A REPORT
ON

IMAGE CAPTIONING AND VIDEO DESCRIPTION GENERATION

Name of the Student	ID Number
Priyanka Verma	***2016B3A70492P***

in partial fulfilment of

LABORATORY PROJECT (CS F366)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
NOVEMBER, 2019

ACKNOWLEDGEMENT

I would like to thank Prof Navneet Goyal for providing me with the opportunity to work on this project. He has constantly guided me through this study as my instructor for the Laboratory Oriented Project- CS F217. Without his constant support & guidance this work would not have been complete.

Table of Contents

1. INTRODUCTION

Classical video description approaches combined subject, object and verb detection with template based language models to generate sentences. Classical approaches were followed by a very short era of statistical methods which were soon replaced with deep learning, the current state of the art in video description Analysis of video description models is challenging because it is difficult to ascertain the contributions, towards accuracy or errors, of the visual features and the adopted language model in the final description. Existing datasets neither contain adequate visual diversity nor complexity of linguistic structures. Finally, current evaluation metrics fall short of measuring the agreement between machine generated descriptions with that of humans.

Deep learning methods have demonstrated state-of-the-art results on caption generation problems. What is most impressive about these methods is a single end-to-end model can be defined to predict a caption, given a photo, instead of requiring sophisticated data preparation or a pipeline of specifically designed models.

Neural Networks are used for various tasks to give better accuracy than traditional methods. There are various architectures possible like Convolution Neural Network , Recurrent Neural Networks, Reinforcement Learning, Generative Adversarial Network (GAN), etc which have been explored through this study. Automatically generating natural language sentences describing the video content has two components; understanding the visual content and describing it in grammatically correct natural language sentences.

The technical platform to implement it is TensorFlow using Keras, OpenCV and NLTK libraries.

1.1 AIM

The aim of this project is to design and implement image captioning and video description generation algorithms using deep neural networks.

1.2 PURPOSE

The purpose of this project is to develop Automated Test Equipment for an aerial flight vehicle, which can perform checkout in a repeatable, consistent and error-free manner.

1.3 Scope

Automatic video description has many applications in human-robot interaction, automatic video subtitling and video surveillance. It can be used to help the visually impaired by generating verbal descriptions of surroundings through speech synthesis, or automatically generating and reading out film descriptions. Currently, these are achieved through very costly and time-consuming manual processes. Another application is the description of sign language videos in natural language. Video description can also generate written procedures for human or service robots by automatically converting actions in a demonstration video into simple instructions, for example, assembling furniture, installing CD-ROM, making coffee or changing a flat tyre. The advancement of video description opens up enormous opportunities in many application domains. It is envisaged that in the near future, we would be able to interact with robots in the same manner as with humans

2. PREREQUISITES

This section provides an overview of the process involved. Automatically generating natural language sentences describing the video content has two components; understanding the visual content and describing it in grammatically correct natural language sentences.

2.1 Technical Specifications

Python SciPy environment, ideally with Python 3. Keras (2.2 or higher) is installed with the TensorFlow. Libraries such as scikit-learn, Pandas, NumPy, and Matplotlib are imported. The code is run on Google Collaboratories which satisfy GPU requirements.

Libraries requirement to map the GPU consumption :-

```
import psutil

import humanize

import os

import GPUtil as GPU
```

2.2 DATA

UCF101 it is an action recognition data set of realistic action videos, collected from YouTube. With 13320 videos from 101 action categories, it has the presence of large variations in camera motion, object appearance and pose, object scale, viewpoint, cluttered background, illumination conditions, etc. It is the most challenging data set to date as it has a **size** of **6.48 GiB .**

Due to the limited computation power in Google Colab, it was replaced by a smaller dataset.

Flickr8k dataset

It consists of 8,000 images that are each paired with five different captions which provide clear descriptions of the salient entities and events. The reason to select is because it is realistic and relatively small to download it and build models on your workstation using a GPU.

- **Flickr8k_Dataset.zip** (1 Gigabyte) An archive of all photographs.
- **Flickr8k_text.zip** (2.2 Megabytes) An archive of all text descriptions for photographs.

3. IMPLEMENTATION

3.1 PREPARE PHOTO DATA

3.1.1 Load Features

Use a pre-trained model to interpret the content of the photos like the Oxford Visual Geometry Group or VGG. Keras provides this pre-trained model directly. Pre-compute the “photo features” using the pre-trained model and save them to file. Then load these features later and feed them into our model as the interpretation of a given photo in the dataset.

Then reshape the loaded photo into the preferred size for the model (e.g. 3 channel 224 x 224-pixel image).

This is **an optimization** that makes **training the models faster** and consume less memory.

3.1.2 Extract Features

A function named *extract_features()* that, given a directory name, will load each photo, prepare it for VGG, and collect the predicted features from the VGG model. The image features are a 1-dimensional 4,096 element vector. The function returns a dictionary of image identifier to image features.

Call this function to prepare the photo data for testing models and save the resulting dictionary to a file named '*features.pkl*'.

The following code snippet shows this task:-

```
# extract features from each photo in the directory
def extract_features(directory):
    # load the model
    model = VGG16()
    # re-structure the model
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    # summarize
    print(model.summary())
    # extract features from each photo
    features = dict()
    for name in listdir(directory):
        # load an image from file
        filename = directory + '/' + name
        image = load_img(filename, target_size=(224, 224))
        # convert the image pixels to a numpy array
        image = img_to_array(image)
        # reshape data for the model
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        # prepare the image for the VGG model
        image = preprocess_input(image)
        # get features
        feature = model.predict(image, verbose=0)
        # get image id
        image_id = name.split('.')[0]
        # store feature
        features[image_id] = feature
        print('>%s' % name)
    return features
```

3.2 PREPARE TEXT DATA

The dataset contains multiple descriptions for each photograph and the text of the descriptions requires some minimal cleaning.

First, load the file containing all of the descriptions. Each photo has a unique identifier. This identifier is used on the photo filename and in the text file of descriptions.

Next, step through the list of photo descriptions. Below defines a function *load_descriptions()* that, given the loaded document text, will return a dictionary of photo identifiers to descriptions. Each photo identifier maps to a list of one or more textual descriptions.

clean the text in the following ways in order to reduce the size of the vocabulary of words we will need to work with:

- Convert all words to lowercase.
- Remove all punctuation.
- Remove all words that are one character or less in length (e.g. ‘a’).
- Remove all words with numbers in them.

Save the dictionary of image identifiers and descriptions to a new file named *descriptions.txt*, with one image identifier and description per line.

Below defines the *save_descriptions()* function that, given a dictionary containing the mapping of identifiers to descriptions and a filename, saves the mapping to file.

4. DEVELOP DEEP LEARNING MODEL

4.1 Loading Data

First, load the prepared photo and text data so to fit the model. Train the data on all of the photos and captions in the training dataset. While training, monitor the performance of the model on the development dataset and use that performance to decide when to save models to file.

The train and development dataset were predefined in the *Flickr_8k.trainImages.txt* and *Flickr_8k.devImages.txt* files respectively, that both contain lists of photo file names. From these file names, we can extract the photo identifiers and use these identifiers to filter photos and descriptions for each set.

The function *load_set()* loads a predefined set of identifiers given the train or development sets filename.

4.2 Defining the Model

The model is described in three parts:

- Photo Feature Extractor. This is a 16-layer VGG model pre-trained on the ImageNet dataset. Already pre-processed the photos with the VGG model (without the output layer) and use the extracted features predicted by this model as input. It expects input photo features to be a vector of 4,096 elements. These are processed by a Dense layer to produce a 256 element representation of the photo.

- Sequence Processor. This is a word embedding layer for handling the text input, followed by a Long Short-Term Memory (LSTM) recurrent neural network layer. It 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.

- Decoder : Both the feature extractor and sequence processor output a fixed-length vector. These are merged together and processed by a Dense layer to make a final prediction. 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.

The function below named *define_model()* defines and returns the model ready to be fit.

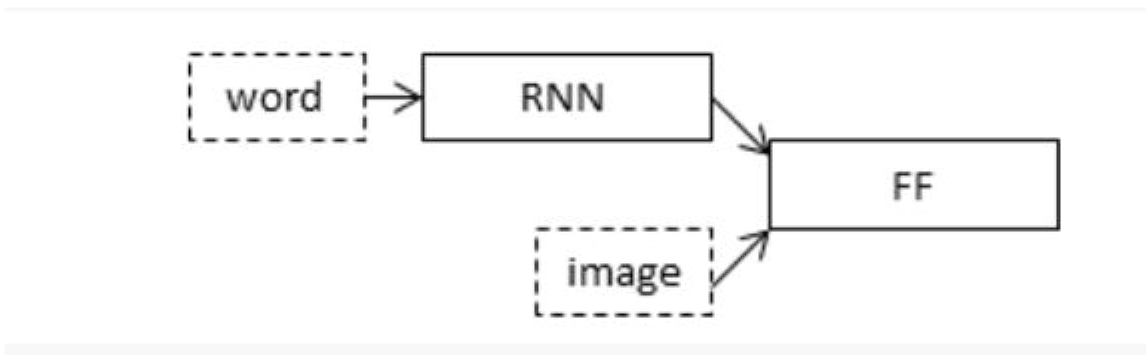
Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 34)	0	
input_1 (InputLayer)	(None, 4096)	0	
embedding_1 (Embedding)	(None, 34, 256)	1940224	input_2[0][0]
dropout_1 (Dropout)	(None, 4096)	0	input_1[0][0]
dropout_2 (Dropout)	(None, 34, 256)	0	embedding_1[0][0]
dense_1 (Dense)	(None, 256)	1048832	dropout_1[0][0]
lstm_1 (LSTM)	(None, 256)	525312	dropout_2[0][0]
add_1 (Add)	(None, 256)	0	dense_1[0][0] lstm_1[0][0]
dense_2 (Dense)	(None, 256)	65792	add_1[0][0]
dense_3 (Dense)	(None, 7579)	1947803	dense_2[0][0]

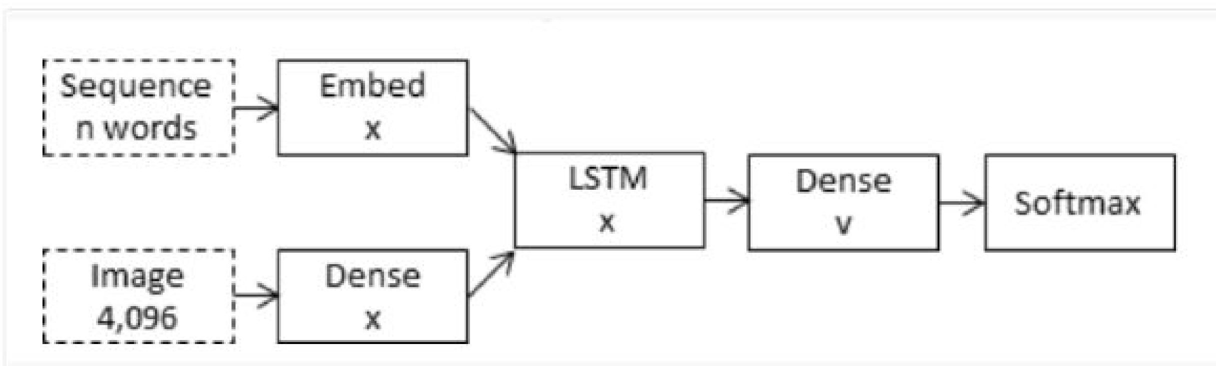
Total params: 5,527,963
Trainable params: 5,527,963
Non-trainable params: 0

4.3 Fitting the Model

The model is fit to 20 epochs, and given the amount of training data, each epoch took 20-30 minutes on GPU.

The underlying architecture is as shown in figures





4.3.1 Optimisation

The training of the caption model does assume you have a lot of RAM. The code in the previous section is not memory efficient.

Therefore, use progressive loading, to train this model. Let the data generator yield one photo's worth of data per batch. This will be all of the sequences generated for a photo and its set of descriptions.

The function below *data_generator()* will be the data generator and will take the loaded textual descriptions, photo features, tokenizer and max length. Essentially, we call the *create_sequence()* function to create a batch worth of data for a single photo rather than an entire dataset.

For the next step, use the *fit_generator()* function on the model to train the model with this data generator.

5. EVALUATE THE MODEL

Evaluate the model by generating descriptions for all photos in the test dataset and evaluating those predictions with a standard cost functio*n.*

First, generate a description for a photo using a trained model.

The function *generate_desc()* implements this behavior and generates a textual description given a trained model, and a given prepared photo as input. It calls the function *word_for_id()* in order to map an integer prediction back to a word.

Generate predictions for all photos in the test dataset and in the train dataset. The function below named evaluate_model() will evaluate a trained model against a given dataset of photo descriptions and photo features. The actual and predicted descriptions are collected and evaluated collectively using the corpus BLEU score that summarizes how close the generated text is to the expected text.

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

The NLTK Python library implements the BLEU score calculation in the corpus_bleu() function. A higher score close to 1.0 is better, a score closer to zero is worse.

```
model = load_model(filename)
# evaluate model
evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
```

```
Using TensorFlow backend.
The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %tensorflow_version 1.x magic: more info.
Dataset: 6000
Descriptions: train=6000
Vocabulary Size: 7579
Description Length: 34
Dataset: 1000
Descriptions: test=1000
Photos: test=1000
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/initializers.py:119: calling RandomUnifor
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling Base
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/backend.py:3994: where (from tensorflow.p
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
BLEU-1: 0.508636
BLEU-2: 0.262012
BLEU-3: 0.176529
BLEU-4: 0.077944
```

Generated Blue scores for Test Data :

BLEU-1: 0.508636

BLEU-2: 0.262012

BLEU-3: 0.176529

BLEU-4: 0.077944

6. CHALLENGES

6.1 Memory Constraints

Gen RAM Free: 9.0 GB I Proc size: 6.7 GB

GPU RAM Free: 244MB | Used: 14835MB | **Util 98%** | Total 15079 MB

```
import GPUtil as GPU
GPUs = GPU.getGPUs()
# XXX: only one GPU on Colab and isn't guaranteed
gpu = GPUs[0]

def printm():
  process = psutil.Process(os.getpid())
  print("Gen RAM Free: " + humanize.naturalsize( psutil.virtual_memory().available ), " I Proc size: " + humanize.natural
  print('GPU RAM Free: {0:.0f}MB | Used: {1:.0f}MB | Util {2:3.0f}% | Total {3:.0f}MB'.format(gpu.memoryFree, gpu.memoryU

printm()
```

```
Collecting gputil
  Downloading https://files.pythonhosted.org/packages/ed/0e/5c61eedde9f6c87713e89d794f01e378cfd9565847d4576fa627d758c554/
Building wheels for collected packages: gputil
  Building wheel for gputil (setup.py) ... done
  Created wheel for gputil: filename=GPUtil-1.4.0-cp36-none-any.whl size=7410 sha256=a0f940a42c25fa789382fff3f9dc6dce287a
  Stored in directory: /root/.cache/pip/wheels/3d/77/07/80562de4bb0786e5ea186911a2c831fdd0018bda69beab71fd
Successfully built gputil
Installing collected packages: gputil
Successfully installed gputil-1.4.0
Requirement already satisfied: psutil in /usr/local/lib/python3.6/dist-packages (5.4.8)
Requirement already satisfied: humanize in /usr/local/lib/python3.6/dist-packages (0.5.1)
Gen RAM Free: 9.0 GB  I Proc size: 6.7 GB
GPU RAM Free: 244MB | Used: 14835MB | Util  98% | Total 15079MB
```

98% of the available memory gets used up while testing 1 video and few images

6.2 Approaches to Memory Optimisation

1) Freeing up tensorflow graph session using the commands

   tf.keras.backend.clear_session()

   tf.reset_default_graph()

   This didn't help. This cleared the backend graphs but was unable to free up RAM memory.

2) Changing the image format (png, jpg, gif)

   No, it does not affect how an image recognition neural net is trained

3) Using the numpy image array captured from video instead of writing it as image and reading it again as array of pixels.

   i.e imwrite() function was removed from code and features were extracted directly from the pixel format of image.

   This helped in greatly reducing memory usage. And was able to work with longer videos of 3 minutes as well.

7. GENERATE NEW CAPTIONS

Use the 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 (e.g. 34).

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. An alternative would be to use our own vocabulary file and mapping to integers function during training.

Create the Tokenizer as before and save it as a pickle file tokenizer.pkl.

Generate a description using the *generate_desc()* function defined when evaluating the model.

7.1 RESULTS

Test 1

Output 1 =

baby with blonder hair is eating snack

```
photo = extract_features(directory +'happychild.jpg')

description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```

```
startseq baby with blonde hair is eating snack endseq
```

code snippet showing the output

Test 2

Output 2 =

Two dogs are running through the water

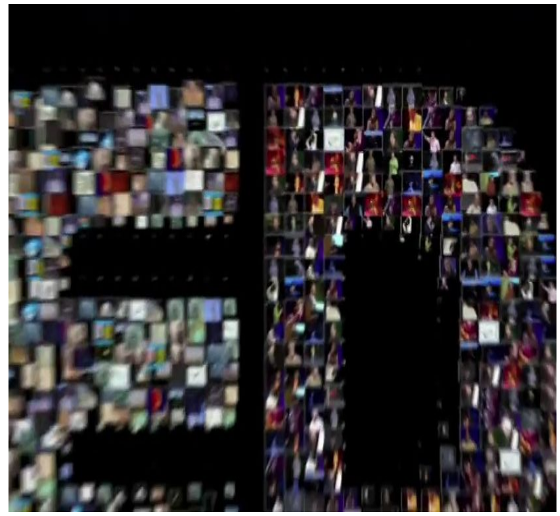
```
[ ] photo = extract_features(directory +'example.jpg')

    description = generate_desc(model, tokenizer, photo, max_le
    print(description)

    startseq two dogs are running through the water endseq
```

Code snippet showing the output

Test 3




```
[ ] print(total)

soccer ball in match endseq', 'startseq man in red shirt is standing in front of the street endseq', 'startseq man in red shirt is standing in front of the road
```

Output generated for the above shown TED Video

['startseq man in black shirt is standing in front of the street endseq',

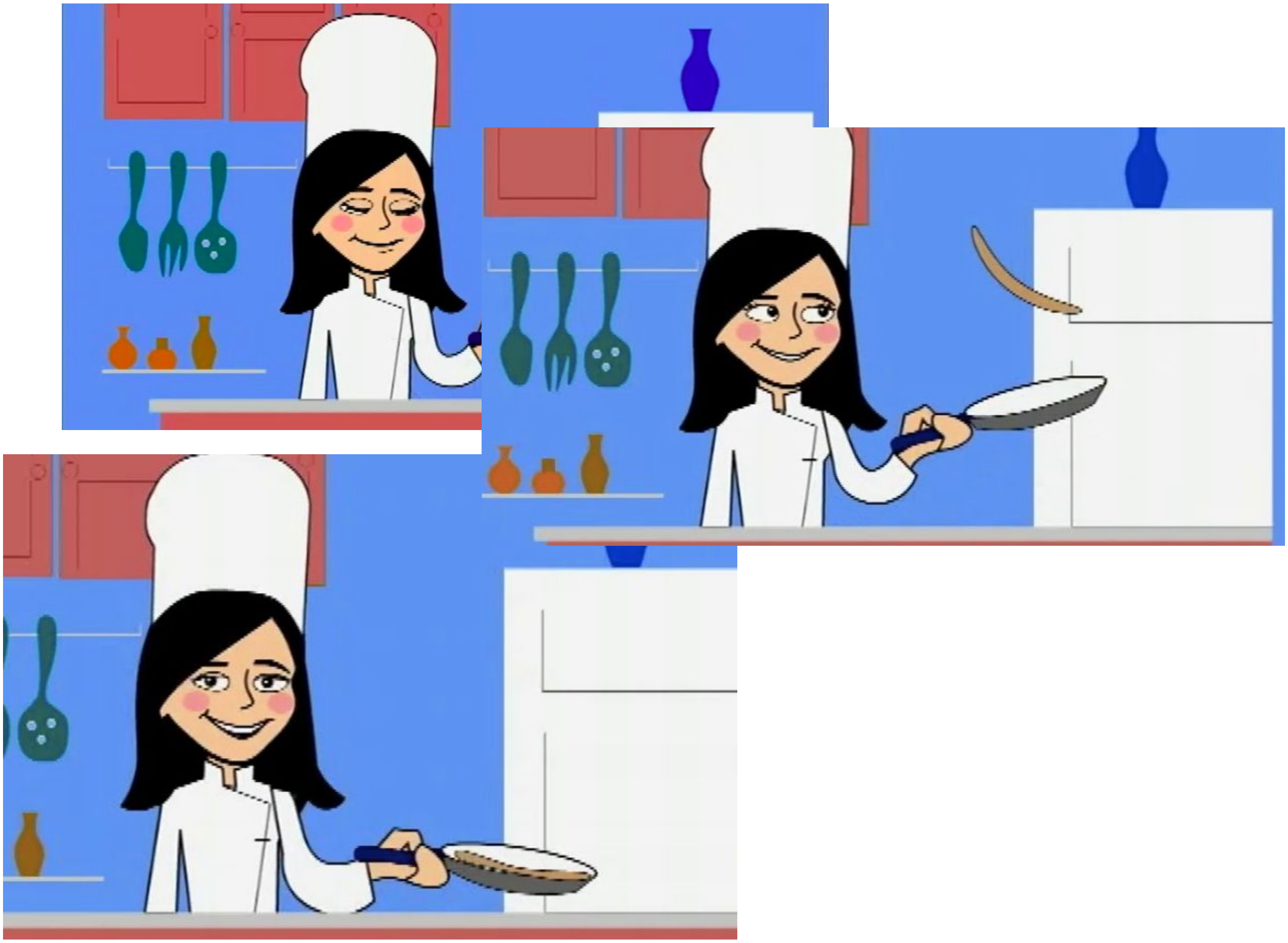
'startseq two children are playing with soccer ball in match endseq',

'startseq two children are playing with soccer ball in match endseq',

'startseq man in red shirt is standing in front of the street endseq',

'startseq man in red shirt is standing in front of the road endseq']

Test 4



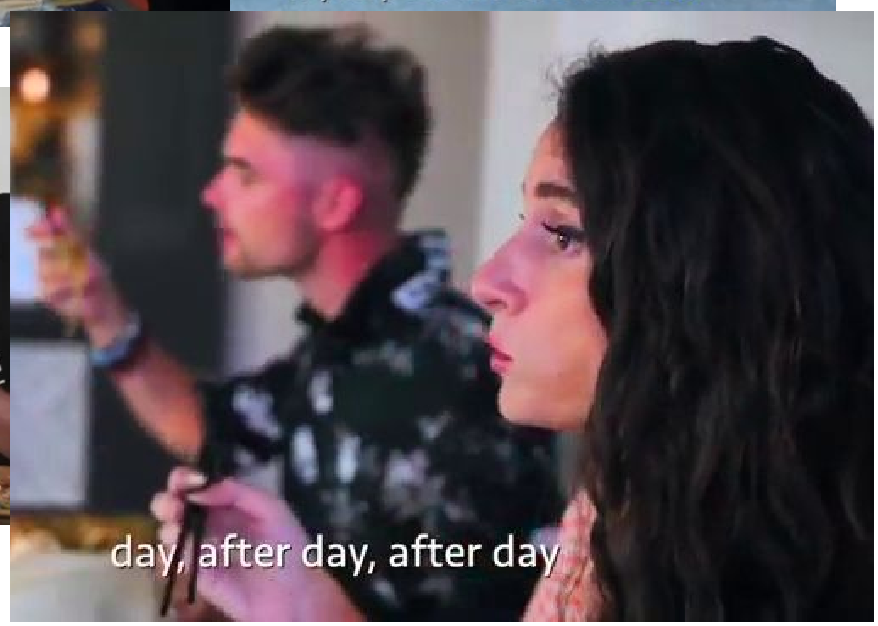
```
for i in range (0,240, 40) :
  photo = extract_features(directory + "vid3-%d.jpg"%i)
  description = generate_desc(model, tokenizer, photo, max_length)
  print(description)
  total.append(description)
```

```
startseq two girls are playing in the water endseq
startseq two girls are playing in the water endseq
startseq two girls are playing with birds in the grass endseq
startseq man in red shirt is standing in front of building endseq
startseq man in black shirt is standing in front of building endseq
startseq two girls are playing with recently funny endseq
```

Test 5


I gave in and changed my diet


If you pour water on a rock


day, after day, after day


And all my friends did too!

OUTPUT GENERATED

```
    photo = extract_features(directory + "acheck2-%d.jpg"%i)
    description = generate_desc(model, tokenizer, photo, max_length)
    print(description)
    im = cv2.imread(directory + "acheck2-%d.jpg"%i)
    cv2.putText(im, description, (100,500) , cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 3, cv2.LINE_AA)
    cv2.imwrite(directory + '1result%d.jpg'%i,im)
    total.append(description)
```

```
startseq man in black shirt is standing in front of building endseq
startseq two girls are sitting on the grass endseq
startseq two young girls are playing in the water endseq
startseq two girls are playing with birds in the grass endseq
startseq two girls are sitting on the sidewalk endseq
startseq man in red shirt is standing in front of the street endseq
startseq two girls are sitting on the grass endseq
```

startseq man in black shirt is standing in front of building endseq

startseq two girls are sitting on the grass endseq

startseq two young girls are playing in the water endseq

startseq two girls are playing with birds in the grass endseq

startseq two girls are sitting on the sidewalk endseq

startseq man in red shirt is standing in front of the street endseq

startseq two boys are sitting on the grass endseq

8. APPENDIX

The following links contain my codes which were implemented as part of this project

- Video Captioning
    - https://colab.research.google.com/drive/1E5Y3gtxqYQLQLR-qEUaPs5hxKtVp0Gsu#scrollTo=aj63mEvnANa9

- Manual Creation of Neural Network
    - https://colab.research.google.com/drive/1ZfL_huY4cEKR9ynEhX7ZAUmqrsRyOaEp#scrollTo=Ha9I1FKsBRXq

- Linear Regression
    - https://colab.research.google.com/drive/1uMq6pZTbtMA12MPcBVeZLUio_dH-E_87

- Prediction for Univariate Time Series using LSTM
    - https://colab.research.google.com/drive/1o_pHSBUG1BUqEtMb4Xr_IILmlr0WjYsT#scrollTo=sUaoBMixOSpd

- Prediction for Multivariate Time Series using LSTMs
    - https://colab.research.google.com/drive/1IfoGRjhtZuQupD1Vi7n9f3aRakqutpaP#scrollTo=ZPkYvk3s4HLi

- Classification of data using LinearClassifier
    - https://colab.research.google.com/drive/1p37tbdI4Qyy8sKoHvw19Ine6GHPVFfA_#scrollTo=h5_LVRWu4pfO

9. REFERENCES

https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5098501/

https://www.youtube.com/watch?v=jrAo6h580PU

http://www.cs.cmu.edu/~christos/PUBLICATIONS/kdd06DTA.pdf

http://aima.cs.berkeley.edu/~russell/papers/nips13-tensor.pdf

https://hackernoon.com/learning-ai-if-you-suck-at-math-p4-tensors-illustrated-with-cats-27f0002c9b32

https://stackoverflow.com/questions/44645849/what-is-the-shape-of-an-image-tensor-in-tensorflow

https://medium.com/tensorflow/introducing-ragged-tensors-ac301c31fd38

https://www.researchgate.net/publication/261483616_Non-homogeneous_spatial_filter_optimization_for_EEG-based_brain-computer_interfaces/figures?lo=1

http://www.cs.cmu.edu/~christos/TALKS/15-SIGMOD-tut/sigmod15-tut-note-part3.pdf

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://glassboxmedicine.com/2019/06/08/regularization-for-neural-networks-with-framingham-case-study/

https://www.sciencedirect.com/topics/engineering/multidimensional-signal-processing