**CS6301 MACHINE LEARNING – MINI PROJECT**

SRIHARI. S T.K.S. ARUNACHALAM

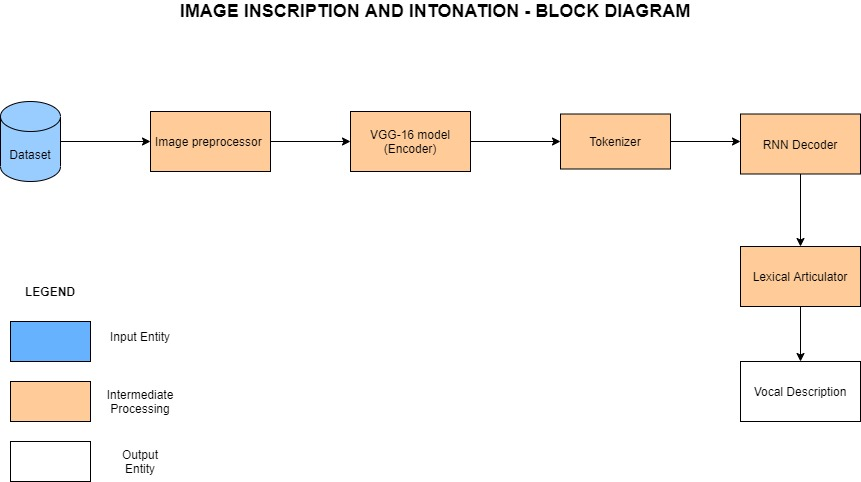
2018103601 2018103616

**IMAGE INSCRIPTION AND INTONATION - A NEURAL NETWORK APPROACH**

**100% IMPLEMENTATION UPDATE DOCUMENTATION**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DATASET USED:** MS-COCO Dataset (<https://cocodataset.org/#download>)



|  |  |
| --- | --- |
| **IMPLEMENTATION PROGRESS** | |
| **Completed Modules** | Image Pre-processor  VGG-16 Model (Encoder)  Tokenizer  RNN Decoder  Lexical Articulator |

|  |  |
| --- | --- |
| **INDIVIDUAL CONTRIBUTION** | |
| **SRIHARI.S** | **ARUNACHALAM.T.K.S.** |
| Caption Pre-processing | Image Pre-processing |
| Transfer Learning on the dataset using VGG16 | Incorporating PyCoCo Tools for dealing with the data |
| Vocabulary construction | Embeddings productions |
| Fabrication of the data generator for training the model | Formulate the Recurrent Network |
| Train the model following Algorithm-1 for 325 epochs (10 hours/day) | Train the model following Algorithm-2 for 325 epochs (10 hours/day) |
| Formulation of the Lexical Articulator | Measurement of performance using 5 metrics |

**MODULE WISE TEST CASES**

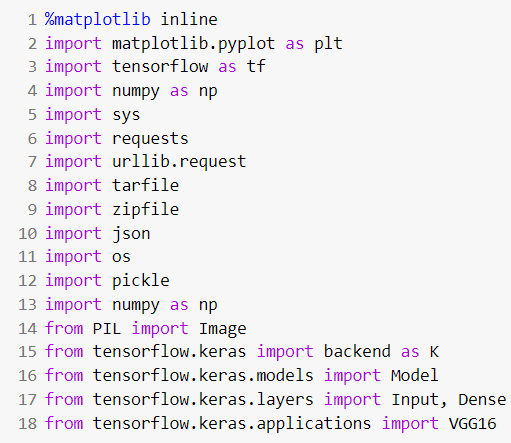
|  |  |  |
| --- | --- | --- |
| **MODULE 1: PRE-PROCESSOR** | | |
| **INPUT** | **SEPARATED IMAGE** | **SEPARATED CAPTIONS** |
| {"license": 5,"file\_name": "COCO\_train2014\_000000057870.jpg","coco\_url": "http://images.cocodataset.org/train2014/COCO\_train2014\_000000000025.jpg","height": 480,"width": 640,"date\_captured": "2013-11-14 16:28:13","id": 57870} |  | ['A giraffe eating food from the top of the tree.', 'A giraffe standing up nearby a tree ', 'A giraffe mother with its baby in the forest.', 'Two giraffes standing in a tree filled area.', 'A giraffe standing next to a forest filled with trees.'] |
| {"license": 5,"file\_name": "COCO\_train2014\_000000384029.jpg","coco\_url": "http://images.cocodataset.org/val2014/COCO\_val2014\_000000003014.jpg","height": 429,"width": 640,"date\_captured": "2013-11-14 16:29:45","id": 384029} |  | ['Three-quarters of a meat-lovers pizza with mushrooms with drink', 'A partially eaten pizza is sitting beside a soda.', 'A table with a partially eaten pizza and blue canned beverage on it.', 'A close up of a pizza and a drink on a table.', 'A pizza with cherry tomatoes has a piece taken out.'] |
| {"license": 1,"file\_name": "COCO\_train2014\_000000222016.jpg","coco\_url": "http://images.cocodataset.org/train2014/COCO\_train2014\_000000000404.jpg","height": 640,"width": 480,"date\_captured": "2013-11-14 16:37:59”,"id": 222016} |  | ['a couple of boats that are in some water', 'A pair of boats docked at a pier is shown.', 'Three boats docked in still water with clouds in the sky. ', 'Three boats are docked together on the cloudy day.', 'some colorful boats sitting next to a dock '] |
| {"license": 3,"file\_name": "COCO\_val2014\_000000391895.jpg","coco\_url": "http://images.cocodataset.org/train2014/COCO\_train2014\_000000000089.jpg","height": 360,"width": 640,"date\_captured": "2013-11-14 11:18:45","id": 391895} |  | ['An oven with a stove on top of it in a kitchen.', 'A stove with a lighted hood in the kitchen.', 'A small light is on above the polished stove top.', 'Smooth top stove with exhaust fan that has light turned on.', 'A stove top is cleaned with a set of knives on the wall.'] |
| {"license": 4,"file\_name": "COCO\_val2014\_000000522418.jpg","coco\_url": "http://images.cocodataset.org/val2014/COCO\_val2014\_000000000073.jpg","height": 480,"width": 640,"date\_captured": "2013-11-14 11:38:44","id": 522418} |  | ['A motorcycle parked in a parking space next to another motorcycle.', 'An old motorcycle parked beside other motorcycles with a brown leather seat.', 'Motorcycle parked in the parking lot of asphalt.', 'A close up view of a motorized bicycle, sitting in a rack. ', 'The back tire of an old style motorcycle is resting in a metal stand. '] |

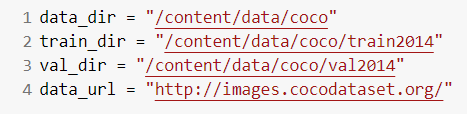
|  |  |
| --- | --- |
| **MODULE 2: ENCODER** | |
| **INPUT** | **OUTPUT – TRANSFER VALUES** |
|  |  |
|  |  |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **MODULE 3: TOKENIZER** | | | |
| **INPUT** | **INTERMEDIATE OUTPUT** | **OUTPUT SEQUENCE** | **EMBEDDINGS** |
| 'Closeup of bins of food that include broccoli and bread.' | 'ssss Closeup of bins of food that include broccoli and bread. eeee' | [2, 844, 5, 2845, 5, 60, 25, 1933, 248, 9, 438, 3] | [[0.41177 , -2.223 , -1.0756 , -1.0783 ,]...[0.15155 , 0.78321 , -0.91241]] |
| 'A giraffe eating food from the top of the tree.' | 'ssss A giraffe eating food from the top of the tree. eeee' | [2, 1, 117, 108, 60, 96, 6, 32, 5, 6, 133, 3] | [[ 0.50451 , 0.68607 , -0.59517,]...[0.60046,-0.13498,.. -0.022801]] |
| 'White vase with different colored flowers sitting inside of it. ' | 'ssss White vase with different colored flowers sitting inside of it. eeee' | [2, 21, 202, 8, 191, 395, 200, 13, 159, 5, 30, 3] | [[ 0.32157 , 0.45894 , -0.32014,]...[0.32015,-0.23156,.. -0.30215]] |
| ‘A cat is lying on its back in a man's lap’ | ssss A cat is lying on its back in a man's lap. eeee | [2, 1, 50, 10, 370, 4, 154, 163, 7, 1, 1248, 584, 3] | [[ 0.09981 , 0.45477 , -0.87890,]...[0.12344,-0.98757,.. -0.00854]] |
| ‘a giraffe standing in tall grass with trees in the background .’ | ssss a giraffe standing in tall grass with trees in the background eeee | [2, 6, 117, 10, 16, 683, 7, 6, 1613, 3] | [[ 0.54687 , 0.63547 , -0.77894,]...[0.30210,-0.65489,.. -0.32154]] |

|  |  |  |
| --- | --- | --- |
| **MODULE 4: DECODER** | | |
| **IMAGE** | **GENERATED CAPTION** | |
| **ALGORITHM - 1**  **3 GRU** | **ALGORITHM - 2**  **2 GRU + 2 LSTM + 1 GRU** |
|  | a plate of food with broccoli and other foods | a close up of a piece of broccoli on a plate |
|  | a living room with a television and a tv | a living room with a couch table and a tv set |
|  | a woman is holding a pink umbrella in a crowd | a woman in a floral swimsuit holds a pink umbrella |
|  | a close up of a sandwich with a slice of cheese | a slice of chocolate cake on a plate with a fork |
|  | a man riding a motorcycle on a city street | a motorcycle parked on the side of a road |

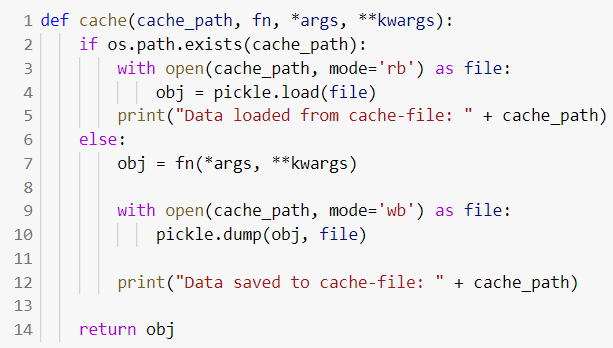
|  |  |  |
| --- | --- | --- |
| **MODULE 5: LEXICAL ARTICULATOR** | | |
| **IMAGE** | **GENERATED CAPTION FROM DECODER** | **ARTICULATED CAPTION** |
|  | a close up of a piece of broccoli on a plate | [Articulated-Caption](https://drive.google.com/file/d/1-INQDGwM4vw3CC-PnasZVp-qbV87j2TE/view?usp=sharing) |
|  | a living room with a couch table and a tv set | [Articulated-Caption](https://drive.google.com/file/d/1-ISFa8sUMeVlsrAsnyRbAVwz-WEqxXGD/view?usp=sharing) |
|  | a woman in a floral swimsuit holds a pink umbrella | [Articulated-Caption](https://drive.google.com/file/d/1-GJ1fRwBpnVOcw0QQxW-ZYcNcJc1NRdl/view?usp=sharing) |
|  | a slice of chocolate cake on a plate with a fork | [Articulated-Caption](https://drive.google.com/file/d/1-1QgRlE8YyEXNd6wm2fcWPVZLsul5sXN/view?usp=sharing) |
|  | a motorcycle parked on the side of a road | [Articulated-Caption](https://drive.google.com/file/d/1-MSUHKwdJ73S1unVSemfIWQN4oirrYim/view?usp=sharing) |

**MODULE - 1 PRE-PROCESSOR**

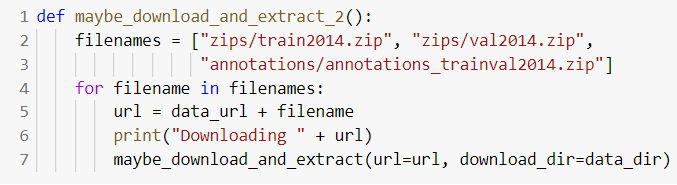


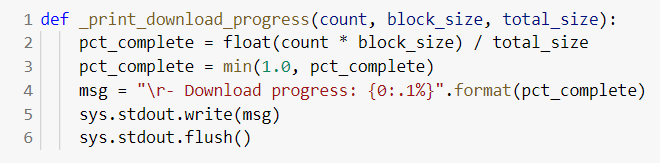
An iterative approach has been chosen for the implementation of the problem statement. The first goal was to understand the nature of the dataset and pre-process it. The input MS-COCO 2014 dataset is of size 25 GB. In-order to deal with this huge dataset and the constrained computing resources we make use of the dynamic programming paradigm by caching the values, the first time the dataset is downloaded, in-order to make access faster the subsequent times. The dataset consisting of both images and captions together is loaded. The images and the corresponding captions are then segregated and stored separately. The images then undergo normalization followed by scaling to finish the pre-processing. On the other hand, the captions are encoded in a dictionary and are thus pre-processed so that it could be used by the tokenizer.

The cached file is stored as a pickle object and the function to accomplish this is defined as below. This is used to persist the data so it can be reloaded very quickly and easily. If the cache-file exists then the data is reloaded and returned, otherwise the function is called and the result is saved to cache.



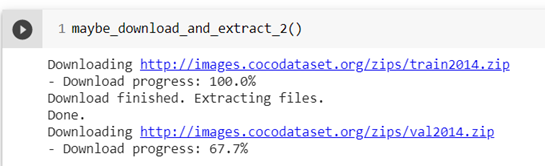
We facilitate the download of the dataset in the desirable format with the aid of the below functions.

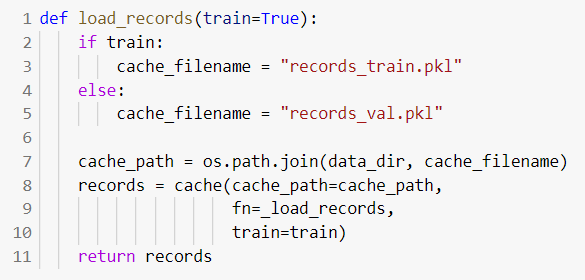


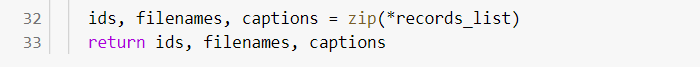


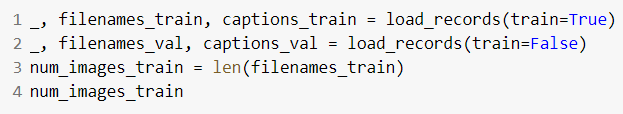


***Downloading the dataset:***



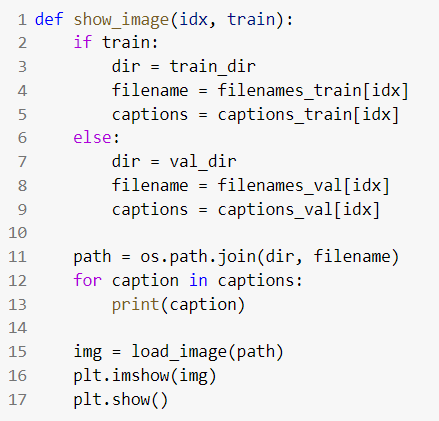
The COCO data-set contains a large number of images and various data for each image stored in a JSON-file. The load\_records function provides the functionality to get a list of image-filenames (but not actually loading the images) along with their associated data such as text-captions describing the contents of the images. 





The below given load\_image function accomplishes the job of image pre-processing. It loads the image from the given file-path and resizes it to the given size. The images are scaled so that their pixels fall between 0.0 and 1.0. It is then plotted with the show\_image function.



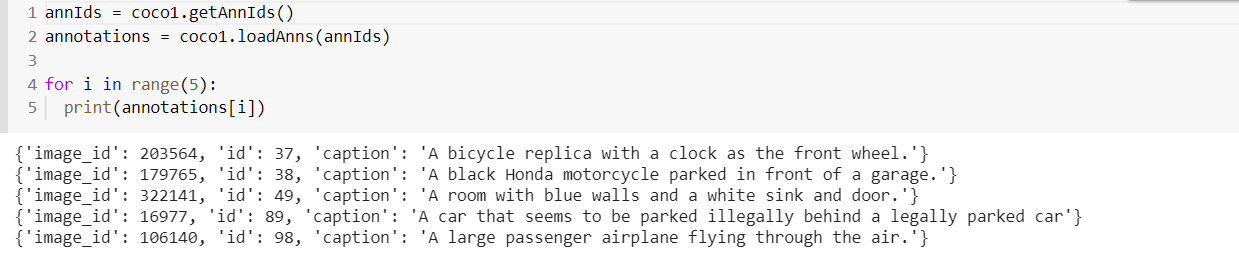


The pycocotools has been put into use. It is a Python API that assists in loading, parsing and visualizing the annotations in COCO. We instantiate the COCO class by passing the json file as an argument.



***Sample of the Annotations:***

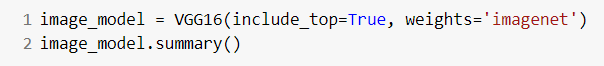
Annotations is a list of dictionaries. The dictionary contains the image\_id, id(caption id), caption as the keys. Here id is the primary key and is used to retrieve a unique caption. img is a dictionary with the following keys. We use the coco\_url to load and display the image

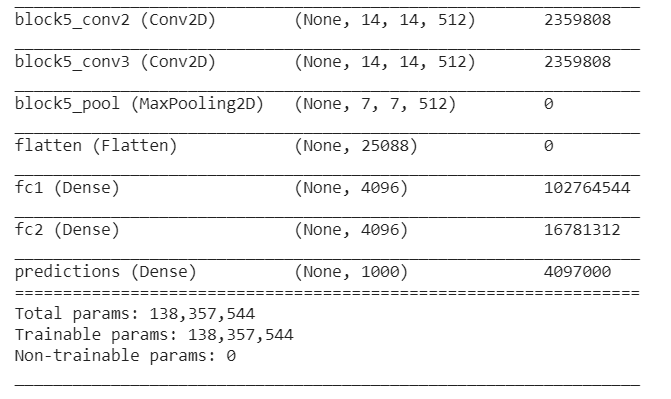
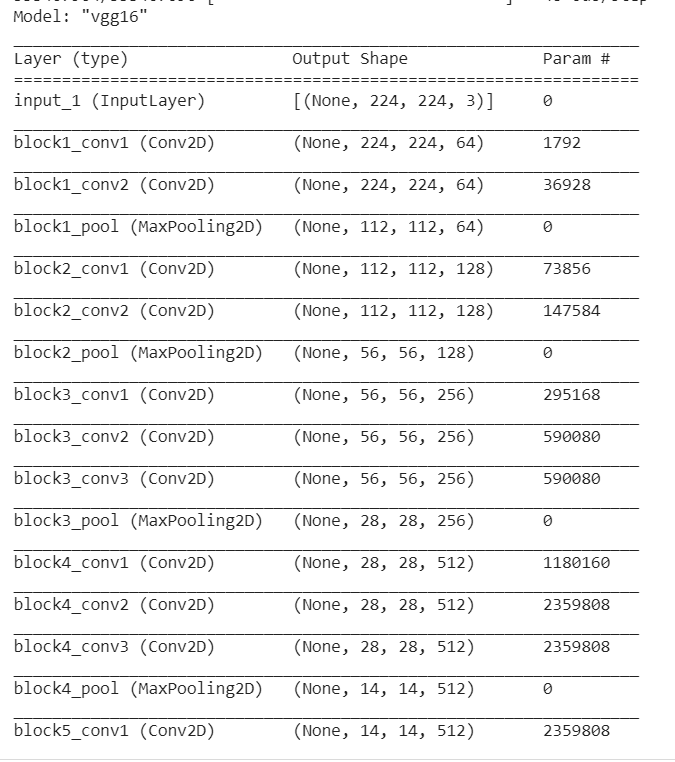


***Using the load\_img method to display an image from the url:***



**MODULE – 2 VGG 16 MODEL ENCODER**





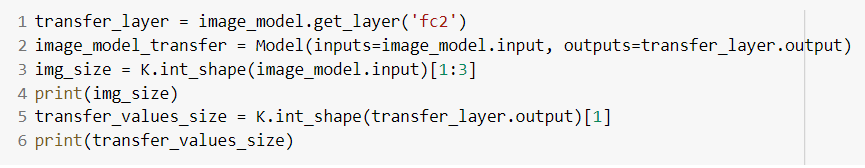
We instantiate the VGG16 architecture by importing it from tensorflow.keras.applications. It loads weights pre-trained on ImageNet. The default input size for this model is 224x224. We remove the last predictions layer and transfer the values of the second fully connected layer.



The get\_layer() method takes as parameters the name of the specific layer which we want and retrieves the layer correspondingly.The transfer\_layer variable has the fc2 layer stored. We make use of the K.int\_shape() function which returns the shape of tensor or variable as a tuple of int or None entries.

1. K.int\_shape(image\_model.input) – Denotes shape of input vector to the model having value (None, 224, 224, 3).
2. K.int\_shape(transfer\_layer.output) – Denotes shape of output vector of fc2 layer having value (None, 4096).

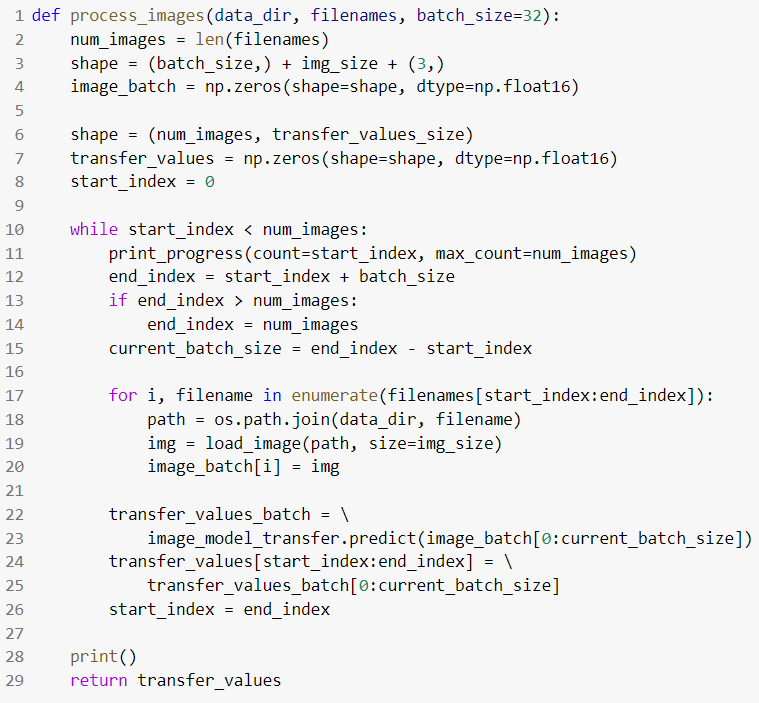
Thus, we assign img\_size and transfer\_values\_size the values (224,224) and 4096 respectively.

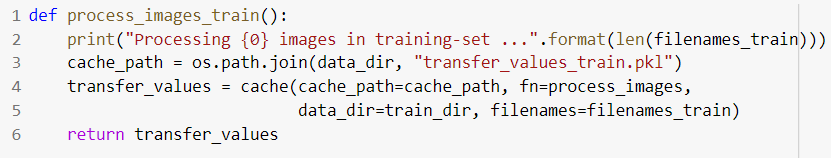


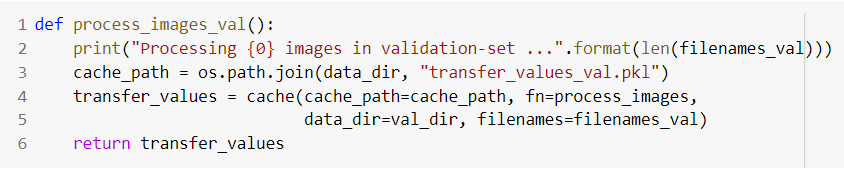
Next step is to process all the images with the vgg16 model and cache the values.

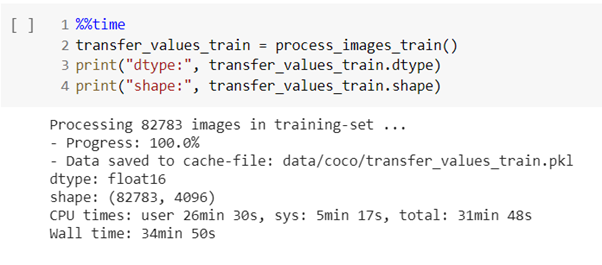
In-order to cache the transfer-values, the cache function is called upon with the path as data/coco/transfer\_values\_train.pkl .

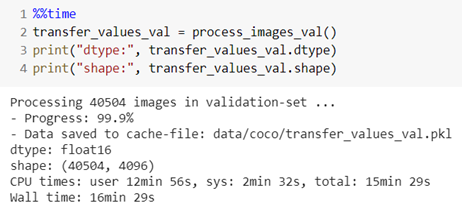
If the cache path i.e. the transfer\_values\_train.pkl file exists, we return the obj. If it doesn't exist the caching of the transfer-values occurs in batches of images. During this process, we load the image using load\_image() function defined previously and the images are resized to meet the expected format for the vgg16 architecture. Thus, we have cached the transfer values for the 82783 images in the training dataset by having the features extracted from the image from the output of the fc2 layer of vgg16 model.

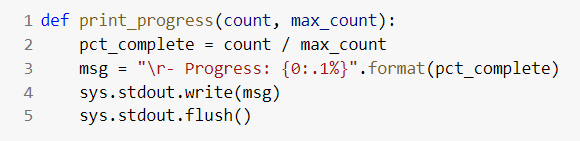








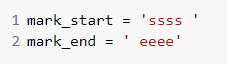




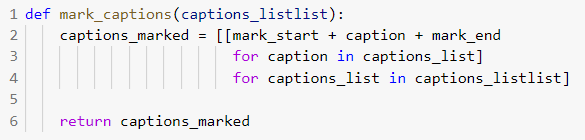
In-order to keep track of the progress, the percentage of completed downloads is constantly updated.

**MODULE - 3 TOKENIZER**

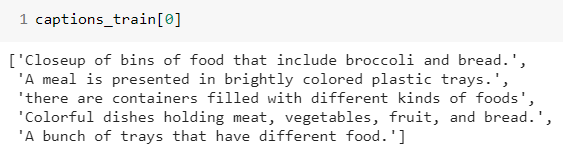
Neural Networks cannot work directly on text-data. We use a two-step process to convert text into numbers that can be used in a neural network. The first step is to convert text-words into so-called integer-tokens. The second step is to convert integer-tokens into vectors of floating-point numbers using a so-called embedding-layer.



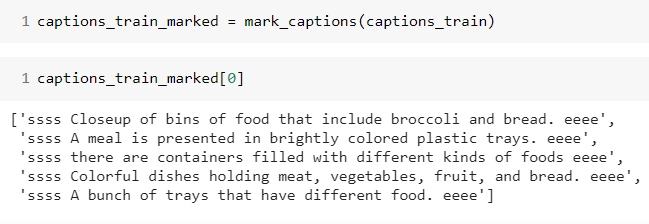
Before we can start processing the text, we first need to mark the beginning and end of each text-sequence with unique words that most likely aren't present in the data.



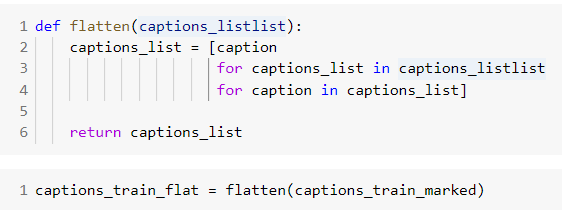
The mark\_captions function wraps all text-strings in the above markers. Since the captions are a list of lists, we use a nested for-loop to process using list-comprehension in python. In the inner loop we iterate over all the captions as we have four to six captions for each image. For each caption we append the start and end marker. Thus, captions\_marked is a list of lists which is returned by this function to the variable captions\_train\_marked.



captions\_train is obtained from the pre-processor module. It’s a tuple of lists. Thus, each element of the tuple captions\_train is as shown above.

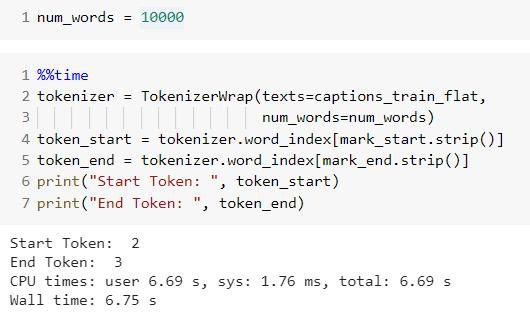


captions\_train\_marked is a list of lists where each caption has the start and end markers added. Thus, each element of the tuple captions\_train\_marked is like shown above.



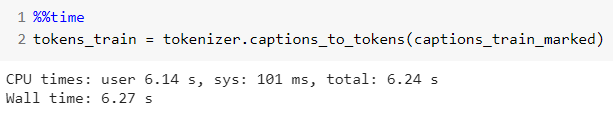
Now process all the captions in the training-set. Next we make a call to flatten() function to which we pass as parameters the marked captions. The functionality achieved here is pretty straightforward and simple.

Basically, we’re converting a list of lists into a single list. But now we unwrap the inner lists to provide a single flattened list. Thus, captions\_list is a python list having all the captions in the training dataset. This is returned to captions\_train\_flat.



The TokenizerWrap is inherited from the Tokenizer class in tensorflow.keras.preprocessing.text as we need more functionality than provided by this Tokenizer class so we wrap it.

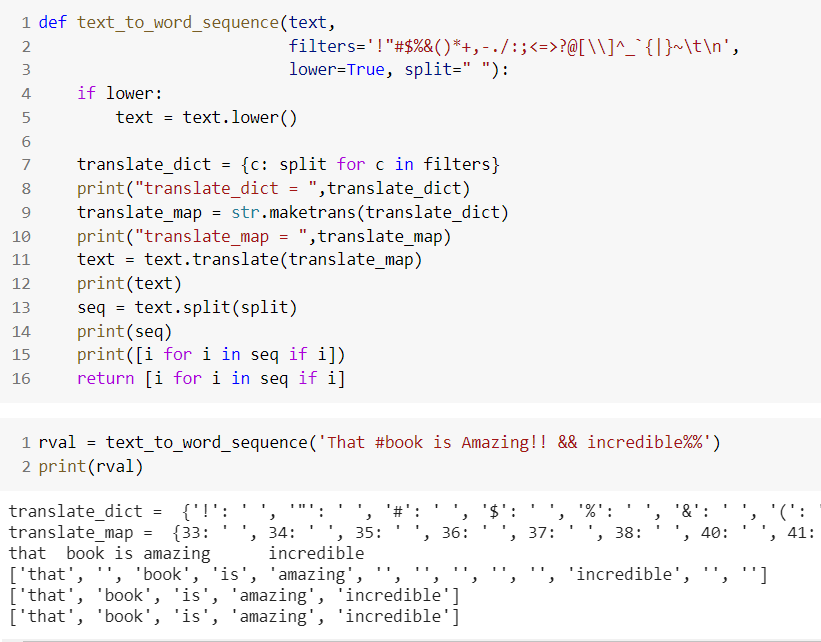
We instantiate a TokenizerWrap object as below by passing the flat list of captions and num\_words as parameters. TokenizerWrap is used to convert a text into sequence of integers. The maximum number of words in the vocabulary is set using the num\_words variable to 10000. This means that we will only use the 10000 most frequent words in the captions from the training-data.



Next we pass the flattened caption to the fit\_on\_texts() function of Tokenizer class. This function updates internal vocabulary based on a list of texts. Then we iterate over the items of the list.

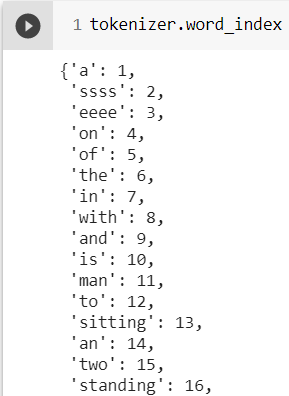
Now create a tokenizer using all the captions in the training-data. The flattened list of captions is used to create the tokenizer because it cannot take a list-of-lists.





This is the sample execution of text\_to\_word\_sequence(). We only pass the text parameter which is each string in the flattened list. Initially we convert it to lowercase. translate\_dict is a dictionary where keys are the punctuations and values are space. The maketrans() method returns a mapping table that can be used with the translate() method to replace specific characters.

In the mapping table we replace the keys in translate\_dict dictionary with their corresponding ASCII values. Using the translate function we replace the instances of the key in the text with space. i.e We turn all the punctuators into spaces. Now we split the text having space as the delimiter. In the end we return a list from the split text consisting of words which are of size at least 1.



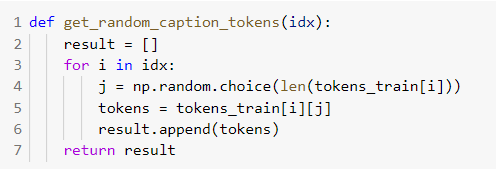
Next passing captions\_train\_marked which is a list of lists of marked captions to the function captions\_to\_tokens, we call the function texts\_to\_sequences() of Tokenizer class for each list in the variable caption\_listlist.



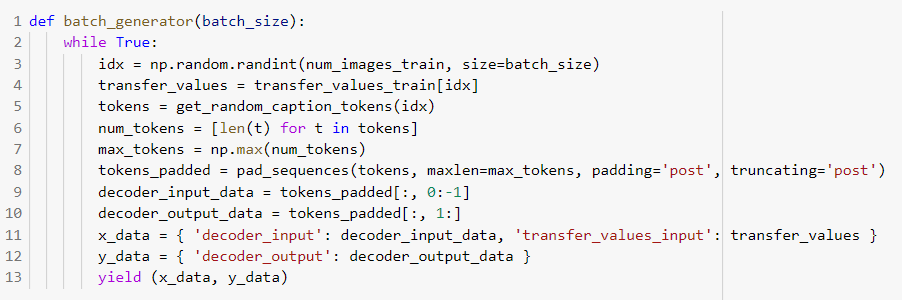
tokens\_train displays the output of the tokenizer (sequence of integers) .



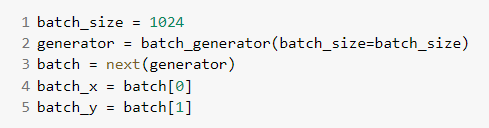
**DATA GENERATOR FOR MODEL TRAINING**



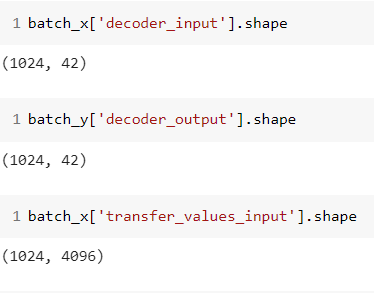
Each image in the training-set has at least 5 captions describing the contents of the image. The neural network will be trained with batches of transfer-values for the images and sequences of integer-tokens for the captions. The above function Given a list of indices for images in the training-set, select a token-sequence for a random caption, and return a list of all these token-sequences.



This generator function creates random batches of training-data for use in training the neural network.it selects the data completely randomly for each batch, corresponding to sampling of the training-set with replacement. This means it is possible to sample the same data multiple times within a single epoch - and it is also possible that some data is not sampled at all within an epoch. However, all the data should be unique within a single batch.



Get the pre-computed transfer-values for those images. These are the outputs of the pre-trained image-model. For each of the randomly chosen images there are at least 5 captions describing the contents of the image. Select one of those captions at random and get the associated sequence of integer-tokens.



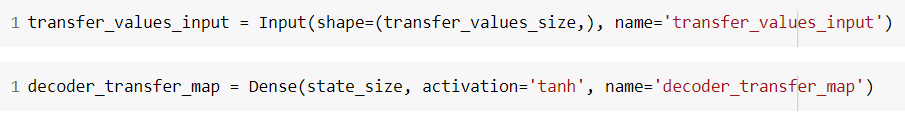
Pad all the other token-sequences with zeros so they all have the same length and can be input to the neural network as a numpy array. Set the batch-size used during training. This is set very high so the GPU can be used maximally - but this also requires a lot of RAM on the GPU. You may have to lower this number if the training runs out of memory. The steps per epoch is calculated depending upon the total number of captions in the training set and the size of each batch.

**MODULE - 4 RNN DECODER**

The Recurrent Neural Network (RNN) is trained to map the vectors with transfer-values from the image-recognition model into sequences of integer-tokens that can be converted into text.

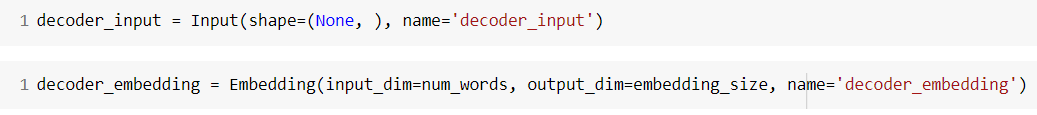


The functional model from Keras is used to build this neural network, because it allows more flexibility in how the neural network can be connected, enabling us to experiment and connect the image-model directly to the decoder. The network construction has been split into two parts: (1) Creation of all the layers that are not yet connected, and (2) a function that connects all these layers.

****

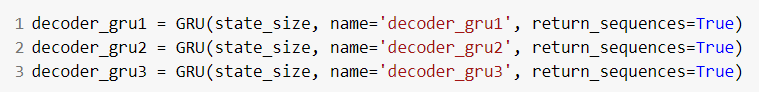
The embedding-layer converts integer-tokens into vectors of this length 128.

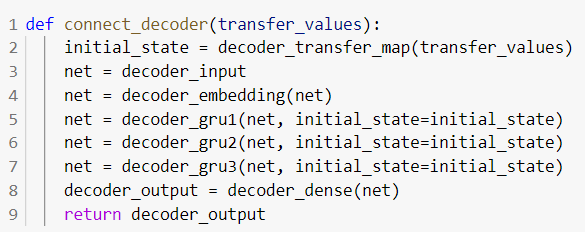
The transfer-values are used to initialize the internal states of the GRU units. This informs the GRU units of the contents of the images. The transfer-values are vectors of length 4096 but the size of the internal states of the GRU units are only 512, so we use a fully-connected layer to map the vectors from 4096 to 512 elements.

****

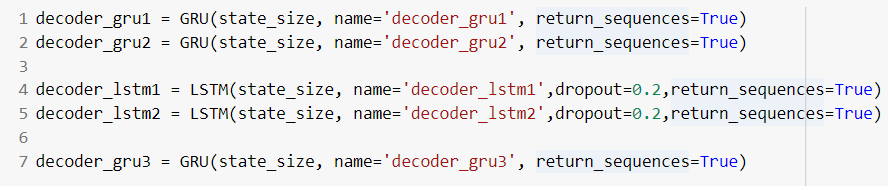
A tanh activation function is used to limit the output of the mapping between -1 and 1. This is the embedding-layer which converts sequences of integer-tokens to sequences of vectors. All the GRU units return sequences because we ultimately want to output a sequence of integer-tokens that can be converted into a text-sequence.

**ALGORITHM - 1 : DECODER CONSISTING OF 3 GRU (GATED RECURRENT UNITS)**

****

****

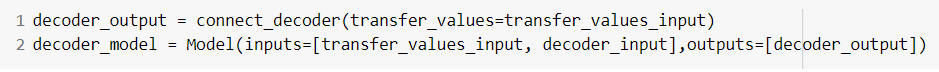
**ALGORITHM - 2 : DECODER CONSISTING OF 2 GRU + 2 LSTM + 1 GRU**

****

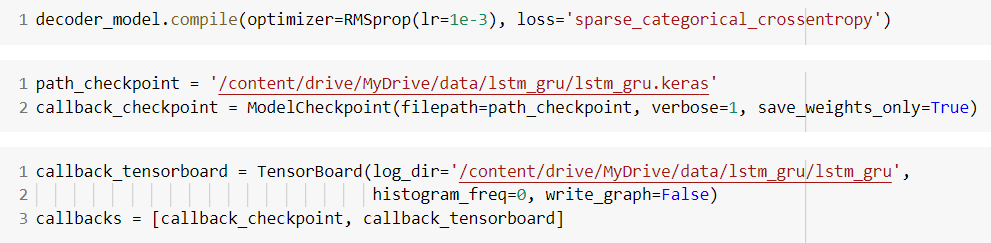
The GRU layers output a tensor with shape [batch\_size, sequence\_length, state\_size], where each "word" is encoded as a vector of length - state\_size. We need to convert this into sequences of integer-tokens that can be interpreted as words from our vocabulary. To accomplish this we convert the GRU output to a one-hot encoded array.

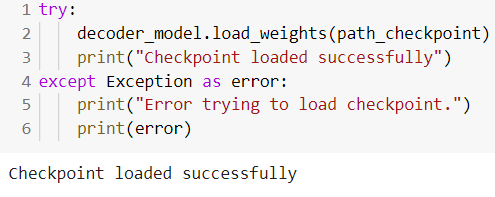
****

This function connects all the layers of the decoder to some input of transfer-values.

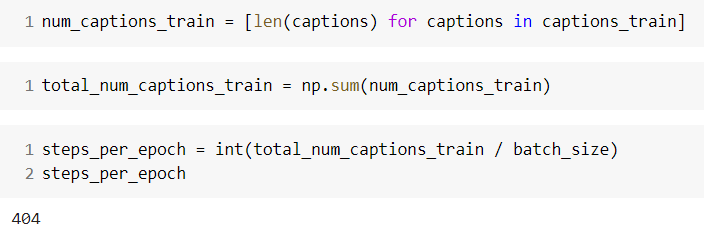
The transfer values are mapped such that the dimensionality matches the internal states of the GRU layers. Hence we use the mapped transfer values as the initial state of the GRU layers. Thus the model takes as input transfer-values and sequences of integer-tokens and outputs sequences of one-hot encoded arrays that can be converted into integer-tokens.****

The output of the decoder is a sequence of one-hot encoded arrays. In order to train the decoder we need to supply the one-hot encoded arrays that we desire to see on the decoder's output, and then use a loss-function like cross-entropy to train the decoder to produce this desired output.

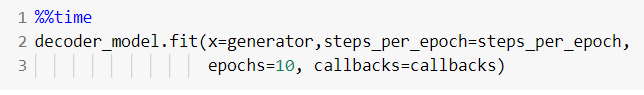
****

However, our data-set contains integer-tokens instead of one-hot encoded arrays. This conversion from integers to one-hot arrays is done in the batch\_generator() above. This is done using sparse cross-entropy loss-function, which does the conversion internally from integers to one-hot encoded arrays. RMSprop optimizer has been used.****

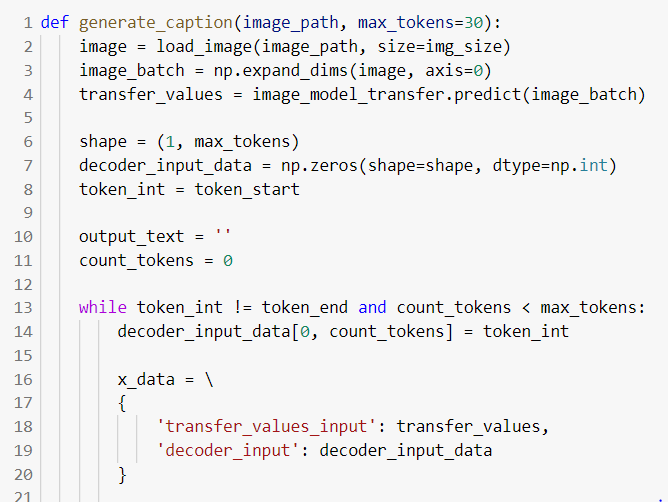
During training we want to save checkpoints and log the progress to TensorBoard so we create the appropriate callbacks for Keras. This is the callback for writing checkpoints during training. The model has been trained for 300 epochs in which each epoch took 10 minutes to run on a GPU.

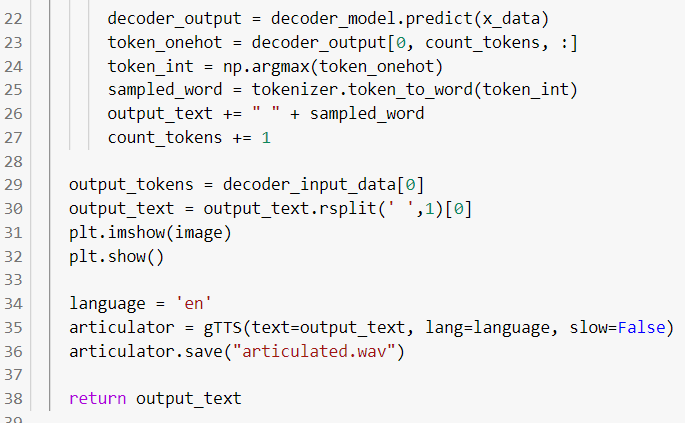
****

The number of steps per epoch is found from the ratio of the total number of captions to the batch size.

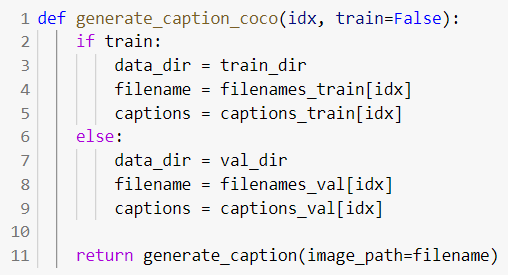
****

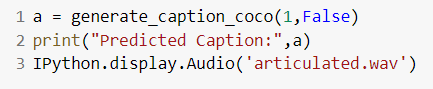
**MODULE - 5 LEXICAL ARTICULATOR**

****

****

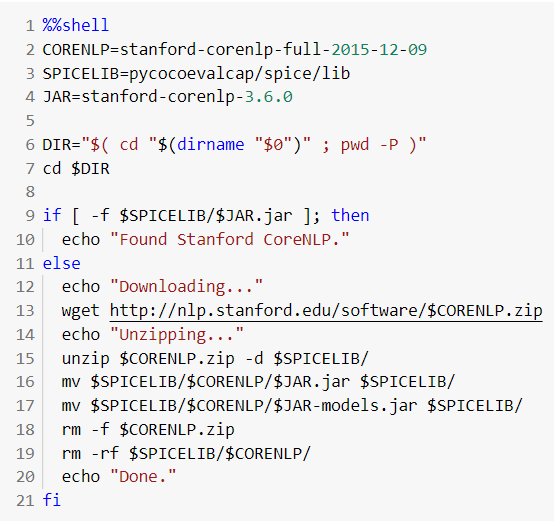
The generate\_caption() function loads an image and generates a caption using the trained model. The google-text-to-speech library is used to convert the generated caption to speech. We set the language to english and set the pace to slow. The converted audio is saved and played using IPython-Audio.

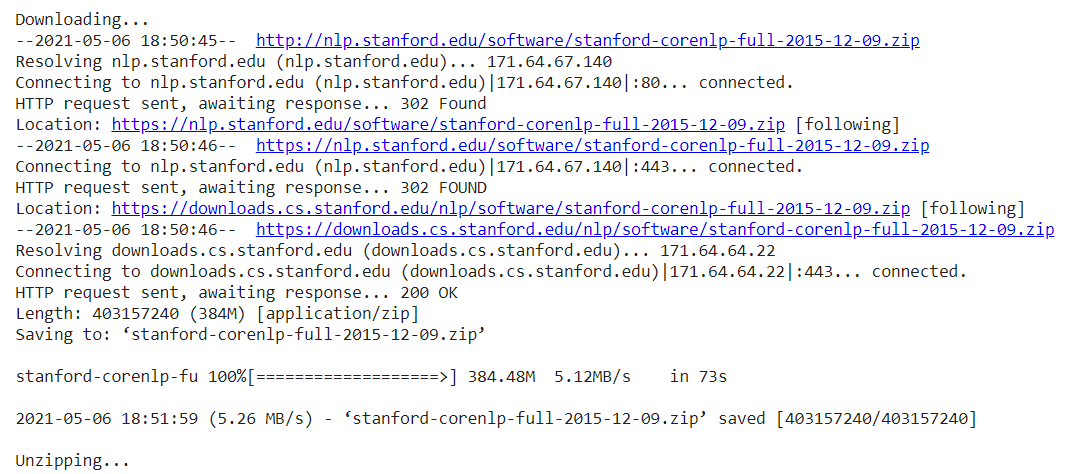
****

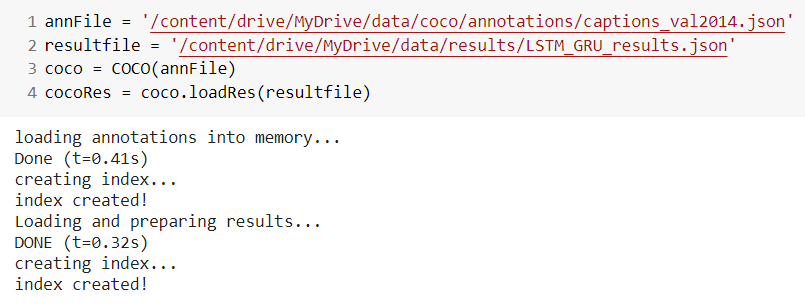
****

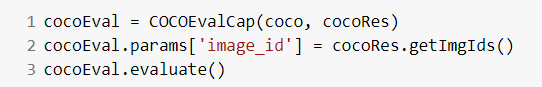
|  |  |
| --- | --- |
| **COMPARISON OF CAPTIONS OF ALGORITHMS** | |
| **3 GRU** | **2 GRU + 2 LSTM + 1 GRU** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**PERFORMANCE METRIC EVALUATION**

****

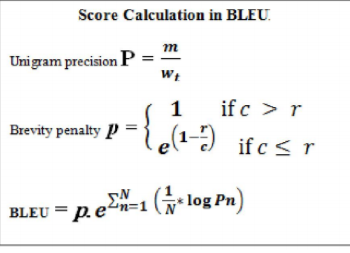
****

****

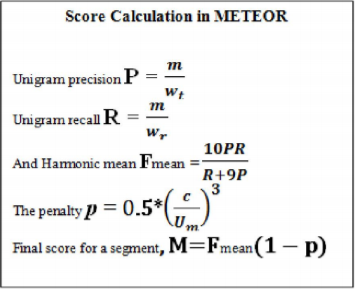
****

|  |  |  |
| --- | --- | --- |
| **COMPARISON OF PERFORMANCE METRICS** | | |
| **METRIC** | **ALGORITHM - 1** | **ALGORITHM - 2** |
| **BLEU-1** | 0.497 | 0.580 |
| **BLEU-2** | 0.297 | 0.406 |
| **BLEU-3** | 0.179 | 0.280 |
| **BLEU-4** | 0.103 | 0.193 |
| **METEOR** | 0.165 | 0.195 |
| **ROUGE-L** | 0.318 | 0.396 |
| **CIDER** | 0.471 | 0.600 |
| **SPICE** | 0.128 | 0.133 |

**BLEU (Bilingual Evaluation Understudy)** is a score for comparing a candidate translation of text to one or more reference translations.Blue score is a number between 0 and 1.This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts. A perfect score of 1 indicates that the candidate is identical to one of the reference translations.



**Meteor (Metric for Evaluation of Translation with Explicit Ordering)** automatic evaluation metric scores machine translation hypotheses by aligning them to one or more reference translations. Alignments are based on exact, stem, synonym, and paraphrase matches between words and phrases. Segment and system level metric scores are calculated based on the alignments between hypothesis-reference pairs.

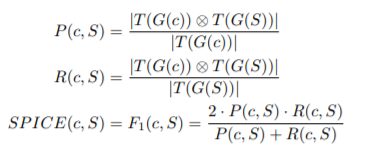


**CIDEr (Consensus-based Image Description Evaluation).**

CIDEr metric measures the similarity of a generated sentence against a set of ground truth sentences written by humans. The metric shows high agreement with consens us as assessed by humans.

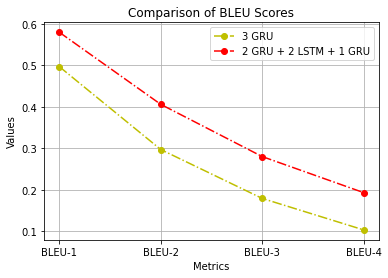
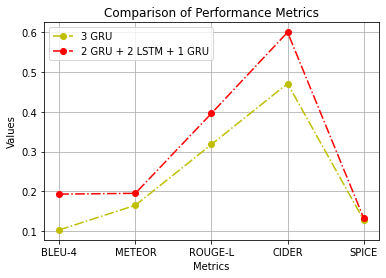
**SPICE (Semantic Propositional Image Caption Evaluation)**

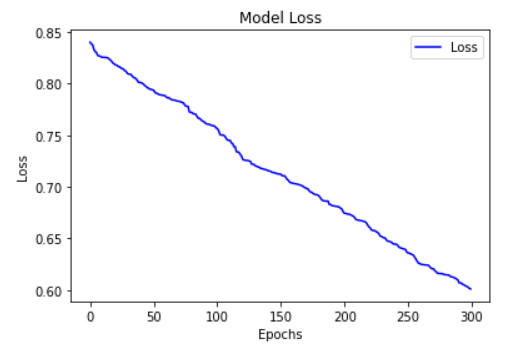
In the first stage, syntactic dependencies between words in the caption are established using a dependency parser pretrained on a large dataset. In the second stage, dependency trees to scene graphs are mapped using a rule-based system Given candidate and reference scene graphs,SPICE metric computes an F-score



**ROUGE-L** (**Recall-Oriented Understudy for Gisting Evaluation**) – measures longest matching sequence of words using LCS. An advantage of using LCS is that it does not require consecutive matches but in-sequence matches that reflect sentence level word order. Since it automatically includes longest in-sequence common n-grams, we don’t need a predefined n-gram length.

***Graphical analysis of Performance Metrics:***

****



***Inferences:***

From the above graph it can be inferred that the algorithm 2 where the decoder is composed of a combination of 2 Gated Recurrent Units, 2 Long Short Term Memory and 1 Gated Recurrent Unit performs better than the algorithm 1 where the decoder is composed of 3 Gated Recurrent Units. The two algorithms are compared among 5 metrics - BLEU, METEOR, ROUGE-L, CIDER and SPICE and the Algorithm 2 outperforms the other in all the metrics.