IMAGE CAPTIONING

Using Attention Based Transformer

Shiv Kiran Bagathi - 200050019
Chilukuri Mani Praneeth - 200050028
Dendukuri Sandeep Verma - 200050032

27-11-2022

- Problem Statement and Related Works
- Datasets
- Workflow, Architecture, Technique
- Results and Analysis
- Demo

- Problem Statement and Related Works
- Datasets
- Workflow, Architecture, Technique
- Results and Analysis
- Demo

Problem Statement

- The problem introduces a captioning task, which requires a computer vision system to describe salient regions in images in natural language.
- Given an image like the one shown below, our goal is to build an attention based model which generates a caption such as “a surfer riding on a wave”

Related Work

- The model architecture we used is inspired by Show, Attend and Tell: Neural Image Caption Generation with Visual Attention - https://arxiv.org/pdf/1502.03044.pdf , but has been updated to use a 2-layer Transformer-decoder

- Our implementation of self and cross attention for transformers is adapted from Attention is All You Need - https://arxiv.org/pdf/1706.03762.pdf

Relevance to NLP

- Image caption generation deals with image understanding and a language description for that image
- Generating well-formed sentences required both semantic and syntactic understanding of the language
- Being able to describe the content of an image using accurately formed sentences is a very challenging task
- The biggest challenge is most definitely being able to create a description that must capture not only the objects contained in an image, but also express how these objects relate to each other

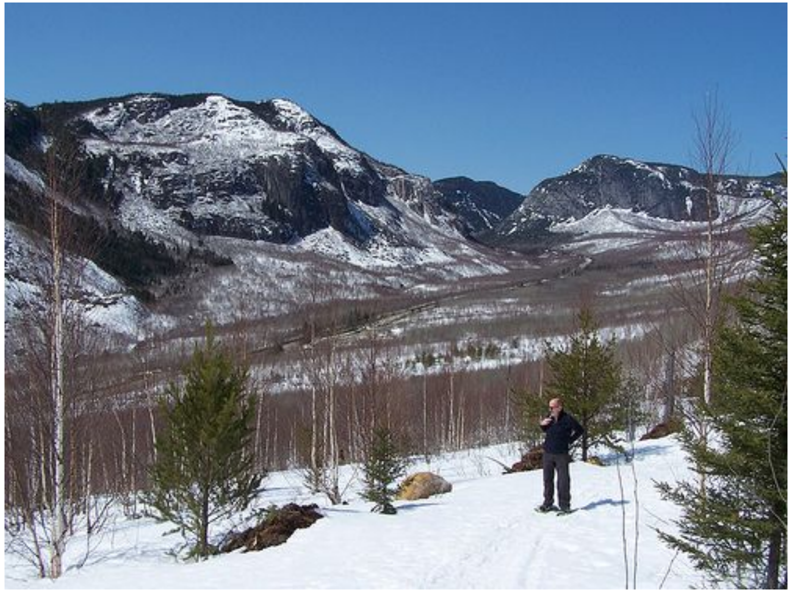
Datasets Used

- Working Training and Testing Datasets
  - Flickr8k
  - A slice of conceptual captions dataset
- Final Training Dataset : The final model is trained on Flickr8k dataset (consists of 8000 images that are each paired with 5 different captions which provide clear descriptions of the salient entities and events)
- We avoided using MS Coco and Flickr30k datasets due to higher training time costs

Preparing Data

- A function prepare_dataset is defined which does the following:
- Load the images
- Images that fail to load are ignored
- Shuffle and rebatch the image, caption pairs
- Tokenize the text, shift the tokens and add label_tokens
- Convert the text from a Ragged Tensor representation to padded dense Tensor representation

An example image-captions pair from Flickr8k



<start> A man is standing on snow with trees and mountains all around him . <end>
<start> A man standing in snow with a mountain in the background . <end>
<start> A man standing near a mountain range . <end>
<start> A man stands on a snowy hill next to a mountain . <end>
<start> A man stands on snow and looks out at mountains and forests . <end>

- Problem Statement and Related Works
- Datasets
- Workflow, Architecture, Technique
- Results and Analysis
- Demo

Workflow

1. Gathering Data
   a. Flickr8k photo caption dataset is downloaded from the github of jbrownlee
   b. Flickr30k
2. Data pre-processing
   a. Explained in the Preparing Data slide
3. Researching the model that will be best for the type of data
   a. The model implemented in Show, Attend and Tell is LSTM. So, to improve on it, we implemented a 2-layer Transformer-decoder model
4. Training and testing the model
   a. The model is trained and tested on the Flickr8k dataset during working phase (owing to the training time cost of Flickr30k)
   b. The final model is trained on Flickr30k
5. Evaluation
   a. BleU scores with 1-gram, 2-gram, 3-gram and 4-gram are used as the metrics for evaluation

Architecture

- The model is Vector-Sequence model
- The model used its decoder as a 2-layer transformer decoder
- Features are extracted from image, and passed to the cross-attention layers of the Transformer-decoder
- The transformer decoder is mainly built from attention layers
- It uses self-attention to process the sequence being generated, and it uses cross attention to attend to the image
- By inspecting the attention weights of cross attention layers, we can see what parts of the image the model is looking at as it generates words

Architecture - Why Transformers

- Transformers are deep neural networks that replace CNNs and RNNs with self-attention
- Self-attention allows Transformers to easily transmit information across the input sequences
- We chose Transformers because they excel at modeling sequential data, such as natural language
- Unlike RNNs, transformers are parallelizable. This makes them faster
- Attention allows each location to have access to the entire input at each layer hence enabling the Transformer to capture distant or long-range contexts and dependencies

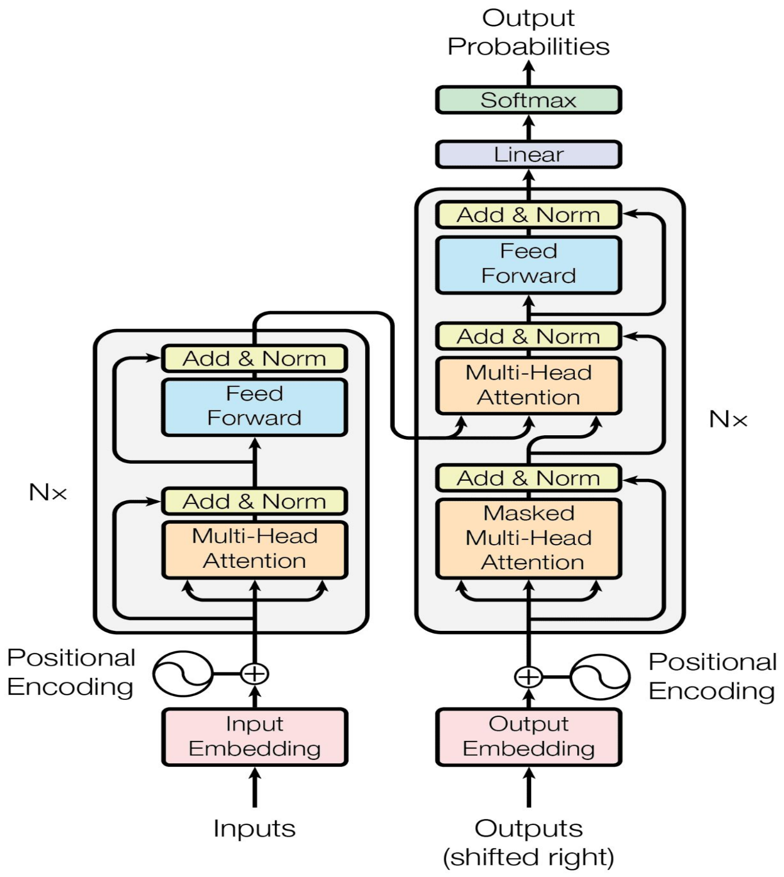
Overview of Transformers

- Recurrent Neural Networks are slow to train and Long sequences lead to vanishing and exploding gradients
- LSTM was introduced in 1991. Skips a lot of processing. The repeating module in an LSTM contains four interacting layers.
- Both in LSTM and RNN’s need to have data in sequential flow. This doesn’t utilize parallel nature of current GPU’s.
- Transformers was introduced in 2017. When input is sent in parallel.
- For transformer there is no sequential nature of input embeddings
- Self attention Blocks of encoder and decoder are passed to another attention layer.
- In this 2nd attention layer is where model is encapsulated.

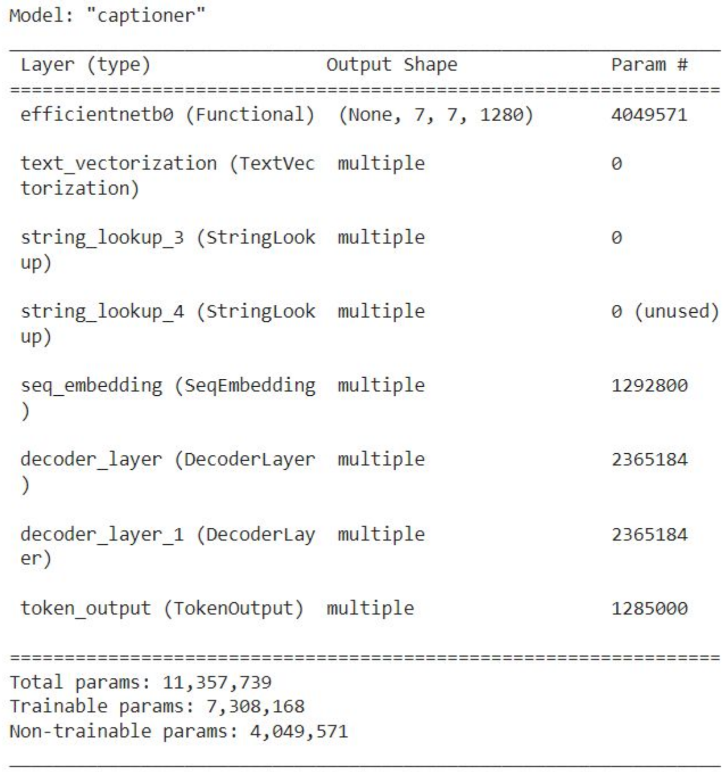
Overview of Transformers

- Positional Vector is added to get context of the the Vector.
- Attention block for encoder calculates attention for each part of the image and then we take weighted average of them.
- In decoder block first Attention block is Masked Multi-Head Attention Block. Because we consider the attention until the word and mask it.
- Then the next attention block generates attention between the input and output attention blocks.
- After each layer batch normalization is applied which smoothens out loss surface.

Architecture Visualised



Model Architecture

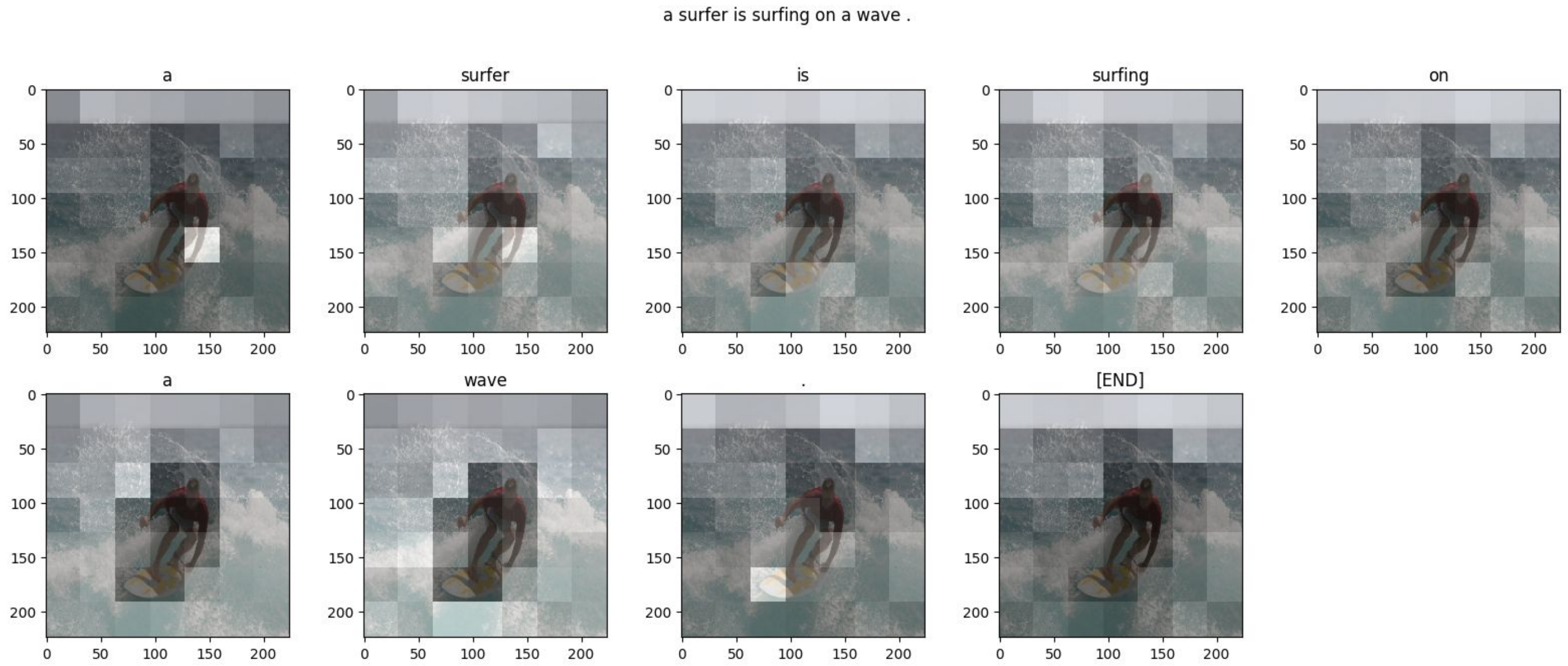


```
Model: "captioner"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb0 (Functional)  (None, 7, 7, 1280)       4049571

 text_vectorization (TextVec  multiple                 0
 torization)

 string_lookup_3 (StringLook  multiple                 0
 up)

 string_lookup_4 (StringLook  multiple                 0 (unused)
 up)

 seq_embedding (SeqEmbedding  multiple                 1292800
 )

 decoder_layer (DecoderLayer  multiple                 2365184
 )

 decoder_layer_1 (DecoderLay  multiple                 2365184
 er)

 token_output (TokenOutput)  multiple                  1285000

=================================================================
Total params: 11,357,739
Trainable params: 7,308,168
Non-trainable params: 4,049,571
_________________________________________________________________
```

Attention Mechanism

- The self-attention mechanism consists of a Single-Head Attention and Multi-Head Attention layer
- To further improve the self-attention mechanism the authors of the paper Attention Is All You Need proposed the implementation of multi-head attention.
- The functionality of a multi-head attention layer is to concatenate the attention weights of *n* single-head attention layers and then apply a non-linear transformation with a Dense layer
- Transformer decoding starts with full input sequence, but empty decoding sequence
- Cross-attention introduces information from the input sequence to the layers of the decoder, such that it can predict the next output sequence token
- The decoder then adds the token to the output sequence, and repeats this autoregressive process until the EOS token is generated

Cross Attention Visualised



Technique - Image Feature Extractor

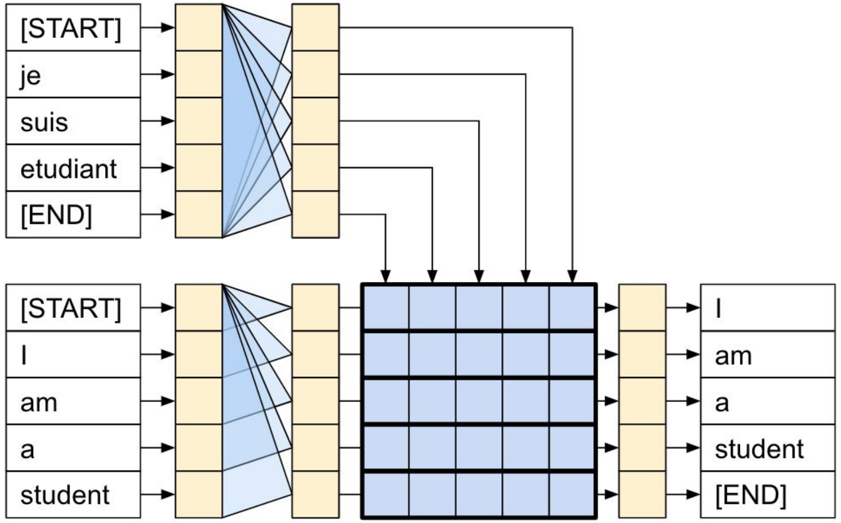
- We used an image model (pretrained on imagenet) to extract the features from each image
- The model was trained as an image classifier, but setting include_top=False returns the model without final classification layer
- Efficient Net model is chosen after proper experimentation
- The model returns a feature map for each image in the input batch
- As the image files are huge for training we are storing the features of extracted images as datacache to reduce load while training in every epoch

Technique - Text Vectorizer

- We transformed the text captions into integer sequences using the TextVectorization layer
- TextVectorization adapt was used to iterate over all captions, split the captions into words, and compute a vocabulary of the top words
- The captions are standardised i.e., lowercasing, removing punctuation, adding start and end tag
- The vocabulary is limited to the top 5000 words
- Each caption is then tokenized by mapping each word to its index in the vocabulary
- All output sequences are padded to length 50
- A word-to-index and index-to-word mapping is also created to display results

Technique - Transformer Decoder Model

- This model assumes that the pretrained image encoder is sufficient, and just focuses on building the text decoder
- We used a 2-layer transformer decoder



Technique - Transformer Decoder Model

- This model assumes that the pretrained image encoder is sufficient, and just focuses on building the text decoder
- We used a 2-layer transformer decoder
- The model is implemented in three main parts:
  1. Input - The token embedding and positional encoding (SeqEmbedding)
  2. Decoder - A stack of transformer decoder layers (DecoderLayer) where each contains:
     a. A casual self-attention layer (CasualSelfAttention), where each output location can attend to the output so far
     b. A cross attention layer (CrossAttention) where each output location can attend to the input image
     c. A feed forward network (FeedForward) layer which further processes each output location independently
  3. Output - A multiclass-classification over the output vocabulary

Technique - SeqEmbedding Layer

- Transformers' attention layers are invariant to the order of the sequence
- So in addition to a simple vector embedding for each token ID, the embedding layer will also include an embedding for each position in the sequence
- The SeqEmbedding layer -
    - looks up the embedding vector for each token
    - looks up an embedding vector for each sequence location
    - adds the two together
    - uses mask_zero=True to initialize the keras-masks for the model
- This implementation learns the position embeddings instead of using fixed embeddings
- This introduces the drawback of not being able to generalize to longer sequences, also known as long dependency issues

Technique - Decoder Layer

- This takes in the input and passes it to a CasualSelfAttention Layer
- The output is then passed as query into the CrossAttention Layer
- The original input is passed as value into this CrossAttention Layer
- The output from this CrossAttention Layer is then fed to a FeedForward Layer

Choosing appropriate dataset

- Flickr30k, and MS Coco datasets aren't tried as they incur huge training time cost
- Even using entire conceptual captions dataset will incur high training cost as it contains 3 million images
- We are left with two choices
  1. Flickr8k
  2. A slice of conceptual captions

Dataset	Trn_Loss	Trn_Acc	Val_Loss	Val_Acc	BleU Score
Flickr8k	2.5968	44.63	2.8575	41.07	0.4494
Conceptual	3.3365	38.27	4.8218	22.73	0.2974

Choosing the better image model

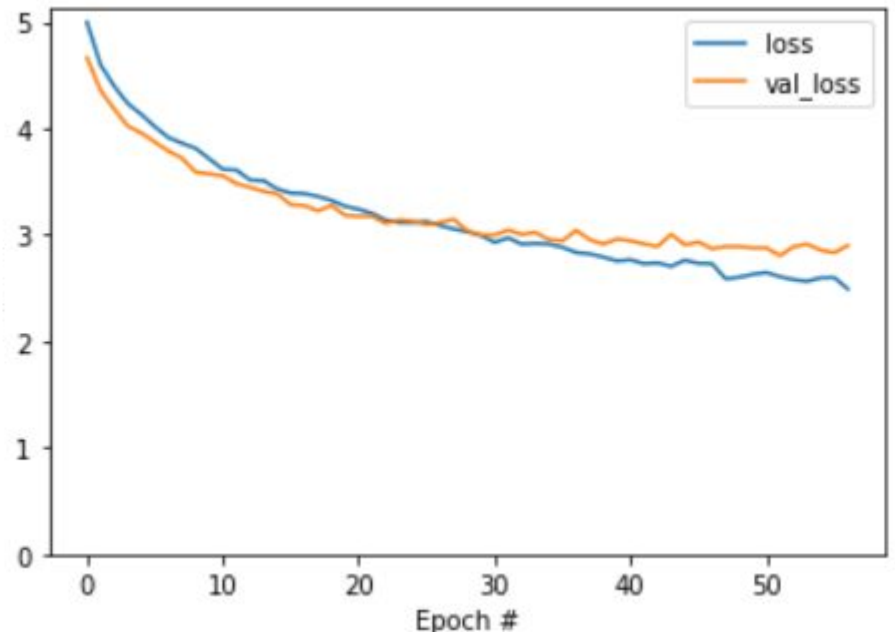
- We need to extract image features from images using transfer learning
- The following are the image models we experimented with:
  1. MobileNetV3Small
  2. Xception
  3. VGG16
  4. EfficientNet
- Each of these image models are tested and the results are presented in the following slides
- Clearly, EfficientNet worked better and hence we proceeded with that

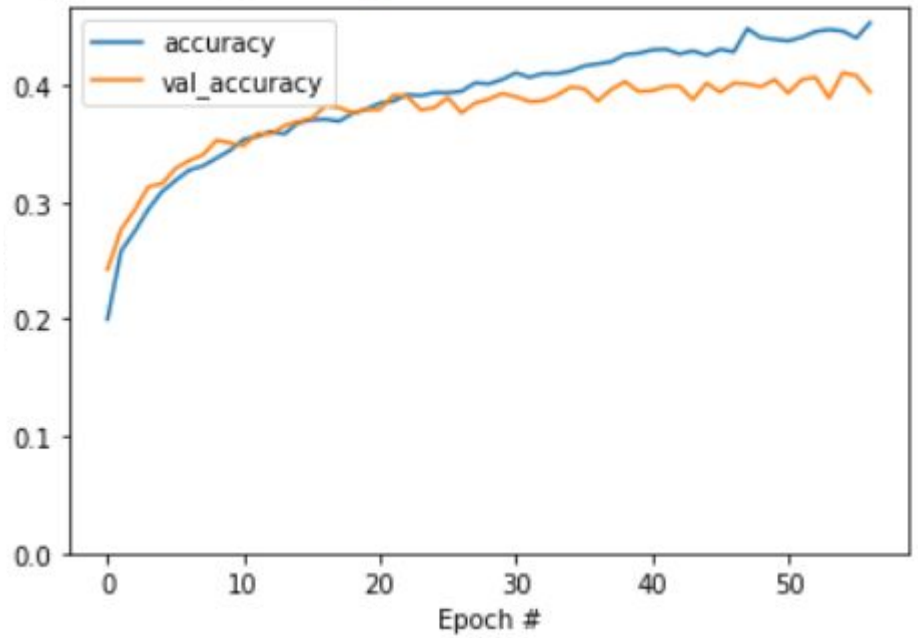
Image Model Comparisons

Image Model	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy	BleU Score
MobileNet	2.7318	42.29	2.8830	39.66	0.4293
Xception	3.1779	37.11	3.2496	35.47	0.3304
VGG16	3.0324	38.86	3.0671	37.64	0.4631
EfficientNet	2.5968	44.63	2.8575	41.07	0.4494

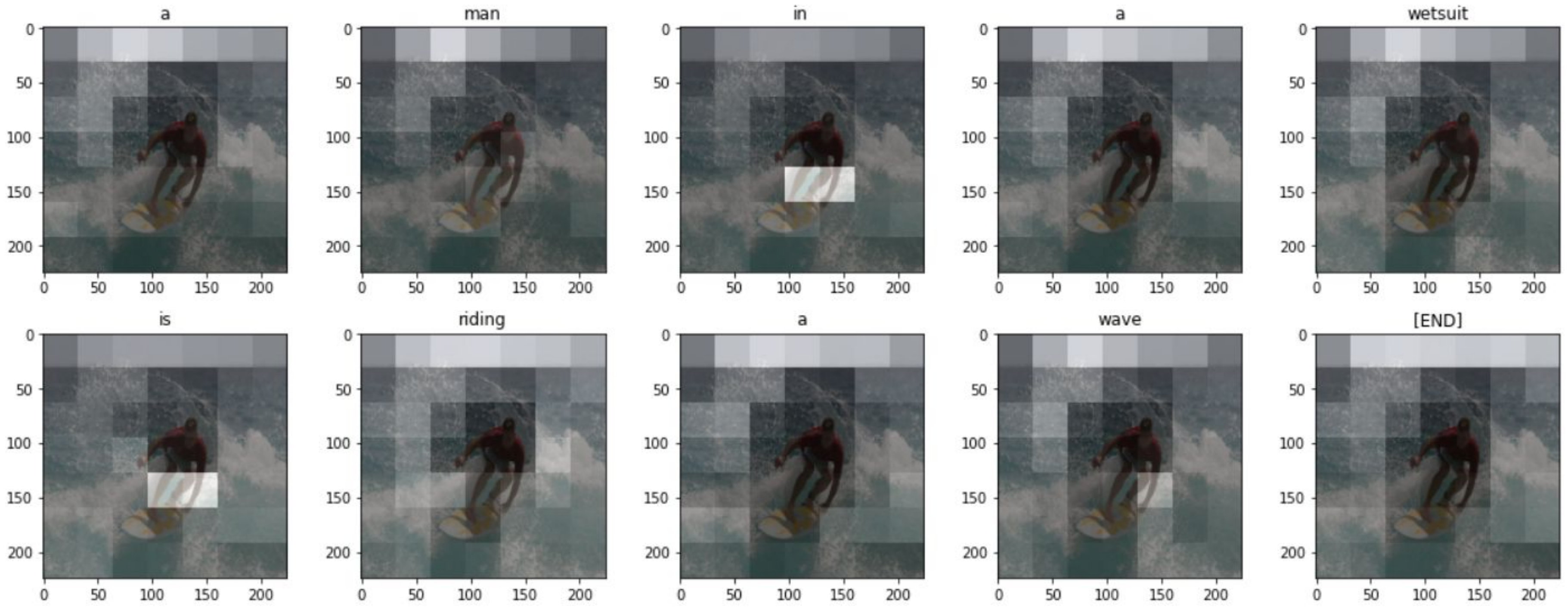
Note: The accuracy vs number of epochs graph for Xception suggests that the number of epochs we've chosen doesn't reach the minima yet, which might be the reason of exceptionally bad performance of XceptionNet

Loss and Accuracy vs No.of Epochs





Attention Map for our problem statement image



- Problem Statement and Related Works
- Datasets
- Workflow, Architecture, Technique
- Results and Analysis
- Demo

Thank You