Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

Name : Varad Sinai Kakodkar, Palak Patel

Net ID: vns2008, pbp2022

Background

What is Image Captioning?

An image is fed into an image captioning application, which produces a short text description of the image's contents. To identify the context of a picture and describe it in a natural language like English, it incorporates computer vision with natural language processing ideas.

Normal Process of Image Captioning

For an application, we begin with image files as the input and extract their key information in a small, encoded form. These will then be fed into an LSTM(Long short-term memory)-based sequence decoder (an artificial neural network that predicts the future from sequences of variable length), which will decode the encoded image and forecast a string of words that best characterizes the image.

Problem with 'Classic' Image Captioning Model

The issue with this approach is that the model usually only manages to describe a small portion of the image when it attempts to generate the next word of the caption. It is unable to fully convey the meaning of the input image. It is not possible to efficiently generate different words for various regions of the image using the entire representation of the image as a criterion. Exactly in this situation is where an attention technique is useful.

Abstract

In this paper, an attention-based methodology for automatically creating captions for images is described. The model has an attention mechanism that enables it to concentrate on important aspects of the image and provide captions that are more in-depth and comprehensive. By obtaining cutting-edge performance on three benchmark datasets—Flickr8k, Flickr30k, and MS COCO—the authors show the model's efficacy. Additionally, they demonstrate that the model is capable of automatically learning to fix its gaze on important items as it produces the matching words in the output sequence, underlining the potential benefit of attention visualization for readability.

Dataset

The datasets include a collection of image files as well as a text file that associates a caption with each image file. Each caption represents a language-specific sentence.

For the research investigation, they used MS COCO, Flickr8k, and Flickr30k.

For our study, we downloaded Flickr8k Dataset from: https://github.com/jbrownlee/Datasets/releases/download/Flickr8k

We have:

- Image files in the folder "Flicker8k Dataset": There are around 8091.jpg files in this folder.
- 'Flickr8k.token.txt' file captions: There are captions for each photograph. There are 5 descriptions per image since the same image can be described in a variety of ways.
- A group of.txt files in the main folder include a list of the training, validation, and test images: The list of image file names to be used for training, validation, and testing can be found in the file "Flickr 8k.trainImages.txt."

For our code, we will use the Flickr 8K dataset for the image caption generator because it takes weeks to train the network on the other two datasets.

Technique

> ### *Image Caption Generation with Attention Mechanism*

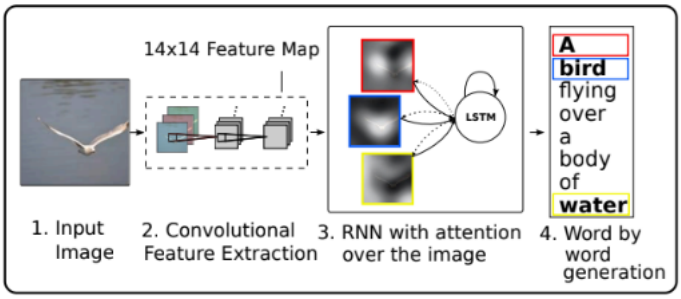


Fig 1: Model Diagram (Image from [1])

CNN(Convolutionl Neural Network):

Convolutional Neural Networks are deep neural networks that are capable of processing data with input shapes similar to a 2D matrix. CNN is particularly helpful when working with photos and can readily express images as a 2D matrix. CNN is primarily used to classify images and determine whether they are a bird, a jet, Superman, etc. Images are scanned from top to bottom and left to right to extract key details, which are then combined to identify the images. It can handle images that have been resized, rotated, translated, and perspective-shifted.

LSTM:

A type of RNN (recurrent neural network) called long short term memory, or LSTM, is helpful for addressing sequence prediction problems. On the basis of the previous paragraph, we can guess the next word. A forget gate allows the LSTM to process inputs while processing pertinent information while rejecting irrelevant information.

i)Model Details

The paper describes hard and soft methods(attention-based methodology) for generating image captions.

ii)Encoder:Convolutional Features

The model creates a caption y encoded as a series of 1-of-K encoded phrases using a raw image as input. The paper also discusses how the model creates a set of feature vectors (also known as annotation vectors) from the image using a convolutional neural network. These feature vectors are then used to create a connection between the feature vectors and different areas of the 2-D image. It should be noted that the decoder can concentrate on particular areas of the image by choosing a subset of the feature vectors because the model takes features from a lower convolutional layer.

iii)Attention Decoder

The model use a convolutional neural network to extract from a photo a set of feature vectors, also known as annotation vectors. A context vector is created by combining these vectors, and a recurrent neural network uses them to create a picture caption.Two models are put out by the authors, one of which employs a "hard" attention mechanism and the other, a "soft" attention mechanism. The "hard" attention method computes a weight for each annotation vector using a multilayer perceptron, and utilizes that weight to compute the context vector. Using a similar methodology, the "soft" attention mechanism combines the annotation vectors with weights to calculate the context vector.

> ### *Learning Stochastic “Hard” vs Deterministic “Soft” Attention*

i)Stochastic "Hard" Attention

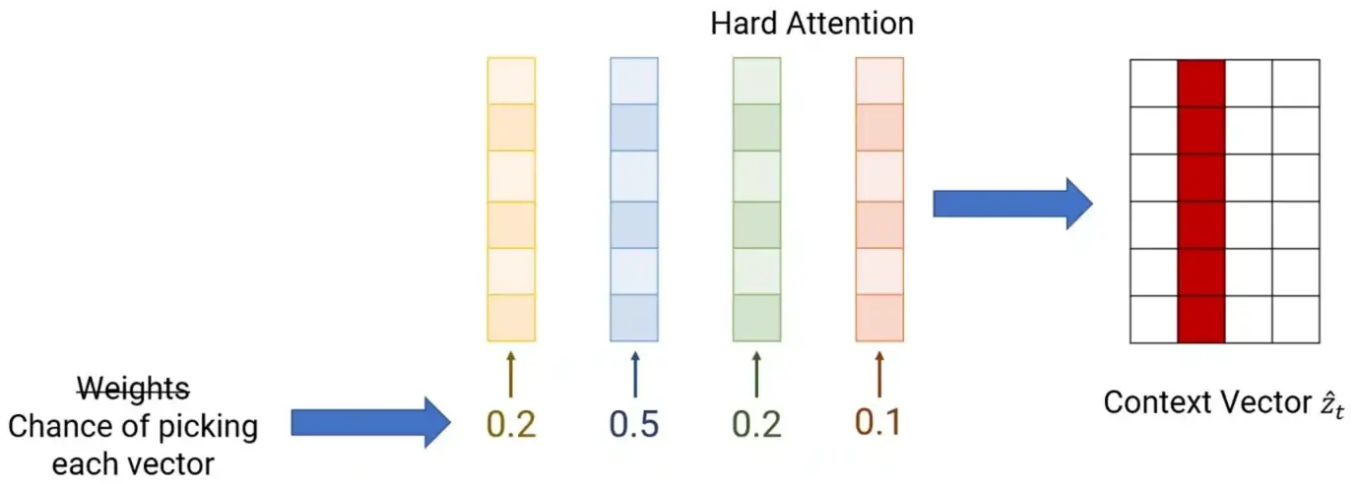


Fig 2: Hard Attention (Image from [3])

When creating a caption, a form of attention mechanism known as stochastic attention uses a random process to decide where to focus attention. The attention location is sampled from a multinouilli distribution defined by the probabilities calculated by the attention model and is treated as a latent variable. Instead of only picking one attention site at each time step, the model can now learn a distribution across all possible attention locations. By using this method, the learning algorithm's robustness can be increased by lowering the variance in the model's predictions of the log likelihood of the target sentence.

ii)Deterministic "Soft" Attention

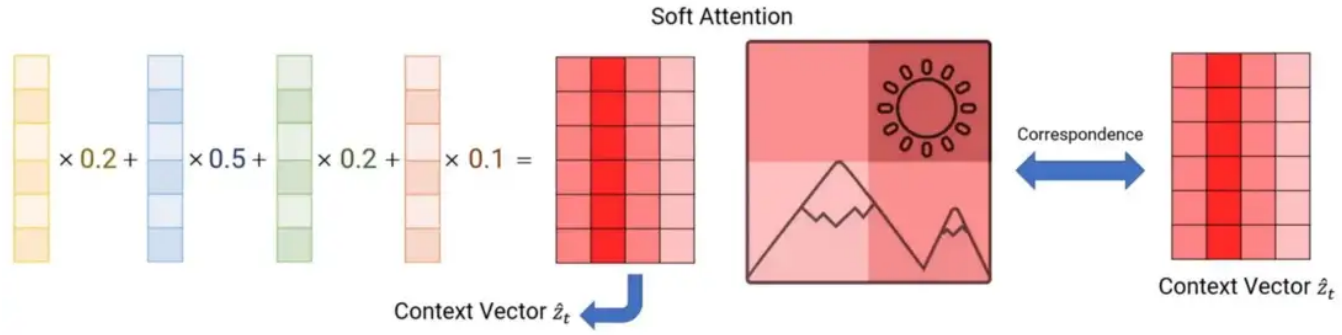


Fig 3: Soft Attention (Image from [3])

Deterministic attention is a form of attention mechanism that chooses where to focus attention based on a deterministic process. Instead of randomly selecting a place from a distribution in this method, the attention model always selects the location with the highest likelihood. As a result, the model's estimations of the log likelihood may be more precise but may also be more susceptible to inaccurate or noisy estimates of the attention probabilities.

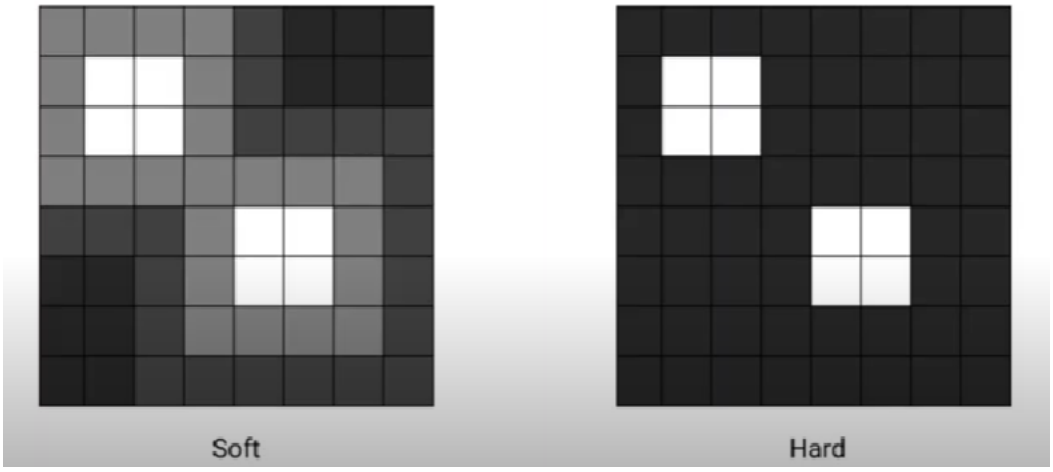


Fig 5: Soft vs Hard Attention (Image from [2])

The soft attention is smooth, whereas the hard attention concentrates solely on the portion it wants to and ignores the remainder. Different weights dependent on the image are employed in soft attention. Hard focus uses only the most crucial portion.

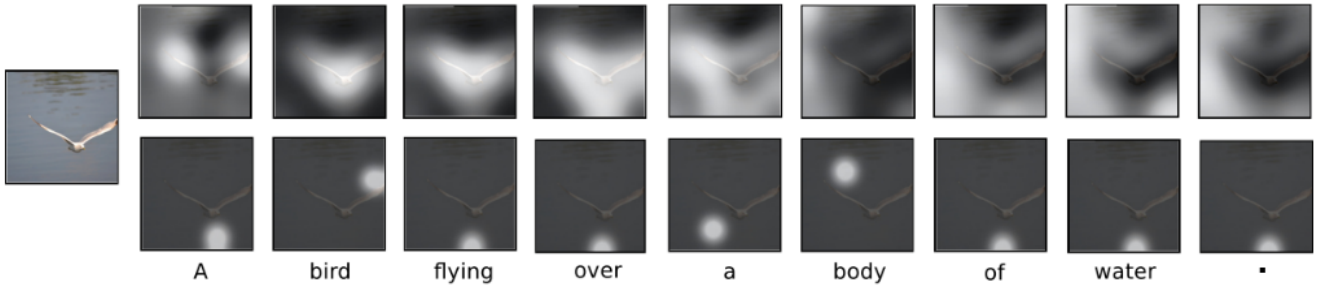


Fig 6: Hard vs Soft Attention example (Image from [1])

Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. "soft" (top row) vs "hard" (bottom row) attention. (Note that both models generated the same captions in this example.)

Conclusion

- The results showed that the model achieved state-of-the-art performance on all three datasets, i.e, Flickr8k, Flickr30k, and MS COCO datasets for the BLEU and METEOR Benchmarks demonstrating the effectiveness of the attention-based approach.

		BLEU				
Dataset	Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR
Flickr8k	Google NIC(Vinyals et al., 2014)$^{\dagger\Sigma}$	63	41	27	—	—
	Log Bilinear (Kiros et al., 2014a)$^{\circ}$	65.6	42.4	27.7	17.7	17.31
	Soft-Attention	**67**	44.8	29.9	19.5	18.93
	Hard-Attention	**67**	**45.7**	**31.4**	**21.3**	**20.30**
Flickr30k	Google NIC$^{\dagger\circ\Sigma}$	66.3	42.3	27.7	18.3	—
	Log Bilinear	60.0	38	25.4	17.1	16.88
	Soft-Attention	66.7	43.4	28.8	19.1	**18.49**
	Hard-Attention	**66.9**	**43.9**	**29.6**	**19.9**	18.46
COCO	CMU/MS Research (Chen & Zitnick, 2014)^{a}	—	—	—	—	20.41
	MS Research (Fang et al., 2014)$^{\dagger a}$	—	—	—	—	20.71
	BRNN (Karpathy & Li, 2014)$^{\circ}$	64.2	45.1	30.4	20.3	—
	Google NIC$^{\dagger\circ\Sigma}$	66.6	46.1	32.9	24.6	—
	Log Bilinear$^{\circ}$	70.8	48.9	34.4	24.3	20.03
	Soft-Attention	70.7	49.2	34.4	24.3	**23.90**
	Hard-Attention	**71.8**	**50.4**	**35.7**	**25.0**	23.04

Fig 7: Experimental Results (Image from [1])

- An additional level of interpretability can be added to a natural language processing (NLP) model's output by visually representing the attention component it has learned. The model's processing and interpretation of the input data can be better understood as a result, and this can also shed light on the model's decision-making process. In conclusion, showing the attention component of an NLP model can help make the model's behavior easier to understand and can enhance interpretability. This can help to find potential problems with the model and to enhance its performance.
- The attention component is used to identify salient regions in the input, which can include both objects and non-object features. This approach is more flexible than systems that rely on object detection, since the model can attend to a wider range of features that may be relevant to the task at hand.
- The model discovers alignments that closely match human intuition.

Source: image_captioning.ipynb

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/text/image_captioning.ipynb#scrollTo=lPdb7I4h9Ulo

Description:

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention [1] is the model that we have adapted from. However, a 2-layer Transformer-decoder model is employed.

The notebook itself is very descriptive and very well explained so the figures are not inserted here again to avoid repetition. The initial cells in the notebook come with the setup required install the dependencies for the model to work(***Setup***).

The section **Data Handling** of the notebook downloads the *Flickr8k* dataset, extracts the data, and does the required preprocessing. By mapping each word to its index in the vocabulary of all conceivable terms, this portion "tokenizes" the text. To make each sequence 50 words long, padding is used. This amazing video greatly assisted us in comprehending this notebook step [4]. We use a pretrained image model on imagenet to **extract the features** from the *Flickr8k* dataset. To be compatible with keras, we prepare the dataset such that we have a (inputs, labels) pairs in the form of ((images, input_tokens), label_tokens) pairs with a 1:1 images to captions mapping. Additionally since we are not changing the image feature extractor, the image features are cached so that we spend lesser time on each epoch during training and validation.

Then we implement the model in 3 parts i.e. Input, Decoder, Output.

Input: consists of token embedding and positional encoding

Decoder: The **transform decoder** comprises of 3 layers the first of which uses self-attention to process the words being generated in the sequence. The second layer uses a cross-attention to attend to the input image. The third layer uses a feed forward network to process each output location independently.

Output: A multiclass-classification that would return a prediction about what the next word should be.

We then build and train this model for the *Flickr8k* dataset. A plot of training and validation loss against is provided for visual interpretation. We also have plots of attention maps which help us understand what part of the image is the model focusing on to generate one particular word in the output.

Code Changes:

There were no changes made to the original code to get it running. The model runs fine on its own. The only changes made were to test the model with a set of test images which is described in the next section following the original working code.

Image captioning with visual attention

View on TensorFlow.org Run in Google Colab View source on GitHub Download notebook

Copyright 2018 The TensorFlow Authors.

[ ] ↳ *6 cells hidden*

Setup

[ ] ↳ *6 cells hidden*

[Optional] Data handling

This section downloads a captions dataset and prepares it for training. It tokenizes the input text, and caches the results of running all the images through a pretrained feature-extractor model. It's not critical to understand everything in this section.

Toggle section

Choose a dataset

This tutorial is set up to give a choice of datasets. Either Flickr8k or a small slice of the Conceptual Captions dataset. These two are downloaded and converted from scratch, but it wouldn't be hard to convert the tutorial to use the caption datasets available in TensorFlow Datasets: Coco Captions and the full Conceptual Captions.

Flickr8k

```
def flickr8k(path='flickr8k'):
  path = pathlib.Path(path)

  if len(list(path.rglob('*'))) < 16197:
    tf.keras.utils.get_file(
        origin='https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip',
```

```
          cache_dir='.',
          cache_subdir=path,
          extract=True)
      tf.keras.utils.get_file(
          origin='https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_text.zip',
          cache_dir='.',
          cache_subdir=path,
          extract=True)

    captions = (path/"Flickr8k.token.txt").read_text().splitlines()
    captions = (line.split('\t') for line in captions)
    captions = ((fname.split('#')[0], caption) for (fname, caption) in captions)

    cap_dict = collections.defaultdict(list)
    for fname, cap in captions:
      cap_dict[fname].append(cap)

    train_files = (path/'Flickr_8k.trainImages.txt').read_text().splitlines()
    train_captions = [(str(path/'Flicker8k_Dataset'/fname), cap_dict[fname]) for fname in train_files]

    test_files = (path/'Flickr_8k.testImages.txt').read_text().splitlines()
    test_captions = [(str(path/'Flicker8k_Dataset'/fname), cap_dict[fname]) for fname in test_files]

    train_ds = tf.data.experimental.from_list(train_captions)
    test_ds = tf.data.experimental.from_list(test_captions)

    return train_ds, test_ds
```

Conceptual Captions

```
def conceptual_captions(*, data_dir="conceptual_captions", num_train, num_val):
  def iter_index(index_path):
    with open(index_path) as f:
      for line in f:
        caption, url = line.strip().split('\t')
        yield caption, url

  def download_image_urls(data_dir, urls):
    ex = concurrent.futures.ThreadPoolExecutor(max_workers=100)
    def save_image(url):
      hash = hashlib.sha1(url.encode())
      # Name the files after the hash of the URL.
      file_path = data_dir/f'{hash.hexdigest()}.jpeg'
      if file_path.exists():
        # Only download each file once.
        return file_path

      try:
        result = requests.get(url, timeout=5)
      except Exception:
        file_path = None
      else:
        file_path.write_bytes(result.content)
      return file_path

    result = []
    out_paths = ex.map(save_image, urls)
    for file_path in tqdm.tqdm(out_paths, total=len(urls)):
      result.append(file_path)

    return result

  def ds_from_index_file(index_path, data_dir, count):
    data_dir.mkdir(exist_ok=True)
    index = list(itertools.islice(iter_index(index_path), count))
    captions = [caption for caption, url in index]
    urls = [url for caption, url in index]

    paths = download_image_urls(data_dir, urls)

    new_captions = []
    new_paths = []
    for cap, path in zip(captions, paths):
      if path is None:
        # Download failed, so skip this pair.
```

```
        continue
      new_captions.append(cap)
      new_paths.append(path)

    new_paths = [str(p) for p in new_paths]

    ds = tf.data.Dataset.from_tensor_slices((new_paths, new_captions))
    ds = ds.map(lambda path,cap: (path, cap[tf.newaxis])) # 1 caption per image
    return ds

  data_dir = pathlib.Path(data_dir)
  train_index_path = tf.keras.utils.get_file(
    origin='https://storage.googleapis.com/gcc-data/Train/GCC-training.tsv',
    cache_subdir=data_dir,
    cache_dir='.')

  val_index_path = tf.keras.utils.get_file(
    origin='https://storage.googleapis.com/gcc-data/Validation/GCC-1.1.0-Validation.tsv',
    cache_subdir=data_dir,
    cache_dir='.')

  train_raw = ds_from_index_file(train_index_path, data_dir=data_dir/'train', count=num_train)
  test_raw = ds_from_index_file(val_index_path, data_dir=data_dir/'val', count=num_val)

  return train_raw, test_raw
```

Download the dataset

The Flickr8k is a good choice because it contains 5-captions per image, more data for a smaller download.

```
choose = 'flickr8k'

if choose == 'flickr8k':
  train_raw, test_raw = flickr8k()
else:
  train_raw, test_raw = conceptual_captions(num_train=10000, num_val=5000)
```

```
Downloading data from https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip
1115419746/1115419746 [==============================] - 85s 0us/step
Downloading data from https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_text.zip
2340801/2340801 [==============================] - 1s 0us/step
```

The loaders for both datasets above return `tf.data.Dataset`s containing `(image_path, captions)` pairs. The Flickr8k dataset contains 5 captions per image, while Conceptual Captions has 1:

```
train_raw.element_spec
```

```
(TensorSpec(shape=(), dtype=tf.string, name=None),
 TensorSpec(shape=(5,), dtype=tf.string, name=None))
```

```
for ex_path, ex_captions in train_raw.take(1):
  print(ex_path)
  print(ex_captions)
```

```
tf.Tensor(b'flickr8k/Flicker8k_Dataset/2513260012_03d33305cf.jpg', shape=(), dtype=string)
tf.Tensor(
[b'A black dog is running after a white dog in the snow .'
 b'Black dog chasing brown dog through snow'
 b'Two dogs chase each other across the snowy ground .'
 b'Two dogs play together in the snow .'
 b'Two dogs running through a low lying body of water .'], shape=(5,), dtype=string)
```

Image feature extractor

You will use an image model (pretrained on imagenet) to extract the features from each image. The model was trained as an image classifier, but setting `include_top=False` returns the model without the final classification layer, so you can use the last layer of feature-maps:

```
IMAGE_SHAPE=(224, 224, 3)
mobilenet = tf.keras.applications.MobileNetV3Small(
    input_shape=IMAGE_SHAPE,
    include_top=False,
```

```
    include_preprocessing=True)
mobilenet.trainable=False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v3/weights_mobilenet_v3_small_224_1.0_floa
4334752/4334752 [==============================] - 0s 0us/step
```

Here's a function to load an image and resize it for the model:

```
def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, IMAGE_SHAPE[:-1])
    return img
```

The model returns a feature map for each image in the input batch:

```
test_img_batch = load_image(ex_path)[tf.newaxis, :]

print(test_img_batch.shape)
print(mobilenet(test_img_batch).shape)
```

```
(1, 224, 224, 3)
(1, 7, 7, 576)
```

Setup the text tokenizer/vectorizer

You will transform the text captions into integer sequences using the TextVectorization layer, with the following steps:

- Use adapt to iterate over all captions, split the captions into words, and compute a vocabulary of the top words.
- Tokenize all captions by mapping each word to its index in the vocabulary. All output sequences will be padded to length 50.
- Create word-to-index and index-to-word mappings to display results.

```
def standardize(s):
  s = tf.strings.lower(s)
  s = tf.strings.regex_replace(s, f'[{re.escape(string.punctuation)}]', '')
  s = tf.strings.join(['[START]', s, '[END]'], separator=' ')
  return s


# Use the top 5000 words for a vocabulary.
vocabulary_size = 5000
tokenizer = tf.keras.layers.TextVectorization(
    max_tokens=vocabulary_size,
    standardize=standardize,
    ragged=True)
# Learn the vocabulary from the caption data.
```

```
tokenizer.adapt(train_raw.map(lambda fp,txt: txt).unbatch().batch(1024))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.8/dist-packages/tensorflow/python/autograph/pyct/static_analysis/liveness.py:83: Analyze
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are used, or at least in the same block. https://github.
```

```
tokenizer.get_vocabulary()[:10]
```

```
['', '[UNK]', 'a', '[START]', '[END]', 'in', 'the', 'on', 'is', 'and']
```

```
t = tokenizer([['a cat in a hat'], ['a robot dog']])
t
```

```
<tf.RaggedTensor [[3, 2, 655, 5, 2, 97, 4], [3, 2, 1937, 10, 4]]>
```

```
# Create mappings for words to indices and indices to words.
word_to_index = tf.keras.layers.StringLookup(
    mask_token="",
    vocabulary=tokenizer.get_vocabulary())
index_to_word = tf.keras.layers.StringLookup(
    mask_token="",
```

```
    vocabulary=tokenizer.get_vocabulary(),
    invert=True)


w = index_to_word(t)
w.to_list()
```

```
[[b'[START]', b'a', b'cat', b'in', b'a', b'hat', b'[END]'],
 [b'[START]', b'a', b'robot', b'dog', b'[END]']]
```

```
tf.strings.reduce_join(w, separator=' ', axis=-1).numpy()
```

```
array([b'[START] a cat in a hat [END]', b'[START] a robot dog [END]'],
      dtype=object)
```

Prepare the datasets

The `train_raw` and `test_raw` datasets contain 1:many `(image, captions)` pairs.

This function will replicate the image so there are 1:1 images to captions:

```
def match_shapes(images, captions):
  caption_shape = einops.parse_shape(captions, 'b c')
  captions = einops.rearrange(captions, 'b c -> (b c)')
  images = einops.repeat(
      images, 'b ... -> (b c) ...',
      c = caption_shape['c'])
  return images, captions
```

```
for ex_paths, ex_captions in train_raw.batch(32).take(1):
  break

print('image paths:', ex_paths.shape)
print('captions:', ex_captions.shape)
print()

ex_paths, ex_captions = match_shapes(images=ex_paths, captions=ex_captions)

print('image_paths:', ex_paths.shape)
print('captions:', ex_captions.shape)
```

```
image paths: (32,)
captions: (32, 5)

image_paths: (160,)
captions: (160,)
```

To be compatible with keras training the dataset should contain `(inputs, labels)` pairs. For text generation the tokens are both an input and the labels, shifted by one step. This function will convert an `(images, texts)` pair to an `((images, input_tokens), label_tokens)` pair:

```
def prepare_txt(imgs, txts):
  tokens = tokenizer(txts)

  input_tokens = tokens[..., :-1]
  label_tokens = tokens[..., 1:]
  return (imgs, input_tokens), label_tokens
```

This function adds operations to a dataset. The steps are:

1. Load the images (and ignore images that fail to load).
2. Replicate images to match the number of captions.
3. Shuffle and rebatch the `image, caption` pairs.
4. Tokenize the text, shift the tokens and add `label_tokens`.
5. Convert the text from a `RaggedTensor` representation to padded dense `Tensor` representation.

```
def prepare_dataset(ds, tokenizer, batch_size=32, shuffle_buffer=1000):
  # Load the images and make batches.
  ds = (ds
```

```
        .shuffle(10000)
        .map(lambda path, caption: (load_image(path), caption))
        .apply(tf.data.experimental.ignore_errors())
        .batch(batch_size))

  def to_tensor(inputs, labels):
    (images, in_tok), out_tok = inputs, labels
    return (images, in_tok.to_tensor()), out_tok.to_tensor()

  return (ds
          .map(match_shapes, tf.data.AUTOTUNE)
          .unbatch()
          .shuffle(shuffle_buffer)
          .batch(batch_size)
          .map(prepare_txt, tf.data.AUTOTUNE)
          .map(to_tensor, tf.data.AUTOTUNE)
          )
```

You could install the feature extractor in your model and train on the datasets like this:

```
train_ds = prepare_dataset(train_raw, tokenizer)
train_ds.element_spec
```

```
WARNING:tensorflow:From <ipython-input-26-03f5d7fa769a>:6: ignore_errors (from tensorflow.python.data.experimental.ops.error_ops) is de
Instructions for updating:
Use `tf.data.Dataset.ignore_errors` instead.
((TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, None), dtype=tf.int64, name=None)),
 TensorSpec(shape=(None, None), dtype=tf.int64, name=None))
```

```
test_ds = prepare_dataset(test_raw, tokenizer)
test_ds.element_spec
```

```
((TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, None), dtype=tf.int64, name=None)),
 TensorSpec(shape=(None, None), dtype=tf.int64, name=None))
```

[Optional] Cache the image features

Since the image feature extractor is not changing, and this tutorial is not using image augmentation, the image features can be cached. Same for the text tokenization. The time it takes to set up the cache is earned back on each epoch during training and validation. The code below defines two functions `save_dataset` and `load_dataset`:

```
def save_dataset(ds, save_path, image_model, tokenizer, shards=10, batch_size=32):
  # Load the images and make batches.
  ds = (ds
        .map(lambda path, caption: (load_image(path), caption))
        .apply(tf.data.experimental.ignore_errors())
        .batch(batch_size))

  # Run the feature extractor on each batch
  # Don't do this in a .map, because tf.data runs on the CPU.
  def gen():
    for (images, captions) in tqdm.tqdm(ds):
      feature_maps = image_model(images)

      feature_maps, captions = match_shapes(feature_maps, captions)
      yield feature_maps, captions

  # Wrap the generator in a new tf.data.Dataset.
  new_ds = tf.data.Dataset.from_generator(
      gen,
      output_signature=(
          tf.TensorSpec(shape=image_model.output_shape),
          tf.TensorSpec(shape=(None,), dtype=tf.string)))

  # Apply the tokenization
  new_ds = (new_ds
            .map(prepare_txt, tf.data.AUTOTUNE)
            .unbatch()
```

```
            .shuffle(1000))

  # Save the dataset into shard files.
  def shard_func(i, item):
    return i % shards
  new_ds.enumerate().save(save_path, shard_func=shard_func)

def load_dataset(save_path, batch_size=32, shuffle=1000, cycle_length=2):
  def custom_reader_func(datasets):
    datasets = datasets.shuffle(1000)
    return datasets.interleave(lambda x: x, cycle_length=cycle_length)

  ds = tf.data.Dataset.load(save_path, reader_func=custom_reader_func)

  def drop_index(i, x):
    return x

  ds = (ds
        .map(drop_index, tf.data.AUTOTUNE)
        .shuffle(shuffle)
        .padded_batch(batch_size)
        .prefetch(tf.data.AUTOTUNE))
  return ds

save_dataset(train_raw, 'train_cache', mobilenet, tokenizer)
save_dataset(test_raw, 'test_cache', mobilenet, tokenizer)
```

```
     188it [03:24,  1.09s/it]
     32it [00:29,  1.09it/s]
```

Data ready for training

After those preprocessing steps, here are the datasets:

[ ] ↳ *6 cells hidden*

A Transformer decoder model

[ ] ↳ *33 cells hidden*

Train

To train the model you'll need several additional components:

- The Loss and metrics
- The Optimizer
- Optional Callbacks

Losses and metrics

[ ] ↳ *2 cells hidden*

Callbacks

[ ] ↳ *6 cells hidden*

Train

Configure and execute the training.

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
          loss=masked_loss,
```

```
            metrics=[masked_acc])
```

For more frequent reporting, use the `Dataset.repeat()` method, and set the `steps_per_epoch` and `validation_steps` arguments to `Model.fit`.

With this setup on `Flickr8k` a full pass over the dataset is 900+ batches, but below the reporting-epochs are 100 steps.

```
history = model.fit(
    train_ds.repeat(),
    steps_per_epoch=100,
    validation_data=test_ds.repeat(),
    validation_steps=20,
    epochs=100,
    callbacks=callbacks)
```

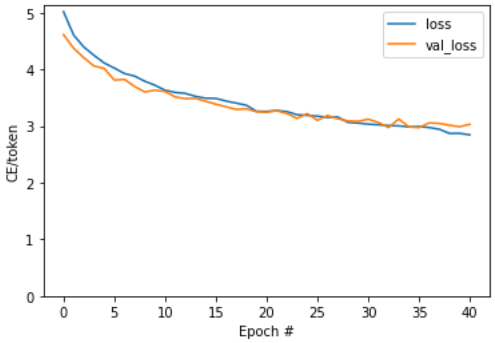
```
 100/100 [==============================] - ETA: 0s - loss: 3.9255 - masked_acc: 0.3271

 a man in a blue shirt is jumping in the water
 a man in a white shirt is standing in the water
 a boy in a boy is through the water

 100/100 [==============================] - 68s 679ms/step - loss: 3.9255 - masked_acc: 0.3271 - val_loss: 3.8223 - val_masked_acc: 0
 Epoch 8/100
 100/100 [==============================] - ETA: 0s - loss: 3.8800 - masked_acc: 0.3308

 a man in a blue shirt is jumping in the water
 a man in a blue shirt is standing in the water
 one young air in the water

 100/100 [==============================] - 68s 681ms/step - loss: 3.8800 - masked_acc: 0.3308 - val_loss: 3.6982 - val_masked_acc: 0
 Epoch 9/100
 100/100 [==============================] - ETA: 0s - loss: 3.7900 - masked_acc: 0.3391

 a man in a blue shirt is jumping in the water
 a child in a blue shirt is running in a pool
 the playground it in a green as the swimming wave against another

 100/100 [==============================] - 65s 654ms/step - loss: 3.7900 - masked_acc: 0.3391 - val_loss: 3.6004 - val_masked_acc: 0
 Epoch 10/100
 100/100 [==============================] - ETA: 0s - loss: 3.7189 - masked_acc: 0.3400

 a man in a blue shirt is jumping in the water
 the man is at the water
 a woman there is backpack during a snowy wave

 100/100 [==============================] - 68s 677ms/step - loss: 3.7189 - masked_acc: 0.3400 - val_loss: 3.6305 - val_masked_acc: 0
 Epoch 11/100
 100/100 [==============================] - ETA: 0s - loss: 3.6307 - masked_acc: 0.3509

 a man in a blue shirt is jumping into the water
 a man is riding a yellow and a pool
 a man with a hat on a leash ball

 100/100 [==============================] - 66s 657ms/step - loss: 3.6307 - masked_acc: 0.3509 - val_loss: 3.6083 - val_masked_acc: 0
 Epoch 12/100
 100/100 [==============================] - ETA: 0s - loss: 3.5917 - masked_acc: 0.3511

 a man in a blue shirt is jumping over a pool
 a boy in a white shirt is riding a pool
 a surfer dressed on the girls walk on the pool

 100/100 [==============================] - 65s 654ms/step - loss: 3.5917 - masked_acc: 0.3511 - val_loss: 3.5117 - val_masked_acc: 0
 Epoch 13/100
 100/100 [==============================] - ETA: 0s - loss: 3.5746 - masked_acc: 0.3516

 a man in a blue shirt is jumping into the water
 a man in a blue is jumping over a wave
 a man surrounding her wheelchair in its neighborhood

 100/100 [==============================] - 65s 648ms/step - loss: 3.5746 - masked_acc: 0.3516 - val_loss: 3.4825 - val_masked_acc: 0
 Epoch 14/100
 100/100 [==============================] - ETA: 0s - loss: 3.5226 - masked_acc: 0.3548
```

Plot the loss and accuracy over the training run:

```
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.ylim([0, max(plt.ylim())])
plt.xlabel('Epoch #')
```

```
plt.ylabel('CE/token')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa4cf2deb50>
```




```
plt.plot(history.history['masked_acc'], label='accuracy')
plt.plot(history.history['val_masked_acc'], label='val_accuracy')
plt.ylim([0, max(plt.ylim())])
plt.xlabel('Epoch #')
plt.ylabel('CE/token')
plt.legend()
```

Attention plots

Now, using the trained model, run that `simple_gen` method on the image:

```
result = model.simple_gen(image, temperature=0.0)
result
```

```
'a man in a red shirt is surfing'
```

Split the output back into tokens:

```
str_tokens = result.split()
str_tokens.append('[END]')
```

The `DecoderLayers` each cache the attention scores for their `CrossAttention` layer. The shape of each attention map is `(batch=1, heads, sequence, image)`:

```
attn_maps = [layer.last_attention_scores for layer in model.decoder_layers]
[map.shape for map in attn_maps]
```

```
[TensorShape([1, 2, 9, 49]), TensorShape([1, 2, 9, 49])]
```

So stack the maps along the `batch` axis, then average over the `(batch, heads)` axes, while splitting the `image` axis back into `height, width`:

```
attention_maps = tf.concat(attn_maps, axis=0)
attention_maps = einops.reduce(
    attention_maps,
    'batch heads sequence (height width) -> sequence height width',
    height=7, width=7,
    reduction='mean')
```

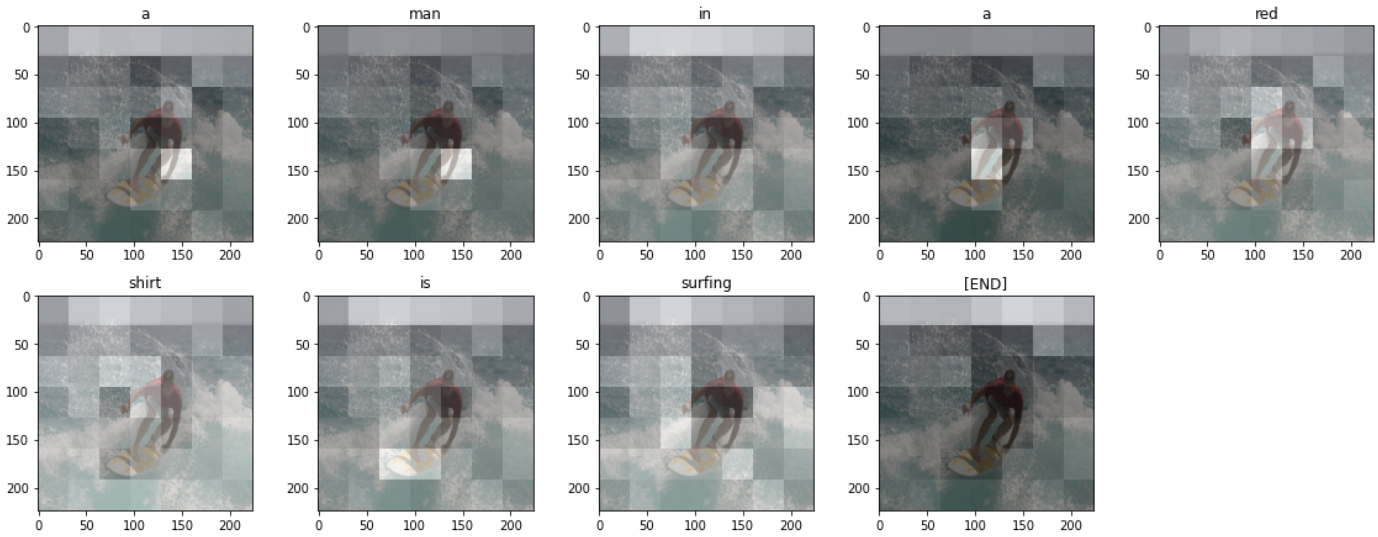
Now you have a single attention map, for each sequence prediction. The values in each map should sum to `1.`

```
einops.reduce(attention_maps, 'sequence height width -> sequence', reduction='sum')
```

```
<tf.Tensor: shape=(9,), dtype=float32, numpy=
array([1.       , 1.       , 1.       , 1.       , 1.       , 1.       ,
       1.       , 1.       , 1.0000001], dtype=float32)>
```

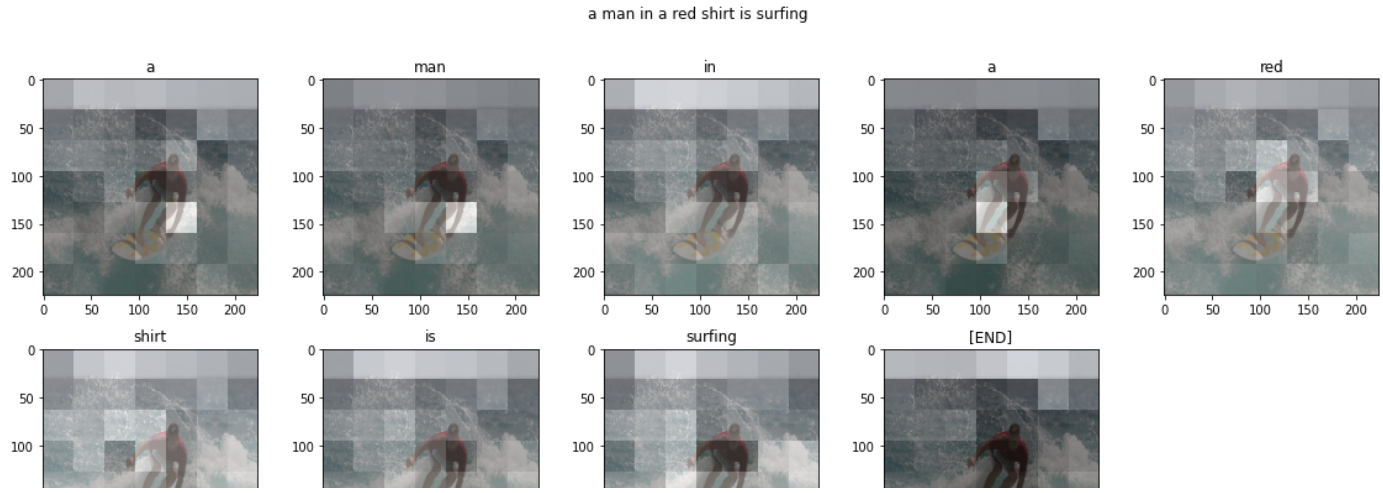
So here is where the model was focusing attention while generating each token of the output:

```
def plot_attention_maps(image, str_tokens, attention_map):
    fig = plt.figure(figsize=(16, 9))

    len_result = len(str_tokens)

    titles = []
    for i in range(len_result):
      map = attention_map[i]
      grid_size = max(int(np.ceil(len_result/2)), 2)
      ax = fig.add_subplot(3, grid_size, i+1)
      titles.append(ax.set_title(str_tokens[i]))
      img = ax.imshow(image)
      ax.imshow(map, cmap='gray', alpha=0.6, extent=img.get_extent(),
                clim=[0.0, np.max(map)])

    plt.tight_layout()


plot_attention_maps(image/255, str_tokens, attention_maps)
```



Now put that together into a more usable function:

```
@Captioner.add_method
def run_and_show_attention(self, image, temperature=0.0):
  result_txt = self.simple_gen(image, temperature)
  str_tokens = result_txt.split()
  str_tokens.append('[END]')

  attention_maps = [layer.last_attention_scores for layer in self.decoder_layers]
  attention_maps = tf.concat(attention_maps, axis=0)
  attention_maps = einops.reduce(
      attention_maps,
      'batch heads sequence (height width) -> sequence height width',
      height=7, width=7,
      reduction='mean')

  plot_attention_maps(image/255, str_tokens, attention_maps)
  t = plt.suptitle(result_txt)
  t.set_y(1.05)


run_and_show_attention(model, image)
```



Try it on your own images

For fun, below you're provided a method you can use to caption your own images with the model you've just trained. Keep in mind, it was trained on a relatively small amount of data, and your images may be different from the training data (so be prepared for strange results!)

[ ] ↳ *2 cells hidden*

Validating a claim

We have taken the following claim from the paper:(as stated in paper)

5.4. Qualitative Analysis: Learning to attend

By visualizing the attention component learned by the model, we are able to add an extra layer of interpretability to the output of the model (see Fig. 1). Other systems that have done this rely on object detection systems to produce candidate alignment targets (Karpathy & Li, 2014). Our approach is much more flexible, since the model can attend to "non object" salient regions.

The 19-layer OxfordNet uses stacks of 3x3 filters meaning the only time the feature maps decrease in size are due to the max pooling layers. The input image is resized so that the shortest side is 256 dimensional with preserved aspect ratio. The input to the convolutional network is the center cropped 224x224 image. Consequently, with 4 max pooling layers we get an output dimension of the top convolutional layer of 14x14. Thus in order to visualize the attention weights for the soft model, we simply upsample the weights by a factor of $2^4 = 16$ and apply a Gaussian filter. We note that the receptive fields of each of the 14x14 units are highly overlapping.

As we can see in Figure 2 and 3, the model learns alignments that correspond very strongly with human intuition. Especially in the examples of mistakes, we see that it is possible to exploit such visualizations to get an intuition as to why those mistakes were made. We provide a more extensive list of visualizations in Appendix A for the reader.

Fig 8: Ref [1], Section 5.4

Claim 1:

The author makes a claim that the attention component can attend or focus on "non object" salient regions. The phrasing of the claim from the author hints that other models which use object detection might not be able to pick this detail.

Claim 2:

The author admits the shortcomings with the model but further states that the mistakes made by the model could be exploited by visualization as to get an intuition into why the model might be predicting what it is predicting.

Example:

Say, a dog is holding a bone. Therefore, when we use the word "dog," we must only pay attention to the dog in the illustration, and when we say the word "holding ," we must pay attention to the dog's hand. Similar to how we must keep our attention just on the bone in the image when we say "bone," This indicates that the words "dog," "holding," and "bone" are made up of various image pixels.

The model must be able to focus its attention on important features in the image in order to predict the correct caption.

To test the validity of this claim, we have chosen a set of images for where the attention model improves explainability. Furthermore we will also make an attempt to test the validity of claim 2 and see if it holds true.

Note: All the code used here is from the colab and no edits were made to the original code. The only change were the test images for which the URL is provided from the website where we found the images!

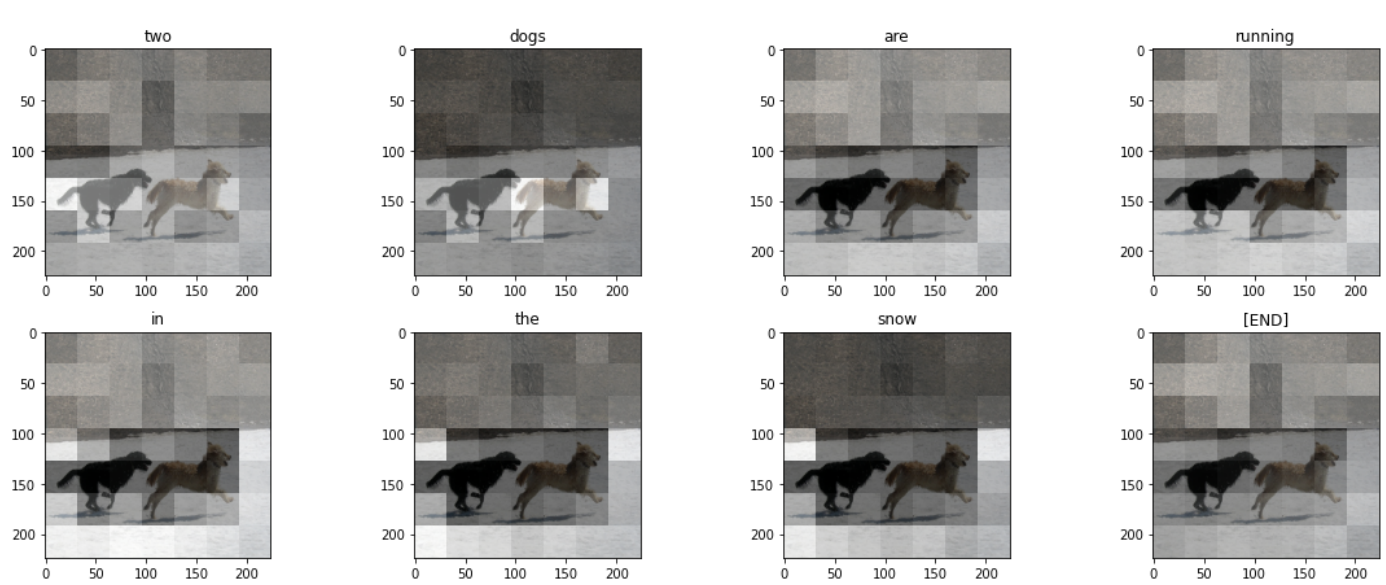
Claim 1:

Results: From the images that were tested for this claim **the model did fairly good at identifying "non salient features" as demonstrated using following examples.**

There were some issues encountered as well such as the model missing to identify the correct object but what was impressive was that the model was able to identify the "salient regions" in the pictures.

```
image_url = 'https://photos.smugmug.com/My-First-Gallery/n-BJWWM3/i-4N9zr47/0/2c7d76c7/S/i-4N9zr47-S.jpg'
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)

run_and_show_attention(model, image)
```

```
Downloading data from https://photos.smugmug.com/My-First-Gallery/n-BJWWM3/i-4N9zr47/0/2c7d76c7/S/i-4N9zr47-S.jpg
51376/51376 [==============================] - 0s 1us/step
```

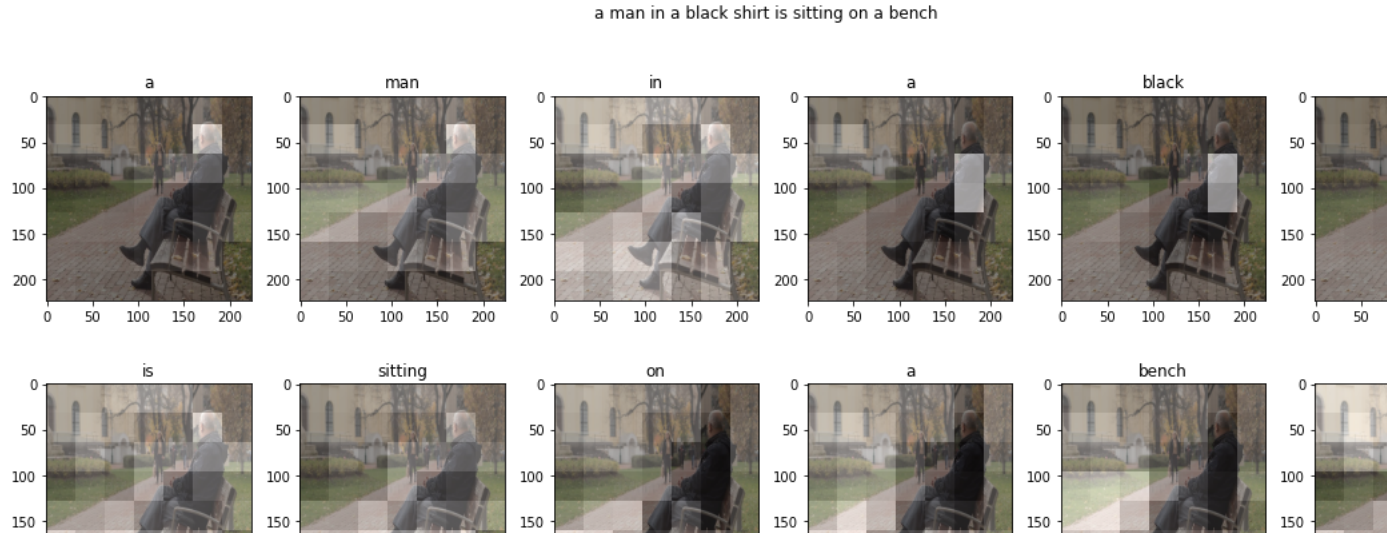


Note: Good

As can be seen in the above image, the model is able to predict "two dogs are running in the snow" The model was able to focus its attention not only on the dogs which are the primary or salient features but also on the background which happens to be the snow!

```
image_url = 'https://thumbs.dreamstime.com/b/old-man-sitting-bench-park-old-man-sitting-bench-park-autumns-day-163880744.jpg'
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)

run_and_show_attention(model, image)
```

```
Downloading data from https://thumbs.dreamstime.com/b/old-man-sitting-bench-park-old-man-sitting-bench-park-autumns-day-163880744.jpg
78748/78748 [==============================] - 0s 0us/step
```

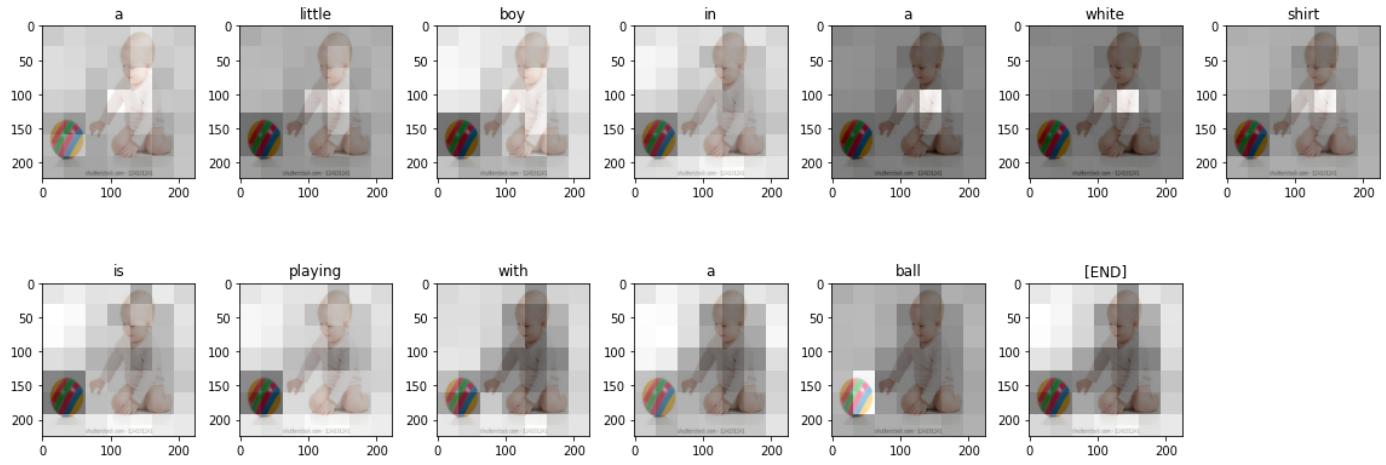


Note: Good

As can be seen in the above image, the model is able to predict "a man in a black shirt is sitting on a bench" The model was able to focus its attention not only on the man sitting on a bench but was also able to correctly identify the color of the shirt of the man which is quite impressive!

```
image_url = 'https://www.shutterstock.com/image-photo/baby-girl-playing-ball-260nw-124131241.jpg'
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)

run_and_show_attention(model, image)
```

```
Downloading data from https://www.shutterstock.com/image-photo/baby-girl-playing-ball-260nw-124131241.jpg
13876/13876 [==============================] - 0s 1us/step
```



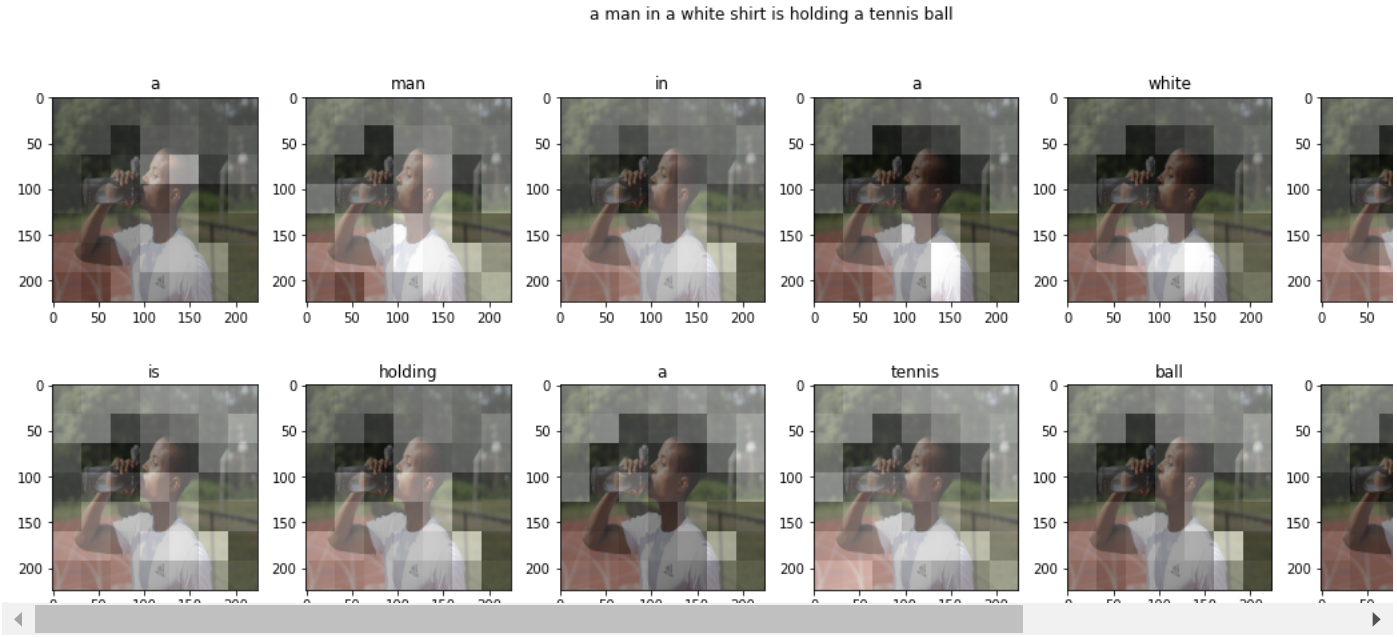
Note: Good

As can be seen in the above image, the model is able to predict "a little boy in a white shirt is playing with a ball" The model was able to focus its attention on the "little boy" and "the ball"

```
image_url = 'https://images.unsplash.com/photo-1600679472233-eabc13b79f07?ixlib=rb-4.0.3&ixid=MnwxMjA3fDB8MHxzZWFyY2h8OHx8ZHJpbmtpbmclMjB3YXR
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)
```

```
run_and_show_attention(model, image)
```

```
Downloading data from https://images.unsplash.com/photo-1600679472233-eabc13b79f07?ixlib=rb-4.0.3&ixid=MnwxMjA3fDB8MHxzZWFyY2h8OHx8ZHJp
72976/72976 [==============================] - 0s 0us/step
```



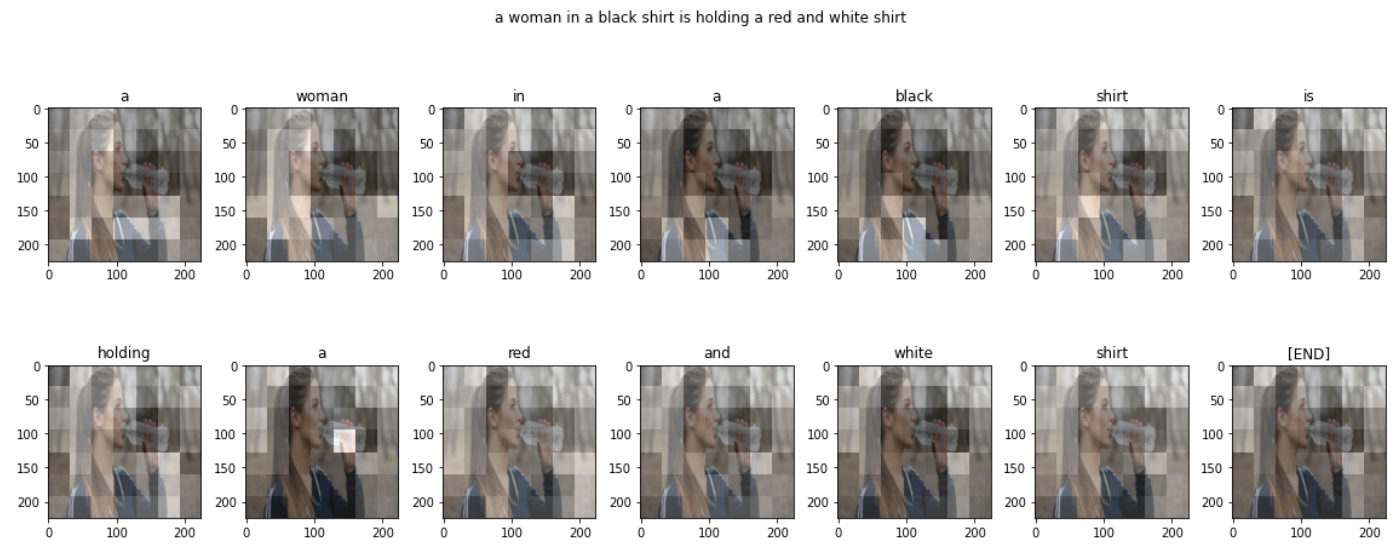
Note: Good???

Prediction: "a man in a white shirt is holding tennis ball"

This is an interesting example as the model was able to focus its attention on the man holding "something". It happens that the object the man is holding is a water bottle but not a tennis ball.

This was experimentally noticed that with another set of images that the model was unable to detect water bottles in general with various wrong predictions being made. This could be due to the traing dataset not having any/enough pictures with a water bottle. This could lead to the model not having "water bottle" in its vocabulary! Interestingly the model was able to focus on important area in the picture although it made a wrong prediction!

```
image_url = 'https://wp.en.aleteia.org/wp-content/uploads/sites/2/2017/09/web3-woman-hiking-water-bottle-water-drinking-forest-shutterstock.j
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)

run_and_show_attention(model, image)
```



Note: Aah that must be it!

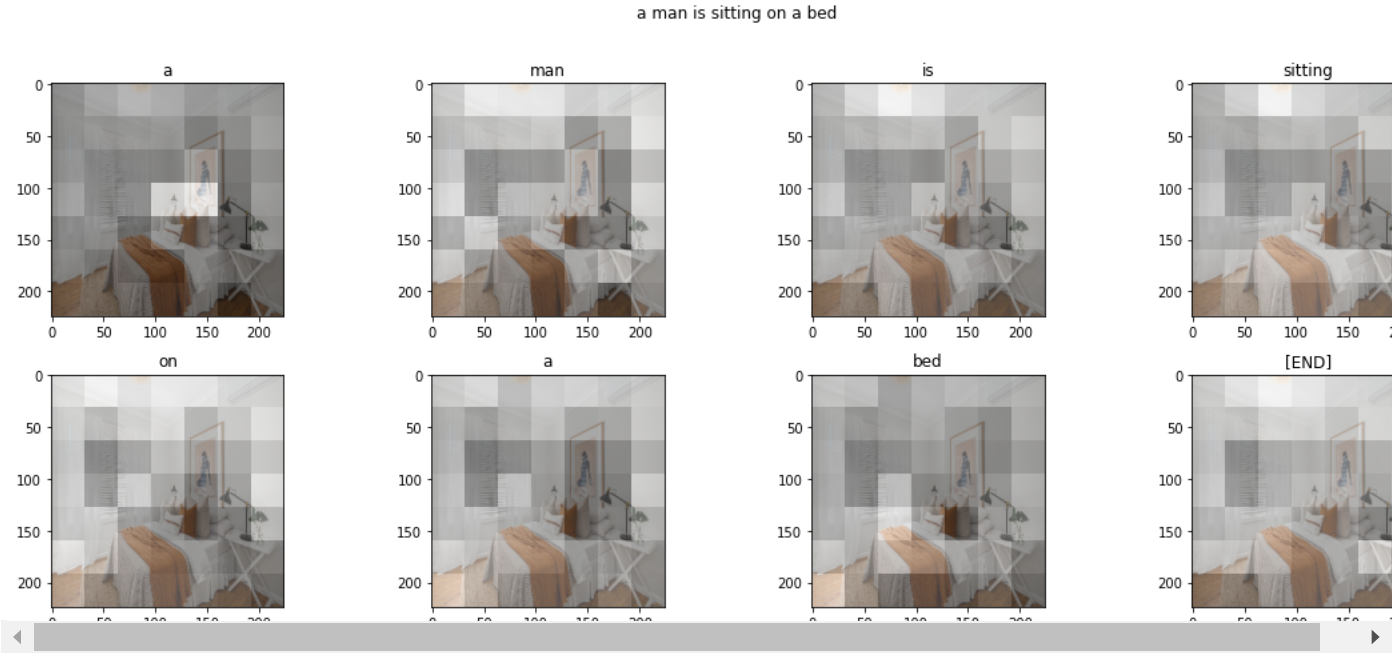
The model is able to focus attention on the water bottle but unable to predict it correctly.

Claim 2

Results: There were very few images where the wrong prediction could have been explained well with the attention plots. i.e. most wrong predictions were unexplained by the attention plots as shown in following sections.

The experimental results did not completely agree with author's claims. This might possibly be due to the model being undertrained as some attention plots which do no correspond to any objects give weird lables for these objects.

```
image_url = 'https://tensorflow.org/images/bedroom_hrnet_tutorial.jpg'
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)

run_and_show_attention(model, image)
```

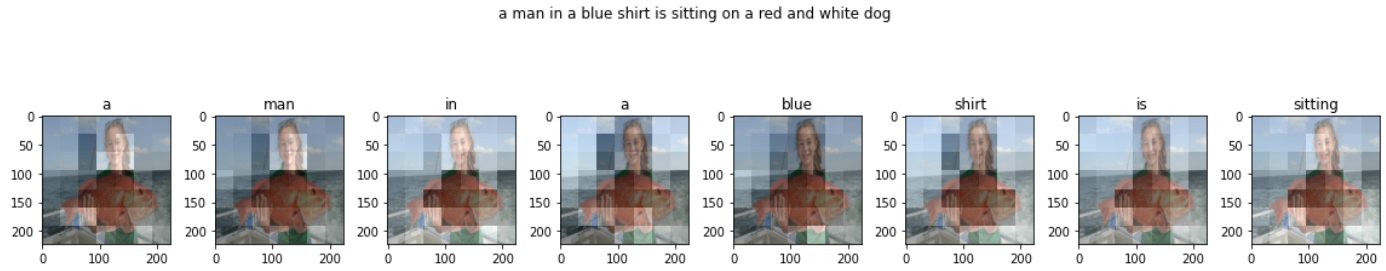


Note: Bad but Good

Prediction: "a man is sitting on a bed"

This is an interesting example as the wrong prediction made by the model can be explained with the second attention plot. This object is causing the model to predict "man" but happens to be a painting.

```
image_url = 'https://www.distractioncharters.com/wp-content/uploads/2019/02/best-charter-boat-orange-beach-gulf-shores-615x411.jpg'
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)

run_and_show_attention(model, image)
```



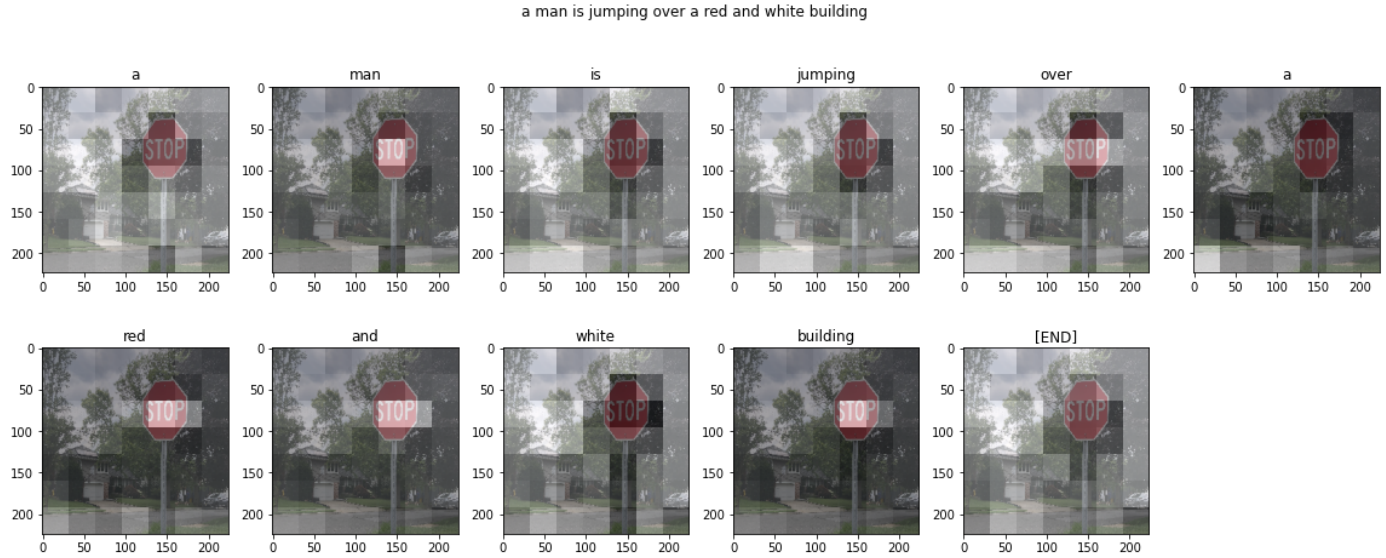

```
# This is formatted as code
```

Note: Bad but somewhat good

Prediction: "a man in a blue shirt is sitting on a red and a white dog

This wrong prediction can somewhat be explained with the help of the attention plots! The corresponding attention plots do not agree with the label "dog" but the rest of the labels are expected.

```
image_url = 'https://media.npr.org/assets/img/2021/09/22/img_4973-1fff64cb2be877fb9678bd9a52bee4ea1e5f03ca.jpg'
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)

run_and_show_attention(model, image)
```



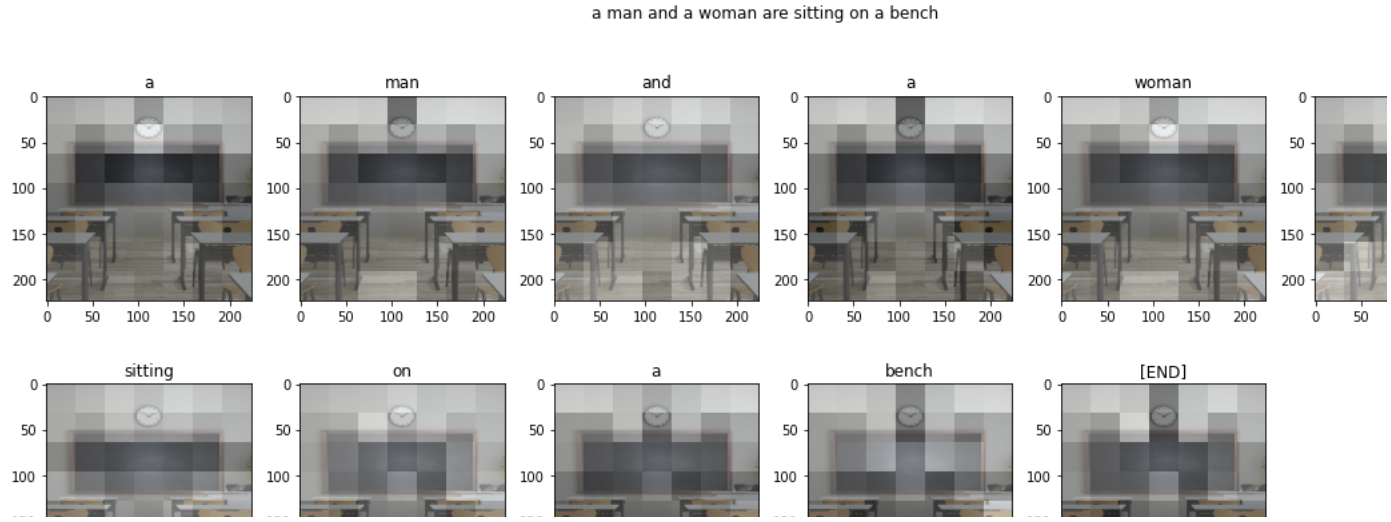
Note: Bad

Prediction: "a man is jumping over a red and a white building"

This wrong prediction cannot be explained with most of the attention plots! The corresponding attention plots do not agree with the labels "man" and "jumping" are these objects are absent in the image.

```
image_url = 'https://images.squarespace-cdn.com/content/v1/5c2d190d5ffd20fcfe3de667/1596565216519-RC73P1L6TP8G0GQMKEVI/BacktoSchool.Teacing.T
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)

run_and_show_attention(model, image)
```

```
Downloading data from https://images.squarespace-cdn.com/content/v1/5c2d190d5ffd20fcfe3de667/1596565216519-RC73P1L6TP8G0GQMKEVI/BacktoS
136285/136285 [==============================] - 0s 0us/step
```



Note: Bad! Cannot be Explained!

Prediction: "a man and a woman are sitting on a bench"

This wrong prediction cannot be explained with the attention plots! There is no man, woman or sitting in this image.

```
image_url = 'https://thumbs.dreamstime.com/b/supermarket-cart-loaded-cardboard-boxes-sales-goods-concept-trade-commerce-online-shopping-high-
image_path = tf.keras.utils.get_file(origin=image_url)
image = load_image(image_path)

run_and_show_attention(model, image)
```

```
Downloading data from https://thumbs.dreamstime.com/b/supermarket-cart-loaded-cardboard-boxes-sales-goods-concept-trade-commerce-online
51131/51131 [==============================] - 0s 0us/step
```

Note: Bad! Cannot be Explained!

Prediction: "a man in a red shirt is sitting on a red and white dog"

There is no "man" "red" "shirt" "sitting" "white" "dog" in this image!!!

References

[1] [2015 ICML] [Show, Attend and Tell] Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

[2] https://www.youtube.com/watch?v=y1S3Ri7myMg

[3] https://sh-tsang.medium.com/review-show-attend-and-tell-neural-image-caption-generation-f56f9b75bf89

[4] https://www.youtube.com/watch?v=cldGihGgtJs

[5] https://towardsdatascience.com/image-captions-with-attention-in-tensorflow-step-by-step-927dad3569fa#:~:text=Image%20Caption%20Model%20with%20Attention&text=It%20consists%20of%20a%20Linear,going%20through%20an%20Embedding%20layer.

[6] https://medium.com/swlh/image-captioning-using-attention-mechanism-f3d7fc96eb0e#:~:text=Problem%20with%20'Classic'%20Image%20Captioning,of%20the%20entire%20input%20image.