Image Captioning with Transformers

and why teaching ANNs to produce audio from video is harder than it sounds

Marcel Banser, Kamran Vatankhah-Barazandeh, Robin Horn
IANNwTF - Osnabrück University - WiSe 20/21

1. Introduction

Fascinated by attention and the Transformer architecture, our goal was to build a neural network that was capable of a cool transformation, such as transforming videos into sound. The goal was something akin to adding sounds to movie scenes [1], but from a dataset of scenic videos with ambient sounds, and without a preexisting sound library. Although we learned a lot trying to achieve this objective, we ultimately decided to take a step down and change our topic to image captioning. There is related work in this area with vanilla methods [2]. This area also has a lot of interesting and useful applications (e.g. annotating images to make content more accessible (text-to-speech for images)), but avoids some of the limitations that we ended up facing. For more details on our change of topic, and what we learned, please refer to section 6. The Image Captioning Model is available on github. The attempted audio architecture, as well as a tool to build datasets for it, are also available there.

2. Related Work

There exist various approaches to image captioning that utilize a Transformer such as the Image Transformer [2], the Meshed-Memory Transformer [3], the Entangled Transformer [4] and the Captioning Transformer with Stacked Attention Modules [5].They all use region detection based on Faster R-CNNs [6]. But since we already had our Transformer implementation coming from our previous attempt we wanted to first build a basic model following a pure Transformer approach [2] before focusing our research on other papers that offered more advanced approaches.

3. Dataset

As it is a common dataset in the field of image captioning, we used the MS COCO dataset [7] in our task. The dataset provides 124000 images with up to 5 corresponding captions each. To fit the dataset into Colab we had to use the train/val split from 2014 and then had to load the train images before the validation images. Otherwise the extracted images plus the corresponding zip file during extraction would take up all of Colabs disk space.

4. The Model Architecture

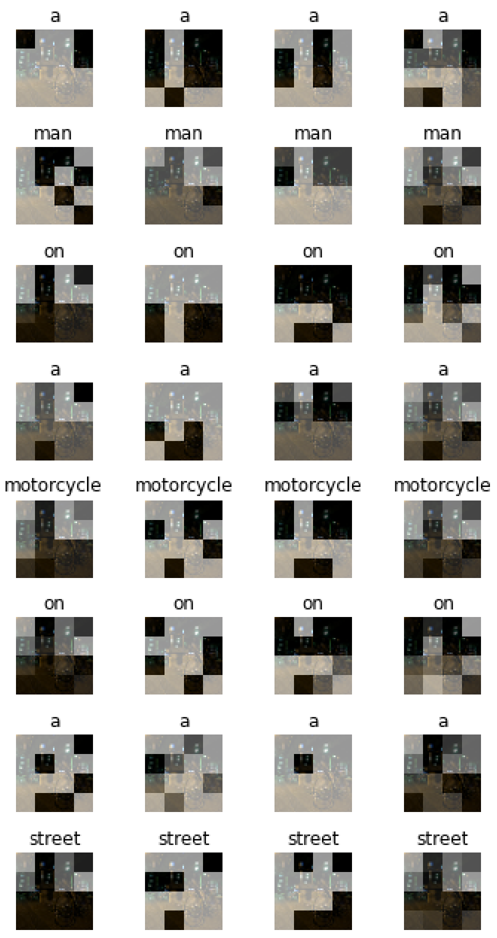
4.1 Main Model

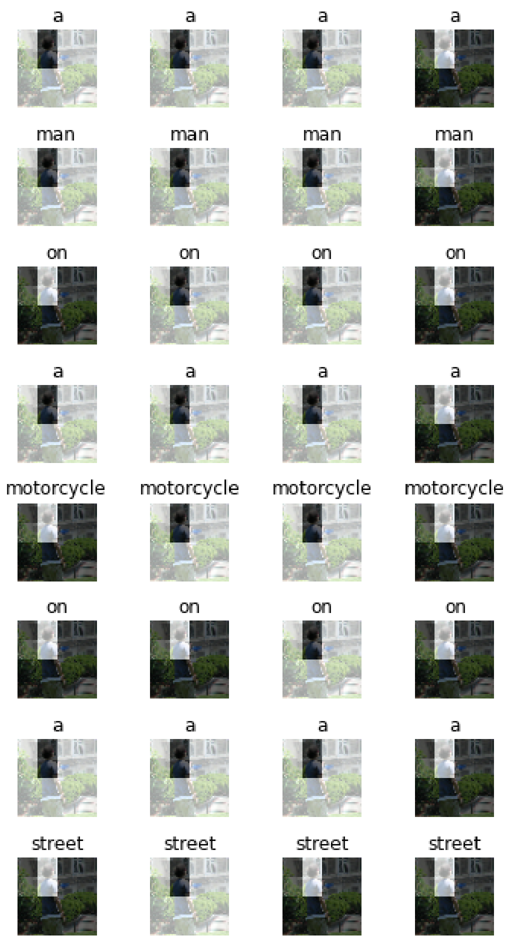
We use a Transformer architecture [8] consisting of multiple encoder and decoder layers. These layers are made up of attention blocks and feed forward blocks which both have residual connections. Our model is presented with a sequence of patches that are obtained from the input image. We explicitly do not extract any features from these patches, because that is the task of the encoder. The images are normalized between [-1, 1]. From these images the Transformer produces logit scores to predict the next word of the caption. To generate captions after the training we apply both a greddy and a beam search to the output.
In the following we will elaborate on the key features of our model and motivate why they are important.

4.2 Attention

In the encoder we use self-attention to get better representations of the input images. Each patch can attend to the other patches and incorporate their information to form a more useful representation.

In the decoder we then use the encoder-decoder attention so that the elements of the decoder which in the highest layer represent the caption words can attend to the different image patches. With this attention mechanism the decoder can incorporate the image information to infer the next word in the caption. Just like the encoder, the decoder also uses self-attention to attend to its own input. But unlike the encoder, it uses a look ahead mask to prevent the decoder from attending to its future prediction targets. We use multi-headed attention to allow the model to attend to different things during the same time step.



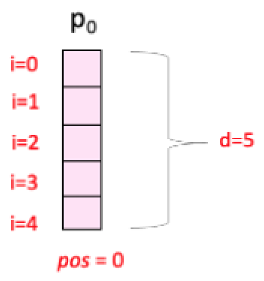


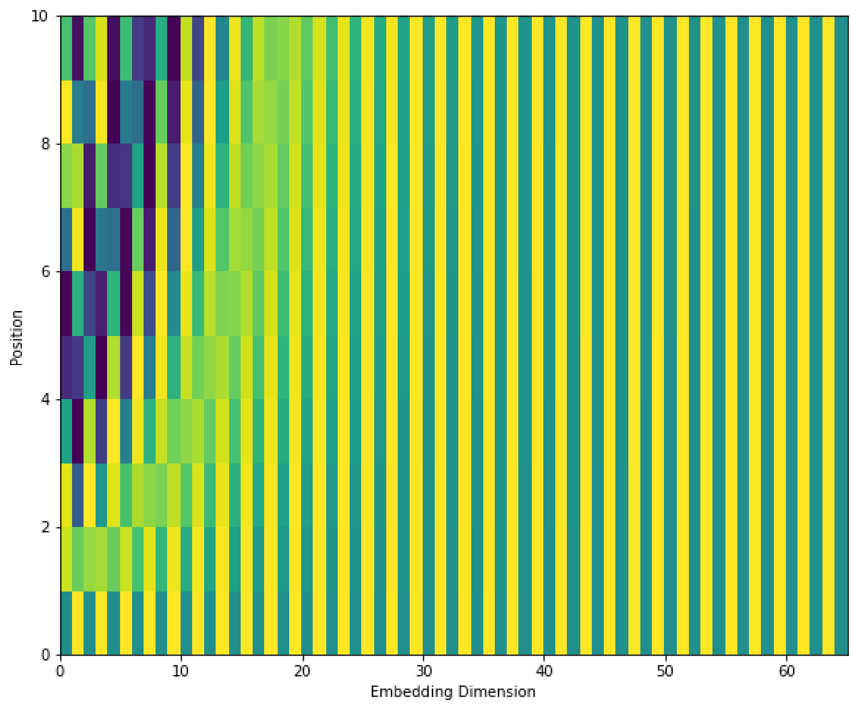
The attention weights are extracted from the last encoder-decoder attention layer. On the x-axis we have the 8 attention heads and on the y-axis are the different decoder outputs. We can see what parts of the image are important for what parts of the caption. Our model generates quite mixed results. Some attention weights seemed interesting (left fig.) while others lacked apparent purpose (right fig.). With regard to image captioning, attention is especially convenient because every caption element references different parts of the image.

4.3 Positional Encodings

Because we feed the input into our Transformer in parallel we need positional encodings [8] to inform the Transformer about the original position of the respective patches as well as the word positions in the captions. The positional encodings are generated based on the sequence position and the position inside the embedding. The encodings are then simply added to the input embeddings.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

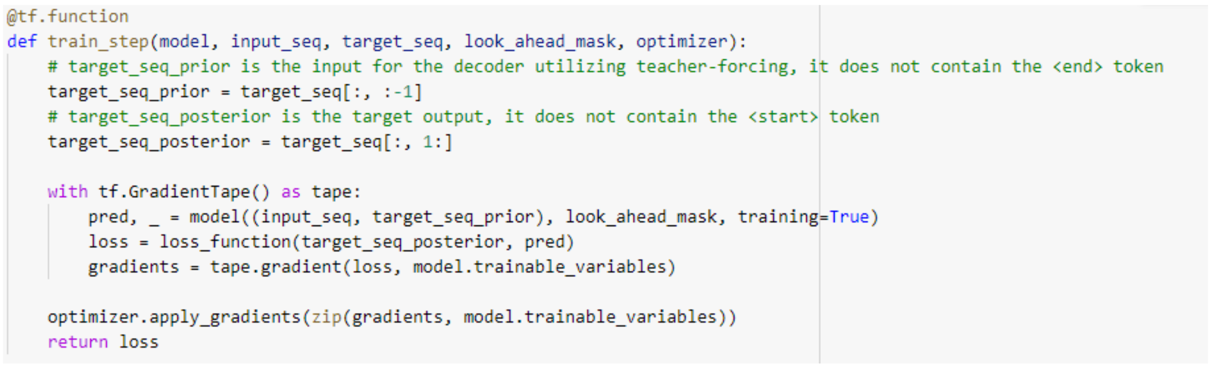




Example of positional encodings for sequence length 10 and embedding dimension 65.

5. Training

Because of hardware limitations we were only able to use a subset of our train and validation dataset.



```
@tf.function
def train_step(model, input_seq, target_seq, look_ahead_mask, optimizer):
    # target_seq_prior is the input for the decoder utilizing teacher-forcing, it does not contain the <end> token
    target_seq_prior = target_seq[:, :-1]
    # target_seq_posterior is the target output, it does not contain the <start> token
    target_seq_posterior = target_seq[:, 1:]

    with tf.GradientTape() as tape:
        pred, _ = model((input_seq, target_seq_prior), look_ahead_mask, training=True)
        loss = loss_function(target_seq_posterior, pred)
        gradients = tape.gradient(loss, model.trainable_variables)

    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    return loss
```

The benefit of the Transformer architecture is that it allows training to be parallelized. This requires a look ahead mask which prevents the decoder from looking at not yet predicted sequence elements. We also employ teacher-forcing which feeds the real targets into the decoder instead of its own predictions which could cascade into large errors the further the sequence progresses. These large errors would not be useful for learning since they arise from already wrong inputs.
For our gradient descent we tried different learning methods. First we tried a custom learning rate schedule inspired by the attention is all you need paper [8]. This yielded mixed results. Then we tried to lower the learning rate of the decoder in relation to the encoder in an attempt to compensate for faster learning of language features, which we inferred from the observation that our model was able to produce natural language, but irrespective of the input image.

6 Hyperparameter Search

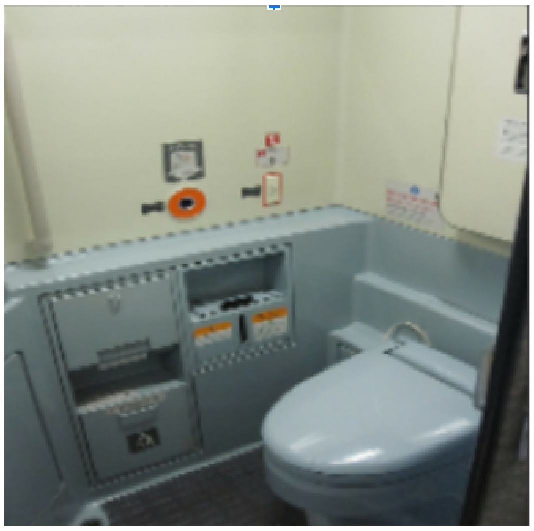
6.1 Troubles

In general, the vast amount of hyperparameters combined with the time required to train our model made it very hard to infer any useful knowledge about which parameters had which effect, if any. Further difficulties came from the fact that sometimes conflicting results were produced by the same exact model (apart from random initialization). The final nail in the coffin was again the hardware limitations. They made it impossible for us to train our model for more than a few epochs, specifically after reaching the Colab time restrictions one too many times (most likely getting our accounts flagged as using a lot of resources). Thus, we had to deal with even stricter runtime limits, that were accompanied by more frequent spontaneous forced restarts of the runtime, which also means redownloading the whole dataset before starting anew.

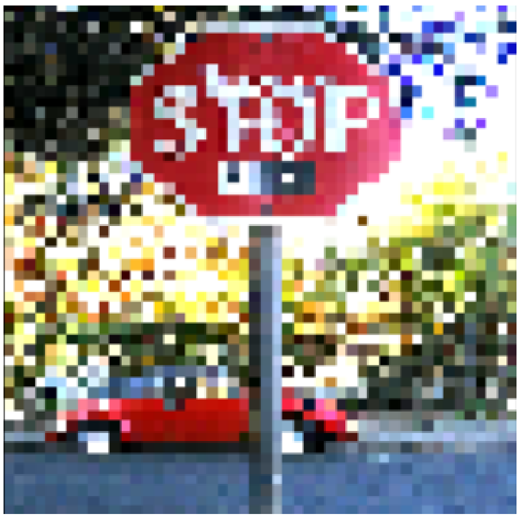
6.2 Findings

All things equal, reducing the learning rate from 0.001 to 0.00001 improved the model significantly. It went from only producing *“a a a a a a a a”* (the most frequent word in our vocabulary) to sentences like *“a giraffe on a giraffe on top of a bench”*. From that we conclude that if the learning rate is too high the model will immediately get stuck in a local optimum and tries to predict the “most common sentence or word”. A solution for this, besides adjusting the learning rate, could be a more diverse dataset. Now, the problem was that our model generated the same caption for every image. We partially overcame this problem by significantly reducing the complexity of our model.

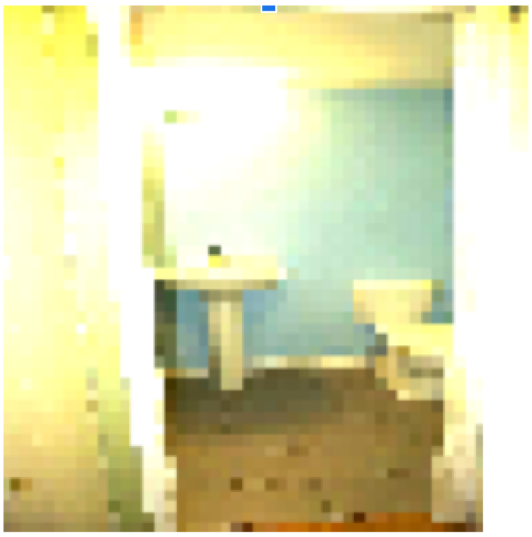
7. Results



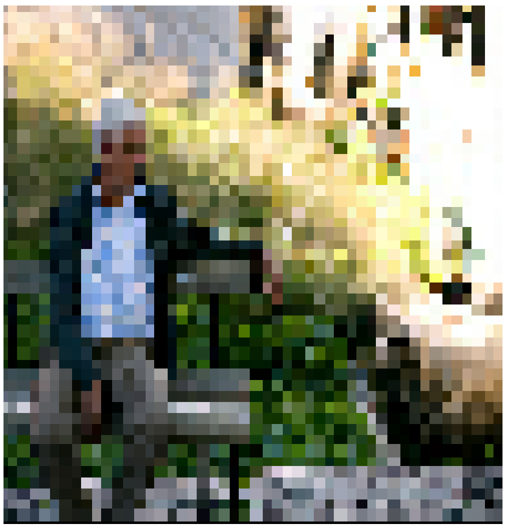
`a group of people standing on a table`



`a man standing on a motorcycle`



`a bathroom with a toilet and a shower`

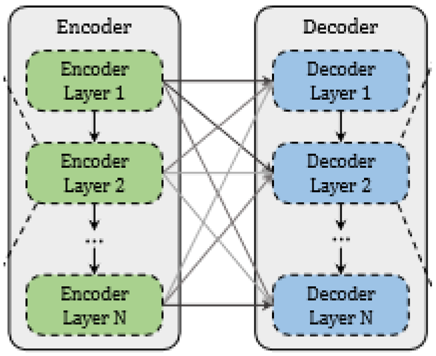


`a man on a man in a man`

As you can see the model was able to produce natural language but the problem was that it always generated the same caption regardless of the input image.

8. Outlook

8.1 Different Architecture



Given some more time our next step would have been to implement a Meshed-Memory Transformer [3]. This architecture has two distinctive features that make it superior to the vanilla Transformer.
The first feature are memory slots which are added to the key and value and allow the encoder elements to attend to something other than themselves and thus incorporate prior knowledge which is independent of the input. The idea is that the encoder successfully recognized a person and a basketball, but to infer the concept of player or game we would need some prior knowledge. The second feature is a mesh between the encoder and decoder. The strings of the mesh each represent an encoder-decoder attention connection which means that every decoder layer can not only attend to the encoder output but to every encoder layer and thus incorporate low as well as high level features from the input images.

8.2 Other Ideas

While we tested our model it often produced the same captions for all images. To prevent this it might help to increase the dropout values in the decoder to incentivise a more complex language representation.

9. Why teaching ANNs to produce audio from video is harder than it sounds

9.1 Huge amounts of data

```
video_len = 1000 #in min
fps = 10
video_h = 200
video_w  = 200
print(f"Memory in GB: {(video_len*60*fps*video_h*video_w*3)/10**9}")
```

```
Memory in GB: 72.0
```

To store video material alone is quite demanding without compression. And even when scaled down - in our case, we resorted to 96p@3fps - getting the data into Colab is a task in itself that can force many runtime restarts. What we learned in handling huge amounts of data is that mounting your Google Drive onto Colab is the most convenient method of accessing your data.
Audio also demands some disk space, as we also need to access the uncompressed sequence, but the size of our sound samples is more of a problem when it comes to the attention (see next section).

9.2 O(N^2) Attention

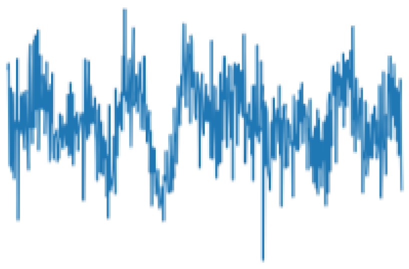
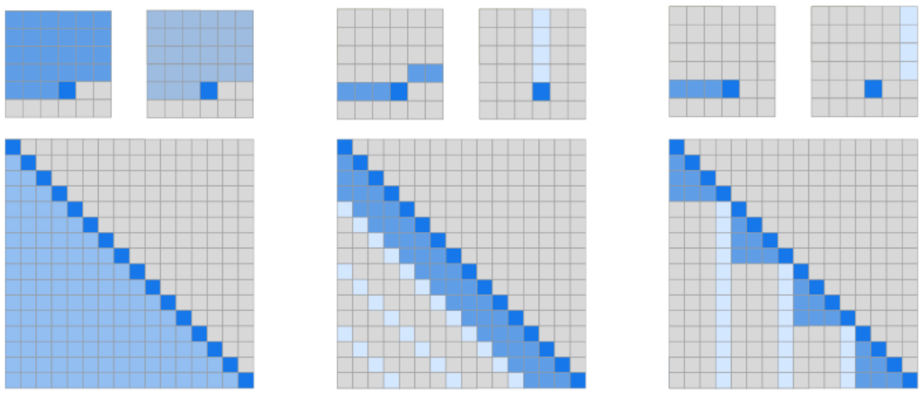


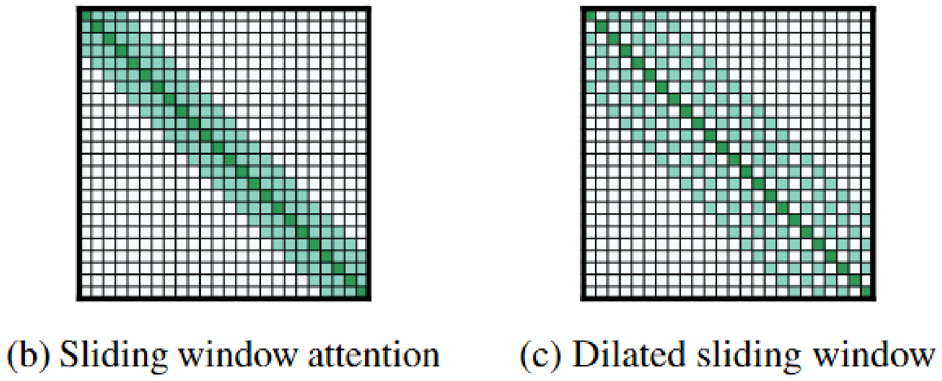
fig. 1 audio in waveform

You are likely familiar with seeing audio represented in a waveform. On the x-axis, we plot time, and on the y-axis, we display the corresponding amplitude.
The resolution of the x-axis is determined by the sample rate. For regular audio (e.g. CDs), this is often 44.1 kHz, which means we have 44,100 samples per second. This ensures perfect sampling for frequencies up to the Nyquist rate of 22,050 Hz (which should cover all frequencies regular human ears can perceive). The resolution of the y-axis is usually 16-bit, meaning we have 65,536 possible values. Again, this means

high resolution audio, likely beyond what we need when creating a proof of concept. Already, this seems like a long sequence with many possible values.
To make the prediction easier for the model, we can lower the resolution of both axes. We reduced it to a sample rate of 16khz at a resolution of 8 bits (256 possible values). This is on the low end of audio quality, but could still display recognizable sound. With this resolution we get some quantization noise and only frequencies below 8khz are detectable, but the audio prediction task becomes more feasible. Now, for one second of audio the model has to predict 16,000 values. Because the decoder of the Transformer takes the previous model output or the previous target sequence as input, it has to self attend to this huge sequence. Self attention being quadratic in nature, we encounter infeasible memory usage and computing time. To give some perspective: For only 1 second of audio, every decoder layer needs an attention matrix with 256 million parameters. Fortunately, there are established approaches that tackle the N^2 problem. The first architecture we came across is called the Sparse Transformer. [9]



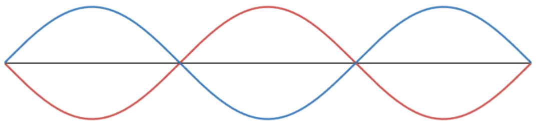
Its name already conveys the main idea. Instead of every element attending to every other previous element, they only attend to their close neighborhood and then to the rest in a striped pattern. This architecture is shown to produce great results with only a $O(N\sqrt{N})$ attention cost. But we can do one better with the Longformer [10]. Here, we ditch the striped pattern and only attend to a sliding window. This window only includes the immediate neighborhood of an element, or can be dilated out to include for example every second element in a range of 6.



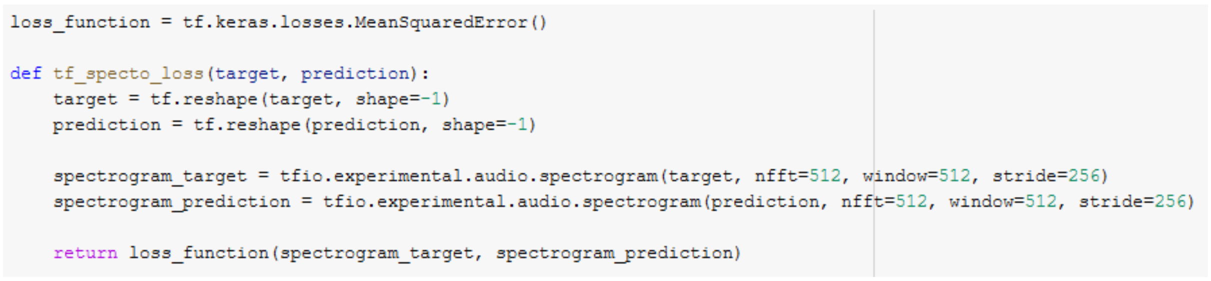
With this tradeoff we end up with only a $O(N)$ attention cost, which makes very long sequences feasible for the longformer while still outperforming other state of the art models. This can be explained with the fact that often attention only looks at the immediate neighborhood anyway. Sadly, tensorflow does not provide sparse matrix multiplication, making any implementation attempt very difficult.

9.3 Loss function for audio

9.3.1 Naive approach



Up until this point, we represented our audio sequence as an audio array, storing the amplitude for each time step given our sample rate. However, imagine a pleasant sounding sine wave. If we shift this wave by half its wavelength and listen to it again, it will sound the same to us. But a loss function comparing the amplitudes of these arrays would tell you that they are nothing alike. To solve this issue of encapsulating the perceived audio difference in a loss function we stumbled across a very interesting paper [11] where an ANN was trained on a “just noticeable difference” dataset and then established as a loss function. Sadly, the implementation was not available for tensorflow 2.x .
So we thought of an alternative:



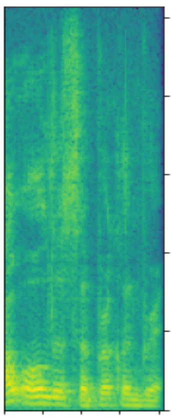
```
loss_function = tf.keras.losses.MeanSquaredError()

def tf_specto_loss(target, prediction):
    target = tf.reshape(target, shape=-1)
    prediction = tf.reshape(prediction, shape=-1)

    spectrogram_target = tfio.experimental.audio.spectrogram(target, nfft=512, window=512, stride=256)
    spectrogram_prediction = tfio.experimental.audio.spectrogram(prediction, nfft=512, window=512, stride=256)

    return loss_function(spectrogram_target, spectrogram_prediction)
```

9.3.2 Spectrograms



To give the model some leeway in its predictions - especially in the time domain - we tried a loss function that compared the audio not as a time vs amplitude array, but looking at the frequency spectrum over time. Spectrograms are plots that capture exactly this property of an audio sequence. This is done not for every sampling step, but over time windows, thus being more lenient if the model happens to be off by a tiny bit.
While this approach seemed promising to us, we are not certain whether it improved the models prediction. For proper validation the model would have needed to produce at least a few meaningful sounds to begin with, and not some noise that might pass as a heavily distorted electric guitar but nothing more. By this point, letting the model train for a couple epochs in order to quickly test out improvements on the loss function or other parameters started becoming difficult, as the Colab runtime started taking quite a bit of time given the large sequences of our in- and output.

References

[1] Sanchita Ghose, and John J. Prevost. AutoFoley: Artificial Synthesis of Synchronized Sound Tracks for Silent Videos with Deep Learning. In *IEEE Transactions on Multimedia (Early Access), 2020*. PDF on arxiv.org

[2] Sen He, Wentong Liao, Hamed R Tavakoli, Michael Yang, Bodo Rosenhahn, and Nicolas Pugeault. Image Captioning through Image Transformer. In *Proceedings of the Asian Conference on Computer Vision, 2020.* PDF on arxiv.org

[3] Marcella Cornia, Matteo Stefanini, Lorenzo Baraldi, and Rita Cucchiara. Meshed-Memory Transformer for Image Captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.* PDF on arxiv.org

[4] Guang Li, Linchao Zhu, Ping Liu, Yi Yang. Entangled Transformer for Image Captioning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 8928-8937.* PDF on openacess

[5] Zhu X, Li L, Liu J, Peng H, Niu X. Captioning Transformer with Stacked Attention Modules. In *Applied Sciences. 2018; 8(5):739.* PDF on https://doi.org/10.3390/app8050739

[6] S. Ren, K. He, R. Girshick and J. Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017.* PDF on arxiv.org

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, C Lawrence Zitnick. Microsoft coco: Common objects in context. *In ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham.* PDF on arxiv.org

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. *In Advances in Neural Information Processing Systems 30 (NIPS 2017).* PDF on arxiv.org

[9] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers. As *arXiv preprint arXiv:1904.10509,* PDF on arxiv.org

[10] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document Transformer. As *arXiv preprint arXiv:2004.05150,* PDF on arxiv.org

[11] Pranay Manocha, Adam Finkelstein, Richard Zhang, Nicholas J Bryan, Gautham J Mysore, and Zeyu Jin. A Differentiable Perceptual Audio Metric Learned from Just Noticeable Differences. As *arXiv preprint arXiv:2001.04460,* PDF on arxiv.org