Image Retrieval Using Caption Generator

Malvika Mohan, Meghana Parab, Prathamesh Mahankal

Research Question:

"How can we use automatic image captioning to enable on-the-go image retrieval with the potential to improve usability and accessibility in smartphone image gallery?"

Previous Work:

Solutions for the Image Captioning problem were considered inconceivable by researchers until the development of Deep Neural Networks. Advancements in Deep Learning has made it relatively easier to build Image Captioning solutions.

To generate captions from images, we first needed to understand how to extract features from images. Previous work in this area guided us towards the Inceptionv3 model which is a pre-trained image classification convolutional neural network trained on 1 million images of the ImageNet dataset. Krizhevsky et. al. (2017) in his paper on "Image Net Classification with Deep CNN", details how his team trained the ImageNet dataset with a neural network that consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax.

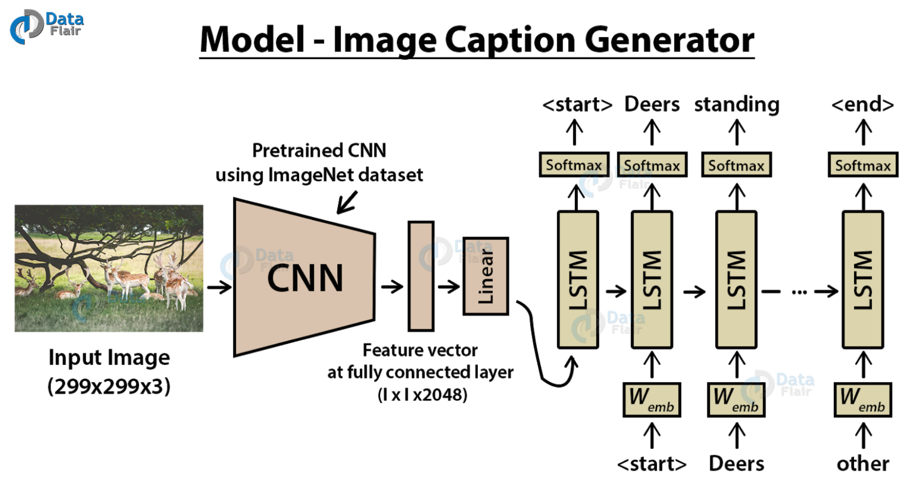
We also looked into Keras's in-built InceptionV3 function which instantiates the InceptionV3 architecture for you. Keras' official documentation on InceptionV3 also points to Szegedy et. al.'s (2016) paper on Rethinking the Inception Architecture for Computer Vision. The report a 3.6% error on the test set using factorized convolutions and aggressive regularization that enhance computational efficiency. Thus, we opted to use Keras' InceptionV3 model for our project.

Next, we wanted to understand how captions can be generated automatically using neural networks. For this, we looked into papers that have studied different types of neural network architectures that work best for this problem.

We looked into Andrej Karpathy's paper on "Deep Visual-Semantic Alignments for Generating Image Descriptions" which is highly recognized in this field. It presents a model that generates natural language descriptions of images and their regions by leveraging datasets of images and their sentence descriptions to learn about the inter-modal correspondences between language and visual data. This paper is highly relevant to our project as it focuses on richer and higher-level descriptions of regions as compared to other research work that we explored in this area that was focused on holistic scene understanding. This paper uses a much simpler Recurrent Neural Network (RNN) model than complicated ones used in other research work including “Explain Images with Multimodal Recurrent Neural Networks” (Mao, 2014) and “A neural image caption generator” (Vinyals, 2014). The simpler model does consequently suffer in performance but is sufficient for the scope of this project.

To build an RNN from scratch, we needed to understand how our model can figure out what words occur after each other. “GloVe: Global Vectors for Word Representation” (Pennington, 2014) helped us figure out the embeddings model we would use for the partial caption generator part of our project.

We conclude our previous work research with the understanding that a combination of CNN-RNN which includes modifications to a pre-trained CNN, and an RNN built from scratch by us, will be the best way to approach our project.



Motivation

The primary motivation of our project is to generate descriptive captions for images which can subsequently be used for image retrieval using a keyword search. Our project will find its application in mobile devices where users can enter one keyword such as “food” or multiple keywords such as “sports” and “football” and the deep learning model will return all relevant images consisting of these keywords in their generated caption. The project aims to improve the usability of image galleries where users can instantly retrieve images of their interest without manually scrolling through large galleries. Our model can also be used to generate captions for new images which could lay the foundation for applications like image tagging allowing faster retrieval of images through the capture of image metadata and album classification.

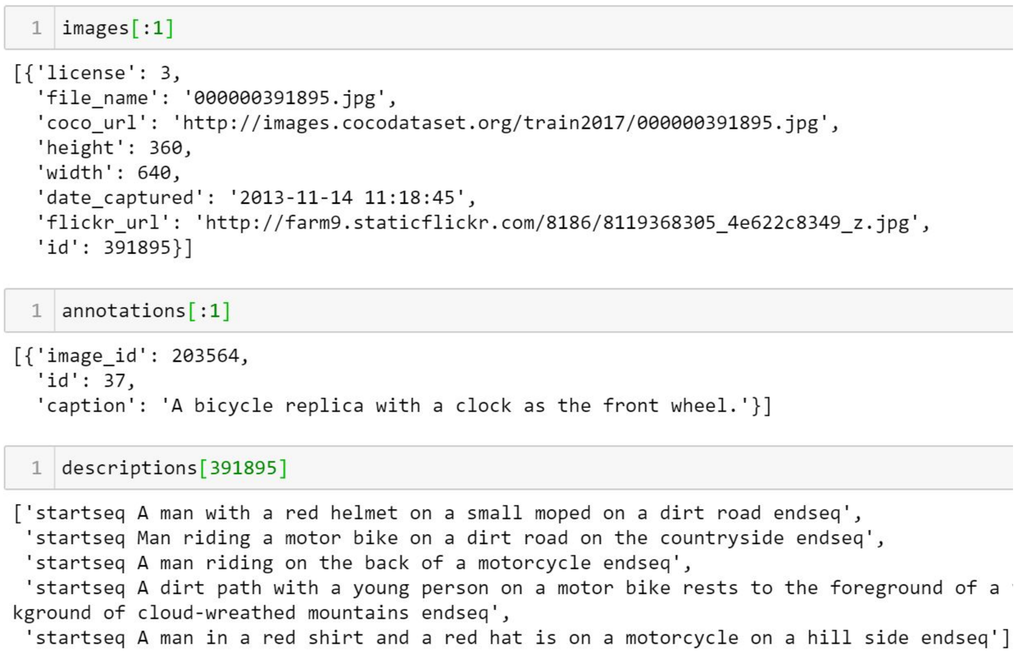
Project Deliverables:

1. Creating descriptive captions for each image in our images gallery.
2. A “Search” option that takes user input (for eg. ‘dog on desk’) and returns all images from the images gallery whose captions contain those particular words.

Data Description

To tackle this problem of image captioning, there are many open-source datasets available on the internet like Flickr 8k, Flickr 30k, and MS COCO dataset. Of these, we used the MS COCO dataset purely because of the number of images it had. MS COCO is large-scale object detection, segmentation, and captioning dataset. This dataset has around 118k images in total, and each image has five unique captions associated with it. Thus, our dataset was clearly unusually large, and thus we had to **randomly sample** 50k images of these 118k images for training (we couldn’t sample more images due to computational restrictions). For validation, we used the MS COCO validation data which has 5k images.

After downloading the dataset, we started trying to understand the data. We saw our training JSON file contains multi-level data. One part of the JSON file contains information about the images, while the other part is the information about the captions. As mentioned earlier, every image has five captions associated with it. We thus built a description dictionary with image id as the key and a list of five descriptions for that image as the value. Here is what our dictionary looks like.



```
images[:1]
```

```
[{'license': 3,
  'file_name': '000000391895.jpg',
  'coco_url': 'http://images.cocodataset.org/train2017/000000391895.jpg',
  'height': 360,
  'width': 640,
  'date_captured': '2013-11-14 11:18:45',
  'flickr_url': 'http://farm9.staticflickr.com/8186/8119368305_4e622c8349_z.jpg',
  'id': 391895}]
```

```
annotations[:1]
```

```
[{'image_id': 203564,
  'id': 37,
  'caption': 'A bicycle replica with a clock as the front wheel.'}]
```

```
descriptions[391895]
```

```
['startseq A man with a red helmet on a small moped on a dirt road endseq',
 'startseq Man riding a motor bike on a dirt road on the countryside endseq',
 'startseq A man riding on the back of a motorcycle endseq',
 'startseq A dirt path with a young person on a motor bike rests to the foreground of a
kground of cloud-wreathed mountains endseq',
 'startseq A man in a red shirt and a red hat is on a motorcycle on a hill side endseq']
```

Once the description dictionary was ready, we started cleaning the individual descriptions. First, we checked our data for any nulls. Next, we tokenized the individual descriptions and converted all the tokens to lowercase. Next, we created a vocabulary of all the unique words in the descriptions and also calculated the maximum length one individual description has (the importance of this step will be explained later in the report). Once we had a cleaned dictionary with the image IDs as keys and a list of their respective descriptions as values, we proceeded towards the analysis step.

Handling An Unusually Large Dataset By Leveraging Cloud Platform Services

Since we wished to train our model on a dataset of 50k images and as each image had 5 captions associated with it, we needed to leverage cloud platform services to extract features and to train our model on these 50k images and a total of 250k captions. We initially chose to tackle this problem by creating a virtual machine on Microsoft Azure that consisted of 2 CPUs, 8GB RAM, and 4 data disks. However, due to the lack of GPU computing capabilities in the system, we faced issues as the virtual machine kept crashing while running our model due to which we needed to restart the system several times and rerun our code. In order to tackle this problem, we needed to find another alternative that could support the processing of our large dataset. Hence we decided to shift to the Google cloud platform where we created a virtual machine consisting of 4 CPUs,15GB memory, and an NVIDIA Tesla V100 GPU processor. We observed that on the addition of GPU computing capabilities our model was able to run much faster and we were able to work smoothly with our large dataset.

Preprocessing Images

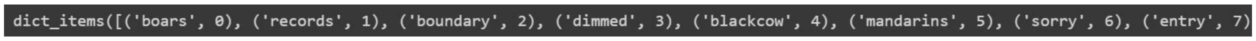
The input for a neural network needs to be a vector. Thus, we need to convert our image into a fixed-length vector which can then be given to our model as an input. To make this possible, we resorted to the concept of Transfer Learning using a Convolutional Neural Network model called InceptionV3. To define Transfer learning, I will use an excerpt from Brownlee, J.'s article named A Gentle Introduction to Transfer Learning for Deep Learning. It defines Transfer Learning as a machine learning method where a model developed for a particular task is reused as the starting point for a model on a second task. It is a technique where pre-trained models are used as the starting point on natural language processing tasks so that one doesn't have to dedicate separate compute and time resources in order to build those starting models right from scratch.

The pre-trained model we used, InceptionV3, is a 48 layer deep convolutional neural network which has been trained on more than a million images from the ImageNet database. This pre-trained network takes in an image input size of 299-by-299 and classifies the input image into one of 1000 unique object categories using a softmax layer. However, the task that we are performing doesn't involve classification. But what we were focusing on was the 2048-length vector that is created in the second last layer of the InceptionV3 network. To obtain this 2048 length vector, we remove the last layer in our neural network so that we get the required vector from the second last layer of our model. This vector is the first feature of our dataset.

Preprocessing Text

Our vocabulary consisted of 18311 words, of which there are some words that do not occur very frequently. We can thus remove such words from our vocabulary. To do this, we will set a threshold for up to how many times the word should occur in order for it to be a part of our vocabulary. Moreover, we also check the maximum length of our descriptions. In our data, the maximum length for a description is **words**.

Next, we need to uniquely identify all the words in our vocabulary. For this, we encode every word into a fixed-length vector. To make this possible, we will create two dictionaries - one to convert a word into an index and the other to convert an index back to the word. They are called wordtoix and ixtoword respectively.



```
dict_items([('boars', 0), ('records', 1), ('boundary', 2), ('dimmed', 3), ('blackcow', 4), ('mandarins', 5), ('sorry', 6), ('entry', 7)
```

In addition to creating a description dictionary, we have also added two tokens in every caption. First, we added a start sequence token with the name ‘startseq’ at the start of every caption. Next, we added an end sequence token at the end of every caption with the name ‘endseq’.

Generator Functions

A Generator function allows us to create a function that works as an iterator. Given that our training data is very large (50k images) along with their 250k captions (5 captions per image), training the data all at once will require a lot of computational resources. Thus, we define generator functions that utilizes SGD computation on each batch rather than entire training data at once. In this way, training efficiency is improved by loading data batch wise. We define two generator functions with identical inputs:

1. Data generator: for training data
2. Validation generator: for validation data

Inputs to this function are:

- **Descriptions:** a pre-processed and cleaned dictionary of every image ID and its corresponding captions
- **Photos:** A 2048 size image feature vector for every image in our gallery
- **Wordtoix dictionary:** A dictionary with every unique word in our vocabulary and its corresponding index.
- **Max_length:** The maximum length of any caption in our entire captions set
- **Num_photos_per_batch:** The number of photos we want to process per batch. We have set this value to 5.

The function will return the output sequence of words (or caption) that is predicted by the model. These two functions will further be given as inputs to the model.fit_generator() function that we use to actually train our model.

Word Embeddings

In order to provide fixed length inputs to our model, we embed every word in our vocabulary to a 200 length fixed vector using a pre-trained Global Vectors for Word Representation (GLOVE) model. This basically tells us the probabilities of two words occurring together in text. If we have one word, using these probabilities we can guess what the next word in our image caption could be. The GLOVE model assigns weights to each word depending on word to word co-occurrence probabilities. We create an embedded matrix of size (n,m) where n is the number of unique words in our vocabulary, and m is 200 (the length of the fixed-sized vector we embed each of the words into). We use this embedded matrix to set weights for our partial caption generator model. Setting these probabilities as weights to our model ensures that given a word the model predicts the next word that is the most likely to occur in the english language.

Model Architecture

The model we are creating is a combination of two neural networks - one for image feature extraction(CNN) and one for caption generation(RNN). Since our model takes two inputs, we use the Keras functional API to create a merge model of these two neural networks.

1. Convolutional neural network - We are using the pre-trained model called Inception V3 by Google. We removed the last layer of this model and used it only to extract features for our images. We then converted the features obtained for each image into a fixed-sized vector of 2048 length that we provide as an input to our model.

   We add the following layers to our model for processing the images :

   a. Input_10 - This is the input layer for our image processing model. The size of this layer is of 2048 length as we pass our image feature vector through this layer.
   b. Dropout_6 - We added a regularization layer with a dropout rate of 0.5 to prevent overfitting of the model. This layer takes input from the input_10 layer.
   c. Dense_7 - We added a fully connected Dense layer consisting of 256 neurons which take the input from the dropout_6 layer.

2. Recurrent neural network - We are implementing the recurrent neural network by creating a Long Short Term memory layer which we use to process our partial captions from the previous state. We add weights to this model that are obtained from the word embedding matrix we created as described previously. The model keeps predicting the next word in the partial

caption until an end sequence is reached or until the maximum length of the caption is reached.

We add the following layers to our model to process the partial caption generated:

a. Input_11 - This is the input layer of our partial caption generation model. The size of this layer is equal to the length of the longest caption in our training dataset (50).
b. Embedding_2 - We added an embedding layer that takes the input from the input_11 layer and maps the index of each word in our vocabulary to a 200 fixed length vector using the GloVe word embedding described previously. The size of this layer is equal to the size of our word embedding matrix.
c. Dropout_7 - We then added a dropout regularization layer with a dropout rate of 0.5 to prevent overfitting of the model. This layer applies a dropout to our embedding layer which is then taken as input to our next later (lstm_2).
d. LSTM_2 - The outputs from the dropout layer are then passed on to the lstm layer which is a specialized kind of recurrent neural network cell that is capable of learning long term dependencies. We define this layer to have 256 neurons and we use it to process our partial captions.

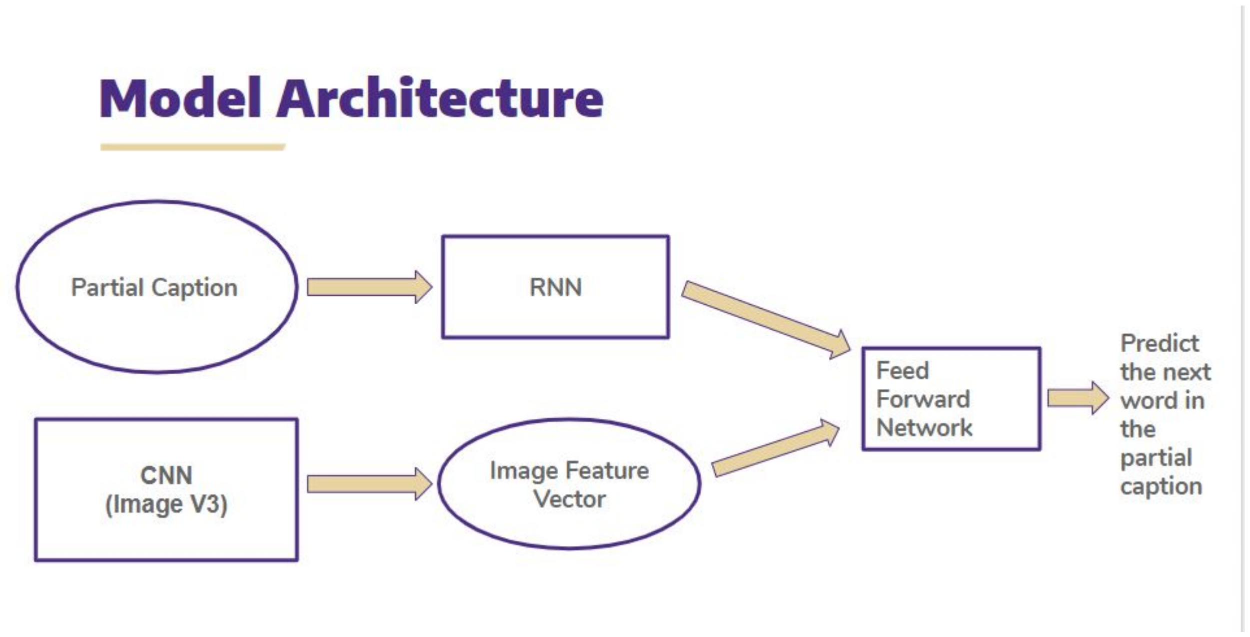
We created three additional layers in order to merge the outputs from the image processing network and caption generation network as follows :

a. Add_2 - Since the shape of the output layers of both our image processing network and caption generation network are the same (256,), we use this layer to merge the output layers from both the networks into a single layer using tensor addition.
b. Dense_8 - We add another dense layer of the same size for more connectivity
c. Dense_9 - This is our output layer which generates the probability distribution across all the words in our vocabulary.

Below is the model summary:

```
____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
input_11 (InputLayer)           (None, 50)           0
____________________________________________________________________________________________________
input_10 (InputLayer)           (None, 2048)         0
____________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 50, 200)      3662200     input_11[0][0]
____________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 2048)         0           input_10[0][0]
____________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 50, 200)      0           embedding_2[0][0]
____________________________________________________________________________________________________
dense_7 (Dense)                 (None, 256)          524544      dropout_6[0][0]
____________________________________________________________________________________________________
lstm_2 (LSTM)                   (None, 256)          467968      dropout_7[0][0]
____________________________________________________________________________________________________
add_2 (Add)                     (None, 256)          0           dense_7[0][0]
                                                                 lstm_2[0][0]
____________________________________________________________________________________________________
dense_8 (Dense)                 (None, 256)          65792       add_2[0][0]
____________________________________________________________________________________________________
dense_9 (Dense)                 (None, 18311)        4705927     dense_8[0][0]
====================================================================================================
Total params: 9,426,431
Trainable params: 9,426,431
Non-trainable params: 0
____________________________________________________________________________________________________
```

Model architecture blueprint



BLEU Score Metric

In order to evaluate the performance of our model, we needed a metric that could evaluate the captions our model generated. Hence we used a metric provided by the NLTK package called **Bilingual Evaluation Understudy** abbreviated as BLEU that compares a model's predicted translation of the text to one or more reference translators. The model uses a modified form of precision to compare the model's generated caption against one or more reference captions. The modified form of precision the metric uses is that for each word in the predicted caption, the metric's algorithm takes a maximum total count ($\mathbf{m}_{\mathbf{max}}$) for that word in any of the reference captions (Wikipedia contributors, 2020). The metric's algorithm calculates the maximum count ($\mathbf{m}_{\mathbf{max}}$) for each word (**w**) in the predicted caption. It then calculates the final BLEU score by summing up the value of ($\mathbf{m}_{\mathbf{max}}$) for each **unique** word in the predicted caption and divides this by the total number of words present in the predicted caption ($\mathbf{m}_{\mathbf{w}}$).
In order to illustrate the calculation of BLEU score we can take a look at the following example :
Predicted Caption: **'two' 'horses' 'standing' 'in' 'field' 'with' 'trees' 'in' 'the' 'background'**
Reference Captions :

1. **'A' 'bunch' 'of' 'people' 'getting' 'geared' 'up' 'with' 'horses'**
2. '**A' 'couple' 'of' 'horses' 'are' 'standing' 'in' 'a' 'field'**

We can see that the word "horses" in the predicted caption appears once in the first reference caption and once in the second reference caption. Hence the maximum total count ($\mathbf{m}_{\mathbf{max}}$) for the word is calculated as the max(1,1) which is 1. Similarly, we can calculate $\mathbf{m}_{\mathbf{max}}$ for the remaining words that come to be 1 for the words "standing", "in", "field" and "with" and 0 for the rest of the words in the predicted caption as they do not appear in either of the reference captions.

We can finally calculate the final BLEU score as (sum of $\mathbf{m}_{\mathbf{max}}$ for each unique word in the predicted caption) / (total number of words in the predicted caption) = 5/10

Hence we get a final BLEU score for the model's predicted caption as **0.5**.

Model Training

Now that our model architecture and evaluation metric is defined, we use the data generators we defined previously to actually start training the model. The model training has 3 main parts:

1. Defining a **Checkpoint** that helps us save model weights after each epoch
2. Creating our own callback that implements early stopping based on the blue score value obtained after each epoch. We set our blue score value threshold to 0.6 (0 is the worst blue score and 1 is best blue score as defined by NLTK). Thus, our model will train until the maximum epoch is reached or until the validation set shows a blue score value of 0.6

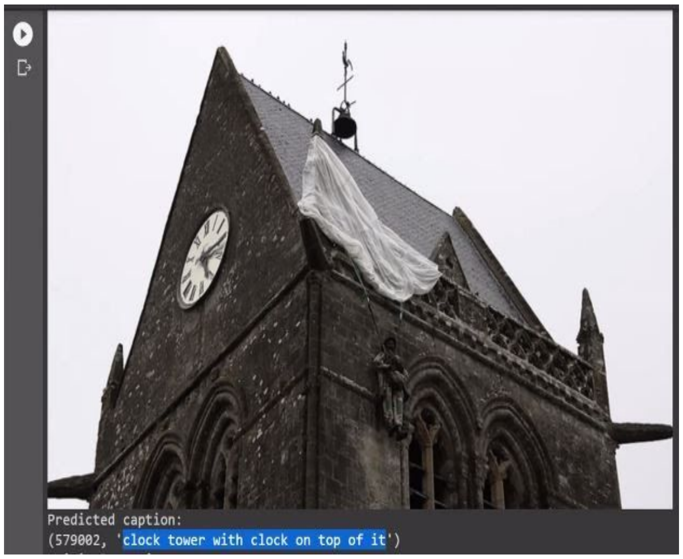
3. The model.fit_generator() function is called to train the model with the following inputs:
   - **Data generator:** generator for training data
   - **Step_size_train:** calculated as (number_of_photos_in_train_data / batch_size)
   - **Validation generator:** generator for validation data
   - **Step_size_val:** calculated as (number_of_photos_in_val_data / batch_size)
   - **Epochs:** since our dataset is large, we set number of epochs to 20 to reduce long training time
   - **Callbacks:** We use the checkpoints and callbacks we defined earlier

Model training took 3.5 hours to run for a batch size of 32 (default value), training data of 50k images and 5k validation images.

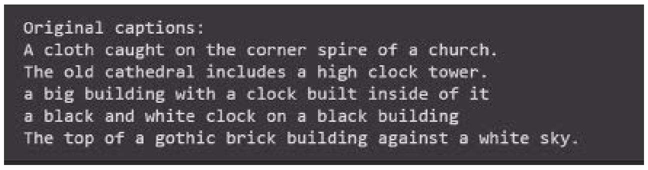
Results

Once our model was ready, we tested this model over a few images in the testing dataset. Here are some of our best results.

Image 1

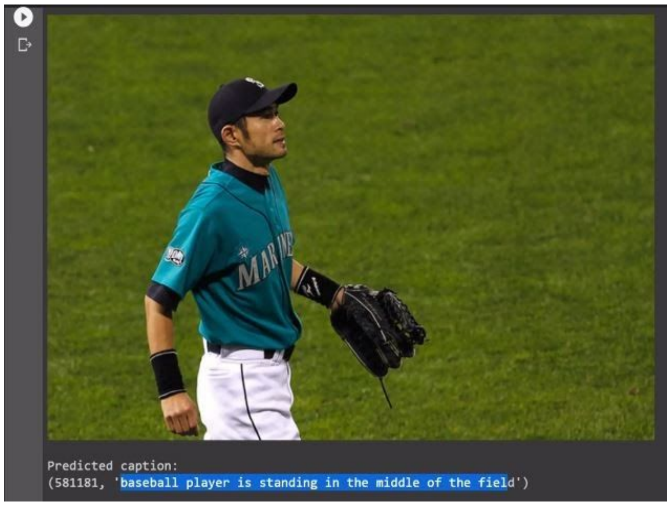


```
Predicted caption:
(579002, 'clock tower with clock on top of it')
```

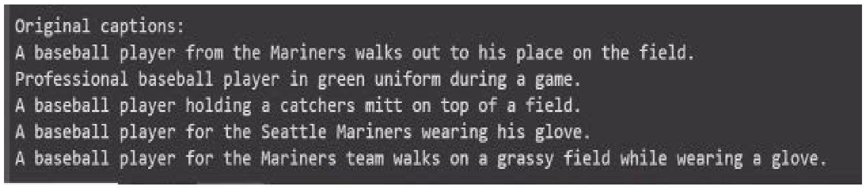


```
Original captions:
A cloth caught on the corner spire of a church.
The old cathedral includes a high clock tower.
a big building with a clock built inside of it
a black and white clock on a black building
The top of a gothic brick building against a white sky.
```

Image 2

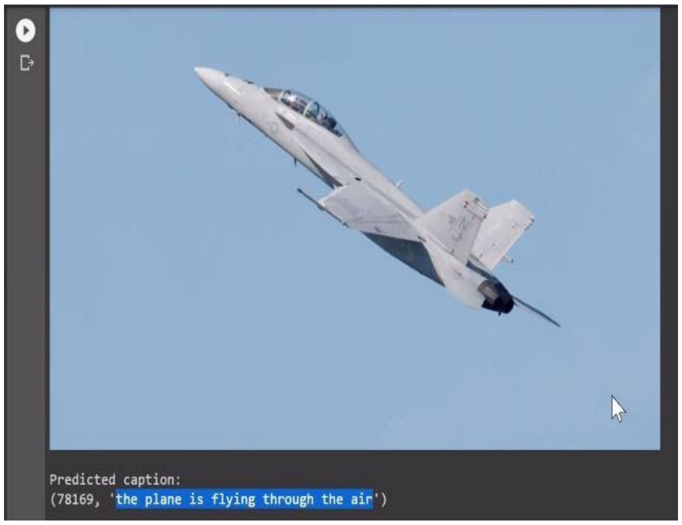


```
Predicted caption:
(581181, 'baseball player is standing in the middle of the field')
```

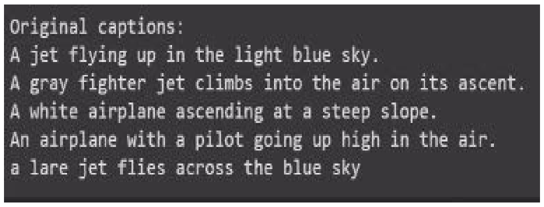


```
Original captions:
A baseball player from the Mariners walks out to his place on the field.
Professional baseball player in green uniform during a game.
A baseball player holding a catchers mitt on top of a field.
A baseball player for the Seattle Mariners wearing his glove.
A baseball player for the Mariners team walks on a grassy field while wearing a glove.
```

Image 3

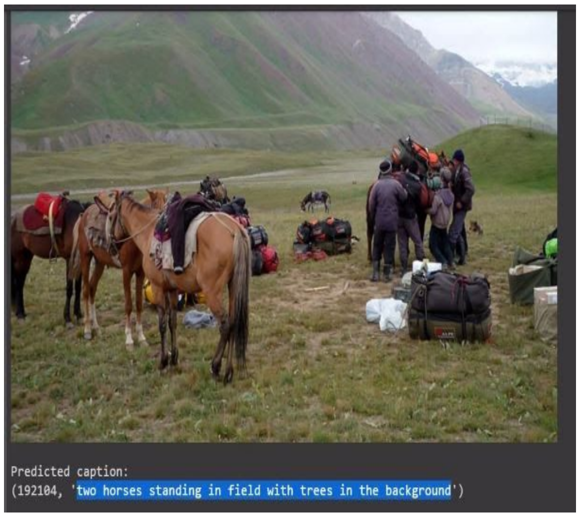


```
Predicted caption:
(78169, 'the plane is flying through the air')
```

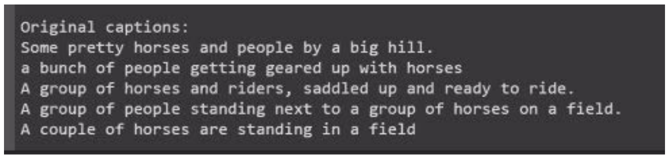


```
Original captions:
A jet flying up in the light blue sky.
A gray fighter jet climbs into the air on its ascent.
A white airplane ascending at a steep slope.
An airplane with a pilot going up high in the air.
a lare jet flies across the blue sky
```

While the above results were great, some of the other results weren't so great.



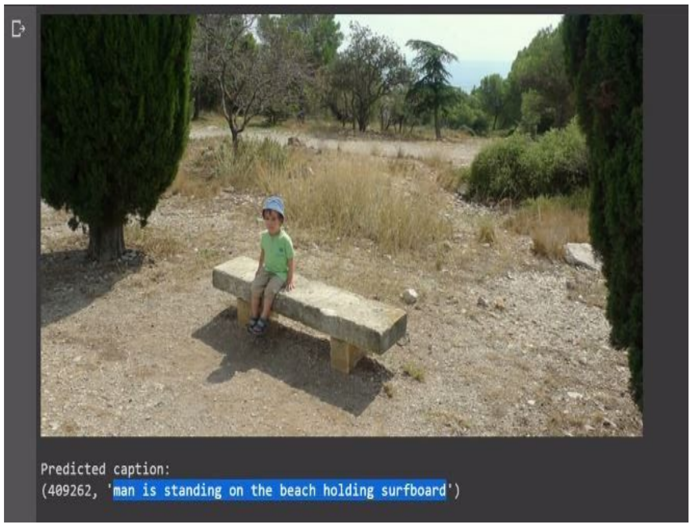
```
Predicted caption:
(192104, 'two horses standing in field with trees in the background')
```



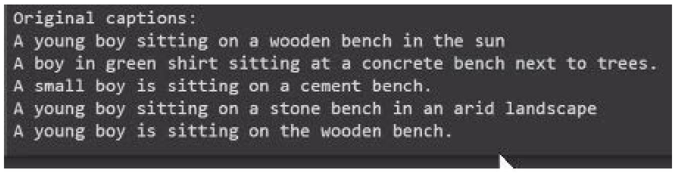
```
Original captions:
Some pretty horses and people by a big hill.
a bunch of people getting geared up with horses
A group of horses and riders, saddled up and ready to ride.
A group of people standing next to a group of horses on a field.
A couple of horses are standing in a field
```

While our model correctly detects the presence of two horses standing in a field, it also incorrectly detects trees in the background. Our assumption is that the model is trained on a lot of tree images, and when it sees a lush green area in the background, it thinks that they would be a large group of trees. Probably adding more images of hills in our training data would help the model better differentiate between hills and trees.

Let us have a look at another such example.



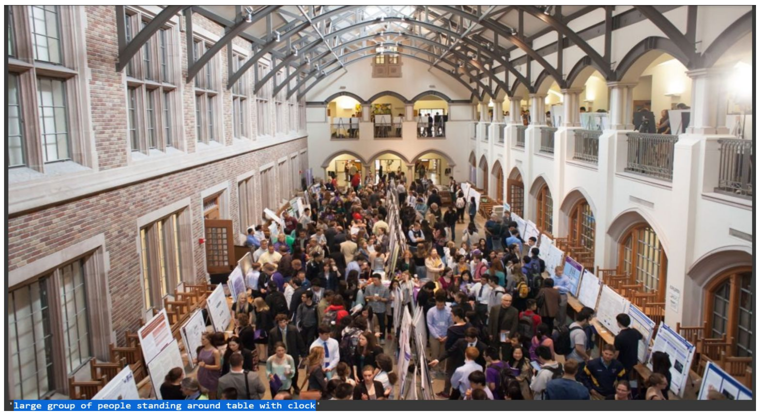
```
Predicted caption:
(409262, 'man is standing on the beach holding surfboard')
```



```
Original captions:
A young boy sitting on a wooden bench in the sun
A boy in green shirt sitting at a concrete bench next to trees.
A small boy is sitting on a cement bench.
A young boy sitting on a stone bench in an arid landscape
A young boy is sitting on the wooden bench.
```

In this example, we see that our model incorrectly classifies the kid as a man. While it is true that the person in the image is a male, taking age into consideration and differentiating between a man and a kid is required. Next, the model predicts the large brown area in the background as sand and thus predicts the location as a beach. Since our model is an RNN model, the prediction of a beach means that the model again incorrectly classifies the bench as a surfboard. While the prediction is clearly not perfect, we can at least build some logical assumptions for the model's poor performance.

Finally, out of curiosity, we wanted to test our model on a UW image. For this, we used the following image of the Mary Gates Hall.



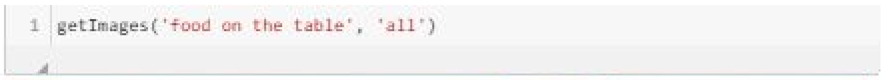
```
'large group of people standing around table with clock'
```

As seen in the prediction, the model correctly detects a large group of people standing. It also detects tables around while incorrectly detecting the presence of a clock in the picture.

Image Retrieval Mechanism

Once we had our test images and their predicted captions, we then created a functionality to retrieve all images that the user desires. The images will be returned in one of the two ways - it can either return all the images with captions that have the exact match of the keywords extracted from the query or it can return all the images with captions that match one or more of the multiple keywords.

To use this functionality, all the user has to do is give the functionality a query, which the functionality will then use to extract keywords. Additionally, the user also has to provide another parameter that enables him to select one of the above two options. Based on the keywords and the option chosen by the user in the second parameter, the functionality will return all the images that satisfy the user's requirements. For example, in the images below, we see that when the user provides the keywords 'table' and 'food' in his query, with 'all' as the second parameter, signifying the need for images with captions that have all of the keywords in the user query. Thus, the user can extract all images he needs from his gallery in just seconds, irrespective of how large the gallery is.



```
getImages('food on the table', 'all')
```

table with some plates of food and some items

table with many different types of different types of food

Individual Contributions:

Previous work: Meghana

Motivation: All

Data collection and setting up VM: Malvika

Analysis: equal contribution by all

Limitations and Future scope: Prathamesh

Limitations and Future Work

1. One of our biggest limitations was a lack of computational resources. Like mentioned in a section above, our training dataset had around 50,000 images, which meant that we had to resort to using cloud platform services for training our model. While we did try using a Virtual Machine built on Microsoft's Azure platform, it was not able to handle the vast amount of data due to a lack of GPU capabilities. While the Google Cloud performed much better, we had to invest some money to ensure we can use the services so that option wasn't feasible either. With better computational resources, we would have managed to train our model even better.
2. The second limitation was a lack of variety in our training data. As we saw in a few examples above, our model was not able to differentiate between a hill and a group of trees, and also between a man and a kid. This was because the model did not have enough examples for a hill or a kid. Thus, it provided the closest possible predictions for those images.
3. Additionally, if we had more time and computational resources, we would have done some more hyperparameter tuning for our RNN model. This would have surely ensured better results for us.

References :

Brownlee, J. (2019b, December 18). A Gentle Introduction to Calculating the BLEU Score for Text in Python. Retrieved June 5, 2020, from https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

Wikipedia contributors. (2020a, January 24). BLEU. Retrieved June 5, 2020, from https://en.wikipedia.org/wiki/BLEU

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1. https://doi.org/10.1109/cvpr.2016.308

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1. https://doi.org/10.3115/v1/d14-1162

Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2017). Show and Tell: Lessons Learned from the 2015 MSCOCO Image Captioning Challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(4), 652–663. https://doi.org/10.1109/tpami.2016.2587640