</button>
</form>

{% if your_result %}
<div class="row">
<div class="col-sm-4">
<video width="300" height="200" controls autoplay>
<source src="{{ url_for('display_video', filename=filename) }}" type="video/mp4"></source>
</video>
<audio controls autoplay>
<source src="{{ url_for('display_audio', filename='caption.mp3') }}" type="audio/mpeg" >
</audio>
</div>
<div class="col-sm-8">
<p class="text"> Generated Description of the uploaded video clip: </p>
<p> {{your_result['caption']}}</p>
</div>
</div>
{% endif %}
</div>
</body>
</html>
Ln 30, Col 36 Spaces: 4 UTF-8 CRLF HTML
Type here to search
30°C Polluted air
07:39
03-05-2022


predict_test.py - RoughWork - Visual Studio Code
EXPLORER
OPEN EDITORS
ROUGHWORK
RoughWork
BEProject \ Video-Captioning
pycache
data
testing_data
feat
video
test_beam.txt
test_greedy.txt
testing_id.txt
training_data
images
model_final
static
templates
index.html
_config.yml
app.py
caption.mp3
config.py
extract_features.py
notepad
predict_realtime.py
predict_test.py
README.md
requirements.txt
train.py
Video_Captioning.ipynb
welcome.mp3
video
OUTLINE
predict_realtime.py
predict_test.py
index.html
app.py
Extension: Python
Video_Captioning.ipynb
Python - Get Started
RoughWork > BEProject > Video-Captioning > predict_test.py > ...
import functools
import operator
import os
import time
import joblib
import numpy as np
from keras.layers import Input, LSTM, Dense
from keras.models import Model, load_model
import config
class VideoDescriptionInference(object):
"""
Initialize the parameters for the model
"""
def __init__(self, config):
self.latent_dim = config.latent_dim
self.num_encoder_tokens = config.num_encoder_tokens
self.num_decoder_tokens = config.num_decoder_tokens
self.time_steps_encoder = config.time_steps_encoder
self.max_probability = config.max_probability
# models
self.encoder_model = None
self.decoder_model = None
inf_decoder_model: NoneType
self.inf_decoder_model = None
self.save_model_path = config.save_model_path
self.test_path = config.test_path
self.search_type = config.search_type
self.tokenizer = None
def load_inference_models(self):
# load tokenizer
with open(os.path.join(self.save_model_path, 'tokenizer' + str(self.num_decoder_tokens)), 'rb') as file:
self.tokenizer = joblib.load(file)
Python 3.7.4 64-bit ('joshua': virtualenv)
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Python
Type here to search
29°C Haze
05:03
03-05-2022



predict_test.py - RoughWork - Visual Studio Code
def load_inference_models(self):
    # load tokenizer

    with open(os.path.join(self.save_model_path, 'tokenizer' + str(self.num_decoder_tokens)), 'rb') as file:
        self.tokenizer = joblib.load(file)

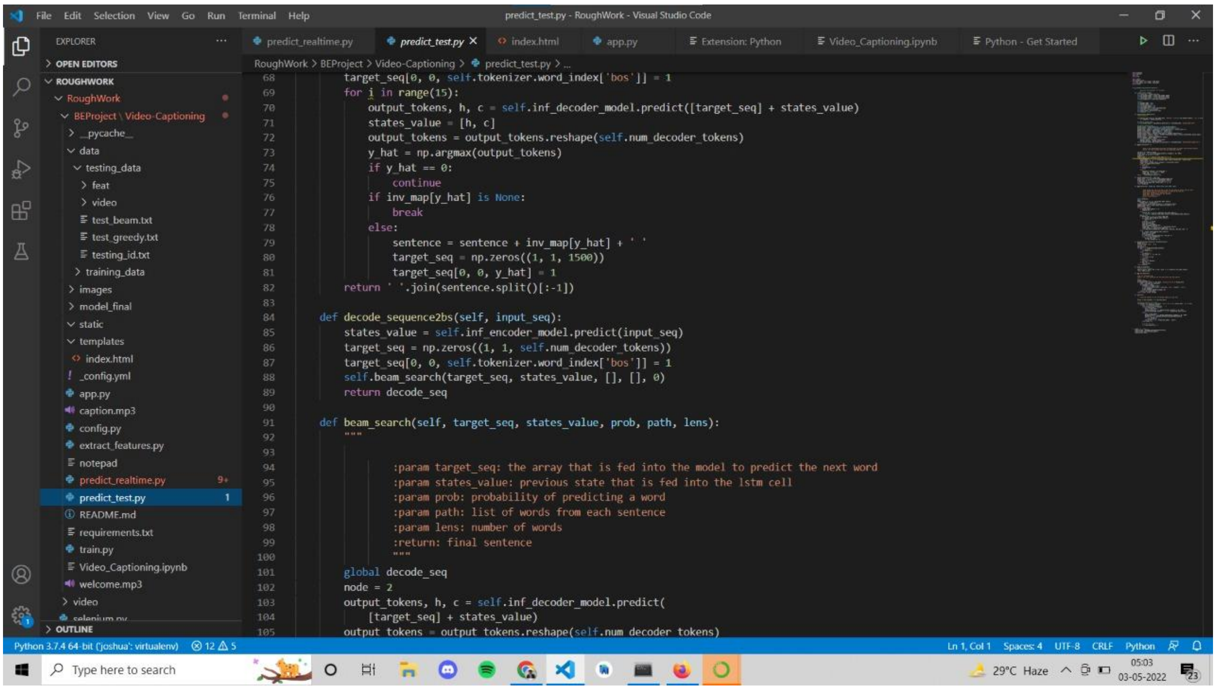
    # inference encoder model
    self.inf_encoder_model = load_model(os.path.join(self.save_model_path, 'encoder_model.h5'))

    # inference decoder model
    decoder_inputs = Input(shape=(None, self.num_decoder_tokens))
    decoder_dense = Dense(self.num_decoder_tokens, activation='softmax')
    decoder_lstm = LSTM(self.latent_dim, return_sequences=True, return_state=True)
    decoder_state_input_h = Input(shape=(self.latent_dim,))
    decoder_state_input_c = Input(shape=(self.latent_dim,))
    decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
    decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs, initial_state=decoder_states_inputs)
    decoder_states = [state_h, state_c]
    decoder_outputs = decoder_dense(decoder_outputs)
    self.inf_decoder_model = Model(
        [decoder_inputs] + decoder_states_inputs,
        [decoder_outputs] + decoder_states)
    self.inf_decoder_model.load_weights(os.path.join(self.save_model_path, 'decoder_model_weights.h5'))

def greedy_search(self, f):
    """

        :param f: the loaded numpy array after creating videos to frames and extracting features
        :return: the final sentence which has been predicted greedily
        """

    inv_map = self.index_to_word()
    states_value = self.inf_encoder_model.predict(f.reshape(-1, 80, 4096))
    target_seq = np.zeros((1, 1, 1500))
    sentence = ''
    target_seq[0, 0, self.tokenizer.word_index['bos']] = 1
    for i in range(15):
        output_tokens, h, c = self.inf_decoder_model.predict([target_seq] + states_value)

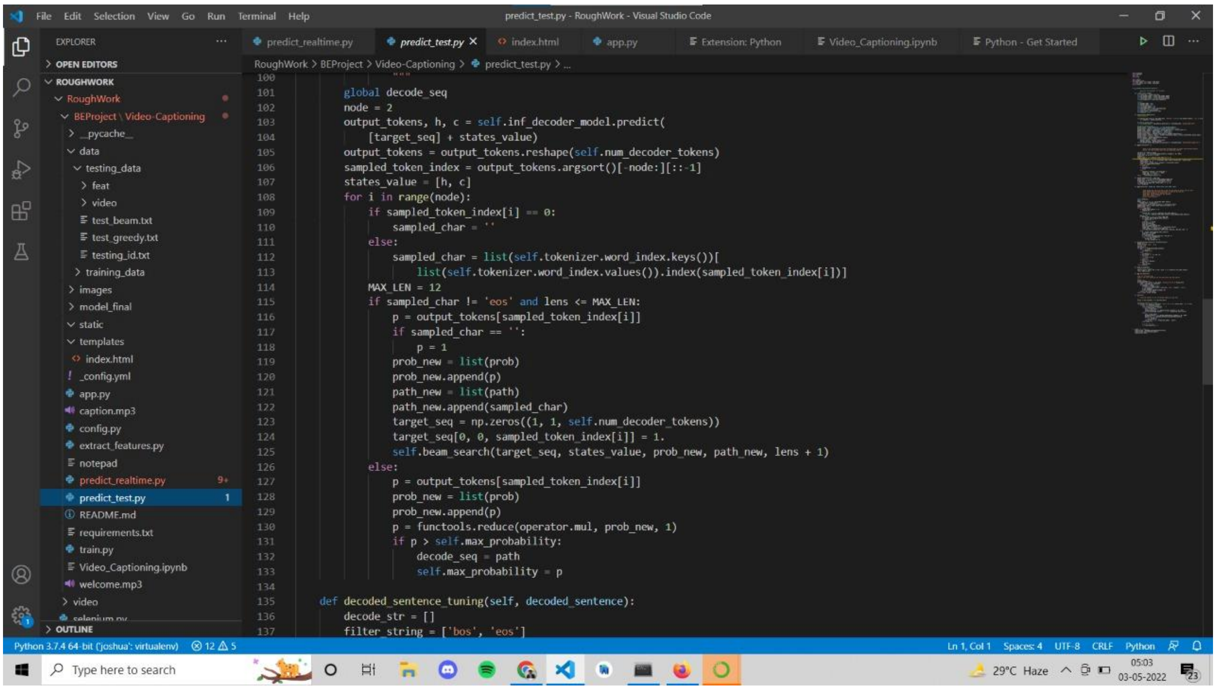


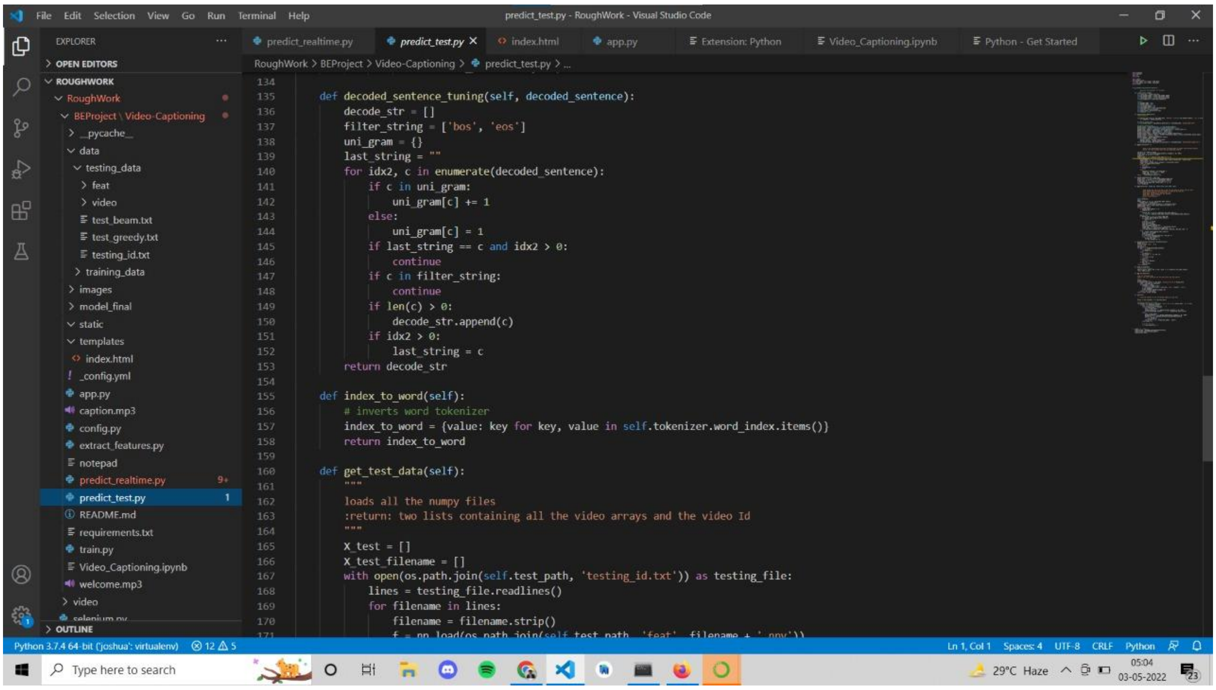
predict_test.py - RoughWork - Visual Studio Code
    target_seq[0, 0, self.tokenizer.word_index['bos']] = 1
    for i in range(15):
        output_tokens, h, c = self.inf_decoder_model.predict([target_seq] + states_value)
        states_value = [h, c]
        output_tokens = output_tokens.reshape(self.num_decoder_tokens)
        y_hat = np.argmax(output_tokens)
        if y_hat == 0:
            continue
        if inv_map[y_hat] is None:
            break
        else:
            sentence = sentence + inv_map[y_hat] + ' '
            target_seq = np.zeros((1, 1, 1500))
            target_seq[0, 0, y_hat] = 1
    return ' '.join(sentence.split()[:-1])

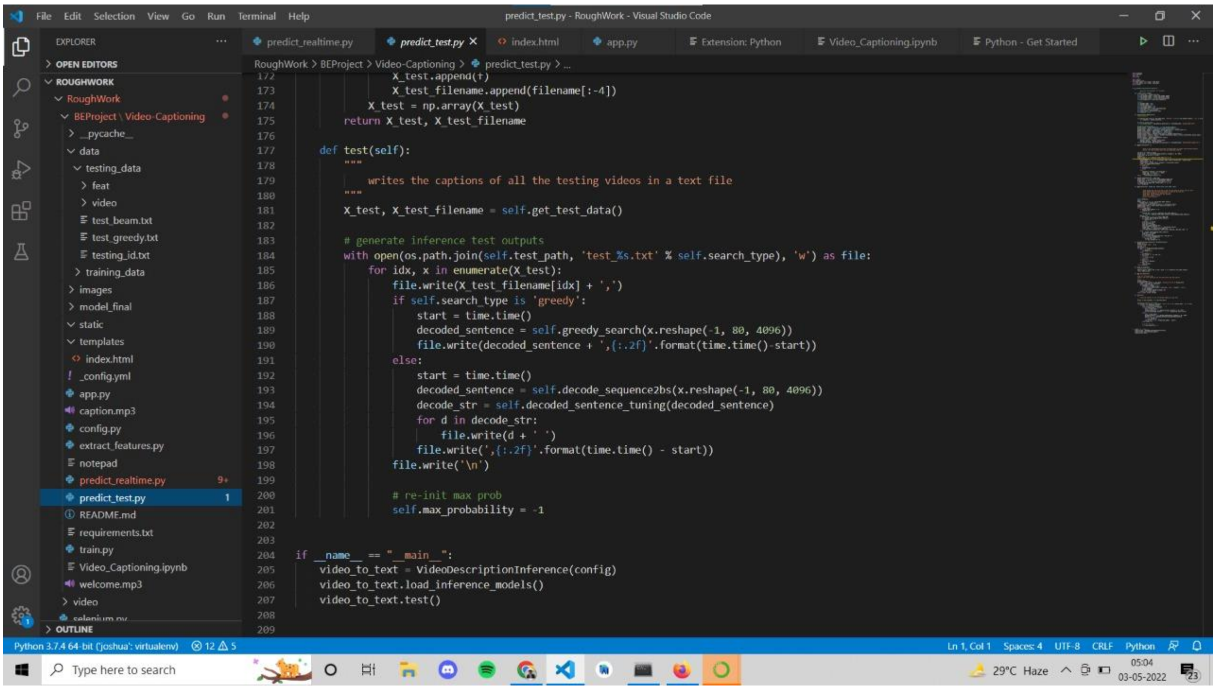
def decode_sequence2bs(self, input_seq):
    states_value = self.inf_encoder_model.predict(input_seq)
    target_seq = np.zeros((1, 1, self.num_decoder_tokens))
    target_seq[0, 0, self.tokenizer.word_index['bos']] = 1
    self.beam_search(target_seq, states_value, [], [], 0)
    return decode_seq

def beam_search(self, target_seq, states_value, prob, path, lens):
    """

        :param target_seq: the array that is fed into the model to predict the next word
        :param states_value: previous state that is fed into the lstm cell
        :param prob: probability of predicting a word
        :param path: list of words from each sentence
        :param lens: number of words
        :return: final sentence
        """
    global decode_seq
    node = 2
    output_tokens, h, c = self.inf_decoder_model.predict(
        [target_seq] + states_value)
    output_tokens = output_tokens.reshape(self.num_decoder_tokens)


predict_test.py - RoughWork - Visual Studio Code
RoughWork > BEProject > Video-Captioning > predict_test.py > ...
global decode_seq
node = 2
output_tokens, h, c = self.inf_decoder_model.predict(
[target_seq] + states_value)
output_tokens = output_tokens.reshape(self.num_decoder_tokens)
sampled_token_index = output_tokens.argsort()[-node:][::-1]
states_value = [h, c]
for i in range(node):
if sampled_token_index[i] == 0:
sampled_char = ''
else:
sampled_char = list(self.tokenizer.word_index.keys())[
list(self.tokenizer.word_index.values()).index(sampled_token_index[i])]
MAX_LEN = 12
if sampled_char != 'eos' and lens <= MAX_LEN:
p = output_tokens[sampled_token_index[i]]
if sampled_char == '':
p = 1
prob_new = list(prob)
prob_new.append(p)
path_new = list(path)
path_new.append(sampled_char)
target_seq = np.zeros((1, 1, self.num_decoder_tokens))
target_seq[0, 0, sampled_token_index[i]] = 1.
self.beam_search(target_seq, states_value, prob_new, path_new, lens + 1)
else:
p = output_tokens[sampled_token_index[i]]
prob_new = list(prob)
prob_new.append(p)
p = functools.reduce(operator.mul, prob_new, 1)
if p > self.max_probability:
decode_seq = path
self.max_probability = p
def decoded_sentence_tuning(self, decoded_sentence):
decode_str = []
filter_string = ['bos', 'eos']


predict_test.py - RoughWork - Visual Studio Code
RoughWork > BEProject > Video-Captioning > predict_test.py > ...
def decoded_sentence_tuning(self, decoded_sentence):
decode_str = []
filter_string = ['bos', 'eos']
uni_gram = {}
last_string = ""
for idx2, c in enumerate(decoded_sentence):
if c in uni_gram:
uni_gram[c] += 1
else:
uni_gram[c] = 1
if last_string == c and idx2 > 0:
continue
if c in filter_string:
continue
if len(c) > 0:
decode_str.append(c)
if idx2 > 0:
last_string = c
return decode_str
def index_to_word(self):
# inverts word tokenizer
index_to_word = {value: key for key, value in self.tokenizer.word_index.items()}
return index_to_word
def get_test_data(self):
"""
loads all the numpy files
:return: two lists containing all the video arrays and the video Id
"""
X_test = []
X_test_filename = []
with open(os.path.join(self.test_path, 'testing_id.txt')) as testing_file:
lines = testing_file.readlines()
for filename in lines:
filename = filename.strip()



Chapter 8

References

[1] H. Aradhye, G. Toderici, and J. Yagnik. Video2text: Learning to annotate video content. In ICDMW, 2009.

[2] H. Huang, Y. Lu, F. Zhang, and S. Sun. A multi-modal clustering method for web videos. In ISCTCS. 2013.

[3] S. Guadarrama, N. Krishnamoorthy, G. Malkarnenkar, S. Venugopalan, R. Mooney, T. Darrell, and K. Saenko. Youtube2text: Recognizing and describing arbitrary activities using semantic hierarchies and zero-shoot recognition. In ICCV, 2013.

[4] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In CVPR, 2015.

[5] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In NIPS, 2014.

[6] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko. Translating videos to natural language using deep recurrent neural networks. In NAACL, 2015.

[7] L. Yao, A. Torabi, K. Cho, N. Ballas, C. Pal, H. Larochelle, and A. Courville. Describing videos by exploiting temporal structure. arXiv:1502.08029v4, 2015.

[8] H. Wang and C. Schmid. Action recognition with improved trajectories. In ICCV, pages 3551–3558. IEEE, 2013.

[9] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using LSTMs. ICML, 2015.

[10] P. Kuznetsova, V. Ordonez, T. L. Berg, U. C. Hill, and Y. Choi. Treetalk: Composition and compression of trees for image descriptions. In TACL, 2014.

[11] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. CVPR, 2015.

[12] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft coco: Com- ´ mon objects in context. In ECCV, 2014.

[13] M. Rohrbach, W. Qiu, I. Titov, S. Thater, M. Pinkal, and B. Schiele. Translating video content to natural language descriptions. In ICCV, 2013.

[14] D. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[15] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In ICML, 2014.

[16] A. Torabi, C. Pal, H. Larochelle, and A. Courville. Using descriptive video services to create a large data source for video annotation research. arXiv:1503.01070v1, 2015.

[17] P. Over, G. Awad, M. Michel, J. Fiscus, G. Sanders, B. Shaw, A. F. Smeaton, and G. Queenot. TRECVID 2012 – an ´ overview of the goals, tasks, data, evaluation mechanisms and metrics. In Proceedings of TRECVID 2012, 2012.