CAPSTONE PROJECT - IMAGE CAPTION GENERATOR

In [1]:

```
#Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os, random, string
from pickle import dump, load
from tqdm import tqdm
```

In [2]:

```
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical, plot_model
from keras.applications.inception_v3 import InceptionV3 , preprocess_input
#from keras.layers.merging import add
from tensorflow.keras.layers import add
from keras.models import Model, load_model
from keras.layers import Input, Dense, LSTM, Embedding, Dropout
from nltk import FreqDist
from nltk.translate.bleu_score import sentence_bleu
```

```
WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\src
\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecate
d. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [3]:

```
#Dataset path
input_dir = 'C:\\Users\\Admin\\DS_CAPSTONE\\'
```

In [4]:

```
def load_captions_dictionary(path):
    file = open(path, 'r')
    captions = file.read().split('\n')
    descriptions = {}
    for text in captions[1:]:
        values = text.split(',')
        img, caption = values[0].split('.')[0], "".join(values[1:])
        if img not in descriptions:
            descriptions[img] = [caption]
        else:
            descriptions[img].append(caption)
    file.close()
    return descriptions

descriptions = load_captions_dictionary(input_dir + 'captions.txt')
```

In [5]:

```
#View first 5 images with its captions
npic = 5
img_size = 299
target_size = (img_size, img_size)
path = input_dir + "Images/"
fig = plt.figure(figsize=(10,20))

count = 1
for img in os.listdir(path)[:npic]:

    filename = path + img
    captions = list(descriptions[img.split(".")[0]])
    image_load = load_img(filename, target_size=target_size)

    ax = fig.add_subplot(npic, 2, count, xticks=[], yticks=[])
    ax.imshow(image_load)
    count += 1

    ax = fig.add_subplot(npic, 2, count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0, 1)
    ax.set_ylim(0, len(captions))
    for i, caption in enumerate(captions):
        ax.text(0, i, caption, fontsize=20)
    count += 1
plt.show()
```

A little girl in a pink dress going into a wooden cabin .

A little girl climbing the stairs to her playhouse .

A little girl climbing into a wooden playhouse .

A girl going into a wooden building .

A child in a pink dress is climbing up a set of stairs in an entry way .

Two dogs on pavement moving toward each other .

Two dogs of different breeds looking at each other on the road .

A black dog and a white dog with brown spots are staring at each other in the street .

A black dog and a tri-colored dog playing with each other on the road .

A black dog and a spotted dog are fighting

Young girl with pigtails painting outside in the grass .

There is a girl with pigtails sitting in front of a rainbow painting .

A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .

A little girl is sitting in front of a large painted rainbow .

A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .

man laying on bench holding leash of dog sitting on ground

A shirtless man lies on a park bench with his dog .

a man sleeping on a bench outside with a white and black dog sitting next to him .

A man lays on the bench to which a white dog is also tied .

A man lays on a bench while his dog sits by him .

The man with pierced ears is wearing glasses and an orange hat .

A man with glasses is wearing a beer can crocheted hat .

A man with gauges and glasses is wearing a Blitz hat .

A man wears an orange hat and glasses .

A man in an orange hat starring at something .

In [ ]:

In [6]:

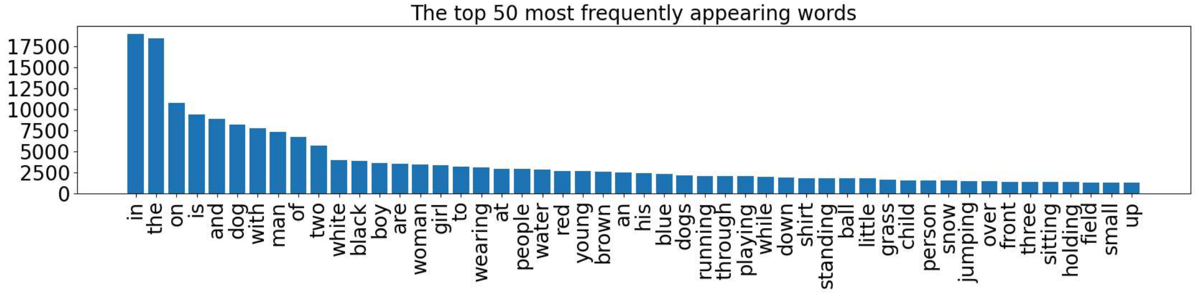
```
#Remove puntuations, convert to lowercase.
def text_cleaning(descriptions):
    table = str.maketrans('', '', string.punctuation)
    for img, caption in descriptions.items():
        for i, img_text in enumerate(caption):
            img_text.replace("-", " ")
            text = [word.lower() for word in img_text.split()]
            text = [word.translate(table) for word in text]
            text = [word for word in text if(len(word) > 1)]
            text = [word for word in text if(word.isalpha())]
            img_text = " ".join(text)
            descriptions[img][i] = img_text
    return descriptions

descriptions = text_cleaning(descriptions)
```

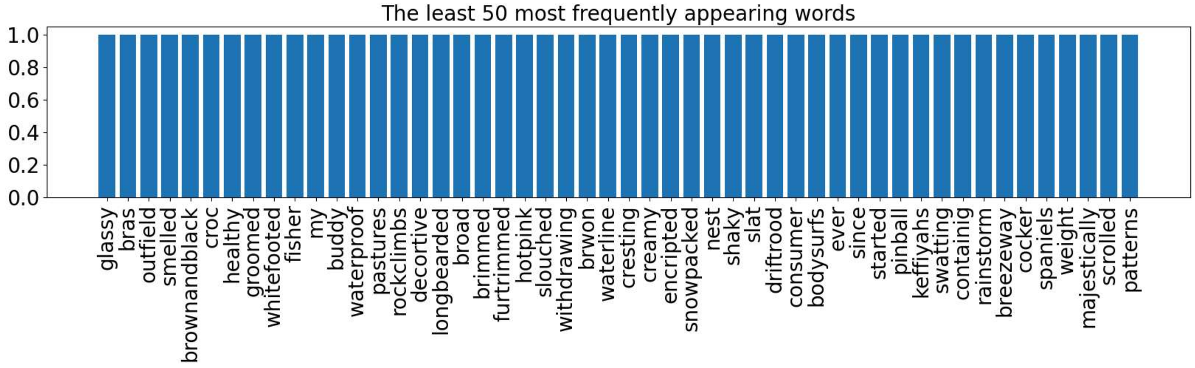
In [7]:
```
def corpus_and_vocab(descriptions):
    corpus = ""
    for img_text in descriptions.values():
        for text in img_text:
            corpus += " "+text
    vocab = set(corpus.split())
    return corpus, vocab

corpus, vocab = corpus_and_vocab(descriptions)
print("Number of unique words = {}".format(len(vocab)))
```

```
Number of unique words = 8763
```

In [8]:
```
#Veiw most frequent and least frequent words
freq_dist = FreqDist(corpus.split())
dfsub = pd.DataFrame(columns = ["word", "count"])
most_common = freq_dist.most_common()
words, counts = [], []
for i in range(len(freq_dist)):
    words.append(most_common[i][0])
    counts.append(most_common[i][1])
dfsub["word"], dfsub["count"] = words, counts
```

In [9]:
```
def plthist(dfsub, title):
    plt.figure(figsize=(20,3))
    plt.bar(dfsub.index,dfsub["count"])
    plt.yticks(fontsize=20)
    plt.xticks(dfsub.index,dfsub["word"],rotation=90,fontsize=20)
    plt.title(title,fontsize=20)
    plt.show()

plthist(dfsub.iloc[:50], "The top 50 most frequently appearing words")
plthist(dfsub.iloc[-50:], "The least 50 most frequently appearing words")
```


The top 50 most frequently appearing words


The least 50 most frequently appearing words

In [10]:
```
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for i, desc in enumerate(desc_list):
            #descriptions[key][i] = desc = "<startseq> " + desc + " <endseq>"
            lines.append(key + '\t' + desc)
    data = "\n".join(lines)
    file = open(filename,"w")
    file.write(data)
    file.close()

save_descriptions(descriptions, "Image_Descriptions_List.txt")
```

In [11]:
```
df_img_caption = pd.DataFrame()
df_img_caption['Image_Name'] = list(descriptions.keys())[:-1]
temps = list(descriptions.values())[:-1]
df_img_caption['Caption'] = [temps[i][random.randint(0,4)] for i in range(len(
df_img_caption.head()
```

Out[11]:

	Image_Name	Caption
0	1000268201_693b08cb0e	little girl climbing into wooden playhouse
1	1001773457_577c3a7d70	black dog and white dog with brown spots are s...
2	1002674143_1b742ab4b8	little girl covered in paint sits in front of ...
3	1003163366_44323f5815	man lays on the bench to which white dog is al...
4	1007129816_e794419615	man in an orange hat starring at something

In [12]:
```
test_images = np.asarray(df_img_caption['Image_Name'][:10], dtype = np.dtype(ob
test_captions = np.asarray(df_img_caption['Caption'][:10], dtype = np.dtype(obj
val_images = np.asarray(df_img_caption['Image_Name'][10:15], dtype = np.dtype(o
val_captions = np.asarray(df_img_caption['Caption'][10:15], dtype = np.dtype(ob
train_images = np.asarray(df_img_caption['Image_Name'][15:], dtype = np.dtype(o
train_captions = np.asarray(df_img_caption['Caption'][15:], dtype = np.dtype(ob
```

In [ ]:

In [13]:

```
#Pre-trained InceptionV3 model
cnn_model = InceptionV3(weights = 'imagenet')
for layer in cnn_model.layers:
    layer.trainable = False  # weights of these layers will not be updated
cnn_model = Model(inputs = cnn_model.input, outputs = cnn_model.get_layer('avg_
                                                            #the 'avg_pool
cnn_model.summary()
```

```
 batch_normalization_63 (Ba  (None, 17, 17, 192)         576       ['conv2
d_63[0][0]']
 tchNormalization)

 batch_normalization_68 (Ba  (None, 17, 17, 192)         576       ['conv2
d_68[0][0]']
 tchNormalization)

 batch_normalization_69 (Ba  (None, 17, 17, 192)         576       ['conv2
d_69[0][0]']
 tchNormalization)

 activation_60 (Activation)  (None, 17, 17, 192)         0         ['batch
_normalization_60[0][0]
                                                                    ']

 activation_63 (Activation)  (None, 17, 17, 192)         0         ['batch
_normalization_63[0][0]
                                                                    ']
```

In [14]:

```
def extract_features(model, images, img_size):
    features = {}
    for img in tqdm(images):
        picture = load_img(input_dir + "Images/" + img + ".jpg", target_size =
        picture = img_to_array(picture)
        picture = np.expand_dims(picture, axis = 0)
        picture = preprocess_input(picture)
        features[img] = model.predict(picture).reshape(2048,)

    return features
```

In [15]:
```
Xtrain_features = extract_features(cnn_model, train_images, 299)
```

```
  0%|
| 0/8076 [00:00<?, ?it/s]

1/1 [==============================] - 2s 2s/step

  0%|
| 1/8076 [00:02<5:03:54,  2.26s/it]

1/1 [==============================] - 0s 150ms/step

  0%|
| 2/8076 [00:02<2:23:10,  1.06s/it]

1/1 [==============================] - 0s 165ms/step

  0%|
| 3/8076 [00:02<1:33:12,  1.44it/s]

1/1 [==============================] - 0s 157ms/step

  0%|
```

In [16]:
```
Xval_features = extract_features(cnn_model, val_images, 299)
```

```
  0%|
| 0/5 [00:00<?, ?it/s]

1/1 [==============================] - 0s 313ms/step

 20%|█████████████████
| 1/5 [00:00<00:01,  2.27it/s]

1/1 [==============================] - 0s 159ms/step

 40%|██████████████████████████████████
| 2/5 [00:00<00:00,  3.05it/s]

1/1 [==============================] - 0s 142ms/step

 60%|███████████████████████████████████████████████████
| 3/5 [00:00<00:00,  3.56it/s]

1/1 [==============================] - 0s 146ms/step

 80%|████████████████████████████████████████████████████████████████████
| 4/5 [00:01<00:00,  3.71it/s]

1/1 [==============================] - 0s 174ms/step

100%|████████████████████████████████████████████████████████████████████████
████████████| 5/5 [00:01<00:00,  3.52it/s]
```

In [17]:
```
Xtrain_features = np.asarray(list(Xtrain_features.values()))
Xval_features = np.asarray(list(Xval_features.values()))
```

In [ ]:

In [18]:

```
tokenizer = Tokenizer(num_words = len(vocab))
tokenizer.fit_on_texts(df_img_caption['Caption'])
word_to_index = tokenizer.word_index
index_to_word = dict([index, word] for word, index in word_to_index.items())
vocab_size = len(tokenizer.word_index) + 1
```

In [19]:

```
train_sequences = tokenizer.texts_to_sequences(train_captions)
val_sequences = tokenizer.texts_to_sequences(val_captions)

def maxLength(sequences):
    return np.max([len(sequence) for sequence in sequences])

max_len = max(maxLength(train_sequences), maxLength(val_sequences))
```

In [20]:

```
def data_generator(features, sequences):
    X_features, X_train, y_train = [], [], []
    for sequence, feature in zip(sequences, features):
        for i in range(1, len(sequence)):
            in_text, out_text = sequence[:i], sequence[i:]
            in_text = pad_sequences([in_text], maxlen = max_len)[0]
            out_text = to_categorical(out_text, num_classes = vocab_size)[0]
            X_features.append(feature)
            X_train.append(in_text)
            y_train.append(out_text)
    return (np.array(X_features), np.array(X_train), np.array(y_train))

Xt_features, Xt_text, yt_text = data_generator(Xtrain_features, train_sequences
Xv_features, Xv_text, yv_text = data_generator(Xval_features, val_sequences)
```

In [21]:

```
print(Xt_features.shape, Xt_text.shape, yt_text.shape)
print(Xv_features.shape, Xv_text.shape, yv_text.shape)
```

```
(66345, 2048) (66345, 28) (66345, 4538)
(40, 2048) (40, 28) (40, 4538)
```

In [ ]:

In [22]:
```
def define_model(vocab_size, max_len):

    #For images
    inputs1 = Input(shape = (2048,))
    x1 = Dropout(0.3)(inputs1)
    x2 = Dense(256, activation = 'relu')(x1)

    #For captions
    inputs2 = Input(shape = (max_len,))
    se1 = Embedding(vocab_size, 256, mask_zero = True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)     #to process the sequential data

    decoder1 = add([x2, se3])
    decoder2 = Dense(256, activation = 'relu')(decoder1)
    outputs = Dense(vocab_size, activation = 'softmax')(decoder2)  # predicted

    rnn_model = Model(inputs = [inputs1, inputs2], outputs = outputs)
    rnn_model.compile(loss = 'categorical_crossentropy', optimizer = 'adam')

    print(rnn_model.summary())
    plot_model(rnn_model, to_file = 'rnn_model.png', show_shapes = True)

    return rnn_model

rnn_model = define_model(vocab_size, max_len)
```

```
WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\src
\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Pleas
e use tf.compat.v1.train.Optimizer instead.

Model: "model_1"
_____________________________________________________________________________
_____________________
 Layer (type)                Output Shape                 Param #   Connected
to
=============================================================================
=====================
 input_3 (InputLayer)        [(None, 28)]                 0         []

 input_2 (InputLayer)        [(None, 2048)]               0         []

 embedding (Embedding)       (None, 28, 256)              1161728   ['input_3
[0][0]']

 dropout (Dropout)           (None, 2048)                 0         ['input_2
[0][0]']

 dropout_1 (Dropout)         (None, 28, 256)              0         ['embeddi
ng[0][0]']

 dense (Dense)               (None, 256)                  524544    ['dropout
[0][0]']

 lstm (LSTM)                 (None, 256)                  525312    ['dropout
_1[0][0]']

 add (Add)                   (None, 256)                  0         ['dense
[0][0]',
                                                                     'lstm[0]
[0]']

 dense_1 (Dense)             (None, 256)                  65792     ['add[0]
[0]']

 dense_2 (Dense)             (None, 4538)                 1166266   ['dense_1
[0][0]']

=============================================================================
=====================
Total params: 3443642 (13.14 MB)
Trainable params: 3443642 (13.14 MB)
Non-trainable params: 0 (0.00 Byte)
_____________________________________________________________________________
_____________________
None
```

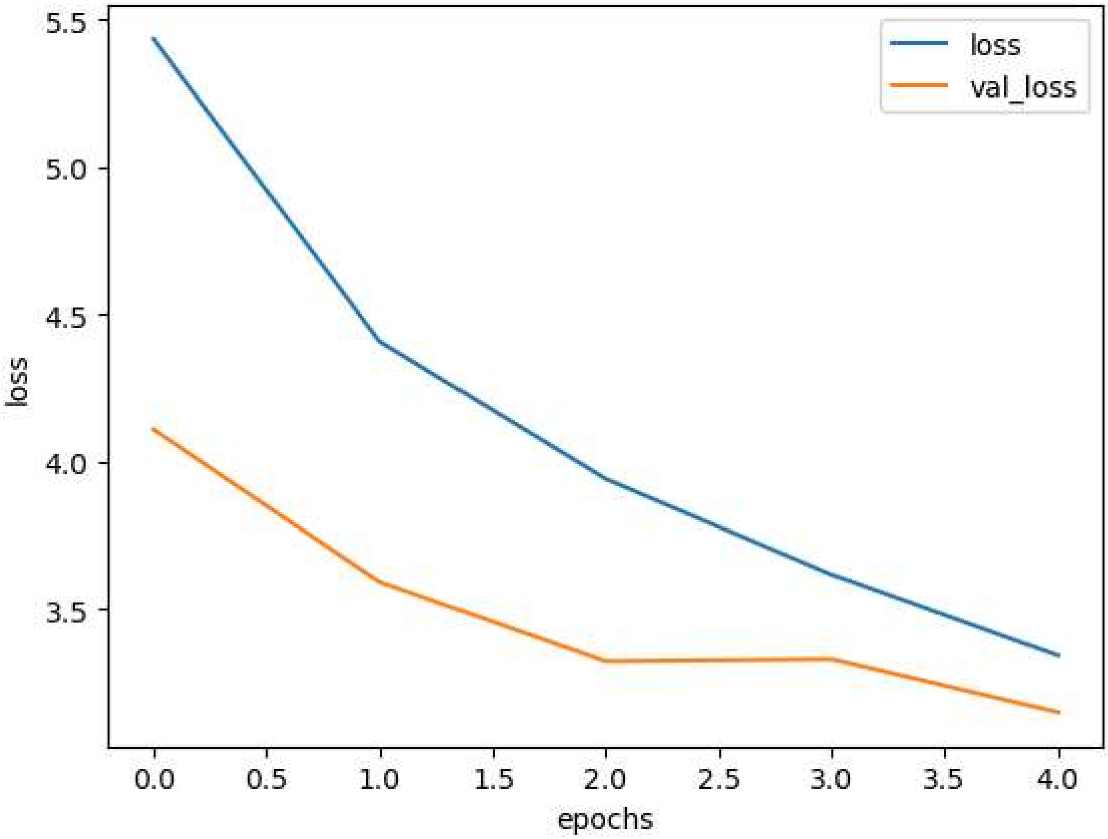
In [23]:

```
model_training = rnn_model.fit([Xt_features, Xt_text], yt_text,
                               epochs = 5, verbose = 2, batch_size = 64,
                               validation_data = ([Xv_features, Xv_text], yv_te
```

```
Epoch 1/5
WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\src
\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. P
lease use tf.compat.v1.ragged.RaggedTensorValue instead.

1037/1037 - 199s - loss: 5.4336 - val_loss: 4.1084 - 199s/epoch - 192ms/step
Epoch 2/5
1037/1037 - 185s - loss: 4.4075 - val_loss: 3.5911 - 185s/epoch - 178ms/step
Epoch 3/5
1037/1037 - 185s - loss: 3.9406 - val_loss: 3.3219 - 185s/epoch - 178ms/step
Epoch 4/5
1037/1037 - 187s - loss: 3.6157 - val_loss: 3.3285 - 187s/epoch - 181ms/step
Epoch 5/5
1037/1037 - 185s - loss: 3.3421 - val_loss: 3.1480 - 185s/epoch - 178ms/step
```

In [24]:

```
for label in ["loss", "val_loss"]:
    plt.plot(model_training.history[label], label = label)
plt.legend()
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()
```



In [25]:

```
rnn_model.save("RNN_Model.h5")
cnn_model.save("CNN_Model.h5")
dump(tokenizer, open('Flickr8K_Tokenizer.p', 'wb'))
```

```
C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\engine\training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. Th
is file format is considered legacy. We recommend using instead the native Ke
ras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have y
et to be built. `model.compile_metrics` will be empty until you train or eval
uate the model.
```

In [26]:

```
def generate_caption(filename):
    img = load_img(filename, target_size = (299, 299))
    img = img_to_array(img)
    img = np.expand_dims(img, axis = 0)
    img = preprocess_input(img)
    features = cnn_model.predict(img)
    in_text = 'startseq'
    for i in range(max_len):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_len)
        pred = rnn_model.predict([features,sequence], verbose=0)
        pred = np.argmax(pred)
        word = index_to_word[pred]
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    return in_text
```

In [27]:

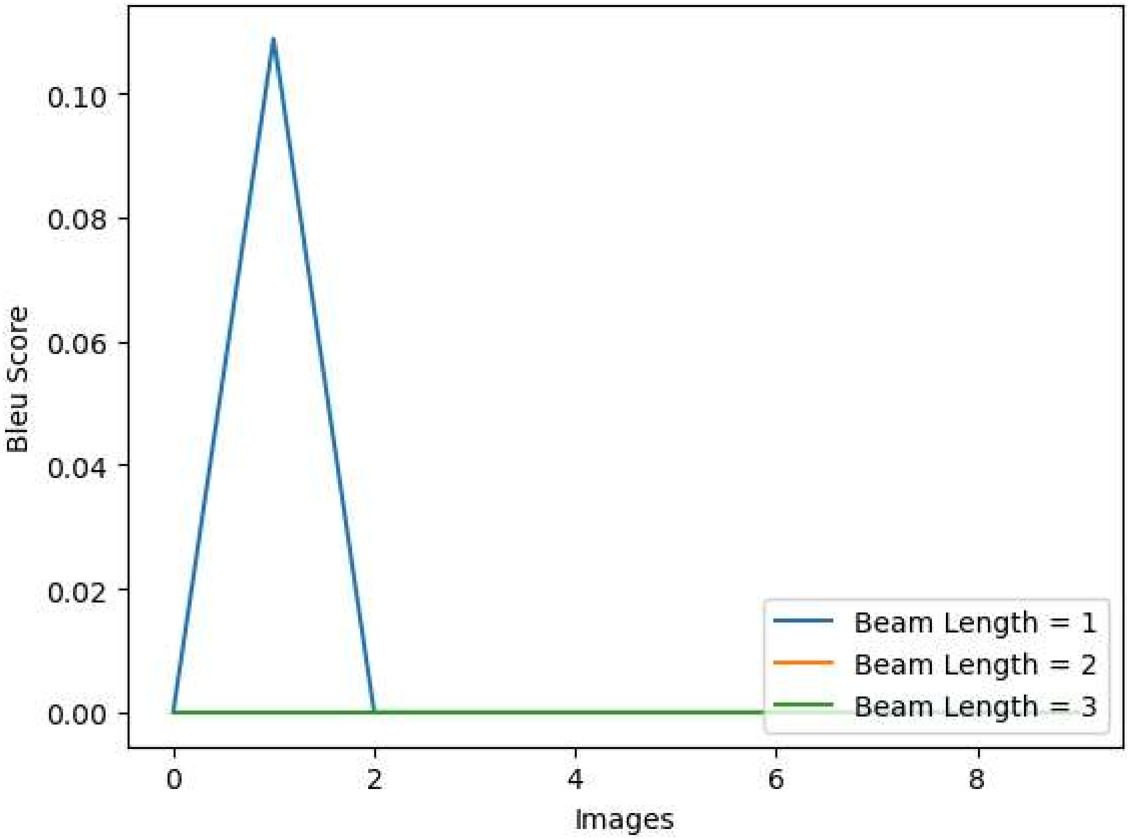
```
def generate_caption_beam_search(filename, max_length, beam_index):
    img = load_img(filename, target_size = (299, 299))
    img = img_to_array(img)
    img = np.expand_dims(img, axis = 0)
    img = preprocess_input(img)
    features = cnn_model.predict(img)
    in_text = [[tokenizer.texts_to_sequences(['startseq'])[0], 0.0]]
    while len(in_text[0][0]) < max_length:
        tempList = []
        for seq in in_text:
            padded_seq = pad_sequences([seq[0]], maxlen=max_length)
            preds = rnn_model.predict([features,padded_seq], verbose=0)
            top_preds = np.argsort(preds[0])[-beam_index:]
            for word in top_preds:
                next_seq, prob = seq[0][:], seq[1]
                next_seq.append(word)
                prob += preds[0][word]
                tempList.append([next_seq, prob])
        in_text = tempList
        in_text = sorted(in_text, reverse=False, key=lambda l: l[1])
        in_text = in_text[-beam_index:]
    in_text = in_text[-1][0]
    final_caption_raw = [index_to_word[i] for i in in_text]
    final_caption = []
    for word in final_caption_raw:
        if word == 'endseq':
            break
        else:
            final_caption.append(word)
    final_caption.append('endseq')
    return ' '.join(final_caption)
```

In [28]:

```python
scores, beam2_scores, beam3_scores = [], [], []
for img, caption in zip(test_images, test_captions):
    hypothesis = generate_caption(input_dir + "Images/" + img + ".jpg")
    scores.append(sentence_bleu([caption.split()], hypothesis.split()))
    hypothesis = generate_caption_beam_search(input_dir + "Images/" + img + ".j
    beam2_scores.append(sentence_bleu([caption.split()], hypothesis.split()))
    hypothesis = generate_caption_beam_search(input_dir + "Images/" + img + ".j
    beam3_scores.append(sentence_bleu([caption.split()], hypothesis.split()))

for i, score in enumerate([scores, beam2_scores, beam3_scores]):
    plt.plot(score, label = 'Beam Length = ' + str(i+1))
plt.legend(loc = 'lower right')
plt.xlabel('Images')
plt.ylabel('Bleu Score')
plt.show()
```

```
1/1 [==============================] - 0s 237ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 173ms/step

C:\Users\Admin\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552:
UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)

1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 171ms/step

C:\Users\Admin\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552:
UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)

1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 157ms/step

C:\Users\Admin\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552:
UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
```

```
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 236ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 157ms/step
```



In [29]:

```
npic = 10
npix = 299
target_size = (npix,npix,3)

count = 1
fig = plt.figure(figsize=(20,20))
for img, true_caption in zip(test_images, test_captions):

    filename = input_dir + 'Images/' + img + ".jpg"
    image_load = load_img(filename, target_size = target_size)
    ax = fig.add_subplot(npic, 2, count, xticks=[], yticks=[])
    true_caption = ' '.join(true_caption.split()[1: -1])
    ax.imshow(image_load)
    count += 1

    caption = generate_caption(filename)
    caption = ' '.join(caption.split()[1: -1])
    ax = fig.add_subplot(npic, 2, count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
    ax.set_ylim(0,1)
    ax.text(0, 0.7, true_caption, fontsize = 20)
    ax.text(0, 0.4, caption, fontsize = 20)
    ax.text(0, 0.1, 'Bleu Score = {}'.format(sentence_bleu([true_caption.split
    count += 1

plt.show()
```

```
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 198ms/step
1/1 [==============================] - 0s 199ms/step
1/1 [==============================] - 0s 231ms/step
1/1 [==============================] - 0s 202ms/step
1/1 [==============================] - 0s 217ms/step
1/1 [==============================] - 0s 202ms/step
1/1 [==============================] - 0s 200ms/step
1/1 [==============================] - 0s 170ms/step
```

```
girl climbing into wooden
two girls are sitting on the floor with two fingers in the background and the other are looking at the camera in the background and smiles in
Bleu Score = 0
```

```
dog and white dog with brown spots are staring at each other in the
and white dog with red collar is running through the snow with stick in its mouth and black dog in the snow with stick in its mouth
Bleu Score = 0.10738037495669005
```

```
girl covered in paint sits in front of painted rainbow with her hands in
little girl in pink shirt and blue shirt is running through the grass with the camera in the background and the other girl in the background and
Bleu Score = 1.1951155126698611e-231
```

```
lays on the bench to which white dog is also
dog is jumping over the air on the ground with two other people in the background and the other are playing in the air with two other
Bleu Score = 4.87371842328988e-155
```

```
in an orange hat starring at
in front of crowd of people in front of crowd of people in front of crowd of people in the background and an umbrella and the other
Bleu Score = 9.50440384721771e-232
```

```
child grips onto the red ropes at the
little girl in red shirt and blue shirt is jumping off swing with two children in the background and the other are sitting on the deck of
Bleu Score = 1.0518351895246305e-231
```

```
runs on the green grass near wooden
and white and white dog running in the grass with stick in its mouth and black and white dog in the grass with stick in its mouth
Bleu Score = 9.50440384721771e-232
```

```
with orange ball at feet stands on shore shaking off
black and white dog running through the grass with stick in its mouth in the background with stick in its mouth and black dog in the snow
Bleu Score = 7.992219124248642e-232
```

```
boy runs aross the
rock man in red shirt and blue shirt is walking down the sidewalk with two other people watch the bottom of the bottom of the side of
Bleu Score = 7.992219124248642e-232
```

```
black dog jumped the tree
dog is running through the grass with stick in its mouth in the background with the camera in its mouth and black dog in its mouth in
Bleu Score = 3.8138907759550345e-155
```

In [ ]:

In [ ]:

In [ ]: