Deep Learning – VQA Report

Pre-Process:

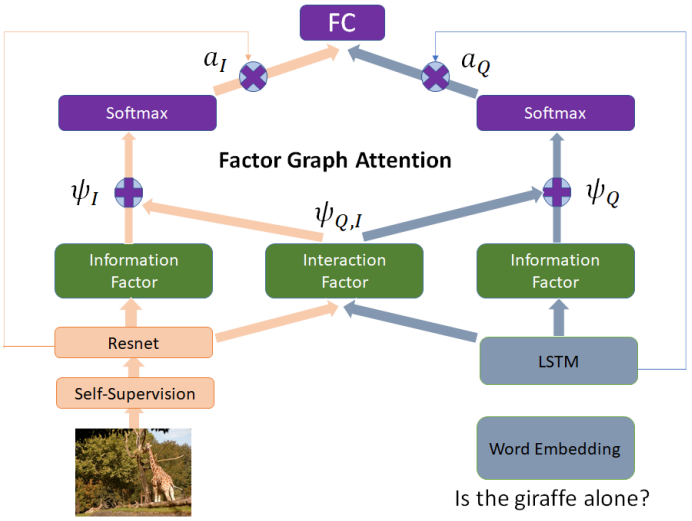
We used the pre-process code that was supplied in the task.
In addition, we applied the following actions:

- transformation to the images by resizing them to a smaller shape.
- building a dataset: for each image in the training data, we found its related answers (and their related questions), following the understanding that once those relevant answers and questions found, we can stop seeking for more, as they appear sequentially in the data.
- building a questions word dictionary which attaches a unique index for every unique word in the questions that appear in the training dataset. We also added an <UNK> key, representing those words that will be new to our model while validating, as well as a 'PAD' key, in order to allow a padding for a fixed question length.

Having built VQA preprocessed datasets for training and validation, we dumped them as pickle files, which will be opened in the model section.

hyper-parameters:

- min answer occurence = 20
- resized image shape = 224 X 224

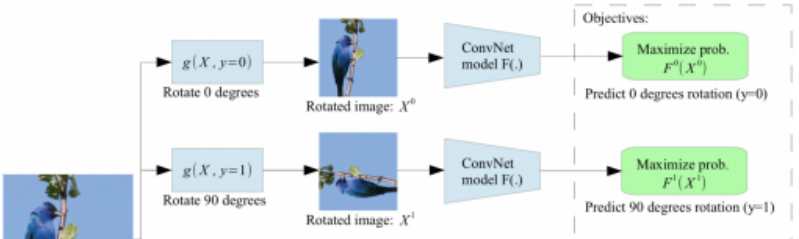


Model Description:

In order to handle the multimodal, visual question answering problem, we created a neural network model that gets image and question as input and produces probabilities for each answer in the VQA dataset.
As can be seen above in the network architecture illustration, the network we built consists of two main building blocks: CNN and LSTM.
As we learned in class, each of these building blocks is able to learn many kinds of data features of the inputs they get, thus creates an encoded form of the images and questions that pass through the entire net.

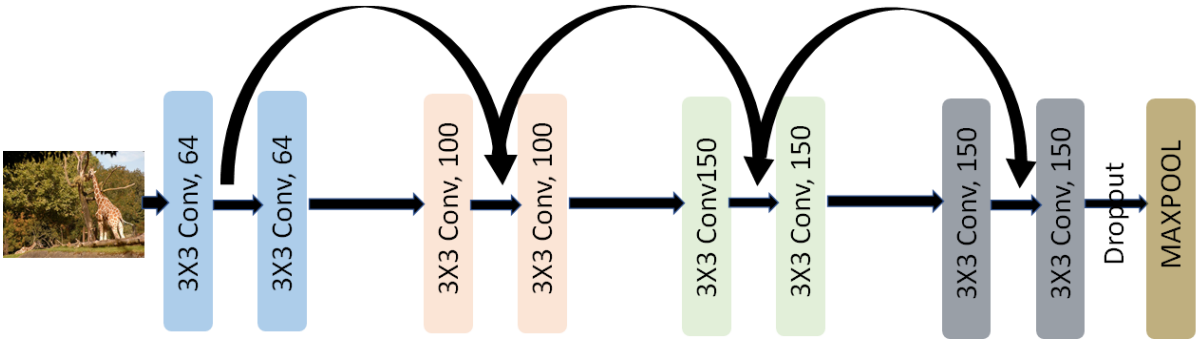
CNN and Self supervised learning:

As for the CNN model for encoding the image we used a model made of 2-dimensional convolutions, BatchNorm, activations, dropout, max pooling and skip connections, similarly to the ResNet model, using the trick of "Residual block", which allows the gradient information of the top layers bypass layers upfront.
The shape of the output of the model is [batch_size, 7, 7, 150] - it means we have 49 regions each represented by a vector of length of 150.
To use some sort of pretrained model we first trained our CNN model on a self-supervised task: we rotated each image with one of the following angles: 0, 90, 180, 270 and let the model predict the angle. As learned in class, in order to



identify the same image with different rotations, the model must learn to recognize high level object parts, thus adding another aspect of image learning to the network.

CNN Illustration:



Questions: For questions encoding, we used the bi-directional LSTM, where each word was eventually represented by a vector with length 200.
This building block is in charge of learning the inter-dependencies between the words in each question. Those words are accepted as an input of the LSTMs, after applying a word embedding using the built-in embedding function of the torch.nn library. This embedding became possible after we applied a padding to the questions tokens, according to the maximum length of a question (which was 23).
These LSTMs learn to classify an answer to each question, using the training data.
hyper-parameters:

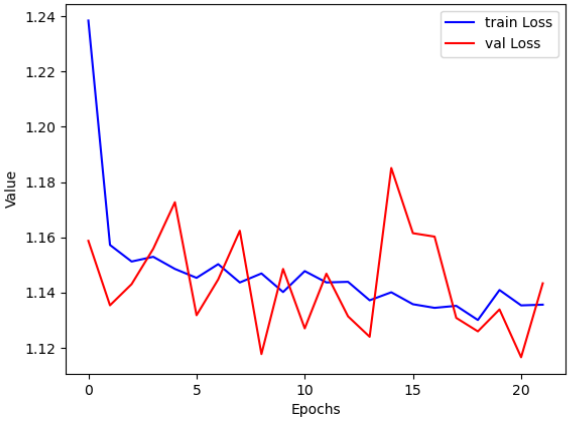
- word embedding dimension = 75
- lstm hidden dimension = 100

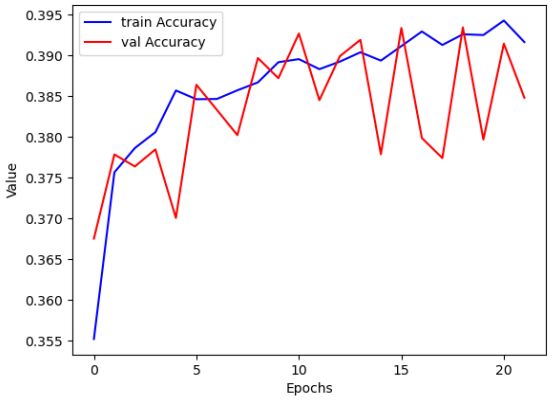
The VQA model we implemented applies an attention belief factorization which consists of three trainable scalers for each of the images and questions: entity information, self message and question-image message. For each object in the model (image or question), we calculated the amount of attention that should be given to each of their units (image-region or word, respectively). Then, their attend representation could be calculated in order to classify the input by a final fully-connected layer.

Training:
In the training part we used a cross-entropy loss with an addition of manipulation using the labels scores: we weighed the loss of each ‘true’ label by its score so that labels with higher scores will affect the model more.
The optimizer we chose is Adam from torch.optim library.

hyper-parameters:

- dropout = 0.3
- batch size = 25
- number of epochs = 21
- learning rate = 0.001
- Adam optimizer: betas = (0.9,0.9), weight_decay = 1e-5
- region embedded dimension = 150
- self interaction dimension = 150
- number of regions: 49 (7X7)
- interaction dimension = 100





Attempts:

- we tried to use Resnets as Resnet34 and Resnet18, but this attempt highly enlarged the number of parameters and training time, with no strong evidence for accuracy improvement.
- we tried to use the transformation RandomHorizontalFlip because we thought it would serve as an augmentation and will lead to better generalization, but it turned out to hurt questions classifying with the answers ‘right’/’left’.

Conclusions:

- While using encoders in such neural nets, one should keep the model size reasonable, so that the number of parameters would not be huge, because it will not necessarily improve its performance, but will definitely slow down the training process.
- One should combine augmentation techniques in deep learning, only after taking into consideration the effect it will have on the classifying task. In our case, flipping the images horizontally, caused our model to be mistaken on questions that ask about the location of an object in the image.
- While observing the output of our model, we figured out that there is a strong effect of the labels frequency in the training data, on their presence in the successful predictions (yes/no labels were very dominant). Therefore, we concluded that label weighting through balancing, could be helpful in terms of converging.

Best val accuracy result: 39.5