CS512 Project Report

Team Members:
Harsh Vora - A20445400
Ninad Parikh – A20427382

AUTO IMAGE CAPTIONING

ABSTRACT:

Automatically describing the content of an image is a fundamental problem in artificial intelligence that connects computer vision and natural language processing. For doing that, the content of an image using properly formed English sentences is a very challenging task, but it could have great impact, for instance by helping visually impaired people better understand the content of images on the web.

In this paper, a generative model is presented based on a deep recurrent architecture that combines recent advances in computer vision and machine translation and that can be used to generate natural sentences describing an image. The model is trained to maximize the likelihood of the target description sentence given the training image.

PROBLEM STATEMENT:

Image Captioning can find its use in different scenarios of life, giving a boost to self-driving cars, aiding the blind and cctv camera for scenario detection and also for image search. This requires computer vision and language processing to work together. This can be achieved through utilizing the pretrained CNN models and a language encoder to generate captions for images.

PROPOSED SOLUTION:

To create an end-to-end system for the Neural Image Captioning problem. We use a pretrained CNN model (Inception v3) which is already pretrained on the ImageNet dataset. Using transfer learning, we use the last hidden layer for the input to the RNN model to decode and generate sentences.

IMPLEMENTATION DETAILS:

1. DATASET USED:

We use the Flicker8k Dataset for the implementation.

*The research paper uses the MS COCO dataset. However, since we don't have the infrastructure to run this massive dataset, we had a choice of selecting from the Flickr8k Dataset (containing 8k images) and Flickr30k (containing 30k images). To stay within the timeline of the project, we chose to implement using the Flickr8k Dataset.

The dataset contains the following:

- Flickr8k_Dataset.zip
  Contains 8000 images. These images are bifurcated as follows:
  Training Set – 6000 images
  Dev Set – 1000 images
  Test Set – 1000 images
- Flicker8k_text.zip
  Contains 8 files from which the most important file is "Flickr8k.token.txt".
  Th file contains the name of each image along with its 5 captions.

```
[ ] # load doc into memory
    def load_doc(filename):
      # open the file as read only
      file = open(filename, 'r')
      # read all text
      text = file.read()
      # close the file
      file.close()
      return text

    filename = "Flickr8k.token.txt"
    # load descriptions
    doc = load_doc(filename)
    print(doc[:300])
```

```
1000268201_693b08cb0e.jpg#0	A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg#1	A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2	A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3	A little girl climbing the s
```

Figure1: Flickr8k.token.txt output

2. DATA PREPROCESSING:

The following steps were carried out in the preprocessing step:

- Data Cleaning: Lower-cased all the words (e.g. "World" becomes "world"), removed punctuations and special tokens (%, $, etc.) and words with numbers ("T12").

- Creating a Vocabulary: In this, all the unique words occurring in the data file (8000x5 captions) is used to create a vocabulary file. We write these captions along with their image names in the new file named "descriptions.txt". We limit the words that occur at least 10 times in the dataset.

Total vocabulary list: 8763.
After limiting the words that occur at least 10 times: 1651.
After appending 0 it becomes 1652

- Wrap descriptions in tokens: We add the words startseq and endseq to the start and end of the caption from the file descriptions.txt.

- Caption data-preprocessing: We create indexes for each unique word in the vocabulary. We then convert each caption into a list of tokens.

- Data generation: We also calculate the maximum length of a caption. The maximum length of a caption is 32. We will then pad the captions in case the length is less than 32, so that all the captions will have equal length.

- Image Data-preprocessing: Convert image to (299,299,3) for using it as input for Inception v3.

3. FEATURE EXTRACTION:

- We will use the Inception v3 model extracting the feature of the images.
- The model will provide us with a 2048 feature vector for each image.
- We will store the image file names and the feature vector in a pickle file named "encoded_train_images.pkl".

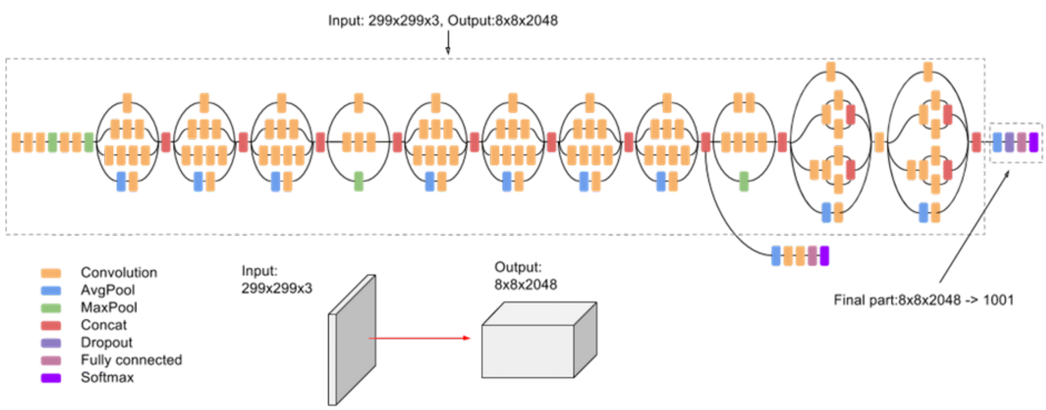


Figure2: Inception v3 model (ref: https://cloud.google.com/tpu/docs/inception-v3-advanced)

- We will map every word in the vocabulary to a 200 long vector using a pretrained GLOVE model to create an embedding matrix to load into the model before training.
- Each sequence in the caption will now have 32 by 200 long vector which will then be used with the 2048 length vector of the image.

4. MODEL BUILD AND TRAINING:

We build the following model:

```
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
```

```
model.summary()
```

```
Model: "functional_11"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_10 (InputLayer)           [(None, 34)]         0
__________________________________________________________________________________________________
input_9 (InputLayer)            [(None, 2048)]       0
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 34, 200)      330400      input_10[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 2048)         0           input_9[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 34, 200)      0           embedding_3[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 256)          524544      dropout_6[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   (None, 256)          467968      dropout_7[0][0]
__________________________________________________________________________________________________
add_3 (Add)                     (None, 256)          0           dense_9[0][0]
                                                                 lstm_3[0][0]
__________________________________________________________________________________________________
dense_10 (Dense)                (None, 256)          65792       add_3[0][0]
__________________________________________________________________________________________________
dense_11 (Dense)                (None, 1652)         424564      dense_10[0][0]
==================================================================================================
Total params: 1,813,268
Trainable params: 1,813,268
Non-trainable params: 0
```

Figure3: Model Summary

We use the following parameters for the model:

- We set the embedding layer weights to the embedding matrix.
- We freeze the embedding layer since we are using the given embedding matrix (trainable = False) so that it does not get updated during back-propogation.

- Loss = categorical_crossentropy
- Optimizer = adam
- Epochs = 20
- Number of pics per bath = 3
- Steps: len(training_descriptions)//number_pics_per_bath

We will give two inputs to the model, Input 1: Partial Caption and Input 2: Image Feature Vector. This will give us an output of an appropriate word next in the sequence of partial caption provided in input 1.

Hyper-parameter tuning after 20 epochs:

- Learning rate = 0.0001
- Epoch = 10
- Number of pics per bath = 3
- Steps: len(training_descriptions)//number_pics_per_bath

We reduce the learning rate so that the model takes lower step during the last epochs for converging. We increase the batch size because overtime helps your gradient updates to be more powerful.

```
] ! mkdir model_weights
  for i in range(epochs):
      generator = data_generator(train_descriptions, train_features, wordtoix, max_length, number_pics_per_bath)
      model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
      model.save('model_weights/model_' + str(i) + '.h5')
```

```
2000/2000 [==============================] - 156s 78ms/step - loss: 4.1290
2000/2000 [==============================] - 152s 76ms/step - loss: 3.4180
2000/2000 [==============================] - 152s 76ms/step - loss: 3.1955
2000/2000 [==============================] - 151s 75ms/step - loss: 3.0633
2000/2000 [==============================] - 151s 76ms/step - loss: 2.9691
2000/2000 [==============================] - 150s 75ms/step - loss: 2.8943
2000/2000 [==============================] - 154s 77ms/step - loss: 2.8389
2000/2000 [==============================] - 154s 77ms/step - loss: 2.7900
2000/2000 [==============================] - 154s 77ms/step - loss: 2.7514
2000/2000 [==============================] - 153s 77ms/step - loss: 2.7166
```

```
] for i in range(epochs):
      generator = data_generator(train_descriptions, train_features, wordtoix, max_length, number_pics_per_bath)
      model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
      model.save('model_weights/model_' + str(i) + '.h5')
```

```
2000/2000 [==============================] - 153s 77ms/step - loss: 2.6827
2000/2000 [==============================] - 153s 76ms/step - loss: 2.6586
2000/2000 [==============================] - 152s 76ms/step - loss: 2.6348
2000/2000 [==============================] - 151s 75ms/step - loss: 2.6180
2000/2000 [==============================] - 153s 77ms/step - loss: 2.5963
2000/2000 [==============================] - 154s 77ms/step - loss: 2.5780
2000/2000 [==============================] - 153s 77ms/step - loss: 2.5644
2000/2000 [==============================] - 152s 76ms/step - loss: 2.5495
2000/2000 [==============================] - 152s 76ms/step - loss: 2.5394
2000/2000 [==============================] - 151s 75ms/step - loss: 2.5240
```

```
] model.load_weights('model_weights/model_9.h5')
```

```
] model.optimizer.lr = 0.0001
  epochs = 10
  number_pics_per_bath = 6
  steps = len(train_descriptions)//number_pics_per_bath
```

```
] for i in range(epochs):
      generator = data_generator(train_descriptions, train_features, wordtoix, max_length, number_pics_per_bath)
      model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
      #model.save('./model_weights/model_' + str(i) + '.h5')
```

```
1000/1000 [==============================] - 82s 82ms/step - loss: 2.4810
1000/1000 [==============================] - 81s 81ms/step - loss: 2.4325
1000/1000 [==============================] - 81s 81ms/step - loss: 2.4152
1000/1000 [==============================] - 80s 80ms/step - loss: 2.4042
1000/1000 [==============================] - 81s 81ms/step - loss: 2.3925
1000/1000 [==============================] - 81s 81ms/step - loss: 2.3831
1000/1000 [==============================] - 81s 81ms/step - loss: 2.3765
1000/1000 [==============================] - 81s 81ms/step - loss: 2.3712
1000/1000 [==============================] - 81s 81ms/step - loss: 2.3634
1000/1000 [==============================] - 81s 81ms/step - loss: 2.3605
```

```
] model.save_weights('model_weights/model_30.h5')
```

Figure 4: Model Training

5. MODEL PREDICTION:

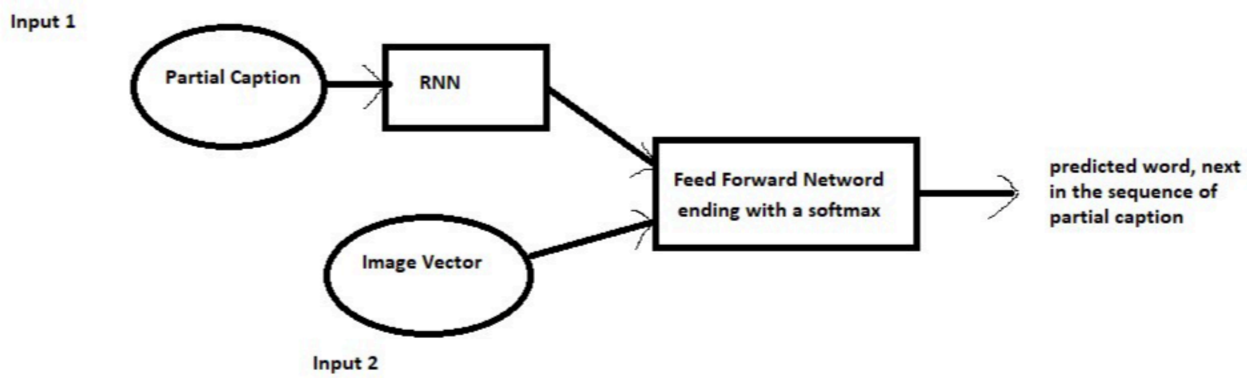
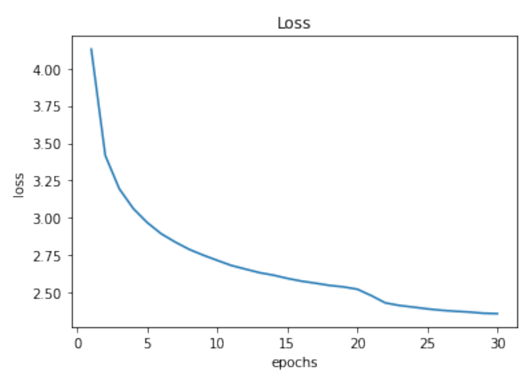


Figure 5: Prediction Model

During prediction, the model takes the image vector and the partial caption, the as input and predicts a word, next in the sequence of partial of caption. The next word prediction is done greedily, selecting the maximum probability given the feature vector and partial caption. This is also called maximum likelihood estimation, and this is done until the model generates a vector of size 12 or when the model comes across an endseq token.

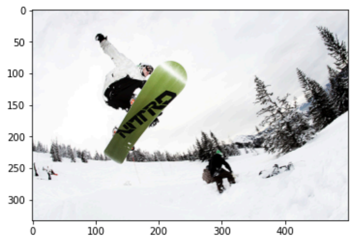
RESULTS AND DISCUSSION:

After training the model, we get the below loss for each epoch. We see that at epoch 20, there is sudden drop in the loss, this is because we changed the hyper-parameters after 20 epochs. We lowered the learning rate and increased the batch size which led to a decrease in the loss at the 20th epoch.



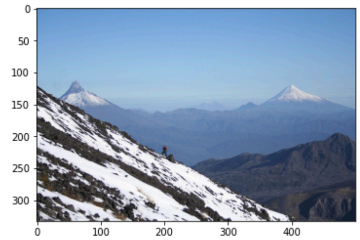
Correct Predictions:

```
[69] z = 2
     pic = list(encoding_test.keys())[z]
     image = encoding_test[pic].reshape((1,2048))
     x=plt.imread(images+pic)
     plt.imshow(x)
     plt.show()
     print("Greedy:",greedySearch(image))
```



Greedy: skier is performing jump on his snowboard

```
z = 3
pic = list(encoding_test.keys())[z]
image = encoding_test[pic].reshape((1,2048))
x=plt.imread(images+pic)
plt.imshow(x)
plt.show()
print("Greedy:",greedySearch(image))
```



Greedy: man in red shirt is standing on top of mountain

```
] z = 4
  pic = list(encoding_test.keys())[z]
  image = encoding_test[pic].reshape((1,2048))
  x=plt.imread(images+pic)
  plt.imshow(x)
  plt.show()
  print("Greedy:",greedySearch(image))
```

Greedy: white dog is running on the grass

```
[72] z = 50
     pic = list(encoding_test.keys())[z]
     image = encoding_test[pic].reshape((1,2048))
     x=plt.imread(images+pic)
     plt.imshow(x)
     plt.show()
     print("Greedy:",greedySearch(image))
```

Greedy: man kayaking through rapids

Incorrect Predictions:

```
z = 1
pic = list(encoding_test.keys())[z]
image = encoding_test[pic].reshape((1,2048))
x=plt.imread(images+pic)
plt.imshow(x)
plt.show()
print("Greedy:",greedySearch(image))
```

Greedy: two children are playing on playground

CONCLUSION:

The model performs well for the given test images and as we can see the from the generated captions, the captions are semantically correct and make sense with respect to the picture. After evaluating the predictions on the test image, we find that the captions generated may not always be similar to the intended image and there may be incorrect predictions. This solution is basic and can be improved upon with different methods. Having a larger dataset would definitely be helpful given more time and infrastructure. Also, hyper-parameter tuning would also improve the results.

REFERENCES:

https://arxiv.org/pdf/1609.06647v1.pdf

https://ieeexplore.ieee.org/abstract/document/8990998

https://papers.nips.cc/paper/2019/file/680390c55bbd9ce416d1d69a9ab4760d-Paper.pdf