**IMAGE CAPTION GENERATOR**

**A PROJECT REPORT**

***Submitted by***

Sushruta Mandal — 12619002060

Arnab Chatterjee — 12619002014

Aryan — 12619004011

*Under the supervision of*

Prof. Uttam Kumar Dash

Department of Information Technology

in partial fulfillment for the award of the degree

of

**BACHELOR OF TECHNOLOGY**

in

**INFORMATION TECHNOLOGY**

HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA

An autonomous Institute under

Maulana Abul Kalam Azad University of Technology

Formerly Known as

West Bengal University of Technology

December, 2022

**ACKNOWLEDGEMENT**

We would take this opportunity to thank Prof. Pranay Chaudhuri, Principal of Heritage Institute of Technology for providing us with all the necessary facilities to make our project work successful.

We would like to thank our Head of the Department for his kind assistance as and when required.

We will be thankful to Prof. our project coordinator for constantly supporting and guiding us, and giving invaluable insights. His guidance and his words of encouragement motivated us to achieve our goal and impetus to excel.

We thank our faculty members and laboratory assistants at the Heritage Institute of Technology for playing a pivotal and decisive role during the development of the project.

Last but not least, we thank all our friends for the cooperation and encouragement they have bestowed upon us.

**Sushruta Mandal**

**Arnab Chatterjee**

**Aryan**

**HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA**

**An autonomous Institute under**

**Maulana Abul Kalam Azad University of Technology**

**Formerly Known as**

**West Bengal University of Technology**

**BONAFIDE CERTIFICATE**

This is to certify that the project reported here, **Image Caption Generator,** has been carried out independently by **Sushruta Mandal, Arnab Chatterjee,** and **Aryan** under the guidance of **Prof. Uttam Kumar Dash** as a project in certification course of Information Technology, and is their original and bonafide work.

**SIGNATURE SIGNATURE**

**Dr. Siuli Roy Prof. Uttam Kumar Dash**

**HEAD OF DEPARTMENT PROJECT GUIDE**

**Department of IT Department of IT**

**Chowbaga Road, Anandapur, Mundapara, Chowbaga Road, Anandapur, Mundapara,**

**Kolkata, West Bengal 700107 Kolkata, West Bengal 700107**

**SIGNATURE**

**EXTERNAL EXAMINER**

**ABSTRACT**

A human can point out and convey an enormous quantity of information about an unfamiliar visual situation with only a cursory glance at it. However, even with today’s state-of-the-art visual recognition algorithms, this basic human skill has proven elusive in the realm of artificial intelligence. To bypass that limitation, the vast majority of prior visual recognition research has concentrated on labeling images with a predetermined set of visual categories (labels). While significant progress has been made in this area, those models frequently rely on hard-coded visual notions and pre-generated language templates, limiting their versatility. Furthermore, they focus on condensing complicated visual sequences into a single sentence, which hinders their effectiveness in certain applications such as developing tools for the visually-impaired. As a result, in this paper, we want to take a step toward producing natural language descriptions of images to convey as much information as possible, rather than a limited single-label mechanical description. This will allow our software to set itself apart by tackling entirely different use-cases from pre-existing models.

**TABLE OF CONTENTS**

1. **Introduction** 
   1. Importance — 6
   2. Objective — 7
   3. Problem Statement — 8
2. **Related Work** —9
3. **Software Used** — 11
4. **Methodology** —12
   1. Data Collection — 13
   2. Feature Extraction — 15
   3. Sequence Processing and Decoding — 17
   4. Training the Model — 19
   5. Testing the Model — 20
   6. Generating Fresh Captions — 21
5. **Implementation** 
   1. Data Collection — 22
   2. Feature Extraction — 23
   3. Sequence Processing and Decoding — 25
   4. Training the Model — 26
   5. Testing the Model — 27
   6. Generating Fresh Caption — 28

1. **Results** — 30
2. **Future Scope** — 31
3. **Conclusion** — 32
4. **References** — 33

**INTRODUCTION**

**1.1) Importance:**

A quick glance at an image is sufficient for a human to point out and describe an immense amount of details about the visual scene. However, this remarkable ability has proven to be an elusive task, even for the current state-of-art visual recognition models. The majority of previous work in visual recognition has focused on labeling images with a fixed set of visual categories (labels) and great progress has been achieved in these endeavors. However, while these mechanisms work great for a single label, blunt definition of an image, they are nowhere close to the i.e. vastly restrictive when compared to the enormous amount of descriptions a human being can compose. Some approaches that address the above stated challenge of generating image descriptions have been developed. However, these models often rely on hard-coded visual concepts and pre-generated sentence templates, which imposes limits on their variety. Moreover, the focus of these works has been on reducing complex visual scenes into a single sentence, which proves to be an unnecessary restriction.

**1.2) Objective:**

In this work, we strive to take a step towards the goal of generating dense descriptions of images i.e. produce the natural language description of images rather than a restricted single label mechanical description.

The primary challenge towards this goal is in the design of a model that is rich enough to simultaneously understand the contents of images as well as their respective description in the domain of natural language. Additionally, the model should be free of assumptions about specific hard-coded templates, rules or categories and instead rely on learning from the training data. The second, practical challenge is that datasets of image captions are available in large quantities on the internet, but these descriptions mix-up mentions of several entities whose locations in the images are unknown.

**1.3) Problem Statement**

In this project, we want to build a system that can describe an image as a complete English sentence, using a CNN and an LSTM-based RNN.

S=f(g(I))

Where I is an image and S is a complete English sentence, g is a function that extracts important features from an image, f is a function that finds the relation between those features and provides a natural language description of the image.

**RELATED WORK**

Many early neural models for image captioning encoded visual information using a single feature vector representing the image as a whole, and hence did not utilize information about objects and their spatial relationships.

Karpathy and Fei-Fei [1], as a notable exception to this global representation approach, extracted features from multiple image regions based on an R-CNN object detector and generated separate captions for the regions. As a separate caption was generated for each region, however, the spatial relationship between the detected objects was not modeled. This is also true of their follow-up dense captioning work, which presented an end-to-end approach for obtaining captions relating to different regions within an image.

Fang et al. [2], generated image descriptions by first detecting words associated with different regions within the image. The spatial association was made by applying a fully convolutional neural network to the image and generating spatial response maps for the target words. Here again, the authors did not explicitly model any relationships between the spatial regions.

Most similar to our work, Anderson et al. [3], addressed this limitation of typical attention models by combining a “bottom-up” attention model with a “top-down” LSTM. The bottom-up attention acts on mean-pooled convolutional features obtained from the proposed regions of interest of a Faster R-CNN object detector. The top-down LSTM is a two-layer LSTM in which the first layer acts as a visual attention model that attends to the relevant detections for the current token and the second layer is a language LSTM that generates the next token. The authors demonstrated state-of-the-art performance for both visual question answering and image captioning using this approach, indicating the benefits of combining features derived from object detection with visual attention.

Geometric attention was first introduced by Hu et al. [4] for object detection. There, the authors used bounding box coordinates and sizes to infer the importance of the relationship of pairs of objects, the assumption being that if two bounding boxes are closer and more similar in size to each other, then their relationship is stronger. The most successful subsequent work followed the above paradigm of obtaining image features with an object detector, and generating captions through an attention LSTM.

**SOFTWARE USED**

Python (Ver. 3.10) — The language we will be using for building our caption generator, thanks to a host of inbuilt tools and open-source libraries for machine learning.

Thonny (Ver. 4.0.1) — The IDE we are using. Lightweight and simple.

**METHODOLOGY**

The series of methods used in this project can be broadly classified under deep learning. The steps involved to build a model are as follows:

1. Data collection and cleaning
2. Feature extraction
3. Sequence processing and decoding
4. Training the model
5. Testing the model
6. Generating fresh captions

After the data is collected it is cleaned, and all null values and outliers are removed. The machine learning algorithms work on numerical data, so we convert all features to numerical. We encode the non-numerical features before feeding it to the model. After that we split the data for training and testing and feed train data to our model, then we check the accuracy of our model by checking its BLEU score, a popular scoring parameter for caption generation in Machine Learning. The NLTK library contains functions for testing BLEU scores.

**4.1) Data Collection and Cleaning:**

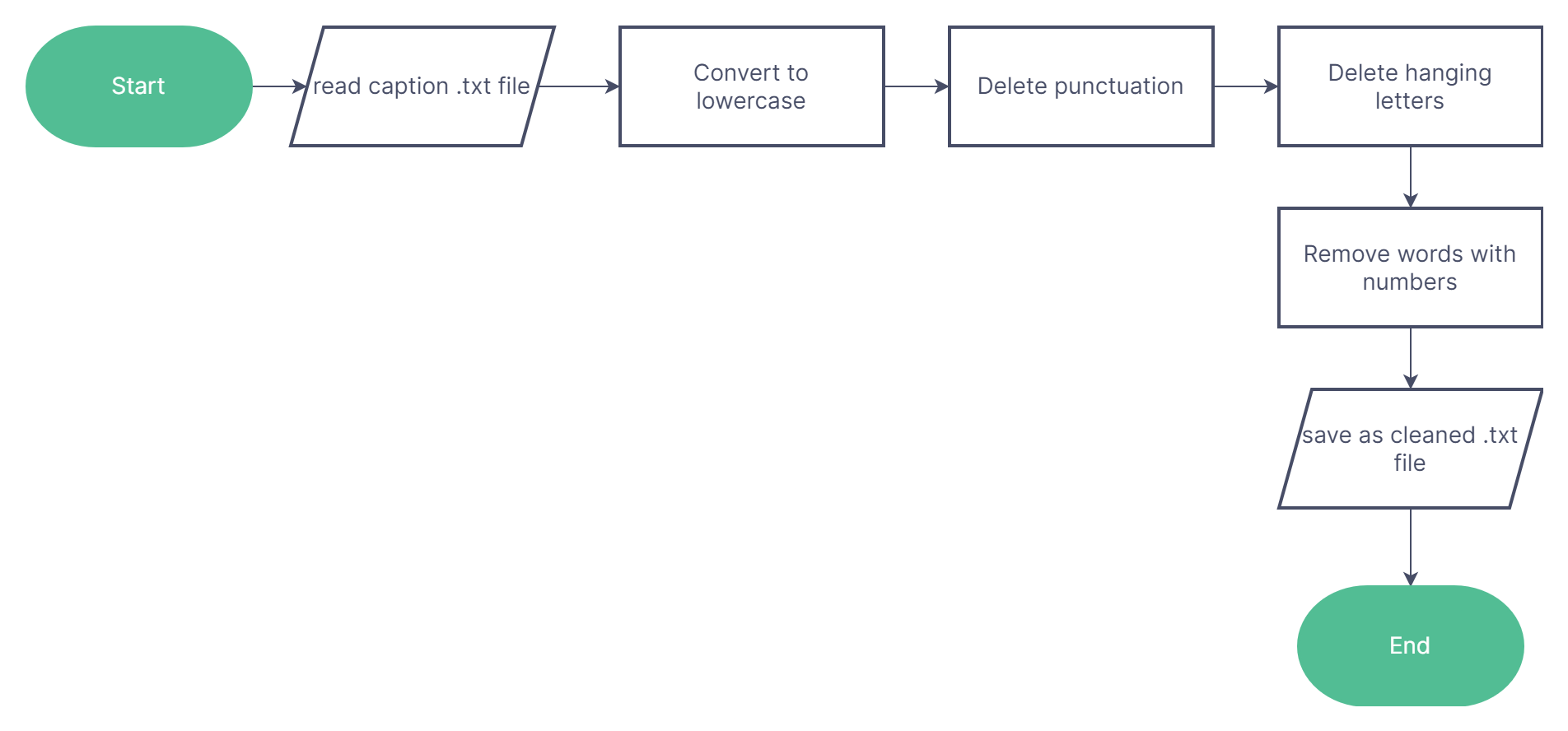
In our project, data has been collected from Kaggle, a data science portal.

We have identified the three most commonly used image caption training datasets in Computer Vision research domain - MSCOCO, Flickr30k, and Flickr8k. These datasets contain 123000, 31000 and 8000 caption annotated images respectively, and each image is labeled with 5 different descriptions. For our purposes, we use the Flickr30k dataset which strikes a balance between accuracy and functionality, by offering a large training pool that we can make use of despite our hardware limitations.

The dataset has a pre-defined training dataset (29,783 images), development dataset (1,000 images), and test dataset (1,000 images).

The caption data must be cleaned before it can be used, and this cleaned data will be used several times. To save time, we use a short program to clean and save the data into a separate text file so the data does not have to be cleaned over and over again.

For our project, we clean the textual data by converting all of it to lowercase, removing punctuation and hanging lettings, and removing tokens with numbers in them.



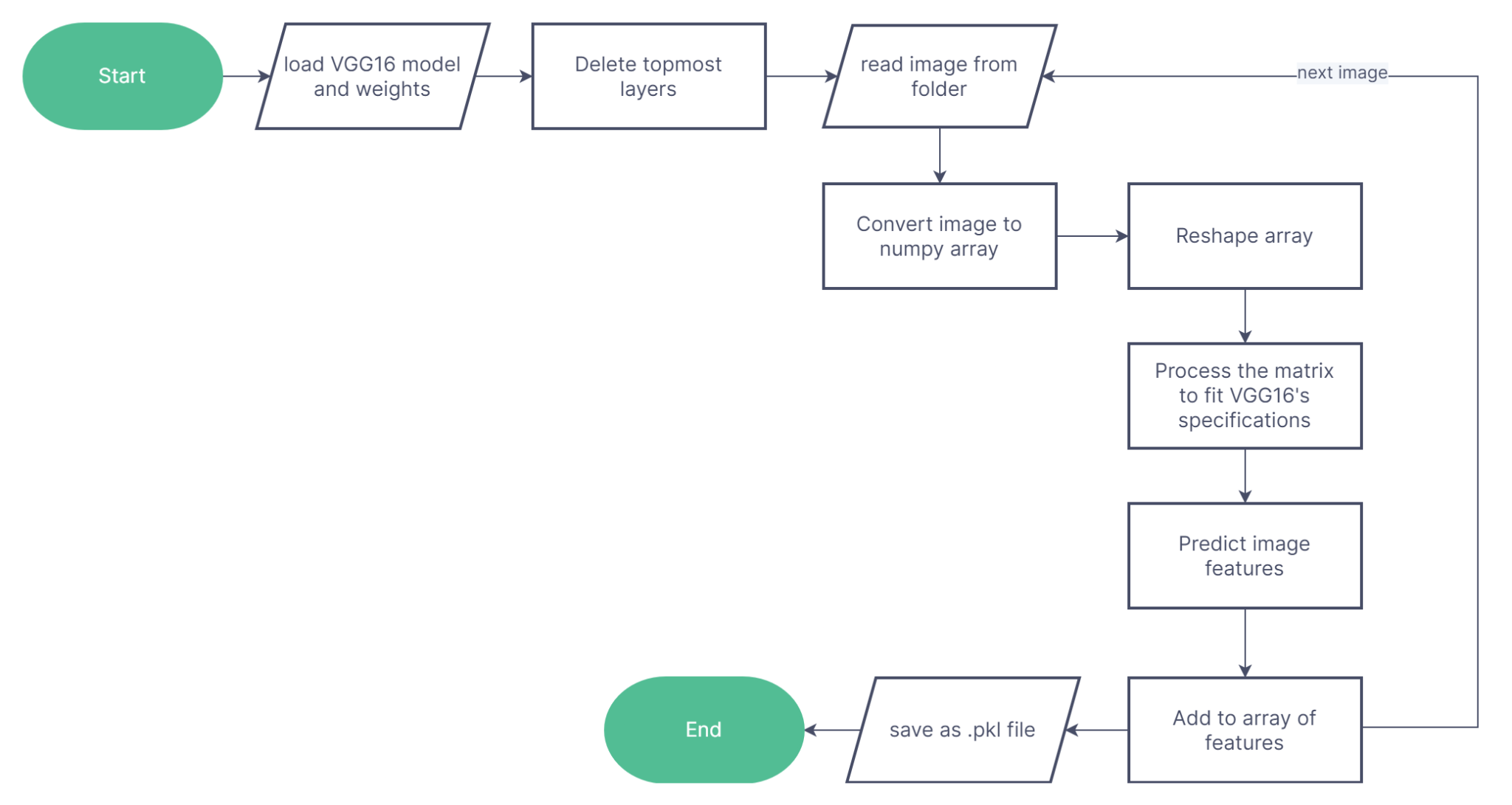
**4.2) Feature Extraction:**

The encoder needs to extract image features of various sizes and encodes them into vector space which can be fed to RNN in a later stage. In our project, we chose to modify the pre-trained VGG-16 model provided by the PyTorch library. For our purposes, CNN is used to encode features instead of classifying images. As a result, we removed the fully connected layers and the max pool layers at the end of the network. Under this new construction, the input image matrix has dimension N×3×256×256, and the output has dimension N×14×14×512.

This is a technique called transfer learning, where we use a pre-trained model VGG16 that has already been trained on large datasets and can extract the features from images so we can use them in our tasks. Keras provides this pre-trained model directly.

We can load the VGG model in Keras using the VGG class. We will remove the last layer from the loaded model, as this is the model used to predict a classification for a photo. We are not interested in classifying images, but we are interested in the internal representation of the photo right before a classification is made. These are the ‘features’ that the model has extracted from the photo.

This forms the core basis of our Photo Feature Extractor. To save on time, we extract all the features from our training and testing dataset beforehand, and save it in a pickle file *features.pkl.*

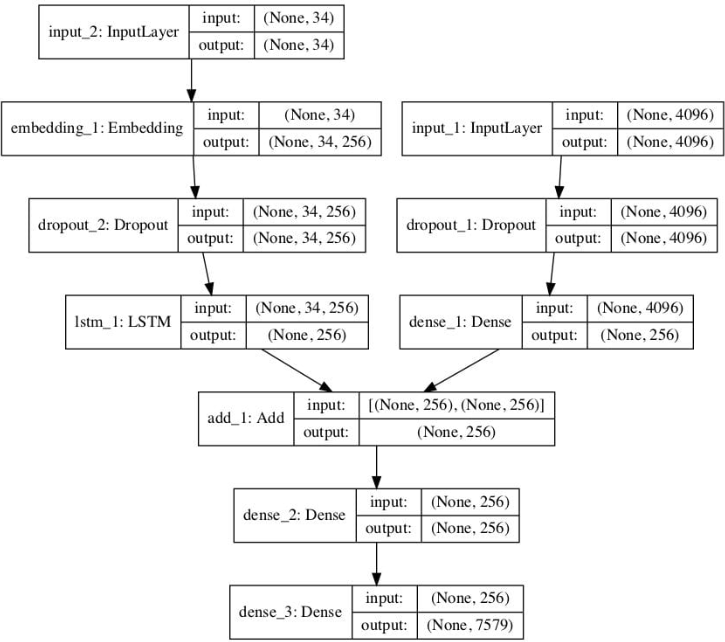
****

**4.3) Sequence Processing and Decoding:**

The **Photo Feature Extractor** is used to generate inputs for the next model in our project — the **Sequence Processor.** This model expects input sequences with a pre-defined length (34 words) which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by an LSTM layer with 256 memory units. Both the input models produce a 256 element vector. Further, both input models use regularization in the form of 50% dropout. This is to reduce overfitting the training dataset, as this model configuration learns very fast.

The decoder needs to generate image captions word by word using an LSTM-based RNN which is able to sequentially generate words. The input for the decoder is the encoded image feature vectors from CNN and the encoded image captions produced in the data-preprocessing stage. The decoder consists of an LSTM cell module, four fully connected layers provided by PyTorch library for the initialization of the states of LSTMcell and word dictionary, and an addition module to implement our CNN. In each iteration of LSTM network, we concatenate the embedded captions of all previous words and the attentioned images and feed them to the LSTM to get the next state of LSTM. Then, fully connected layers can predict the probabilities of current word embedding based on the current state and append it to the word embedding prediction matrix.

The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a Dense 256 neuron layer and then to a final output Dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence.

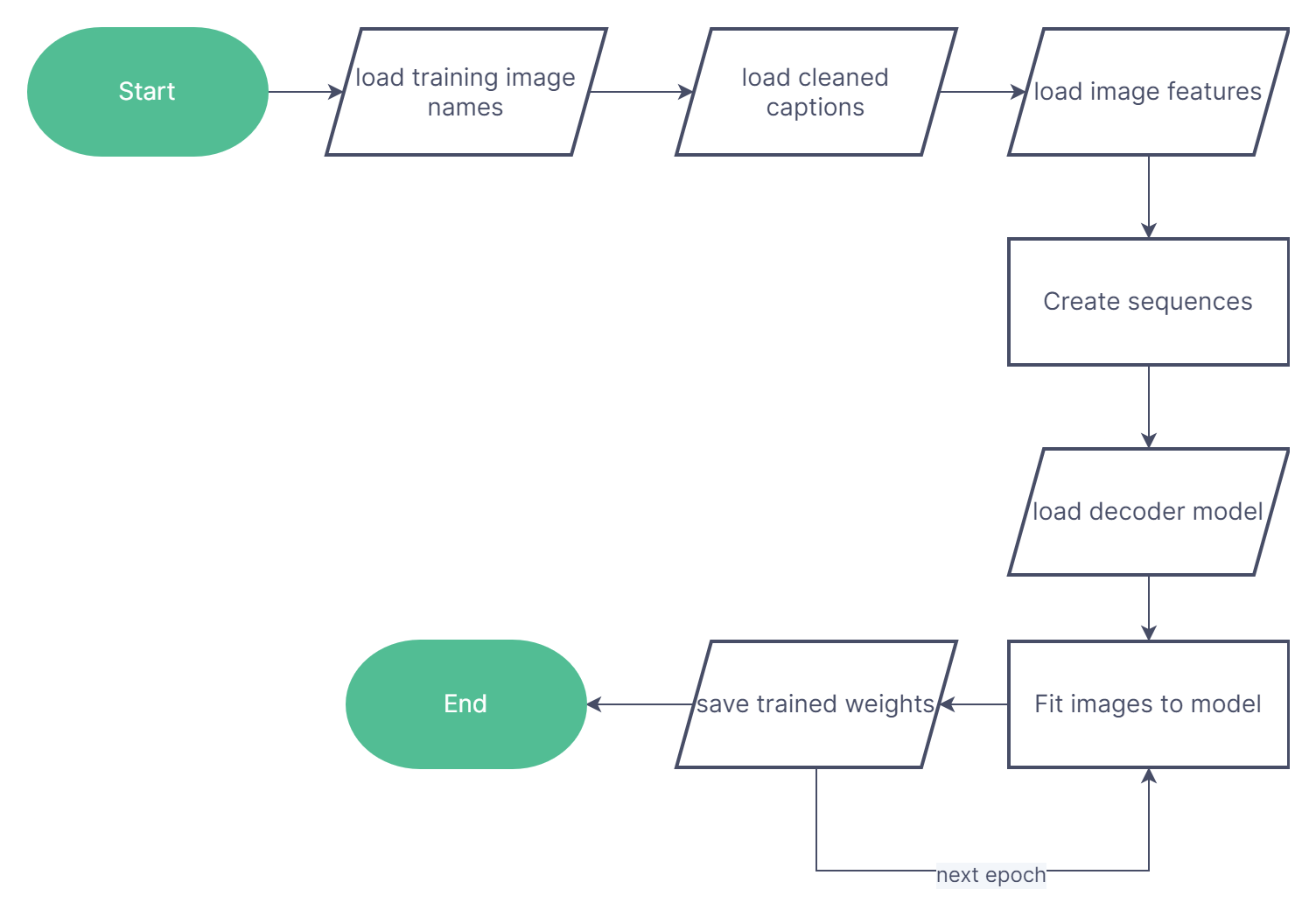


**4.4) Training the Model:**

The model learns fast and quickly overfits the training dataset. For this reason, we will monitor the skill of the trained model on the holdout development dataset. When the skill of the model on the development dataset improves at the end of an epoch, we will save the whole model to file.

At the end of the run, we can then use the saved model with the best skill on the training dataset as our final model.

We can do this by defining a *ModelCheckpoint* in Keras and specifying it to monitor the minimum loss on the validation dataset and save the model to a file that has both the training and validation loss in the filename.



**4.5) Testing the Model:**

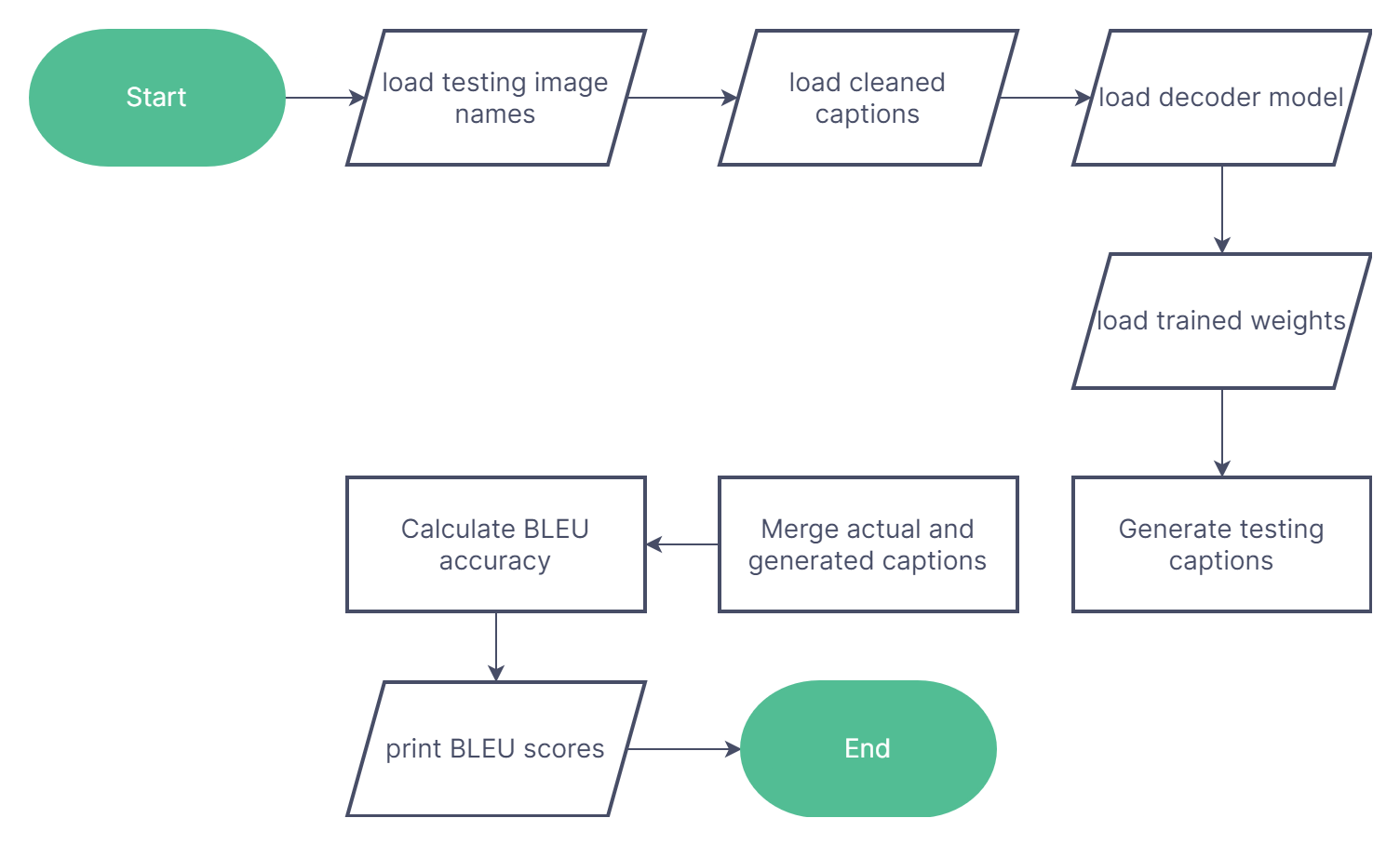
BLEU scores are used in text translation for evaluating translated text against one or more reference translations.

Here, we compare each generated description against all of the reference descriptions for the photograph. We then calculate BLEU scores for 1, 2, 3 and 4 cumulative n-grams.

Some ballpark scores for skillful models are—

* BLEU-1: 0.401 to 0.578.
* BLEU-2: 0.176 to 0.390.
* BLEU-3: 0.099 to 0.260.
* BLEU-4: 0.059 to 0.170.

BLEU-N refers to the accuracy of the model in predicting sequences of N lengths at a time, from individual words (BLEU-1), to four subsequent words (BLEU-4).

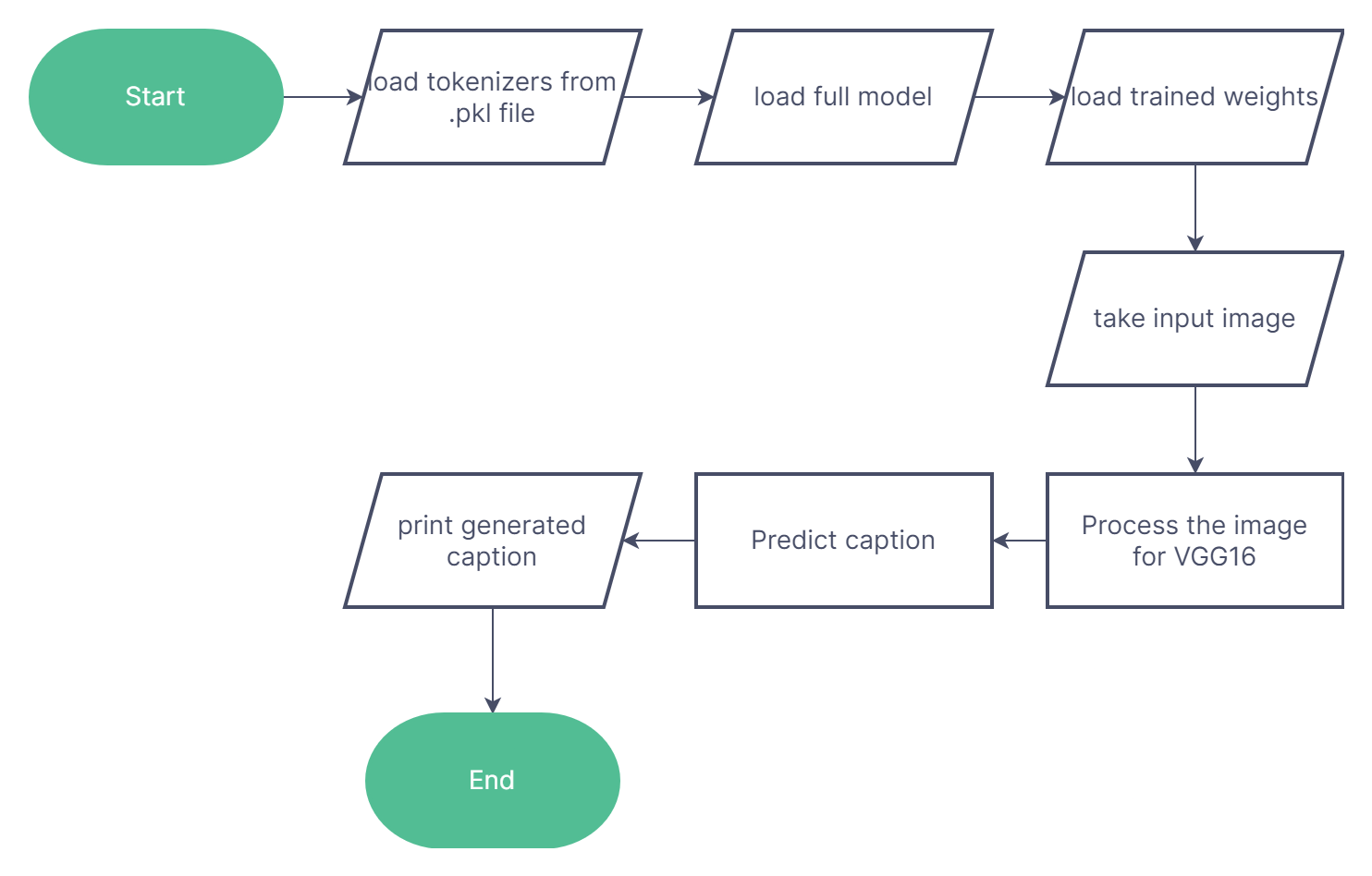


**4.6) Generating Fresh Captions:**

Almost everything we need to generate captions for entirely new photographs is in the model file. But we also need a Tokenizer for encoding generated words for the model while generating a sequence, and the maximum length of input sequences, used when we defined the model.

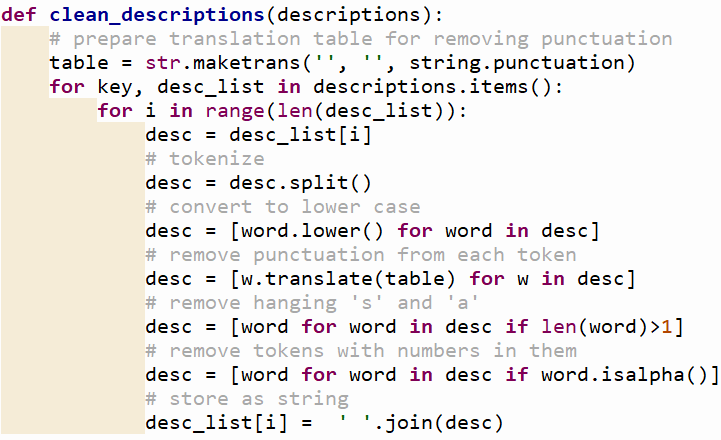
We can hard code the maximum sequence length. With the encoding of text, we can create the tokenizer and save it to a file so that we can load it quickly whenever we need it without needing the entire Flickr8K dataset. In our case, we are saving it as a pickle file *tokenizer.pkl*.

Finally, to generate fresh captions using our model, we need to merge our previous two models into a dynamically-running program, and provide an image to generate a caption for.



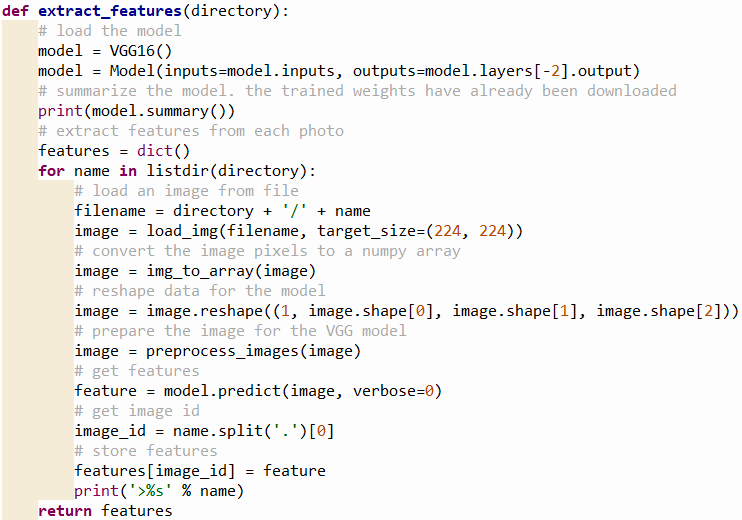
**IMPLEMENTATION**

**5.1) Data Collection and Cleaning:**



First, we need to clean the data we have downloaded from Kaggle. To do this, we use a small function clean\_descriptions() to make our data as uniform as possible. Removing capitalization, punctuation, and numbers are the primary tasks.

**5.2) Feature Extraction:**

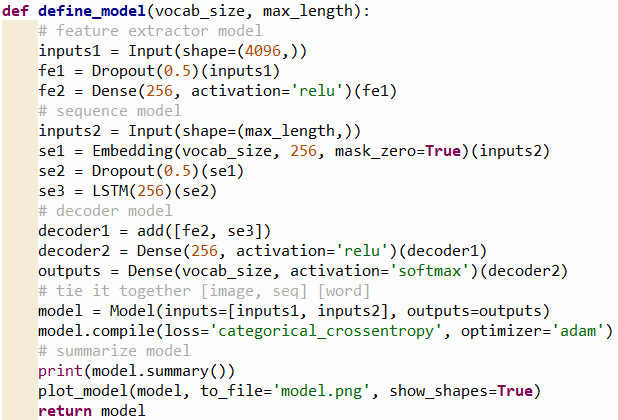


After loading the outline of the VGG16 model, we use the Model class to fully group its layers into an object with training and inference features.

The load\_img() function loads an image in the Python Imaging Library format. Then, img\_to\_array() is used to convert the PIL image to a Numpy array. After resizing the image using image.reshape(), we use preprocess\_images() to adjust the image data to fit the VGG16 model’s specifications. Then, the predict() function is used to generate output predictions for the input samples, based on the layers present in the Model object calling it.

This is our **Photo Feature Extractor.**

**5.3) Sequence Processing and Decoding:**



The Input() function is used to instantiate a Keras tensor, which is a symbolic tensor-like object which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model.

The two Dropout(0.5) layers randomly set 50% of the input units to 0. This helps prevent overfitting.

The Dense() layers are the most fundamental layers of a Neural Network. They take the input units and implement a mathematical operation using a weight matrix created by the layer, an optional bias vector, and an activation argument.

The Embedding() layer allows us to convert textual data into fixed length vectors of a defined size. The resultant vectors condense a lot of information into smaller values, which allows us to save on storage space and increases model efficiency.

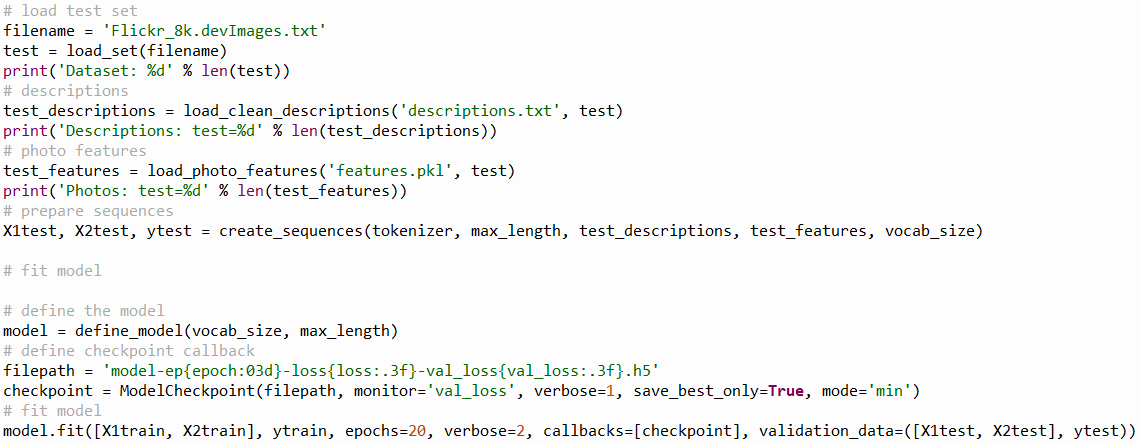
The LSTM() layer implements a ‘memory state’ for the network, which captures information from previous steps. This is done via different gates, which control the data that is added to the memory state.

The add() layer simply takes a list of tensors, all of the same shape, and merges them together into a single tensor. We do this to add together the outputs of both the **Image Feature Extractor** and the **Sequence Processor** in order to feed it to the Decoder.

Then, after creating a Model object incorporating all the different layers, we compile() it to finalize the model and make it ready to use, by specifying an optimizer and a loss function.

Finally, we visualize the model using the plot\_model function, which saves it as an image representation of the data flowchart.

**5.4) Training the Model:**

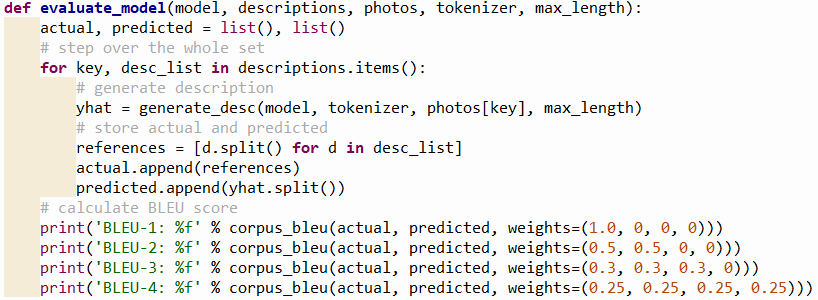


Here, load\_set() loads a predefined list of photo identifiers used to match image data to its corresponding textual data, load\_clean\_descriptions() loads a text file with the cleaned and preprocessed caption data, and load\_photo\_features() loads a pickle file of photo features that we have previously obtained via our VGG16 model and saved in order to increase efficiency.

Then, using create\_sequences(), we generate sequences of images, input sentences and output words for an image in order to train the model.

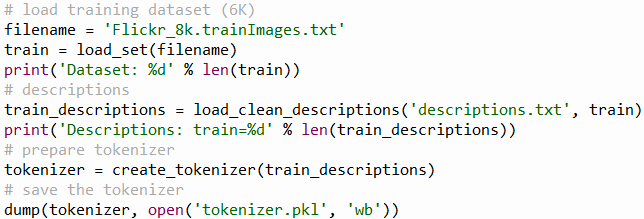
The define\_model() function is the same one outlined in the previous section. We use model.fit() to feed the training sequences into our model for several iterations (epochs), and save the resulting weights from each epoch using the ModelCheckpoint() function.

**5.5) Testing the Model:**

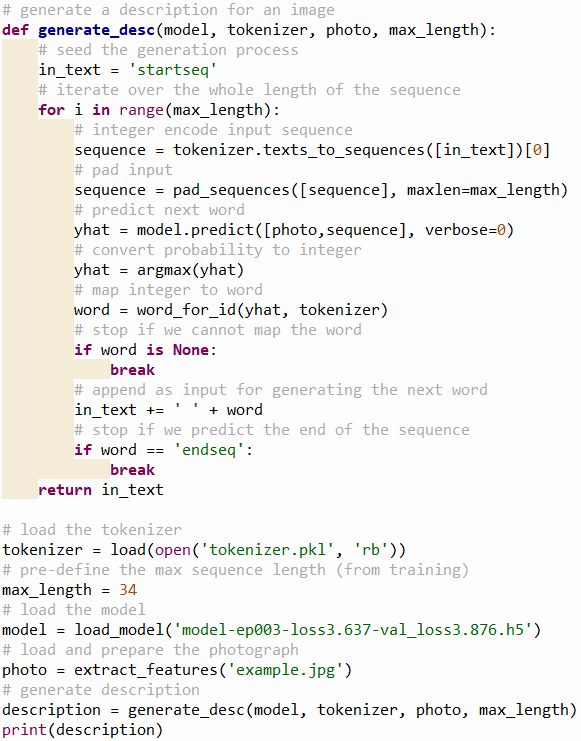


The NLTK library provides the corpus\_bleu() function, which calculates the BLEU score for multiple sentences or paragraphs. Here, we concatenate all the testing captions into one list, and our generated captions into another list, and feed those two lists into the corpus\_bleu() function.

**5.6) Generating Fresh Captions:**



Using this *tokenizer.pkl,* we can generate new captions without needing access to the training dataset’s list of captions.



The generate\_desc() function is an optimized function of our model which only requires the tokenizer file, the trained set of weights, and an input image to generate a new caption.

**RESULTS**

**FUTURE SCOPE**

Deep Learning (DL) has a big scope in the future and the biggest reason for it is that DL doesn’t require any kind of feature engineering. Deep Learning extracts the features from the data itself instead of us giving it the features after extracting it from the data. This way it solves our biggest problem of feature engineering. Also, since features are learned by the model itself, it has a better probability of producing a model which is more generalized than the feature engineered models. These reasons alone are sufficient to prefer DL over other technologies. With the recent development in DL, now we have more state-of-the-art results on various tasks including Natural Language Processing, Language Translation, Automatic Speech Recognition, Multi Label Image Recognition/Classification, Language generation among others, AI has grown more stronger in past 8 years with DL than it has developed in 20 years before that.

There are several ways to improve the accuracy of a model such as ours. Some of those methods are—

1. Using a larger dataset.
2. Changing the model architecture, e.g. include an attention module.
3. Doing more hyper parameter tuning (learning rate, batch size, number of layers, number of units, dropout rate, batch normalization etc.).
4. Using Beam Search instead of Greedy Search during Inference.

**CONCLUSION**

Neural networks are a very fundamental deep learning technique. In this overview, we have compiled all aspects of the image caption generation task, discussed the model framework proposed in recent years to solve the description task, focused on the algorithmic essence of different attention mechanisms, and summarized how the attention mechanism is applied. We summarize the large datasets and evaluation criteria commonly used in practice.

**REFERENCES**

1. A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2015.
2. H. Fang, S. Gupta, F. Iandola, R. K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt, et al. From captions to visual concepts and back. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2015.
3. P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2018.
4. H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei. Relation networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2018.