Deep Learning- CSE641

Assignment - 4

Name: Akanksha Shrimal **Roll No:** MT20055
Name: Vaibhav Goswami **Roll No:** MT20018
Name: Shivam Sharma **Roll No**: MT20121

FILES SUBMITTED : submitted .py file , .ipynb file and readme.pdf and output.pdf

References:

1. https://github.com/AaronCCWong/Show-Attend-and-Tell
2. https://github.com/jiangqn/IAN-pytorch

UTILITY FUNCTIONS :

1. Save and Load models using Pickle

```
# Saving and Loading models using pickle
def save(filename, obj):
 with open(filename, 'wb') as handle:
     pickle.dump(obj, handle, protocol=pickle.HIGHEST_PROTOCOL)

def load(filename):
 with open(filename, 'rb') as handle:
     return pickle.load(handle)
```

2. Pre Processing data - Normalization and One Hot Encoding

```
# Utility function to normalize the data and one hot encode the
labels
def pre_process_data(train_x, train_y, test_x, test_y):
   # Normalize
   train_x = train_x / 255.
   test_x = test_x / 255.
   enc = OneHotEncoder(sparse=False, categories='auto')
   train_y = enc.fit_transform(train_y.reshape(len(train_y), -1))
   test_y = enc.transform(test_y.reshape(len(test_y), -1))
   return train_x, train_y, test_x, test_y
```

3. Plotting functions

```
# function to plot double line graph
# Plot double line using X1 , Y1 and X2 , Y2
def plot_double_line_graph(X1,Y1,label1 ,X2 ,Y2,label2
,title,y_name):
 fig = plt.figure(figsize=(7,5))
 plt.subplot(111)
  plt.plot(X1,Y1 ,label=label1 ,marker = "x" , color="blue")
 plt.plot(X2, Y2 , label=label2 ,marker = "x" , color="red")
 plt.title(title)
 plt.ylabel(y_name)
```

```
 plt.xlabel('Epochs')
 plt.legend( loc='upper left',prop={'size': 13})
 plt.show()


# Plot simgle line using X1 , Y1
def plot_single_line_graph(X1,Y1,label1, title,name_y):
 fig = plt.figure(figsize=(7,5))
 plt.subplot(111)
  plt.plot(X1,Y1 ,label=label1 ,marker = "x" , color="blue")
 plt.title(title)

 plt.ylabel(name_y)
 plt.xlabel('Epochs')
 plt.legend( loc='lower right',prop={'size': 13})
 plt.show()
```

4. Plotting ROC Curve

```
# (7,7)
#https://www.dlology.com/blog/simple-guide-on-how-to-generate-roc-
plot-for-keras-classifier/
def plot_roc(classes, y_test, y_score, figsize=(7,7)):
 n_classes = len(classes)
 # Plot linewidth.
 lw = 2

 # Compute ROC curve and ROC area for each class
 fpr = dict()
 tpr = dict()
 roc_auc = dict()
 for i in range(n_classes):
     fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
     roc_auc[i] = auc(fpr[i], tpr[i])

 # Compute micro-average ROC curve and ROC area
 fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(),
y_score.ravel())
 roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

```
 # Compute macro-average ROC curve and ROC area

 # First aggregate all false positive rates
 all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))

 # Then interpolate all ROC curves at this points
 mean_tpr = np.zeros_like(all_fpr)
 for i in range(n_classes):
     mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

 # Finally average it and compute AUC
 mean_tpr /= n_classes

 fpr["macro"] = all_fpr
 tpr["macro"] = mean_tpr
 roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

 # Plot all ROC curves
 plt.figure(1)
 plt.figure(figsize=figsize)
 plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

 plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

 colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
 for i, color in zip(range(n_classes), colors):
     plt.plot(fpr[i], tpr[i], color=color, lw=lw,
              label='ROC curve of class {0} (area = {1:0.2f})'
              ''.format(i, roc_auc[i]))

 plt.plot([0, 1], [0, 1], 'k--', lw=lw)
 plt.xlim([0.0, 1.0])
```

```
 plt.ylim([0.0, 1.05])
 plt.xlabel('False Positive Rate')
 plt.ylabel('True Positive Rate')
 plt.title('Some extension of Receiver operating characteristic to
multi-class')
 plt.legend(loc="lower right")
 plt.show()
```

6. Plotting Confusion Matrix

```
def confusion_matrix_find(y, y_hat, nclasses):

    y = y.astype(np.int64)
    y_hat = y_hat.astype(np.int64)

    conf_mat = np.zeros((nclasses, nclasses))

    for i in range(y_hat.shape[0]):
        true, pred = y[i], y_hat[i]
        conf_mat[true, pred] += 1
    return conf_mat
# Plotting confusion matrix
def confusion_matrix_plot(cm, classes, title='Confusion matrix',
cmap=plt.cm.Blues, figsize=(7,7), path=None, filename=None):

    cm = cm.astype(np.int64)
    plt.figure(figsize=figsize)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = 'd'
    thresh = cm.max() / 2.
```

```
   for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
   plt.ylabel('True label')
   plt.xlabel('Predicted label')
   plt.tight_layout()
```

1. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention using soft attention

1.1 Dataset description:

1.2 Task Description:

- To build a model that can generate a descriptive caption for an image.
- The model uses an encoder-decoder architecture.
- The task is also achieved by using attention in images to find relevant and appropriate information in the image as required.
- The model also uses a pre-trained model for quick learning and better performance than a model trained from scratch.

1.3 Model Description

1.3.1 Pre-trained Model

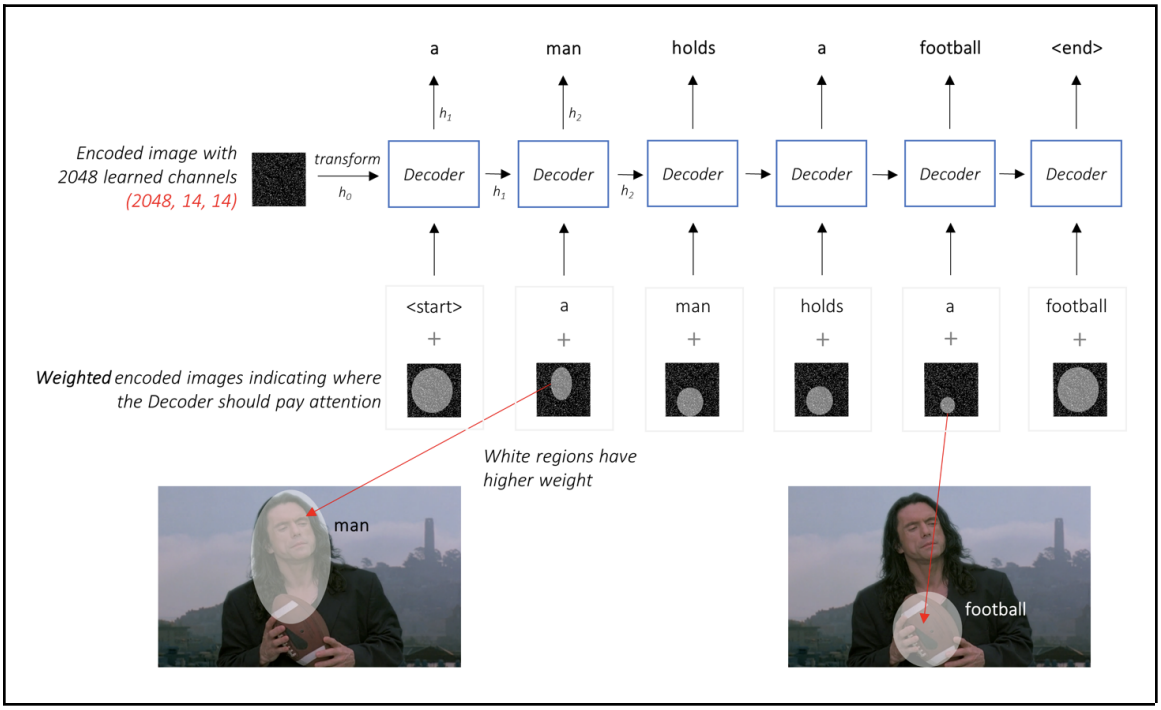
- VGG19 is used as a pre-trained model for encoding images in desired representation.
- The images are resized along with parameters like mean and standard deviation are provided in accordance with the pre-trained model's specifications for it to be able to train and process the images.
- This method is known as transfer learning.

1.3.2 Encoder

- The Encoder encodes the input image with 3 color channels into a smaller image with "learned" channels.
- The encoded image is a representation of all the useful parts of the original image.
- The encoder is implemented by a Convolutional Neural Network (CNN).
- VGG19 is used to encode the images.
- The model progressively creates smaller and smaller representations of the original image, and each subsequent representation is more "learned", with a greater number of channels.
- The last layer or two, which are linear layers with softmax function, are removed.

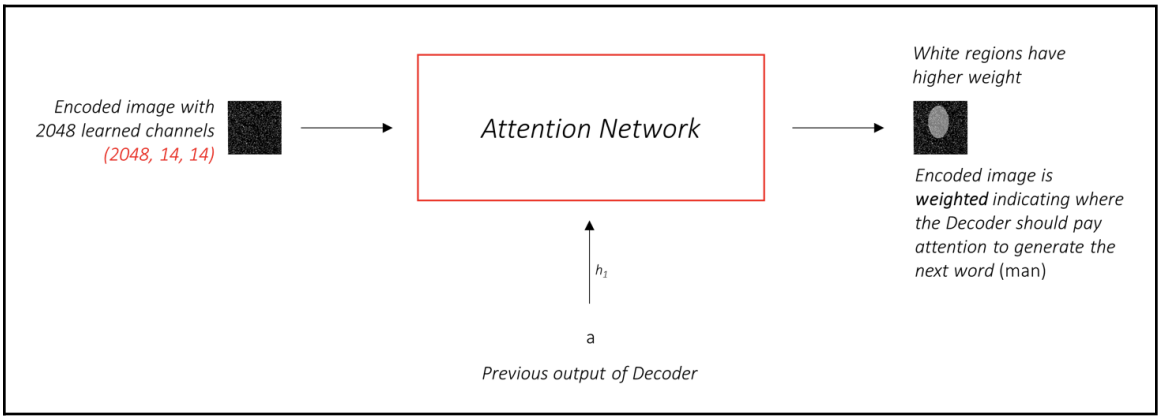
1.3.3 Decoder

- The Decoder's job is to look at the encoded image and generate a caption word by word.
- As a sequence is generated, Recurrent Neural Network (RNN) should be used for this purpose.
- Long Short Term Memory (LSTM) is used in the implementation as it learns the sequential nature of data along with overcoming vanishing/exploding gradient problems in vanilla RNN.
- Attention is used with the decoder so that the decoder can look at different parts of the image at different points in the sequence.
- Weighted average was used across all the pixels which can be used with the previous word to generate the next word.



1.3.4 Attention

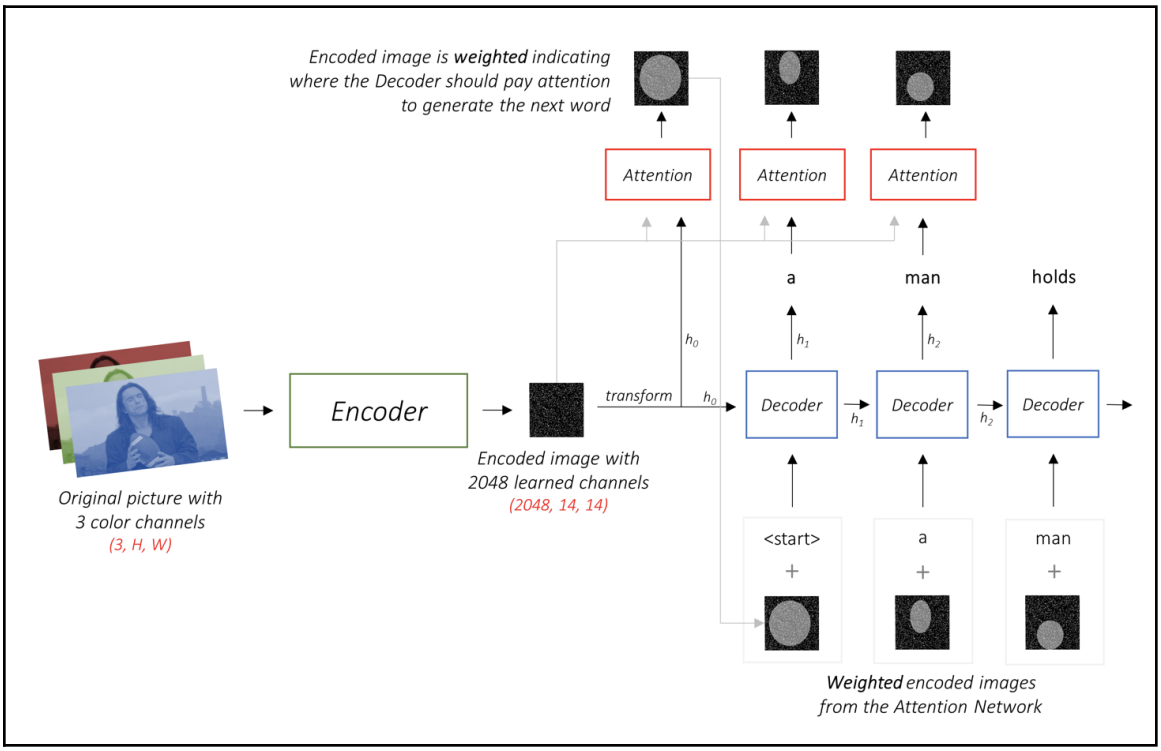
- The attention network is used to compute all the attention weights.
- The attention weights are used to generate the next word with the help of decoder.
- It considers the sequence till now and focuses/attends to the part of the image that needs to be described next.
- Soft attention is used in the implementation.
- It is where weights of pixels are added up to 1.
- It can be interpreted as calculating the probability of a pixel to be an appropriate place in the image needed to generate the next word.



1.3.5 Overall Summary

- The encoder generates the encoded images. VGG19 is used for encoding of images which is a pre-trained CNN model.
- Once the Encoder generates the encoded image, we transform the encoding to create the initial hidden state h (and cell state C) for the LSTM Decoder.
- At the decoding stage,

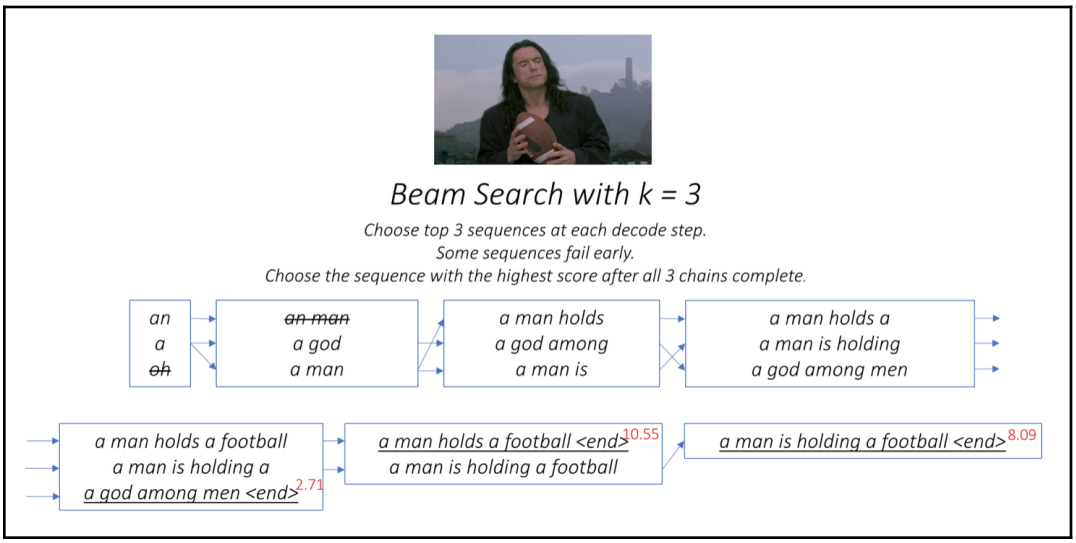
- the encoded image and the previous hidden state is used to generate weights for each pixel in the Attention network.
- the previously generated word and the weighted average of the encoding are fed to the LSTM Decoder to generate the next word.



1.4 Beam Search

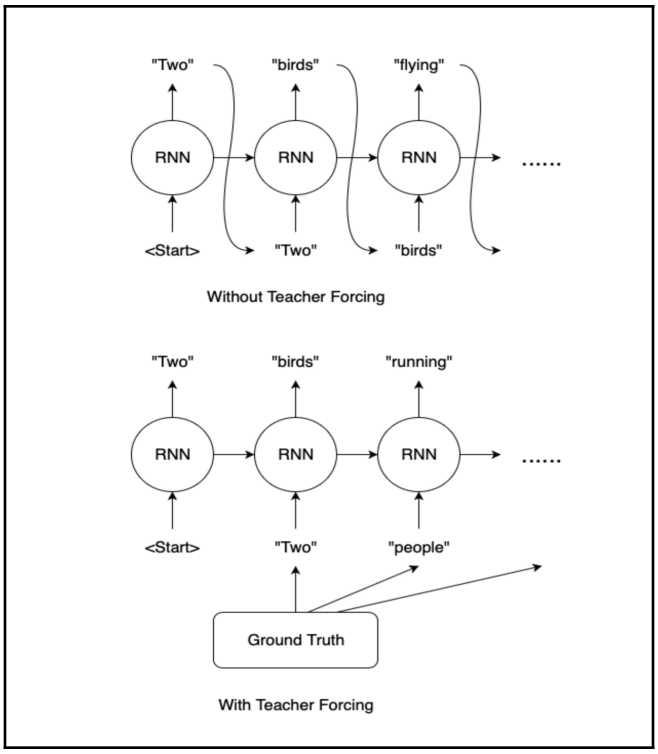
- A linear layer is used to transform the Decoder's output into a score for each word in the vocabulary.
- The straightforward – and greedy – option would be to choose the word with the highest score and use it to predict the next word.
- But this is not optimal because the rest of the sequence hinges on that first word you choose.
- If that choice isn't the best, everything that follows is sub-optimal. And it's not just the first word – each word in the sequence has consequences for the ones that succeed it.
- It would be best if we could somehow *not* decide until we've finished decoding completely, and choose the sequence that has the highest *overall* score from a basket of candidate sequences.
- This is exactly what Beam Search does.
    - At the first decode step, consider the top k candidates.
    - Generate k second words for each of these k first words.
    - Choose the top k [first word, second word] combinations considering additive scores.
    - For each of these k second words, choose k third words, choose the top k [first word, second word, third word] combinations.
    - Repeat at each decode step.
    - After k sequences terminate, choose the sequence with the best overall score.

- Some sequences may fail early, as they don't make it to the top k at the next step. Once k sequences generate the <end> token, we choose the one with the highest score.



1.5 Teacher Forcing

- Teacher forcing is a method for quickly and efficiently training recurrent neural network models that use the ground truth from a prior time step as input.
- It passes ground truth instead of prediction from the previous timestamp to find output at the next timestamp.
- Using predicted outputs can lead to a sequence of errors. If one predicted output is wrong, it is used as input to the next timestamp and the output of that would also be wrong. Therefore, it can lead to a series of errors.
- Teacher Forcing overcomes the above-mentioned problem by passing ground truth values instead of predicted values to calculate output.



1.6 Working Model

1.6.1 Input and Pre-Processing

- The images along with captions are given in the assignment, split into train, validation and test sets.
- As we are using a pre-trained encoder (VGG19), we would need to pre process the images in required format and specifications by VGG19.
- Specifications are as follows:
    - Mean = [0.485, 0.456, 0.406]
    - Standard Deviation = [0.229, 0.224, 0.225]
    - The images are resized to size of (256 x 256)
    - Then the images are cropped to size of (224 x 224)
- The captions are both targets and input as the previous word is used to generate the next word.
- To generate the first word, a zeroth word is used which is as <*start*>
- The last word is kept as <*end*> which indicated the decoder to stop decoding during inference.
- The captions are padded to equal lengths. If a caption is less than the maximum length, it is padded with <*pad*> tokens.
- A word mapping/word indexing is created for every word including <*start*>, <*end*> and <*pad*>.
- Therefore, captions fed to the model are integer type.

1.6.2 Files and Dataset Class

- 8 files have been created and pickled as follows:
    - train_img_paths.json - This file contains the paths of images in the train dataset.
    - train_captions.json - This file contains the captions of respective images in the training dataset.
    - test_img_paths.json - This file contains the paths of images in the test dataset.
    - test_captions.json - This file contains the captions of respective images in the test dataset.
    - val_img_paths.json - This file contains the paths of images in the validation dataset.
    - val_captions.json - This file contains the captions of respective images in the validation dataset.
    - word_dict.json - This file is for word indexing, containing a dictionary of words with index. The key is the word and item is the index number in the dictionary.
    - word_dict_reverse.json - This file is for reverse word indexing, containing a dictionary of words with index, but in reverse manner to the above-mentioned. The key is the index and item is the word in the dictionary.
- Dataset class has been created which is used to initialize and fetch data.
- It initializes the required data by loading the respective images and captions, from the file paths.
- It returns the required image, correct caption and all the 5 captions for the image.

1.6.2 Encoder

- Pre-trained VGG19 is imported from pytorch library.

- The last layer is removed which is a linear layer with softmax function as we don't need it for prediction but to just encode images and extract features from the images.

1.6.3 Attention

- It is a simple architecture, composed of linear layers and activation functions.
- Separate linear layers transform both the encoded image and the hidden state (output) from the Decoder to the same dimension, viz. the Attention size.
- Activation function is applied and then a linear layer is used to transform the dimension and then softmax is used to generate weights.

1.6.4 Decoder

- The output of the Encoder is received here and flattened.
- LSTM with two separate linear layers is initialized and used.
- The sequence is processed one word (timestamp) at a time and sorting is done to pick top images, using top outputs from previous outputs.
- Weights and attention-weights are computed at each timestamp using attention mechanism.

1.7 Evaluation Metrics

1.7.1 BLEU[1-4]

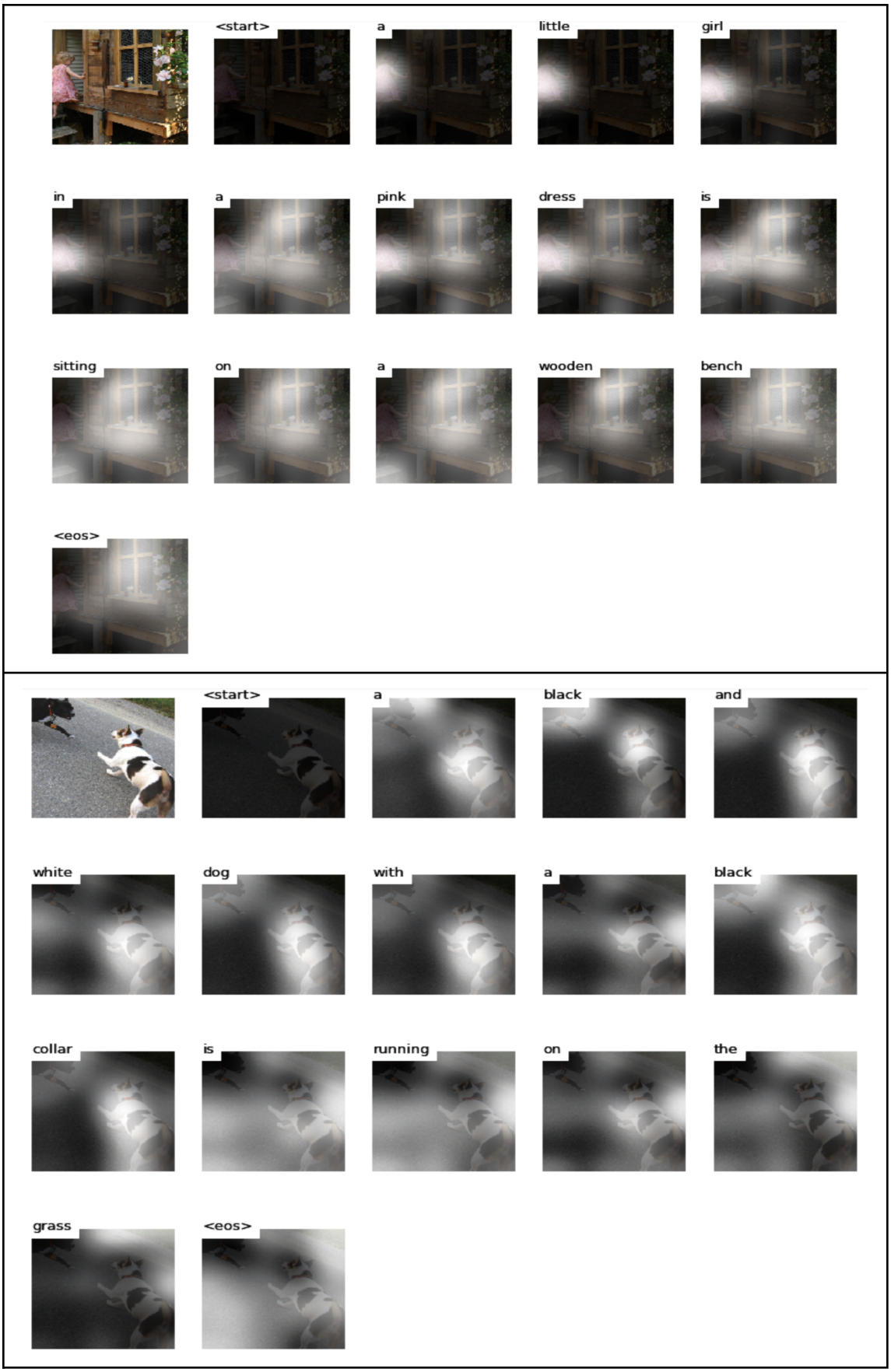
- The Bilingual Evaluation Understudy Score, or BLEU for short, is a metric for evaluating a generated sentence to a reference sentence.
- The BLEU score evaluates the quality of text that has been translated by a machine from one natural language to another.
- A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0.
- The weights for the BLEU-4 are 1/4 (25%) or 0.25 for each of the 1-gram, 2-gram, 3-gram and 4-gram scores.

1.7.2 METEOR

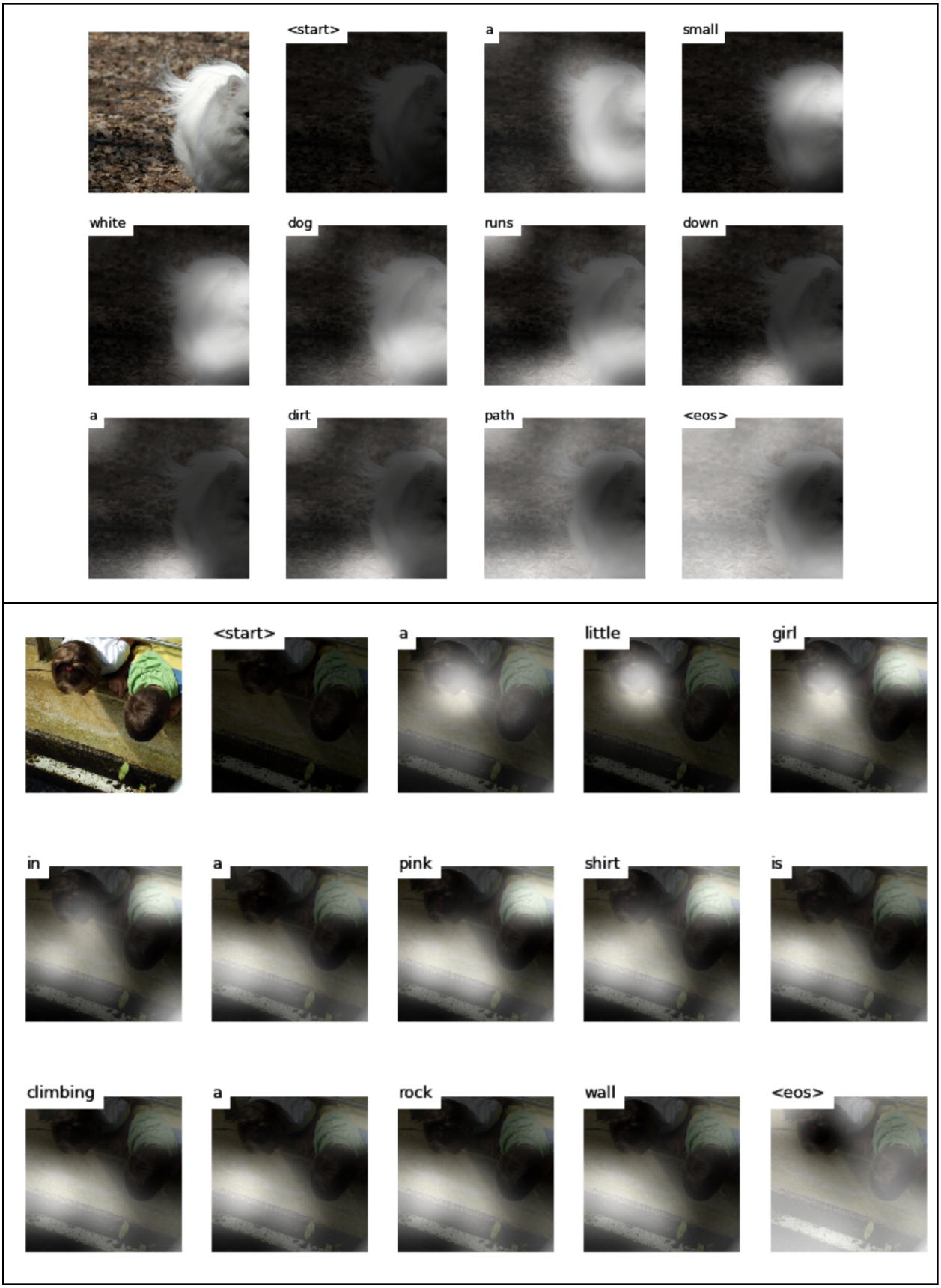
- The *Metric for Evaluation of Translation with Explicit ORdering* (METEOR) is a precision-based metric for the evaluation of machine-translation output.
- It overcomes some of the pitfalls of the BLEU score, such as exact word matching whilst calculating precision.
- The METEOR score allows synonyms and stemmed words to be matched with a reference word.

1.8 Attention Weights

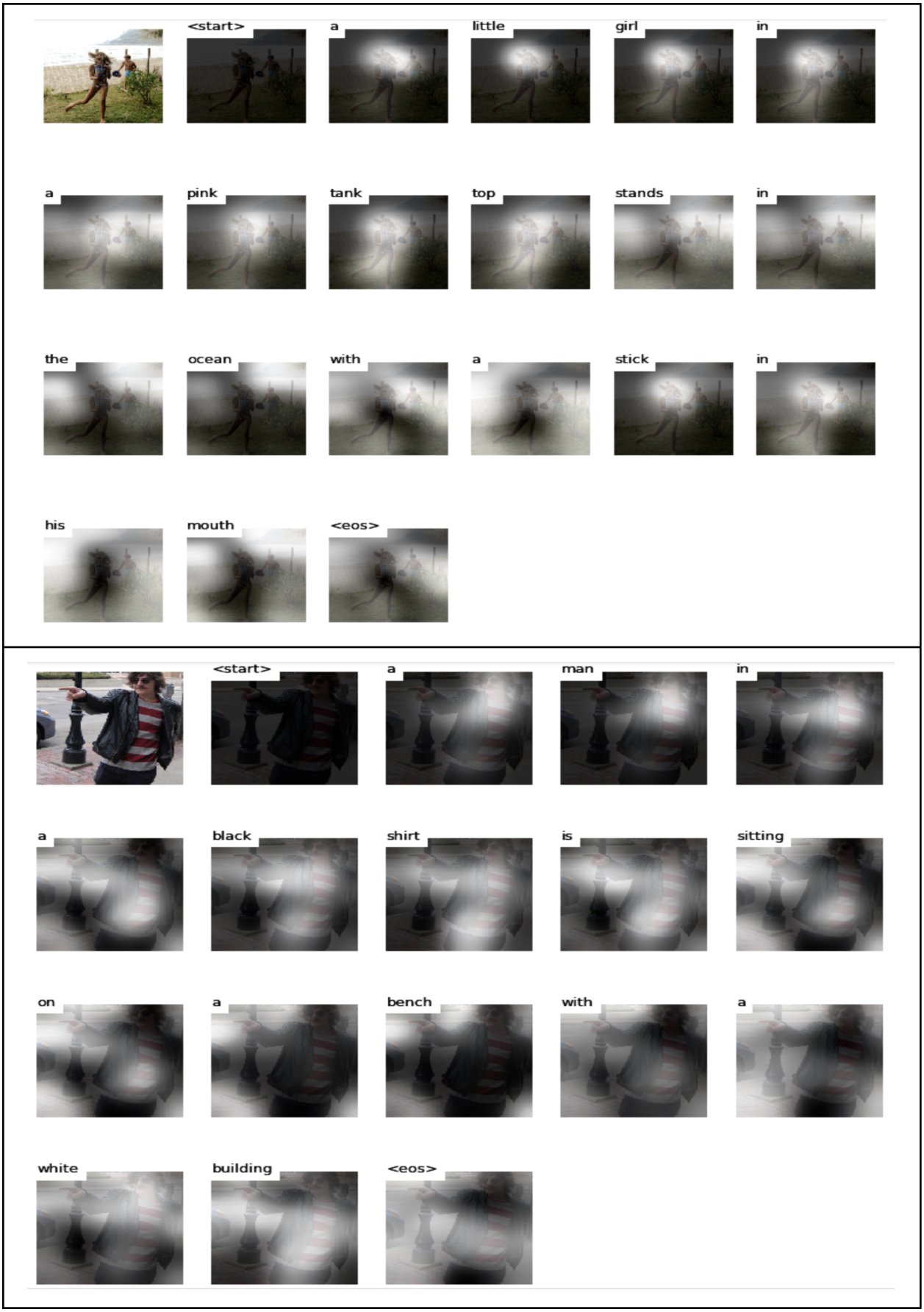
1.8.1 Train Dataset

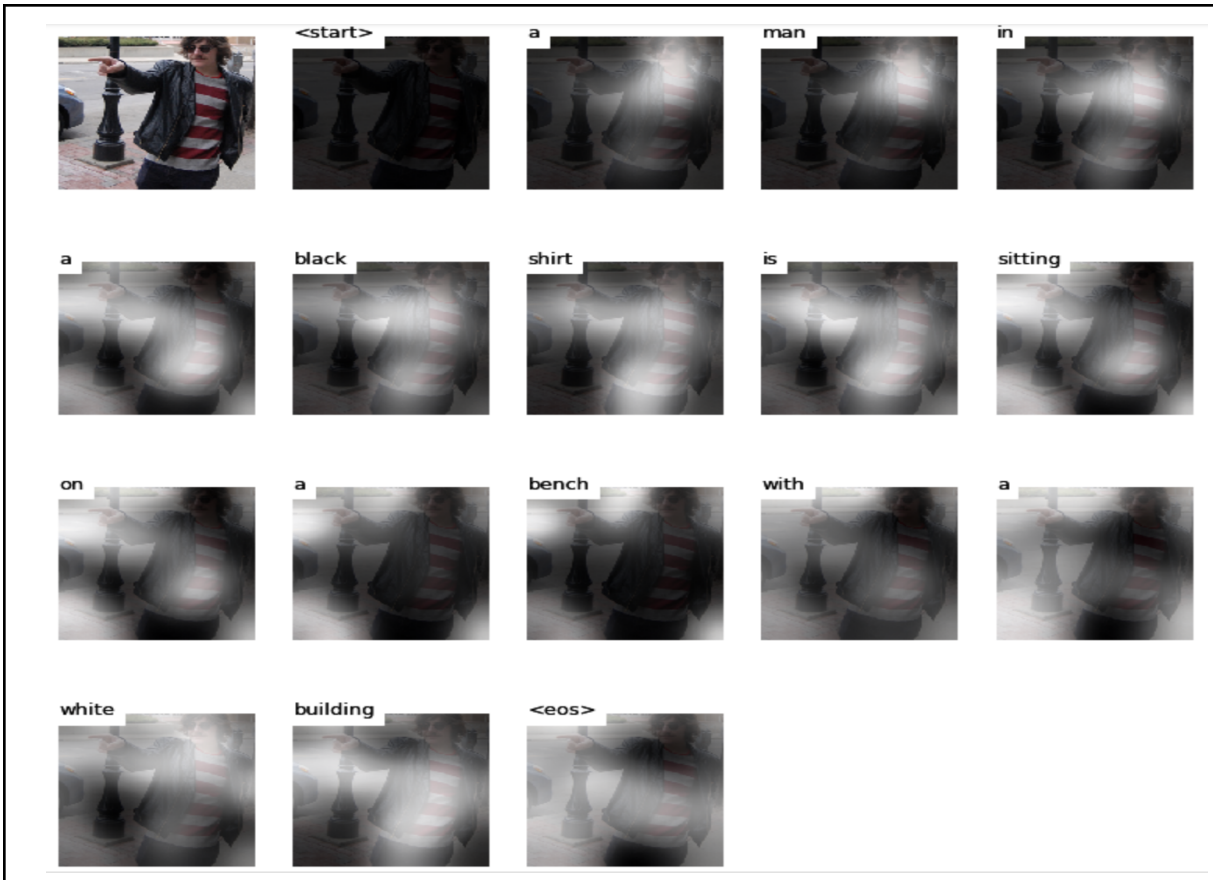


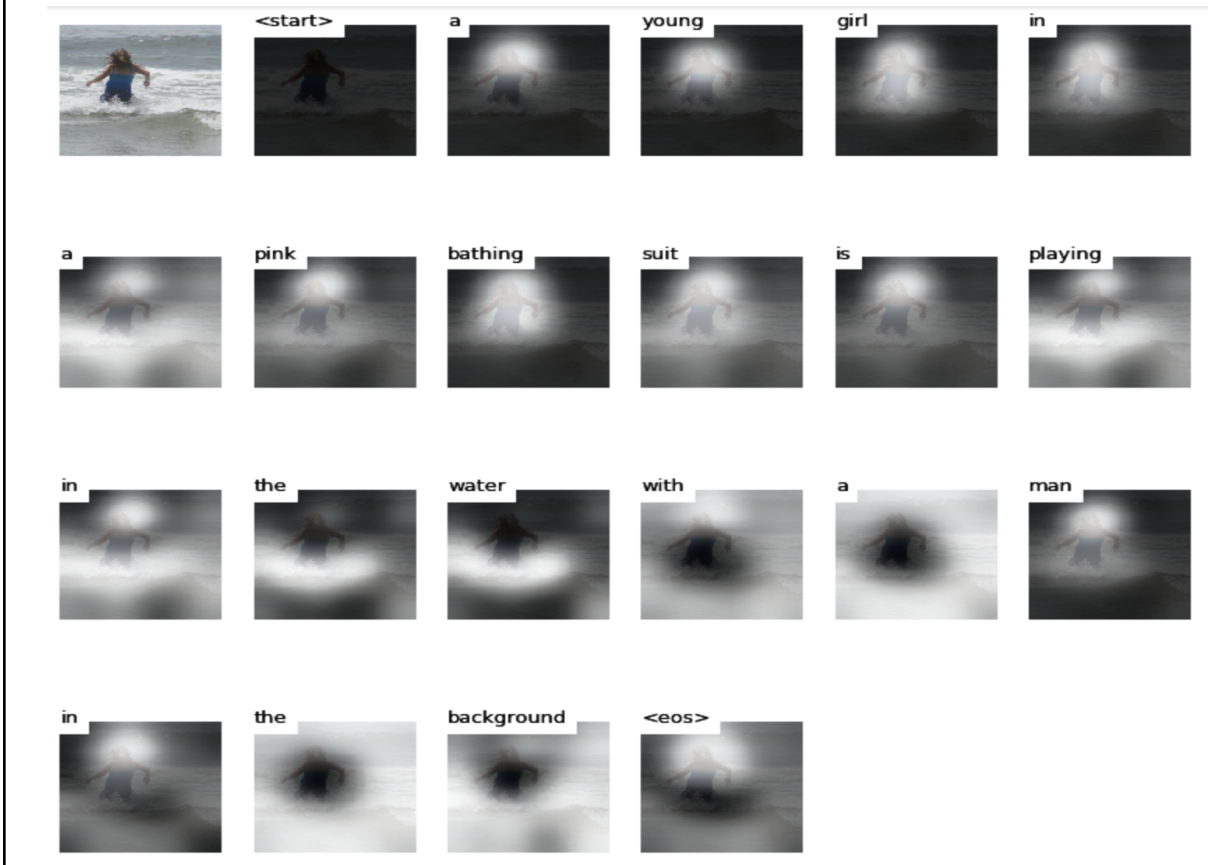
1.8.2 Validation Dataset



1.8.3 Test Dataset




<start> a man in
a black shirt is sitting
on a bench with a
white building <eos>


<start> a young girl in
a pink bathing suit is playing
in the water with a man
in the background <eos>

1.9 Results

1.9.1 Validation Dataset

- Number of epochs - 25
- Validation Loss - 2.8601
- Accuracy:

Metric	Accuracy (%)
BLEU-1	58.2
BLEU-2	37.9
BLEU-3	24.9
BLEU-4	15.9
METEOR	48.2

1.9.2 Test Dataset

- Number of epochs - 25
- Test Loss - 2.5919

- Accuracy:

Metric	Accuracy (%)
BLEU-1	59.6
BLEU-2	38.9
BLEU-3	25.5
BLEU-4	16.3
METEOR	53.1

Contribution of Each Member

1. **Shivam Sharma**
   Implemented (2) and report
2. **Akanksha Shrimal**
   Implemented (1) and report
3. **Vaibhav Goswami**
   Implemented (2) and report.