VISUAL QA

SLIDE 1: TITOLO
This project consists in the development of architectures that address the visual question answering (VQA) task,

SLIDE 2: TASK
It is a semantic task that aims at answering questions based on an image. Given an image and a natural language question about the image, the task is to provide an accurate natural language answer. For many questions, a simple “yes” or “no” response is sufficient. However, other questions may require a short phrase.
It can be treated both as a multiple choice task or an open-ended answering task. We choose for the latter (since for part of the dataset (Abstract scenes) the data for multiple choice weren’t available).
This task requires to combine Computer Vision (CV), Natural Language Processing (NLP), and Knowledge Representation & Reasoning (KR).

SLIDE 3: PROPOSED APPROACHES
We’ll see 4 different approaches to this task: random baseline and prior yes that are trivial baselines; a more consistent and used approach that exploits CNN and LSTM; and finally a state-of-the-art generative approach.

SLIDE 4: DATASET
The dataset used in the project is Visual Q&A v2.0.
It contains images both from COCO and Abstract Scenes. MS COCO dataset has real images characterized by multiple objects and rich contextual information.
Since the VQA task with real images requires the use of complex and often noisy visual recognizers. To attract researchers interested in exploring the high-level reasoning required for VQA, but not the low-level vision tasks, it was implemented an abstract scenes dataset that we also used in our project.

For each image or scene at least three questions were collected. Each question was answered by ten subjects along with their confidence in answering (Yes if they were sure of the answer they gave, Maybe or No). Also, there are 3 plausible (but likely incorrect) answers per question.

SLIDE 5: PREPROCESSING
This task is very complex and requires a lot of resources. So, in order to lighten up the computational burden and preserve the RAM for training, we made some adjustments. First we build a dictionary where each image_filename was associated with its image_id. It was necessary since we don’t extract the images from zips all at once, but one batch at the time during training. Therefore the dictionary allowed to access directly the image_filename, that is the one we used to identify an image, since the image_id wasn’t unique between coco and abstract scenes.
Also we decided to consider only a portion of the dataset, specifically a tenth. Samples were randomly selected from the whole dataset.
Finally, we performed some data preprocessing and save it in a file, in order to directly retrieve them for the following dataset creation. The data saved on the file have the shown

format. For both approaches 1 and 2, to each question, identified through the question id, is associated the image_filename, through the dictionary we talked about. ‘quests’ represents the question in words, which was processed by removing punctuation. Regarding the answers, things differentiate. In the case of approach one, for each question, we return a vector of indexes. Each index represents the corresponding answer in the dictionary of the K most frequent answers. The answers not belonging to the most frequent ones are discarded. The samples with no answers belonging to the most frequent ones are completely discarded. For approach 2, ‘ann-vector’ is the most frequent answer for that question. Each word is represented through the correspondent index in the vocabulary of all the words in questions and answers from the training set. If a word is not present, it is associated to the index given to the '[UNK]' token.
Finally, regarding approach 1, we also need the confidence score for each answer. We associated 1 if confidence is Yes, 0.5 for Maybe, 0.0 for No.

For the trivial baselines, each sample has the same format as in approach 2. The only difference is that the answers aren’t processed in any way, since for the baselines WE generate the answer by ourselves.

SLIDE 6: DATASET CREATION
So, saved on the file, the prepared data are loaded back and, during dataset creation, we perform some further processing, before batching and splitting the validation set in test and val. For approach 1, questions are embedded through Glove. For approach 2, questions are encoded using LXMERT tokenizer and padded to the same length. (perchè ci sono domande con più parole e domande con meno)

SLIDE 7: EVALUATION
Multiple different answers may be correct for a certain question. Human subjects may also disagree on the “correct” answer.
In order to have a metric robust to inter-human variability in phrasing the answers, for the open-ended task, the generated answers are evaluated using the following accuracy metric: FORMULA so that an answer is deemed 100% accurate if at least 3 workers provided that exact answer.

Before performing the evaluation, all answers are processed mading them lowercase, converting numbers to digits and removing punctuation and articles.

SLIDE 8: Trivial baselines: Random and Prior yes
As trivial baselines, we have chosen to implement
1) the Random baseline, where the output answer is taken randomly from the K (1000) most frequent answers
2) the Prior yes baseline. Here for every sample, the prediction is yes. From this baseline we expected better results since most questions are answered with a 'yes' or a 'no' and, among the 'yes'/'no' questions, with a bias towards 'yes'.

SLIDE 9: Trivial baselines: evaluation
1) We can see how the random baseline leads to very poor results, due to the variety of possible answers. In general, the accuracies for each answer type (yes/no, other, number) are all below 0.3%. By looking at the accuracies per question type, instead, just for

questions starting with "Are there..." and "How many people are there..." the accuracy values go above 1%. The overall accuracy is just 0.14%.

2) Differently, in the case of Prior yes: As expected, the overall accuracy reached is higher, a almost 25%. Clearly, even in this case, the accuracy for answers of type “other” and “number” are very low, while the accuracy for questions with “yes/no” answers is 63.43% since, as specified before, among the questions with yes/no answers there is a bias towards yes.

SLIDE 10→15: Approach 1 CNN + LSTM

(10) The first non trivial approach developed consists in a 2-channel vision (image) + language (question) model that outputs the most probable answer across the K most frequent ones in the training dataset.

(11) The vision channel is represented by pre-trained VGG, from which we extract the activation values of the last hidden layer, that are l2 normalized and are used as 4096-dim image embeddings.

(12) The image embedding is then transformed to 1024-dim by a fully connected layer with tanh non-linearity.

(13) The language channel, instead, is represented by a LSTM with two hidden layers. Differently from the reference paper, in which the input vocabulary to the embedding layer is made up by all the question words seen in the training dataset, we have directly used pre-trained Glove embeddings (in particular, 50-dim and 300-dim embeddings). The last cell state and the last hidden state representations from each of the two hidden layers are concatenated to obtain the question embedding, that is 2048-dimensional.

(14) Again, we use a linear layer with tanh non-linearity to transform it to a 1024-dim embedding.

(15) The image and question embeddings, which have the same size, are combined through element-wise multiplication. The combined embedding is passed to an MLP with a linear layer of 1000 hidden units followed by tanh non-linearity and 0.5 dropout, and a final layer with K units followed by the softmax activation function, to obtain a probability distribution over the K most frequent answers. In the global variables section, we defined K=1000.

SLIDE 16: Approach 1: loss

CrossEntropy loss function has been used.

None of the papers and repositories that implement the CNN+LSTM approach use the preferences that are provided together with the answers. They consider as ground truth answer for a given question the most frequent one. Therefore, if the model predicts one of the other possible answers, it will be considered as completely wrong. For this reason, in addition to reproducing the training with this standard loss, we propose a novel loss leveraging preferences/confidences.

In particular, we define the ground truth for a certain question as a vector of K elements where the elements corresponding to a possible answer are set to 0, 0.5, or 1, depending on whether they have preference 'no', 'maybe', or 'yes', respectively. In this way, we consider all

possible answers, but we take into account the fact that some of them may be potentially wrong, by weighting them differently.

SLIDE 17: Approach 1: evaluation
(Standard loss) On the left there are the results obtained with the model trained considering as label of each sample the most frequent answer among the possible ones, without considering preferences and using 300-dimensional GloVe embeddings.
Because of the limited resources at our disposal (RAM limit), we managed to train the model for just 3 epochs. This is clearly not enough to obtain satisfying results. However, even with such a short training, we managed to overcome the performances obtained with both trivial baselines. In fact, the overall accuracy is 25.79%.

(Novel loss) On the right, the model has been trained using preferences in the loss computation and 300-dimensional GloVe embeddings in the embedding layer.
As we can see, despite training the model for just 3 epochs, using preferences seems to improve the training, leading to slightly higher overall accuracy of 26.45%. However, due to the impossibility of training the model for a sufficiently large number of epochs, we do not know how the training will evolve and if it will actually lead to better results.

We have tried using smaller embeddings to see whether we managed to train the model for more epochs, thus reaching better results. However, the training was still very heavy and did not allow us to reach higher performances. For this reason, we leave here just the results obtained using the 300-dimensional embeddings.

SLIDE 18: Approach 1: Results
In this slide we have reported an example of a correct answer computed using the novel loss.

SLIDE 19: Approach 2: Generative LXMERT
This paper extends LXMERT, which is treated as a multimodal encoder, with a decoder that allows to generate text.

SLIDE 20: Approach 2: ENCODER
The encoder is an LXMERT model, which has been used pretrained and hasn't been further trained, mainly because of the limited resources available.

SLIDE 21→23: Encoder: LXMERT model
(21) Entering in the details of the encoder, it is a pre-trained vision-and-language cross-modality framework. In fact, the LXMERT model architecture has three components. The first one is an object-relationship encoder (with a faster RCNN backbone), that receives in input the Region of Interest (RoI) features and bounding boxes extracted by a FasterRCNN and processes them using a BERT-style encoder. Also Faster RCNN has been used pretrained on COCO images dataset and has been frozen during training. (Faster RCNN has been executed block by block, in order to be able to extract the features after the last pooling layer before predicting the box and class for each roi proposal).

(22) LXMERT language encoder receives as input the input_ids, token_type_ids and attention_mask obtained by applying LxmertTokenizer to questions. Also the latter has been used as pre-trained, without performing any additional training epoch.
(The language encoder is essentially the same as a BERT style self-attention based language encoder).

(23) A cross-modality encoder allows language and visual embeddings to attend to each other, fusing information together for downstream visual-language related tasks.

SLIDE 24: Approach 2: BRIDGING LAYER
We concatenate language and visual hidden states produced by LXMERT in order to have a unique hidden state representation. This is passed to a bridging layer, which transforms it into a form amenable for our decoder.

SLIDE 25: Approach 2: DECODER
The decoder consists of a 3-layer GRU RNN with a 300-dimensional word embedding layer and final vocab layer of dimension ||V||, where ||V|| is the size of the vocabulary that contains all the different words that appear in questions and answers of the training set. Initially, the decoder takes as input the [CLS] token and the representation given as output by the bridging layer (hid_gru_0). At each timestep, the GRU outputs two vectors: out_gru, which contains the final features obtained from the last layer of the GRU, and hid_gru which corresponds to the final hidden representation. The former is given to the vocabulary layer and the softmax is applied to generate one token. This output token, together with hid_gru are given as input to the GRU at the next time.

SLIDE 26: Approach 2: TEACHER FORCING
Once a token has been generated by the decoder, with probability r we set the input token for the next decoding step equal to the ground truth one. Teacher forcing has been implemented since, mainly in the first epochs of training, the decoder won't be able to predict correctly the next token. Therefore at next iteration, the GRU would take in input a wrong token and in this way would never learn to correctly predict.

SLIDE 27: Approach 2: modification and evaluation
Actually, we have tried making some modifications to improve the training and avoid this behavior, but because of the impossibility of training the model for a larger number of epochs, it is impossible to see if there is an actual improvement or if things change from epoch to epoch. Adding gradient clipping, since the fact that the model always predicts the same token may indicate that there is an exploding gradient. However, even if this made the model predict random words for a longer time, towards the end the model always predicts the [PAD] token.

- Weighting the [PAD] token less when computing the loss. In this way, the loss tries to correct the prediction of the [PAD] token less, so that the model is not tempted to always predict the [PAD] token, which is the most probable one.
- Testing different optimizers:
    - Adam led to having all [PAD] tokens very soon during the training.
    - SGD made the model always predict MAX_ANSW_LEN random tokens.
    - AdamW allowed to have the model predict all [PAD] tokens later during training, but the resulting behavior is exactly the same obtained with Adam.

SLIDE 28: Possible improvements (with more resources)
If we had more resources, we would have made of course some modifications in order to improve the results:

- ★ Train for more epochs
- ★ Use the entire dataset during training (at the moment we use only 10%)
- ★ Increase the batch size: now we use accumulate_grad_batches because the batches are made of 2 samples.
- ★ Fine-tune LXMERT and the embedding layers instead of using pre-trained ones
- ★ Try to combine word2vec and GloVe embeddings
- ★ Use teacher forcing with a higher probability in the first epochs and stabilize the 0.5 probability after a certain number of epochs.

LASCIATELE GRAZIE!
Each question in the question archives comes with the image_id of the image it is referred to. Since we do not extract all the images from at once, but we extract them batch by batch, we need to be able to find the image we are looking for using its filename. To this aim, we build a dictionary that maps each image_id to the corresponding filename. We save this dictionary in a json file so that we do not need to compute it every time we run the code.

In approach 1, we perform the classification over the K (set to 1000) most frequent answers. Here, we are computing a dictionary with all the possible answers in the training set and the number of times each of them appears. In the validation and test sets, we do not consider any answer that is not among the K most frequent ones. If a question has all answers that are not among these K, the question is not considered. For further details see the implementation of approach 1 below.

For approach 2, we need to build a dictionary made up of all the different words that appear in the questions and answers in the training dataset. This will be used in the vocabulary layer within the network.

In order to not overload the RAM when creating the dataset and starting the training, we perform some data preprocessing and prepare data for the actual creation of the dataset which will be done later. The preprocessed dataset is then saved, so that we will just have to retrieve data from there.

PREPARE_DATA (quella che salva su un file e poi è passata alla dataset creation)
-randomly consider just a portion of the total samples (10%)
-process the question: remove punctuation
-associate to each question the corresponding image filename
-associate 1, 0.5, 0 to the weights
-associate to each question the list of answers and confidences
 -in the case of approach1, only the answers in the 1000 and pad. If no answer in 1000, exclude the sample. Replace the answer with the corresponding index
-approach2: for each question, we take the most frequent answer, process it and split it into words. We encode the words in the answer using the dictionary with all the words in the questions and answers of the training set. If a word is not present, it is associated to the index given to the '[UNK]' token.
-return *img_filenames, qst_ids, quests, preferences, ann_vectors*

DATASET CREATION:
approach 1: embed question with glove
approach 2: pass question to lxmert tokenizer, extracting input_ids, attention_mask e token_type_ids.
Pad all the encoded answers to the same length
return `image_filename, question_id, question, preferences, labels`
return `image_filename, question_id, question, labels, question_input_ids, question_attention_mask, question_token_type_ids`