

AI 825: Visual Recognition (Part 2) Mini Project: Image Captioning

Team Details-Group Code: **AVB**

Akshay Mahesh Gudiyawar (MT2020137)

Vikrant Punia (MT2020049)

Bhaskar Yadav (MT2020136)

Problem Statement: Build an Image Captioning system which takes Images as input and provides meaningful captions describing the Image. The system should be built from CNN + LSTM models, where CNN does the task of obtaining features from Images, i.e. it will convert Image to vector information. LSTM (Long Short Term memory) is a recurrent neural network that is capable of learning long term dependencies in data. We will be using it to learn / predict captions of the Input Image.

Solution Approach: To perform Image Captioning, we are using a combination of CNN (Convolution Neural Network) and LSTM (Long Short Term Memory) models. When an Image is supplied as input, the data will be in the form of pixels (pixel arrays). We need to convert it to feature vector for further processing. CNN takes Image pixel data and converts them into feature vectors here. Then these feature vectors along with the captions is fed into an LSTM. For CNN we have use pre-trained Inception model. We had constructed an LSTM network with keras containing 256 units with linear activation function (by default).

An LSTM is a recurrent neural network architecture that is commonly used in problems with temporal dependences. It succeeds in being able to capture information about previous states to better inform the current prediction through its memory cell state. Since we are going to predict the next word using the previous words we are setting “return_sequences = True” in LSTM code. In a sentence language model, an LSTM is predicting the next word in a sentence, given the context of previously seen characters. When we feed image embeddings from inception to LSTM, this becomes the first previous state to the language model, influencing the next predicted words.

At each time-step, the LSTM considers the previous cell state and outputs a prediction for the most probable next value in the sequence. When training this token, we are also tokenizing the input, meaning mapping each unique word to unique numeric id. In Data generator, we are taking the encoding of an image and use a start word to predict the next word. After that, we will again use the same image and use the predicted word to predict the next word. So, the image will be used at every iteration for the entire caption. This is how we will generate

the caption for an image. Here in the code, “Image_model” trains on the images and “caption_model” trains on captions and then we concatenate these two into a “final_model”. We have set size of embedding as 300, which was decided as optimal value after experimenting - helps in minimizing the no of parameters. The input size for “image_model” is defined as 2048 because it is the vector size of the encoded training images. They are processed by the dense layer to produce a 256 element representation of the Image.

The “final_model”, overall contained 9,115,032 parameters. For prediction, we have used two methods: Argmax search and Beam search. Argmax Search is where the maximum value index(argmax) in the 8256 long predicted vector (which is the vocab size) is extracted and appended to the result. This is done until we hit <end> or the maximum length of the caption. Beam Search is where we take top k predictions, feed them again in the model and then sort them using the probabilities returned by the model. So, the list will always contain the top k predictions. In the end, we take the one with the highest probability and go through it till we encounter <end> or reach the maximum caption length.

Architecture of models used:

Final Model Summary: The “final_model” obtained after concatenating “image_model”, which is trained on Images, and “caption_model” which trains on captions. Final model architecture is as shown below. It contains about 90 lakh parameters.

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 40)]	0	
input_2 (InputLayer)	[(None, 2048)]	0	
embedding (Embedding)	(None, 40, 300)	2476800	input_3[0][0]
dense (Dense)	(None, 300)	614700	input_2[0][0]
lstm (LSTM)	(None, 40, 256)	570368	embedding[0][0]
repeat_vector (RepeatVector)	(None, 40, 300)	0	dense[0][0]
time_distributed (TimeDistribut	(None, 40, 300)	77100	lstm[0][0]
concatenate_2 (Concatenate)	(None, 80, 300)	0	repeat_vector[0][0] time_distributed[0][0]
bidirectional (Bidirectional)	(None, 512)	1140736	concatenate_2[0][0]
dense_2 (Dense)	(None, 8256)	4235328	bidirectional[0][0]
activation_94 (Activation)	(None, 8256)	0	dense_2[0][0]
Total params: 9,115,032			
Trainable params: 9,115,032			
Non-trainable params: 0			

Image model: Which was trained on image inputs. This was later used to concatenate into final model

```
In [24]: image_input = Input(shape = (2048,))
x = layers.Dense(embedding_size, activation='relu')(image_input)
image_output = layers.RepeatVector(max_len)(x)
image_model = Model(inputs=image_input, outputs = image_output)
image_model.summary()
```

Model: "model_1"

Layer (type)	Output Shape	Param #
=====		
input_2 (InputLayer)	[(None, 2048)]	0
dense (Dense)	(None, 300)	614700
repeat_vector (RepeatVector)	(None, 40, 300)	0
=====		
Total params: 614,700		
Trainable params: 614,700		
Non-trainable params: 0		

Caption model: This was trained on captions of Images. This together with image_model are concatenated into final model.

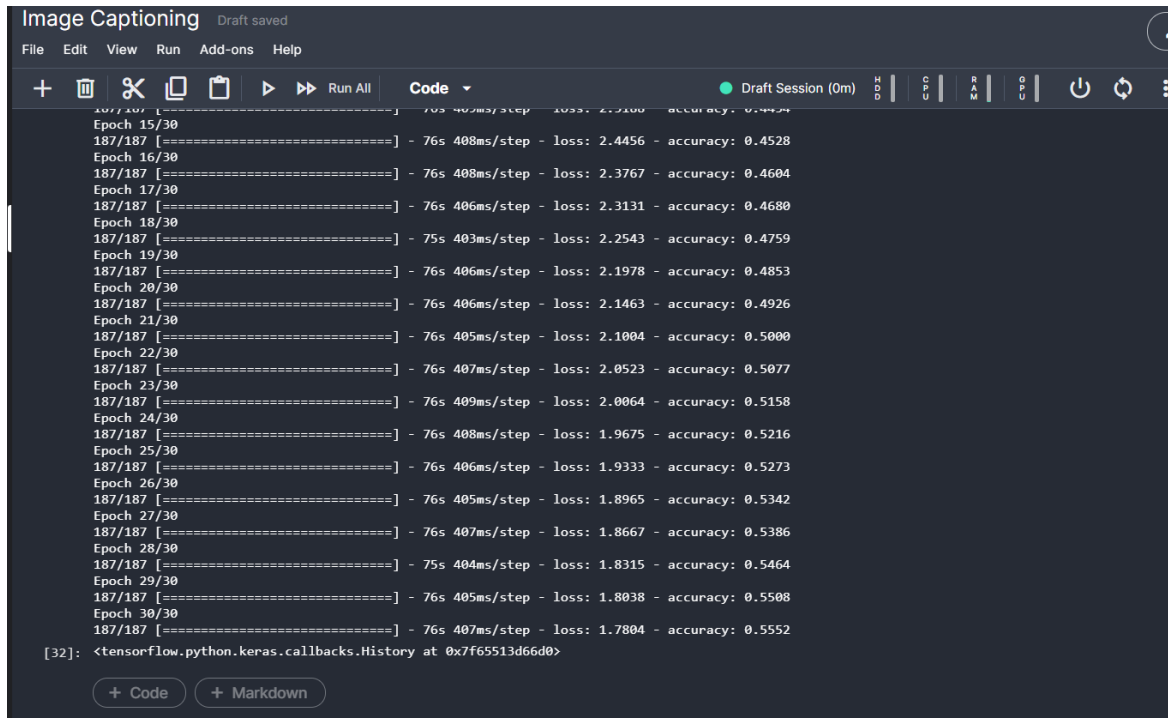
Layer (type)	Output Shape	Param #
=====		
input_3 (InputLayer)	[(None, 40)]	0
embedding (Embedding)	(None, 40, 300)	2476800
lstm (LSTM)	(None, 40, 256)	570368
time_distributed (TimeDistri	(None, 40, 300)	77100
=====		
Total params: 3,124,268		
Trainable params: 3,124,268		
Non-trainable params: 0		

Training methodology: We trained the model using Flickr Image Captioning dataset <https://drive.google.com/drive/folders/1RQ5qHm0aVFqWDG9VBISnXINPI5T15Wf> . The dataset contains text files filled with captions for training Image also. Flickr8K has 6000 training images, 1000 validation images and 1000 testing images. For creating the model, the captions had to be put in an embedding. We have set embedding size as 300 in our model. The optimizer used was Adam with loss function being 'Categorical Cross Entropy'. We had set batch size to 2048 while training. We trained this system for 30 epochs and got final loss value of 1.7804 and accuracy value 0.5552 meaning approximately 55.2% at the end of 30th epoch.

Results Obtained:

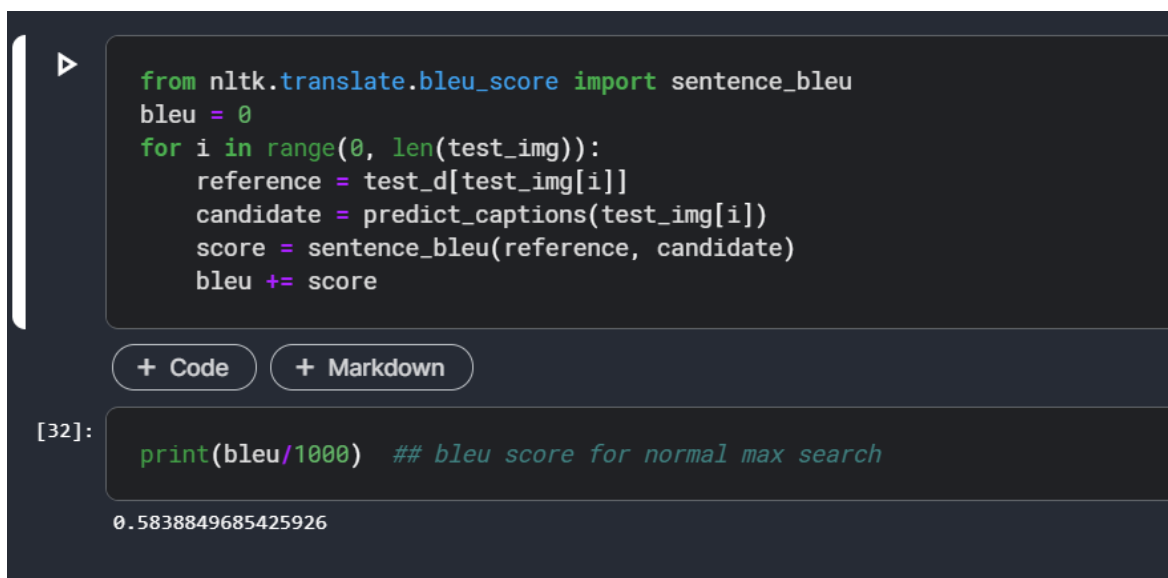
The objective was to maximize bleu score for the resultant model on test data. We tried predictions with normal max search, beam search with index 3, 5 and 7. We tried obtaining captions for some random Images and also calculated BLEU scores for test data on these methods individually.

As we had trained the model for 30 epochs with configurations mentioned earlier, below is the Figure showing metrics obtained at the end of 30th epoch. We got loss as 1.7804 and accuracy as 55.52 %.



```
Image Captioning Draft saved
File Edit View Run Add-ons Help
+ [Icons] Run All Code Draft Session (0m) CPU RAM GPU
Epoch 15/30
187/187 [=====] - 76s 408ms/step - loss: 2.4456 - accuracy: 0.4528
Epoch 16/30
187/187 [=====] - 76s 408ms/step - loss: 2.3767 - accuracy: 0.4604
Epoch 17/30
187/187 [=====] - 76s 406ms/step - loss: 2.3131 - accuracy: 0.4680
Epoch 18/30
187/187 [=====] - 75s 403ms/step - loss: 2.2543 - accuracy: 0.4759
Epoch 19/30
187/187 [=====] - 76s 406ms/step - loss: 2.1978 - accuracy: 0.4853
Epoch 20/30
187/187 [=====] - 76s 406ms/step - loss: 2.1463 - accuracy: 0.4926
Epoch 21/30
187/187 [=====] - 76s 405ms/step - loss: 2.1004 - accuracy: 0.5000
Epoch 22/30
187/187 [=====] - 76s 407ms/step - loss: 2.0523 - accuracy: 0.5077
Epoch 23/30
187/187 [=====] - 76s 409ms/step - loss: 2.0064 - accuracy: 0.5158
Epoch 24/30
187/187 [=====] - 76s 408ms/step - loss: 1.9675 - accuracy: 0.5216
Epoch 25/30
187/187 [=====] - 76s 406ms/step - loss: 1.9333 - accuracy: 0.5273
Epoch 26/30
187/187 [=====] - 76s 405ms/step - loss: 1.8965 - accuracy: 0.5342
Epoch 27/30
187/187 [=====] - 76s 407ms/step - loss: 1.8667 - accuracy: 0.5386
Epoch 28/30
187/187 [=====] - 75s 404ms/step - loss: 1.8315 - accuracy: 0.5464
Epoch 29/30
187/187 [=====] - 76s 405ms/step - loss: 1.8038 - accuracy: 0.5508
Epoch 30/30
187/187 [=====] - 76s 407ms/step - loss: 1.7804 - accuracy: 0.5552
[32]: <tensorflow.python.keras.callbacks.History at 0x7f65513d66d0>
+ Code + Markdown
```

In predictions wrt test dataset, average BLEU score of 0.58388 was obtained on normal max search as shown below



```
▶
from nltk.translate.bleu_score import sentence_bleu
bleu = 0
for i in range(0, len(test_img)):
    reference = test_d[test_img[i]]
    candidate = predict_captions(test_img[i])
    score = sentence_bleu(reference, candidate)
    bleu += score

+ Code + Markdown

[32]:
print(bleu/1000) ## bleu score for normal max search

0.5838849685425926
```

With using beam search (index = 3), we got average BLEU of 0.59807 as shown below:

```
▶ from nltk.translate.bleu_score import sentence_bleu
bleu = 0
for i in range(0, len(test_img)):
    reference = test_d[test_img[i]]
    candidate = beam_search_predictions(test_img[i], beam_index=3)
    score = sentence_bleu(reference, candidate)
    bleu += score
    |
print(bleu/1000)    ## bleu score for beam search with index = 3

0.5980732144941163
```

With using beam search (index=5), we got BLEU score of 0.58724 as shown below:

```
from nltk.translate.bleu_score import sentence_bleu
bleu = 0
for i in range(0, len(test_img)):
    reference = test_d[test_img[i]]
    candidate = beam_search_predictions(test_img[i], beam_index=5)
    score = sentence_bleu(reference, candidate)
    bleu += score #this is on scale of 0 to 1. to get it on scale of 0 to 100 multiply it by 100

/opt/conda/lib/python3.7/site-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 4-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
warnings.warn(_msg)
```

+ Code + Markdown

```
1: bleu/1000# to get average of 1000
2: 0.5872486106489746
```

With using beam search (index=7), we got BLEU score of 0.570 as shown below:

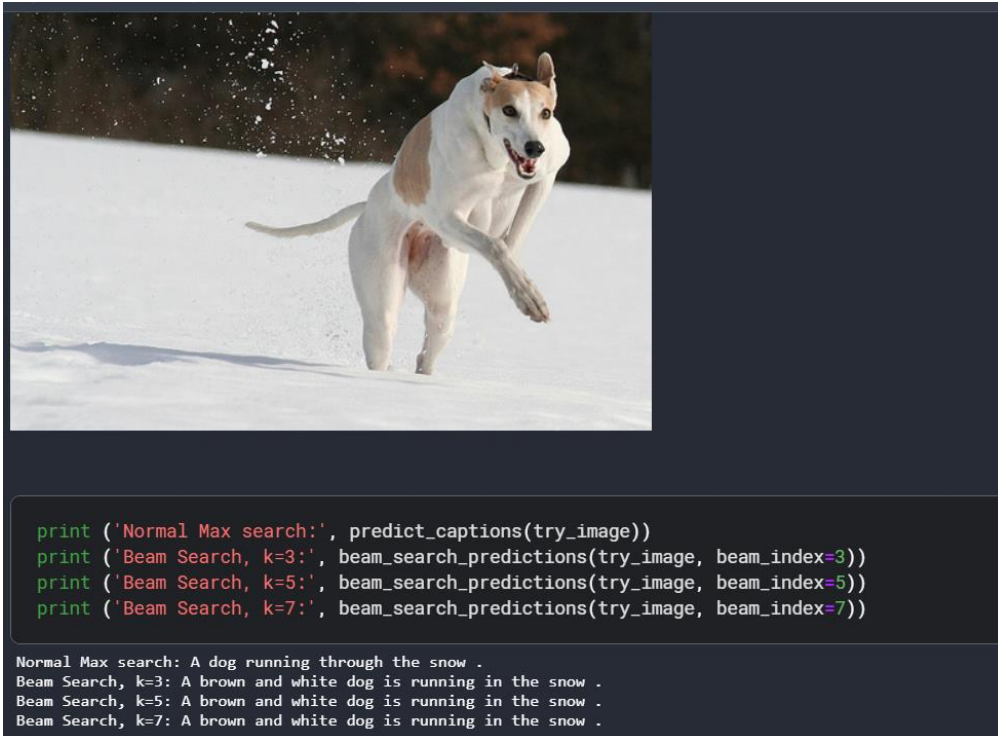
```
[31]: from nltk.translate.bleu_score import sentence_bleu
bleu = 0
for i in range(0, len(test_img)):
    reference = test_d[test_img[i]]
    candidate = beam_search_predictions(test_img[i], beam_index=7)
    score = sentence_bleu(reference, candidate)
    bleu += score #this is on scale of 0 to 1. to get it on scale of 0 to 100

[32]: bleu/1000# to get average of 1000

[32]: 0.5700085619350284
```

BLEU score of beam search with index = 3 gave the max score in our experiments.

Some resultant captions obtained on random Images from test data are shown below (Subjective Images results are attached in later section):



A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .
A little girl is sitting in front of a large painted rainbow .
A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
There is a girl with pigtails sitting in front of a rainbow painting .
Young girl with pigtails painting outside in the grass .

Each caption output is from normal max search, beam search (index 3), beam search (5) and beam search (7) respectively.



```
[52]: print ('Normal Max search:', predict_captions(try_image))
      print ('Beam Search, k=3:', beam_search_predictions(try_image, beam_index=3))
      print ('Beam Search, k=5:', beam_search_predictions(try_image, beam_index=5))
      print ('Beam Search, k=7:', beam_search_predictions(try_image, beam_index=7))
```

Normal Max search: A boy in a blue shirt is standing on railroad tracks .
 Beam Search, k=3: A little girl in a blue shirt is standing on railroad tracks .
 Beam Search, k=5: A little girl in a blue shirt is standing on railroad tracks .
 Beam Search, k=7: A little girl in a blue shirt is standing on railroad tracks .



+ Code + Markdown

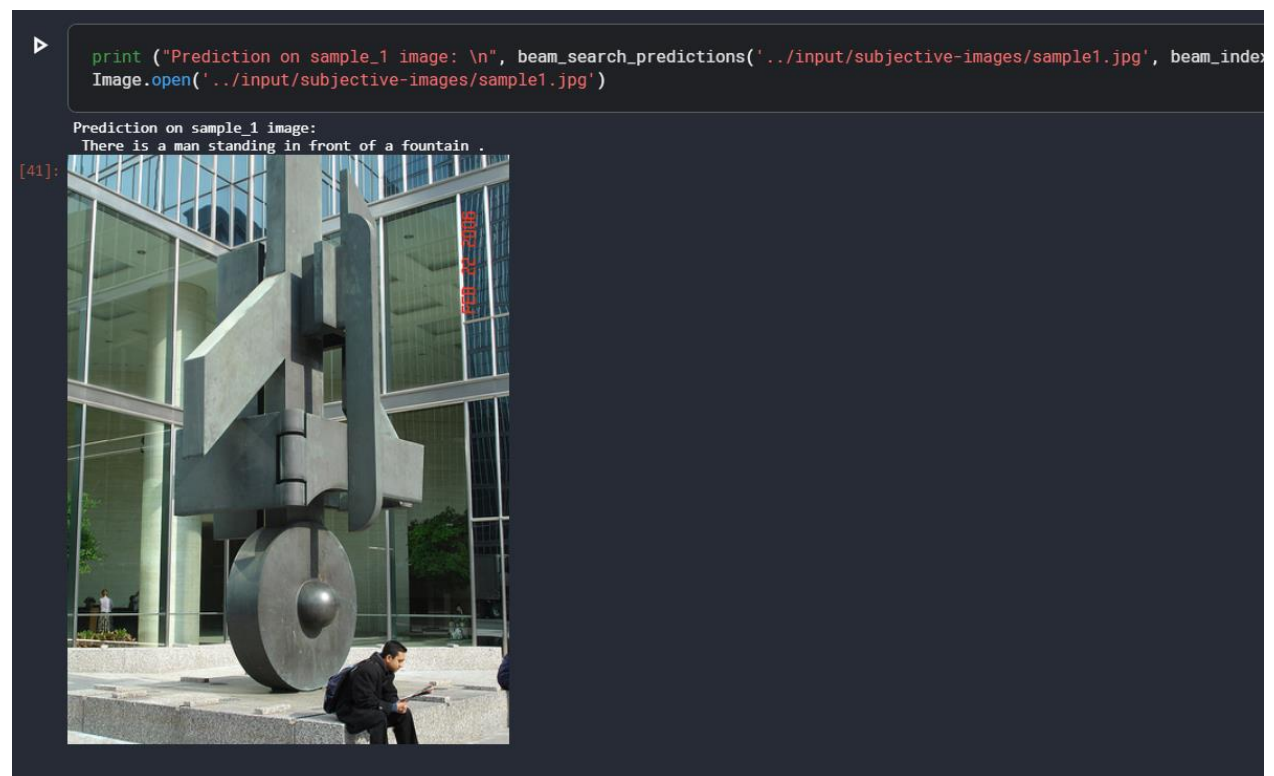
```
[44]: print ('Normal Max search:', predict_captions(try_image))
      print ('Beam Search, k=3:', beam_search_predictions(try_image, beam_index=3))
      print ('Beam Search, k=5:', beam_search_predictions(try_image, beam_index=5))
      print ('Beam Search, k=7:', beam_search_predictions(try_image, beam_index=7))
```

Normal Max search: A man and a man are sitting in the water .
 Beam Search, k=3: A man in a yellow kayak is paddling through the water .
 Beam Search, k=5: A man in a yellow kayak is paddling through the water .
 Beam Search, k=7: A man in a yellow kayak is paddling through the water .

Subjective Images result:

Sample 1 Image caption result: It produced the caption as – “There is a man standing in front of a fountain”. We can see that it didn’t properly analyze the situation in Image. The man is

infact sitting and there is no water fountain behind him. But the model may have assumed it because the shape of object is similar to fountain in structure. And it also successfully picked up person in the Image and the relative position of object and person as “front and back”.



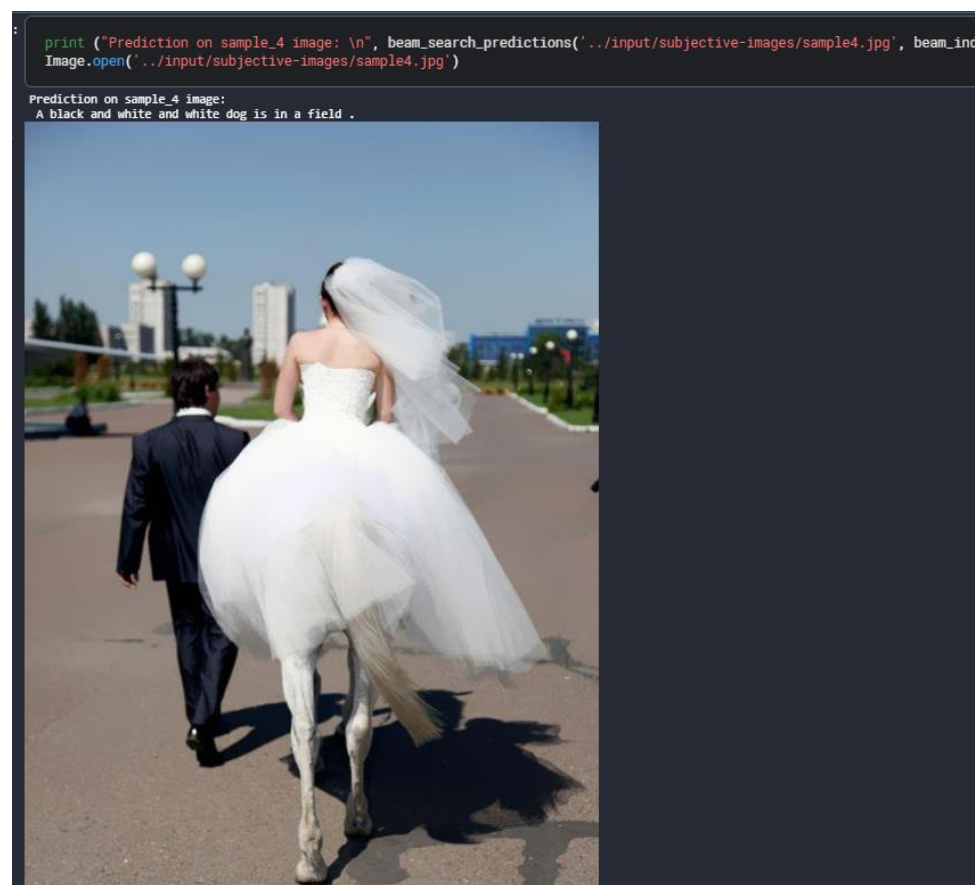
Sample 2 Image caption result: The caption predicted by beam search is “A group of people on a street”. The model correctly analyzed the image and we can see that caption is right.



Sample 3 Image caption result: The caption predicted is “A group of people standing in front of a crowd”. While the system correctly identified group of people, and also got their relative position correct, it’s not the best description of the below Image.



Sample 4 Image caption result:



Here in Sample 4 Image, the caption produced was “A black and white dog is in a field”. We can see that it’s not correct as there is no dog, but two persons and one of them riding a white horse. But because the lady is sitting on it and it makes an abstract shape of dog, the system recognized it as dog. And it got the colors right as black and white.

Sample 5 Image caption result: Here the caption produced was – “A brown dog is running through the snow”. While the dinosaur shape resembles a dog running, it didn’t capture the person in Image. But we can reason that the image itself is not normal, as in it doesn’t appear to be real in nature. Hence the system tried its best to predict the caption by recognizing it as a dog running.

