Visual Question Answering Project

Artificial Neural Network and Deep Learning

A Deep Learning model for Visual Question Answering

Andrea Boselli, Carlo Ghiglione, Leonardo Perelli

25-01-2021

Contents

1 Introduction

The aim of the project is to develop a Visual Question Answering model able to answer questions about the content of images. The provided dataset has the following composition:

1. 29333 images;
2. 58832 training and 6372 test questions associated to a image;
3. 58 possible answers for each question belonging to 3 possible categories: 'yes/no' answers, 'counting' answers (from 0 to 5) and 'other' (e.g., colors, objects, ecc.)

Analysing the provided dataset, we observed two main issues:

1. the answers classes are very unbalanced;
2. the majority of the questions is quite short but some are very long.

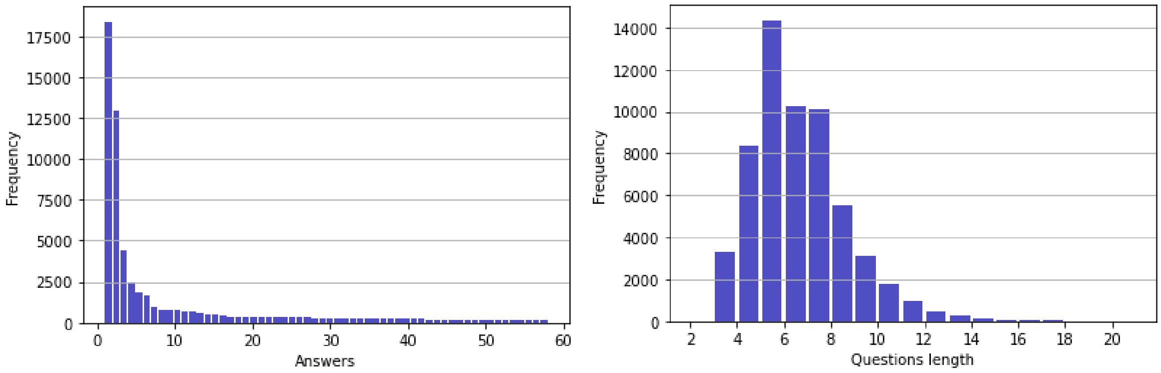

Figure 1: Answers (sx) and questions length (dx) frequency

2 First Model

2.1 Architecture

We started with a very simple architecture. For the image processing, we implemented a CNN fed with 256x256 pixels input images having the following structure:

```
-----------------------------------------------------------------
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1.1 (Conv2D)          (None, 256, 256, 32)      896
-----------------------------------------------------------------
conv2d_1.2 (Conv2D)          (None, 254, 254, 32)      9248
-----------------------------------------------------------------
max_pooling2d_1 (MaxPooling2 (None, 127, 127, 32)      0
-----------------------------------------------------------------
conv2d_2.1 (Conv2D)          (None, 127, 127, 64)      18496
-----------------------------------------------------------------
conv2d_2.2 (Conv2D)          (None, 125, 125, 64)      36928
-----------------------------------------------------------------
max_pooling2d_2 (MaxPooling2 (None, 62, 62, 64)        0
-----------------------------------------------------------------
conv2d_3.1 (Conv2D)          (None, 62, 62, 128)       73856
-----------------------------------------------------------------
conv2d_3.2 (Conv2D)          (None, 60, 60, 128)       147584
-----------------------------------------------------------------
max_pooling2d_3 (MaxPooling2 (None, 30, 30, 128)       0
-----------------------------------------------------------------
conv2d_4.1 (Conv2D)          (None, 30, 30, 256)       295168
-----------------------------------------------------------------
conv2d_4.2 (Conv2D)          (None, 28, 28, 256)       590080
-----------------------------------------------------------------
conv2d_4.3 (Conv2D)          (None, 26, 26, 256)       590080
-----------------------------------------------------------------
max_pooling2d_4 (MaxPooling2 (None, 13, 13, 256)       0
-----------------------------------------------------------------
flatten_1 (Flatten)          (None, 43264)             0
-----------------------------------------------------------------
dense_1 (Dense)              (None, 128)               5537920
=================================================================
Total params: 7,300,256
-----------------------------------------------------------------
```

For the sentence processing, we implemented an encoder of this kind:

```
Model: "Encoder Model"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 21)]              0

_________________________________________________________________
embedding_1 (Embedding)      (None, 21, 32)            148544

_________________________________________________________________
lstm_1 (LSTM)                [(None, 128), (None, 128) 82432
=================================================================
Total params: 230,976

_________________________________________________________________
```

The outputs of the two parts are concatenated and then pass through a dense layer of 256 units and the softmax for the final classification.
We always used *ReLU* activation function, *CategoricalCrossEntropy* function, the *Adam* optimizer, a learning rate of 10^3, batch size equal to 1 and validation split of 20%.

2.2 First results and considerations

The resulting model has 7 million parameters and, trained with the first 1500 instances, reached 0.48 of validation accuracy.
The main problems were:

1. overfitting issues quite early;
2. lack of improvement quite early;
3. the large majority of the parameters is provided by the CNN, so question processing is under-represented.

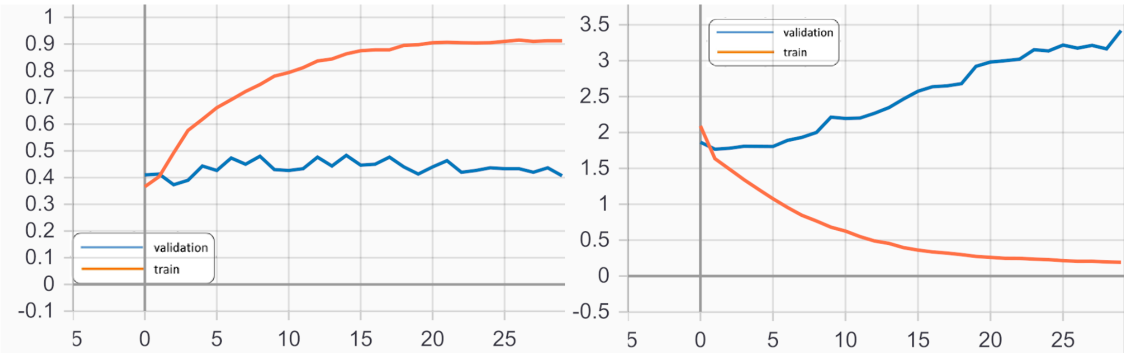

Figure 2: Accuracy (sx) and loss (dx) of the model

2.3 Implemented improvements

To address the overfitting issues, we passed from 1500 input instances to 24000 in the training. To deal with our limited memory capacity, we had to reduce the size of input images to 64x64 pixels. To augment the influence of the input sentences in the classification, the embedding size was set to 128.

In order to deal with class imbalance, we weighted the answers inversely proportionally to their cardinality in the training dataset. Moreover, since we noticed that 99% of the training questions is 13 words long maximum, we decided that this was a sufficient size for the input of the embedding.

These improvements led to a model with only 2.5 million parameters that scored 0.55 of validation accuracy, obtaining almost the same result on the test dataset. We can still see that there are overfitting issues and the model stops improving quite early.

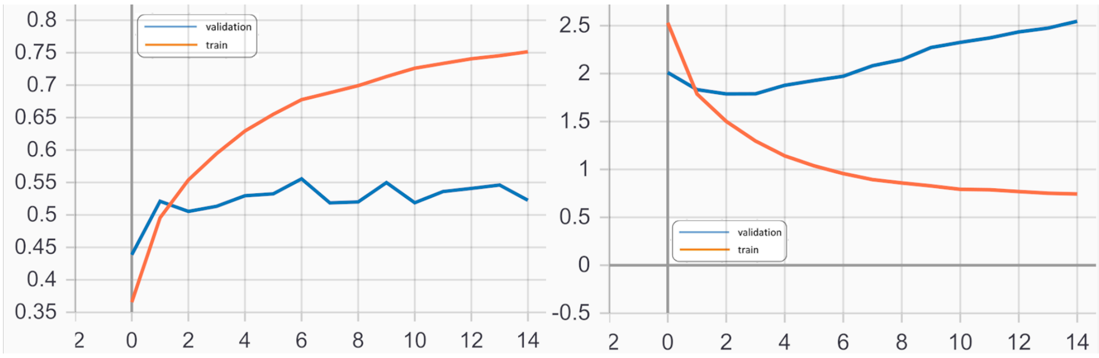

Figure 3: Accuracy (sx) and loss (dx) of the model

3 VGG16 Model

We decided afterwards to change the convolutional part, implementing transfer learning with VGG16 in order to rely on a strong feature extractor without the constraint of large input images.

3.1 Implementation

We tuned the embedding size in the recurrent part with values 16, 64 and 128: we noticed that the optimal one was 64, scoring almost 60% of test accuracy, and that further improvements were not beneficial.

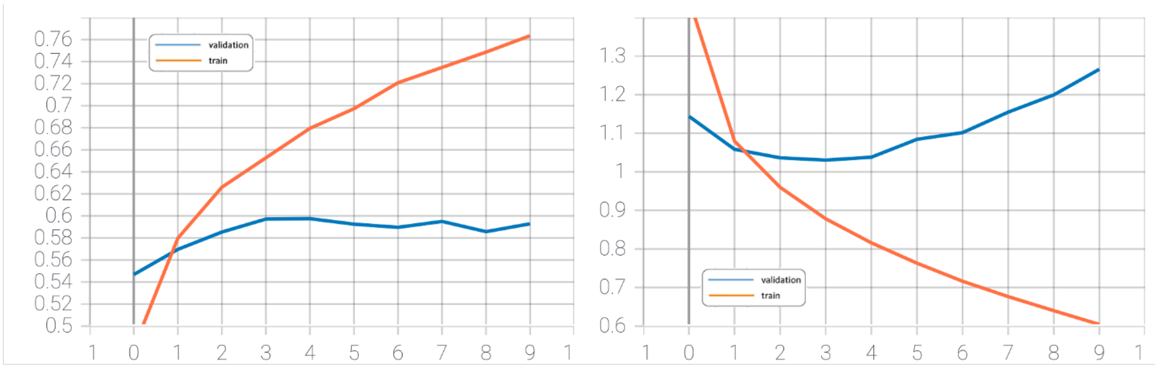

Figure 4: Accuracy (sx) and loss (dx) of the model

3.2 Considerations

We even tried to perform fine tuning, freezing the first 15 layers of VGG16, but this resulted only in a huge growth of training time, with no improvements in performance. All these attempts made clear that the actual bottleneck of the network was the recurrent part, so we focused on that.

4 Final Model

4.1 Training improvements

To address our ubiquitous overfitting issues, we used all the data available iterating the training over chunks of 2000 instances for 10 epochs each. As we can see in the graphs, this had remarkable benefits.

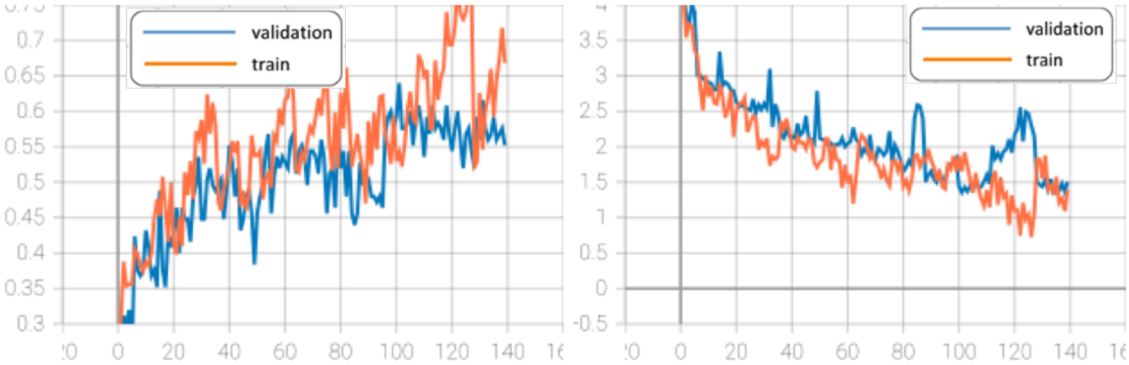

Figure 5: Accuracy (sx) and loss (dx) of the model

4.2 Introducing Attention Mechanism

Trying to better integrate the vision and the text part, we implemented an attention-inspired mechanism, doing a pointwise multiplication of the outputs of both parts instead of concatenating them.

In particular, the LSTM output is used as key to select the important features of an image by multiplying it element-wise with the dense layer output of the vision model and then doing the final classification. This should enable to select certain features of the image based on the encoding of the question.

To help to recognize simple features from the image related to the question, we implemented a sort of skip connections adding a GAP layer after each max pooling directly concatenated to the final output of the vision model.

The model reached comparable results of the previous one without using a pre-trained network.

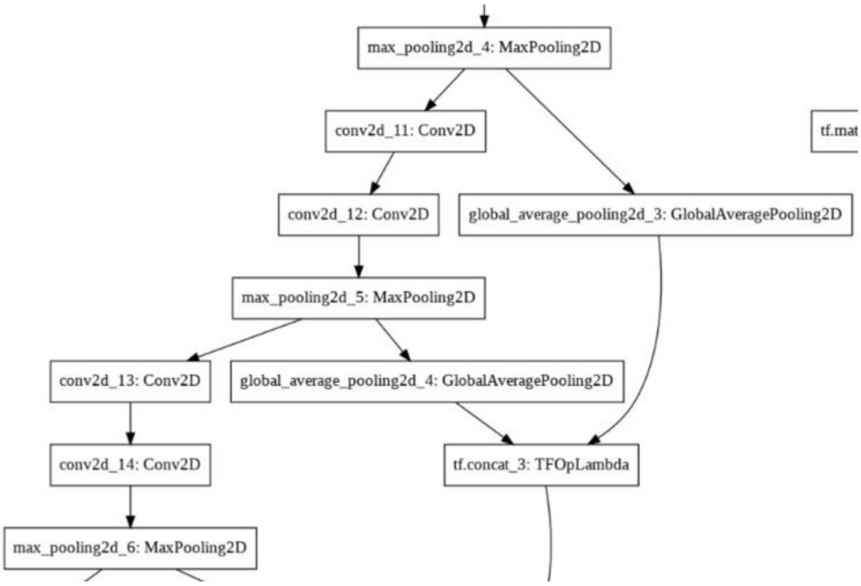
max_pooling2d_4: MaxPooling2D
conv2d_11: Conv2D
tf.mat
conv2d_12: Conv2D
global_average_pooling2d_3: GlobalAveragePooling2D
max_pooling2d_5: MaxPooling2D
conv2d_13: Conv2D
global_average_pooling2d_4: GlobalAveragePooling2D
conv2d_14: Conv2D
tf.concat_3: TFOpLambda
max_pooling2d_6: MaxPooling2D

Figure 6: Graph of the implemented architecture

5 Conclusions

As further developments, we would:

1. upgrade the attention mechanism;
2. develop a less memory intensive pipeline.

We are satisfied by our results and that we had the opportunity to experiment Deep Learning techniques in a real-world framework.