Emile Breyne r0629298

IRSE project: part 1

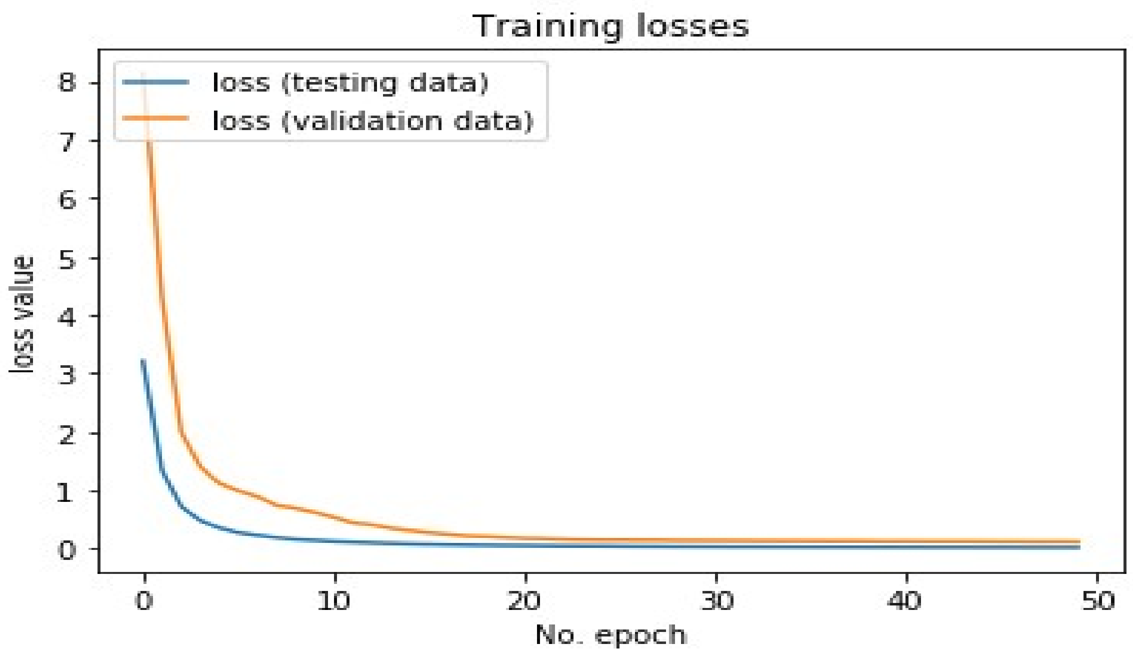
I have implemented this part of the project alone. I based most of my approach on the original paper by L. Wang, Y. Li, and S. Lazebnik, “Learning deep structure-preserving image-text embeddings”. I used keras as my main framework, with tensorflow as a backend.

I created a simple 2 branch network. The dimensionality of the output embedding space is fixed to 512. As the images were already pre-processed and represented by vectors in 2048 dimensions, I directly plugged them to the network like this.

Image Pre-Processing : For the text on the other hand, I did a little bit of pre-processing. For each image, I took all of the words from the 5 corresponding sentences as one document and created a bag of words. I filtered out stopwords (‘a’, ‘the’, ‘and’, etc.) and applied **tf-idf weighting.** I implemented this part myself (can be found in *txt_preprocessing_emile.py*) to show that I easily understand the principles but then replaced it by scikit-learn’s tfidfvectorizer for ease of use, and because it is better optimized. I included support for using stemming (Lancaster stemmer) too, but left it out in the final results as I felt it didn’t bring much benefits. The size of the vocabulary was still almost 30000. This would result in a very big neural network with a lot of weights, and very sparse representations of the text documents. Wang et al. also argued that this sparsity leads to instabilities in the learning process. As a result, I chose to reduce the size of the text document representation by using a truncated SVD transformation. This can be seen as doing a Principal Component Analysis of the Tf-Idf BOW representation matrix, and keeping the n most important dimensions. I did this twice, once reducing to (n=) 2048 dimensions and once to 4092. I will then compare the results to see if using a bigger representation (which also results in a bigger neural network) yields better results on this particular dataset.

Network Architecture: My network is very similar to the one proposed by Wang et al. in their paper. When using a text representation of 2048 dimensions (same as the images), the 2 branches are identical: It consists of 2 fully connected layers, one hidden with size 1024 and one output with size 512, respectively. I used a rectified linear activation unit (ReLu) activation function to introduce non-linearities, and added drop-out layers to prevent over-fitting. At the end, I normalize the outputs using a batch normalization layer.
For the part where the text input size is 4092, I just increase the size of the hidden layer to 2048, and keep the rest the same.
I used the exact loss described in the assignment. As this had to run on the gpu, I had to model the loss function entirely using tensorflow matrix operations exclusively, which proved to be the most challenging part of this project. The model converged in approximately 20 iterations. Training and validation loss (for text dimension 4092) can be seen in the next figure (wrong legend: blue is training data):

Emile Breyne r0629298



Retrieval: In order to find the image (or text) embeddings that are closest to a given text (or image) embedding, I just compute the distance to all image (or text) embeddings in the given dataset and rank them accordingly. This is feasible for the test dataset with 1000 images, but not in a big real-life application and further development has to be done on this part.

Results : I learned the embeddings of both images and captions of the test set and ranked the corresponding images for each text document (that is, the bag of words of the 5 corresponding sentences). I use the Mean Reciprocal Rank (MRR) and HITS@n (fraction of examples where the relevant image is in the first n ranked images). Results can be seen in the next table:

	MRR	HITS@1	HITS@3	HITS@5	HITS@10	HITS@20
Text dim 2048	48.628	0.142	0.284	0.366	0.494	0.622
Text dim 4092	48.145	0.138	0.289	0.375	0.503	0.623

Conclusions: We can see that we almost didn't lose any information by reducing the size of the text representation. Although these numbers seem far from optimal, manual evaluation shows that the model is very able to retrieve images related to a given caption and consistently provides related images. Regarding our test setup, in a majority of cases the model ranks the correct image at the top, which is already quite an achievement. It however has some difficulties with more ambiguous images and with abstract words, which results in some images ranking badly and tipping the MRR to the wrong side. All things considered results are pretty satisfying. Manually trying out the image retrieval (running the *eval.py script* lets you write captions) strengthens this opinion.