Dual Transfer Learning for Caption Generation

Xavier Reyes Tejedor

Abstract

Transformers are widely used to convert words to meaningful vectors, as we see in the Transformer Encoder architecture (see BERT), or to generate new words from other words, as we see in a Transformer decoder or in the full Transformer architecture (see GPT-2) or in the full Transformer encoder-decoder architecture. Now, observing at the current state of research, we can see that the Transformer architectures do not allow adding any non textual context to the text it generates.

It is for this reason that this project will be centered on generating descriptive sentences using only meaningful vectors extracted from images, using the power of Convolutional Neural Networks (specifically EfficientNet) and the Language Model based on the Transformer Decoder architecture GPT-2. The final purpose is then, to generate captions from a given image.

Introduction

Taking a look at current research, it can be seen that recently the research on caption generation for images has been poor. While there are some papers on it, the majority of them are based on traditional recurrent neural networks trained directly from annotated datasets and are from around 5 years ago. It can be seen that current results can be improved, by using models based on the new very relevant breakthroughs on Attention and the not that relevant but still notable improvements in image recognition.

In this paper, we present a method for merging the outputs of two big and well known models to be able to extract a sentence from a given image. The models used are GPT-2 on it's small variant and the EfficientNet B4 using the AdvProp improvement. The merging is done using a simple multi layer perceptron with skip connections as checkpoints to improve convergence. The merging done is done using the COCO dataset in its 2017 version, consisting of more than 150k unique images with about 5 captions per image.

It is important to note that the network we are trying to construct is really heavy on its backbone, and tricks such as gradient checkpointing, gradient accumulation and FP16 training among others have been tested or used to be able to The results are not only comparable to the current state-of-the-art but hugely better, specially on the precision on low n-grams when compared to the best models available.

this motorcycle has pictures of dollar bill and an us flag on it.
a parked motorcycle sitting in front of a tall building.
we are looking at a low angle shoot of a motorcycle.
motorcycle with american flag motif parked on a brick-paved lot.
motorcycle with patriotic features covering the fuel tank

In this image we see a sample image with different valid captions for it. Extracted from the COCO dataset.

Approach

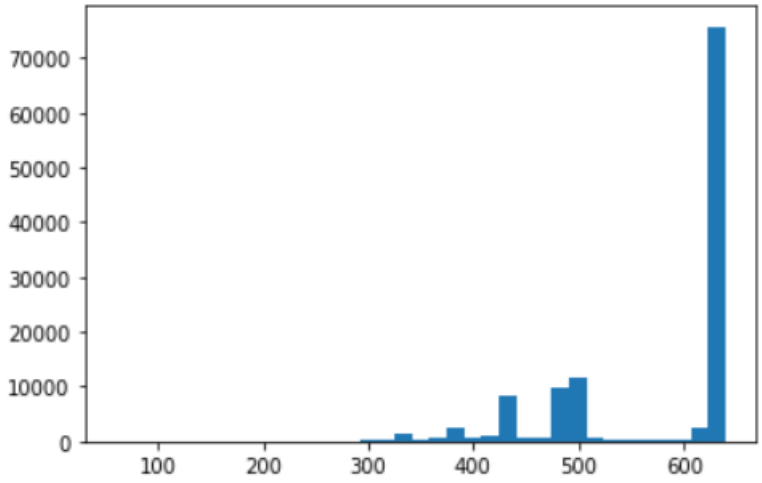
The approach used to obtain the final model is composed of multiple steps:

Dataset

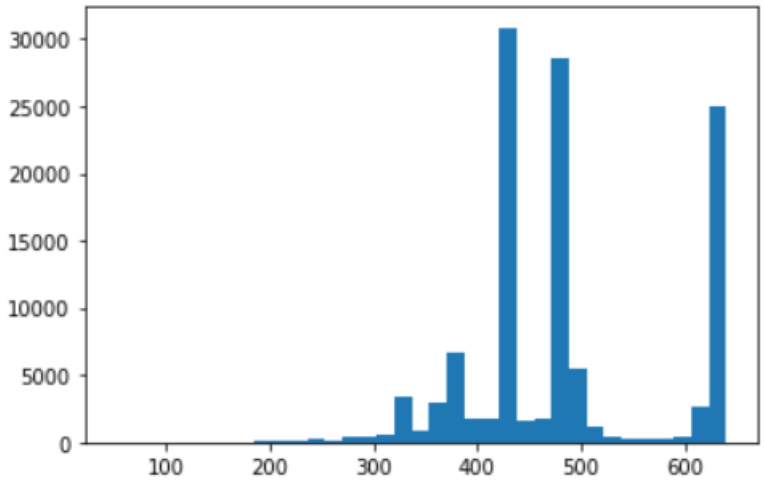
First things first, the COCO dataset was first downloaded and preprocessed, obtaining all images uncompressed and stored in a directory in the computer and a file with all the Captions stored. It is important to note that the train and validation images itself weigh almost 20 GB, which was too much for the RAM available in the PC used. The approach to overcome this problem was to read the images directly from the disk in training time. While it could be very inefficient, due to the size of the model used and that the images are stored in a Solid State Drive, the overhead applied is not very big. It may be important to note that I could have reduced the dataset by increasing the number of workers in the PyTorch Dataset object, but this feature is not compatible with the Operating System being used (Windows).

The images are transformed so that they are square (by only taking the central part of the image) and have the adequate size for optimal performance of the EfficientNet being used (380x380px for B4). We can say that there is no loss in quality, since the great majority of the images have more than 380 pixels in both dimensions. It may be worth noting that in general the images are more horizontal than vertical (more pixels on the horizontal axis). The color transformation used for training the AdvProp variant is also applied to maximize compatibility and convergence.

Widths hist:

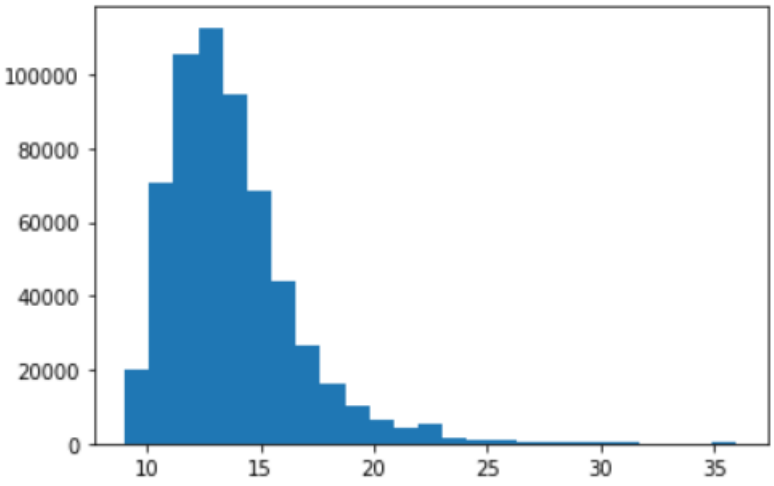


Heights hist:



Distribution of widths and heights for the images of the COCO dataset used.

The captions are transformed so that they start and end with the EOS token for the GPT-2 model, so that the model has an initial token and knows how and when to end the generation, respectively. For memory and time constraints, the maximum length of generated sentences has been reduced to 36, with longer sentences ending prematurely. An histogram with the distribution of lengths for training captions is provided below.



Distribution of caption lengths in GPT-2 tokens (including EOS) for the model being trained.

As it can be seen, very few captions are actually cropped (on the order of 0.1%). The mean caption length is of about 12 GPT-2 tokens (which is in general similar to the amount of words in the caption) excluding the two EOS tokens.

Image Model

As it has already been said, the Image model used for the project is EfficientNet in its B4 variant and using the AdvProp pretrained weights. As it has been said, the images fed to it are transformed so that they are 380x380 and transformed according to the method of the authors of AdvProp.

The EfficientNet model outputs a vector of size 1000, which describes the different features in the image being processed.

It is important to note that gradient checkpointing has been used on the final model on each convolutional block to reduce the amount of memory that the model needs while preserving the maximum performance possible.

Language Model

The language model which has been used is the small version of the GPT-2 model, having 122 million parameters. Elementally, it is fed a vector of tokens and for each element of the vector produces the probability distribution of the next token. In order to be able to have batches, all instances have to be fixed to a given number of tokens (in this case, 36).

For the task in hand, we tear the final transposed embedding layer off so that instead of having probabilities, we end up with the hidden vector of the model, formed by 768 features.

Gradient Checkpointing has also been used to

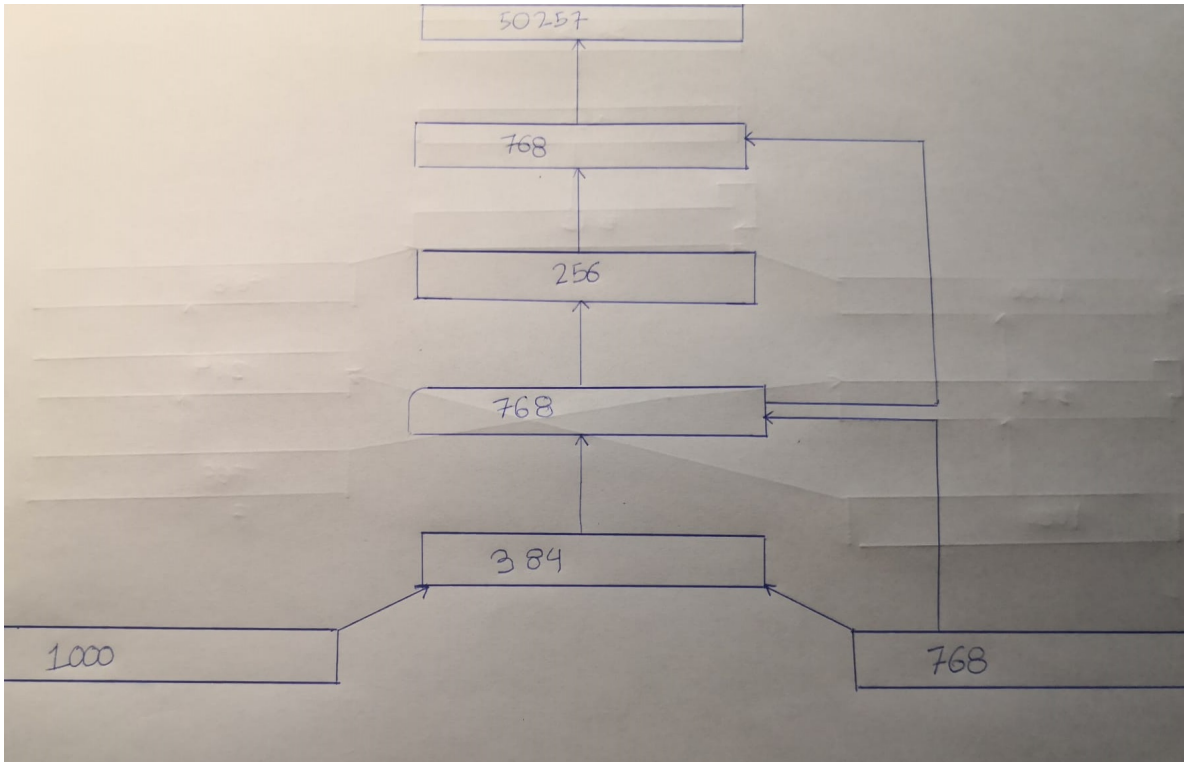
Merging Model

At this point, we have two latent vectors at each time step. On one hand, the 1000 size vector output of the EfficientNet for the image we are creating the caption about and on the other hand the 768 final hidden vector of the GPT-2 model.

What we want to do is to train a MLP to merge those two vectors into a single 768 sized vector that is in an acceptable format to apply the transposed embedding layer that we pulled out previously from the GPT-2 model.

We use skip connections in order to improve convergence, and in a more informal way to alter the distribution provided by GPT using the image, instead of trying to rebuild the distribution from scratch. This skip connection approach allows us to have intermediate layers of less size (384, 256) than the full feature vector we are constructing (768), which is useful considering the amount of training samples is not sufficient to train very large networks.

Two hidden layers instead of one are used to make sure that the merging model is not limited by the lack of nonlinearities. Each hidden layer is provided with ReLU activation functions to create nonlinearities.



The architecture of the merging model used.

Experiments

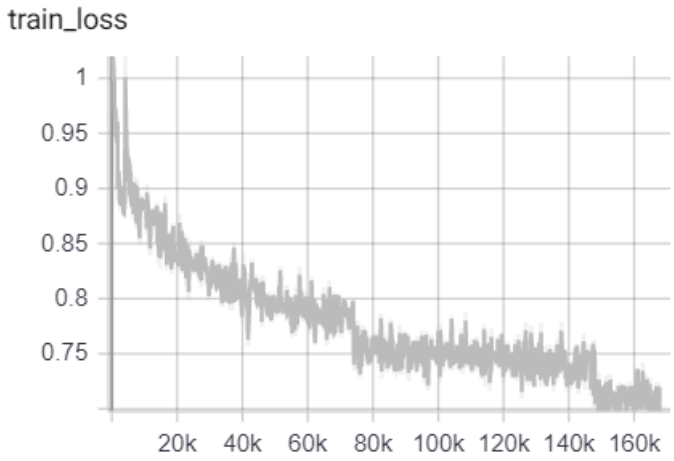
Additional Hyperparameters

Some of the hyperparameters have been already disclosed while describing the architecture. In this section the rest of the hyperparameters will be disclosed.

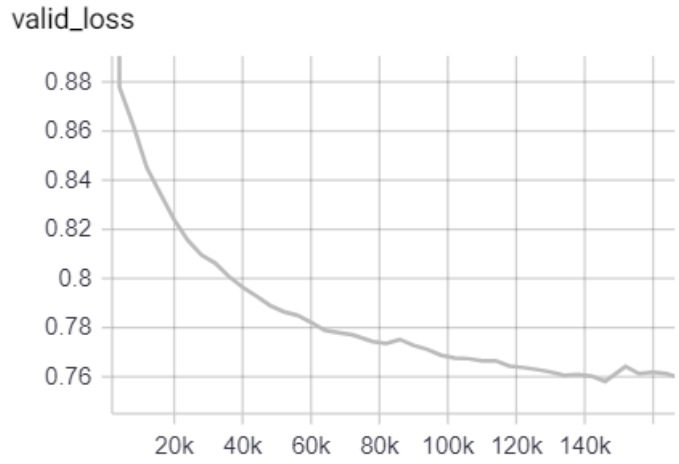
- **Optimizer:** The Adam optimizer, with no weight decay. It is a good optimizer that needs little tuning.
- **Learning Rate:** `3 * 10^-5`. It is low since we don't want to "destroy" the already pretrained weights, but redirect them to a disposition that maximizes the performance for the task in hand.
- **Batch Size:** The batch size used for training is 8. I tried to make it as high as possible as my GPU memory would allow, since at this level an increase in batch size is a training speed improvement with no performance loss at all (or even the opposite, in some cases). The batch size for validation is 16, but this can be as high as the memory allows.
- **Epochs:** As we will see later, the optimal number is epochs for this architecture is 2. After that, the validation loss starts going up slowly but steadily.

Training

The training was performed on an RTX 2060 and took around 20 hours. The required time per epoch was 20 hours, which limited our ability to test different architectures. The first epoch reduced significantly the validation loss. On the second, the improvement stagnated but it was still improvement. After starting the third epoch, it could be seen that the loss started going up slightly. It is easy to appreciate significant drops in training loss over each epoch. This happens because the training dataset is large enough and enables us to see the drops and the model has an outstanding ability to fit the samples really fast.



The Training loss over time for the model being used.



Evolution of the Validation loss for the model used.

Evaluation Method

I have evaluated the model using three criterions.

- **Validation Loss:** The main metric, being the mean cross entropy loss for each token over the entire validation dataset. It must be said that it is only applicable over models trained with this exact number of maximum tokens, since the masked tokens also account for the loss, effectively showing a lower than real loss result.
- **BLEU score:** This is the main metric to compare with other approaches, and the reason it can be said that the model is state of the art. We present BLEU scores in different beam search configurations, and across the first 4 n-grams (1-4), which is the standard for comparing this caption generation models and in general language generation tasks. All provided metrics is done on execution of the model on the 5000 validation images of the MS COCO dataset.
- **Own Criterion:** Finally, the results can't be fully decided without actually testing the model yourself and comparing the output of the model with the description that you would generate if you were asked to.

Results

Validation Loss

The final Cross Entropy validation loss on the main model has been **0.758**. This loss is not too descriptive by itself but it tells us that the probability of choosing the most likely token at a given time step is significant.

BLEU score

The BLEU score of the model using an isolated n-gram from 1 to 4 using a beam search of 5 is:

```
B1: 0.924
B2: 0.798
B3: 0.665
B4: 0.577
```

On the other hand, the BLEU score using a higher beam search of 64 have been the same but slightly higher. So we have worse results.

Comparing now on the results extracted from "What Convnets Make for Image Captioning?", which is state of the art for this task (using beam search of 5) is:

Method	B-1	B-2	B-3	B-4
Karpathy et al. [12]	0.625	0.450	0.321	0.230
mRNN [19]	0.670	0.490	0.350	0.250
NIC [26]	-	-	-	0.277
LRCN [3]	0.669	0.489	0.349	0.249
gLSTM [8]	0.670	0.491	0.358	0.264
Bi-LSTM [27]	0.672	0.492	0.352	0.244
VNet-ft-LSTM [28]	0.680	0.500	0.370	0.250
Soft-Attention [29]	0.707	0.492	0.344	0.243
Hard-Attention [29]	**0.718**	0.504	0.357	0.250
Jin et al. [10]	0.697	0.519	0.381	0.282
ATT-FCN [31]	0.709	0.537	0.402	**0.304**
Ours	0.707	**0.548**	**0.410**	**0.304**

As it can be seen, the model improves the BLEU score by a very significant margin, driving the precision to even scaring levels. This model could be used a a teacher for littler student models.

Own Criterion

The samples generated are of really good quality. Even if the BLEU score was lower for higher beam search, we happen to find higher beam searches more accurate. In particular, let's see some examples using a beam search of 256.

```
A glass vase filled with flowers on top of a table
```

A group of people standing around a living room.

A group of people sitting around a table.

Conclusions

We have presented a model which outperforms the current state-of-the-art by a significant margin, which is a good improvement for the current research on the topic. This approach demonstrates the powerful capabilities of transfer learning and may invite other researchers to use this powerful pretrained models in other tasks.

We have had a lot of difficulty at the training stage of the model, and preprocessing and avoiding out of memory errors was the most consuming part of this work.

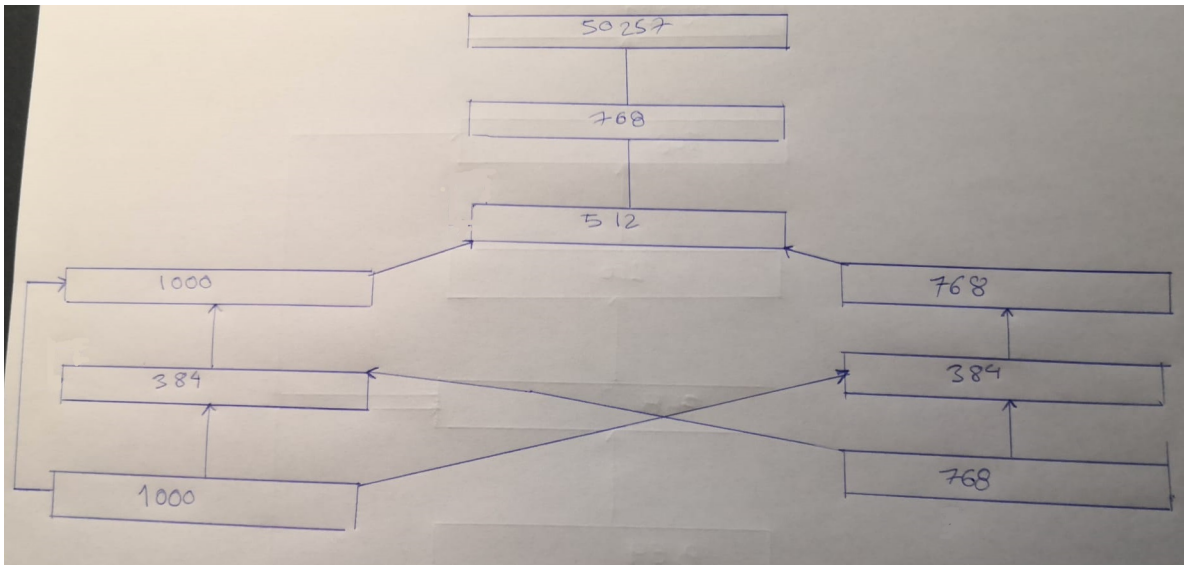
The model alone can be used for recreational purposes to see the type of captions generated. We also devise that the model can be used for automating the generation of captions, for example, which may save a lot of time. Evolved models may of this one may be useful for blind people, so that they can have a constant voice that describes the environment for them

While this architecture is really heavy, it can be used to tag an unsupervised image dataset and train a smaller task-specific student to achieve similar results to it, while improving the performance dramatically.

Further Work

On the presented architecture

The model could be improved by using a more sophisticated merging network architecture, which I expect it to improve the loss by a bit, as can be seen by the initial training loss and validation curve. This architecture can be seen in the figure below. It has not been trained due to lack of time, since training the model for the two needed epochs takes about 40 hours on my PC.



The slightly better merging architecture.

Another way of improving the model in a more significant way would be to use a bigger pretrained ImageNet and GPT-2. This has not been done since the my GPU has not more memory available and trying to use bigger models would imply making too many sacrifices in training time and convergence.

We may also use the same architectures but in its improved version. For the language model, this one is the brand new GPT-3 on the small version. For the Image model, it would be the current state-of-the-art EfficientNet in its FixEfficientNet variant and using the same B4 model.

Maybe using the EfficientDet model instead of EfficientNet could be useful, since the former is used for object detection and the captions generated seem to fail at knowing relative locations and number of elements of the same type. The way of fitting the EfficientDet outputs to the current architecture should be studied.

On new possible architectures

The inverse version of this task, being **generation of images from text captions** is more difficult, since we have less data as input and have to generate a big output. The network would have to convert a very small vector of tokens to a full sized image. This could be accomplished by converting the **tokens to a meaningful vector using a language model** and then to convert

that **vector to an image using a transposed convolutional network**.

The language model for this case would be better if it is **bidirectional** (like BERT), being a Transformer encoder. This type of models can convert a variable sized text into a single vector, that may be used as an "encoded sentence". Now, to convert that vector into an image it would be harder. In my opinion, using a GAN might work to train the fresh transposed neural network. We would use the same COCO dataset, but inverting the inputs and outputs.

The way of doing this would be to use the captions along the real and fake generated images (using the generator) architecture to train the Discriminator, which would be a mix between a CNN (to encode the input image) and MLP (to use the BERT-like encoded sentence also as input). The Generator, on the other hand, would be trained using the available captions (and obviously the output of the Discriminator). It would be a transposed convolutional network, whose input is the vector representation of the caption.

Using the network described in this work would be very useful in order to generate training data, since we could potentially generate Captions on all the ImageNet dataset and then use those generations to train our GAN. This converts the problem in hand from an supervised one to a unsupervised one.

References

- BERT: https://arxiv.org/abs/1810.04805
- GPT-2: https://openai.com/blog/better-language-models/
- Attention is all you need: https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
- EfficientNet: https://arxiv.org/abs/1905.11946
- AdvProp improvement: https://arxiv.org/abs/1911.09665
- COCO dataset: http://cocodataset.org/#home
- What Convnets Make for Image Captioning?: https://www.researchgate.net/publication/311990686_What_Convnets_Make_for_Image_Captioning

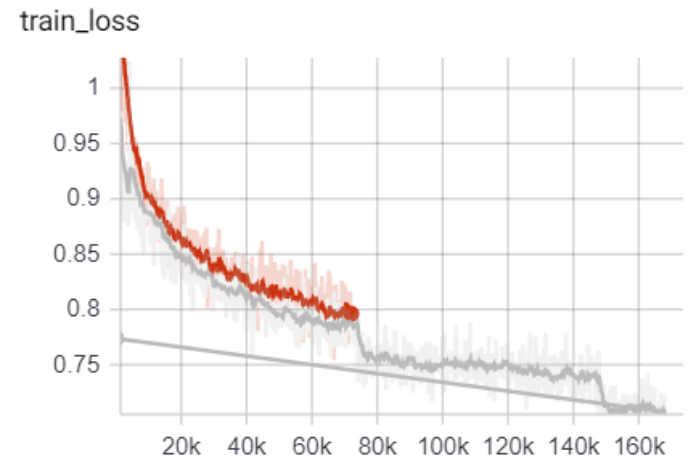
Appendix

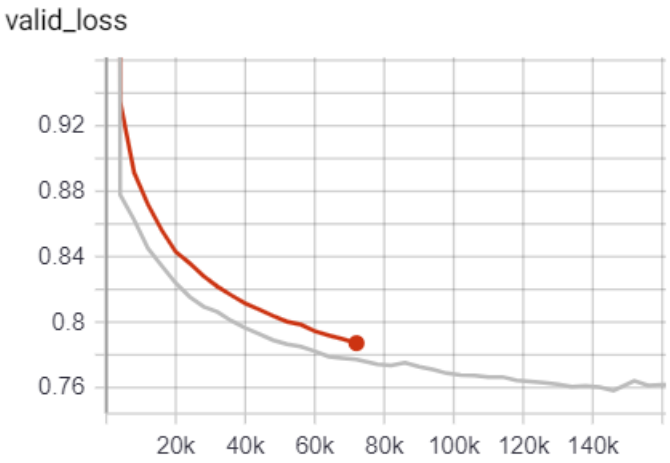
Some extra tricks for reducing memory usage have been tried:

- **FP16 training:** It was discarded since the accuracy of the model was affected, the performance did not change since Windows can't make use of the Nvidia Tensor Cores in my RTX 2060 GPU and lastly, the reduction in memory used was not sufficient to justify using it on the final model.
- **Gradient accumulation:** I tried to accumulate the gradient to have batch sizes of 16 instead of 8, but at the end I ended up discarding the approach since the reduction in loss per time step seemed to be worse using this approach.

Tricks for improving the merging model were also tried:

- **Batch Normalization:** It was not useful since the batch size was not high enough to take advantage of it, plus the implementation was hard since we don't want to normalize all time steps since it can give the model info about future time steps.
- **Layer Normalization:** Here, the hidden vectors at each time step are normalized individually. It was not useful, since the loss obtained was equal or even worse when compared with the initial approach.
- **RNN:** I tried the GRU variant of RNN, trying to feed the previous time steps to the current time step. The results were not favorable, since all the needed information seems to already be in the 768 length vector of GPT-2 and providing previous data just confuses the model. The results are on par but slightly worse when comparing to the regular approach.





The train and validation loss for the main model (gray) and for the RNN model (red).

Repository where the code used can be found:

https://github.com/xaviolo99/dual-transfer-caption-generation