Visual Question Answering

Prof. Anna Rohrbach,
Prof. Marcus Rohrbach,
Jonas Grebe,
Marcel Klemt

TECHNISCHE UNIVERSITÄT DARMSTADT

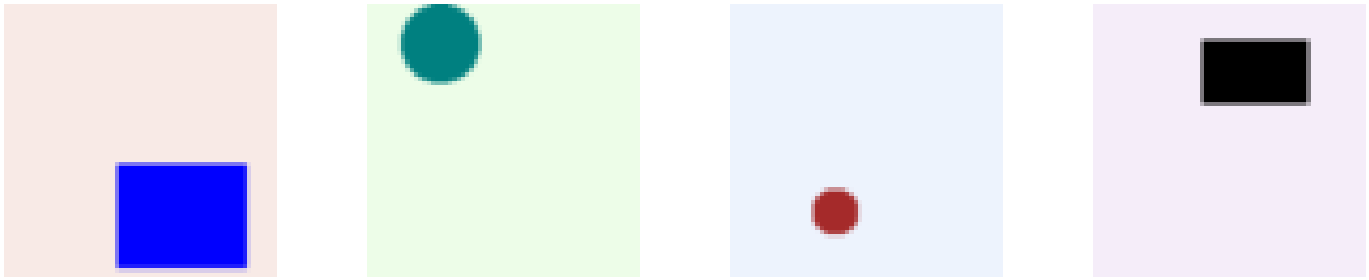
Summer Semester 2024 – Multimodal Artificial Intelligence
Deadline Wednesday 29th May, 2024
Sheet 2

Visual Question Answering

In this exercise, we are starting of multiple modalities! You will implement a simple PyTorch model for **Visual Question Answering (VQA)**. In this task, the model is supposed to predict the answer $a \in \mathcal{A}$ to an image-question pair (x, q). For the purpose of this exercise, we will be using the easy-VQA toy dataset, which comes with $|\mathcal{A}| = 13$ total possible answers (e.g. circle, green, red, yes, ...), $5,000$ images and roughly $50,000$ questions.

In our scenario, we model the VQA task as a simple 13-class classification problem. Every possible answer is treated as one class.

We have created a Jupyter Notebook for this exercise that you can find on our Multimodal AI Lab Github page and import into your Google Colab.



(a) What is the blue shape? → rectangle

(b) Is there a triangle in the image? → no

(c) What is the color of the shape? → red

(d) Is there a teal shape in the image? → no

Figure 1: Training samples from the easy-VQA dataset

Task 2.1: Preprocess the questions

We have our data (images, questions, and answers) available. But the questions are still in the wrong format. Our model can not handle words in a textual form. Therefore, please preprocess each question by removing punctuation (given in the notebook), tokenizing it, removing the stop words, and padding each question to a fixed length. And then really transform it to the embedding representation provided by *word2vec*. The final output shape for the training dataset should be $[38575, 7, 300]$.

Task 2.2: Create your own custom VQA dataset

After embedding the questions, we finished preprocessing. Thus, we are ready to create our custom VQA dataset (help on how to generate a custom PyTorch dataset can be found here).

```
class CustomVQADataset(torch.utils.data.Dataset):

    def __init__(self, img_paths, qs_embs, ans_encodings, qs_img_ids, img_transform):
        # initialize the relevant variables

    def __getitem__(self, idx: int):
        # return the elements of the dataset needed for training the VQA model

    def __len__(self):
        # return the length of the entire dataset
```

We can return the question and answer easily as we preprocessed them. However, we have to apply the given image transformations to the image belonging to the question-answer pair.

Task 2.3: Create the VQA model

Our multimodal VQA model consists of an Image Encoder, a Text Encoder, and a part that combines both modalities. We provided the Text Encoder part, but the Image Encoder's forward pass and the forward pass of combining both modalities are missing.

2.3a) Forward pass of the Image Encoder

Please implement it by passing the input image through a convolution with the ReLU activation function followed by a MaxPooling layer three times. Finally, flatten the input (but keep the batch dimension) and apply the fully connected layer followed again by ReLU.

```
class ImgEncoder(torch.nn.Module):

    def __init__(self, emb_size=32)
        super(ImgEncoder).__init__()
        ...

    def forward(self, img):
        # Your Python code here
```

2.3b) Forward pass of combining both modalities

Please implement the forward pass of predicting the answer given the image and question. Therefore, we need to encode the image and the question via the Image and Text Encoder, respectively. After that, both encodings need to be combined via element-wise multiplication, followed by ReLU, the given dropout layer, and finally, the fully connected layer.

```
class VQA(torch.nn.Module):
    def __init__(self, num_answers, emb_size=32):
        super(VQA, self).__init__()
        ...

    def forward(self, img, q):
        # Your code here
```

Task 2.4: Implement the Training and Validation Loop

Parts of the training loop are already given, e.g., how we monitor our loss and accuracy values this time, but the main part is missing. Please implement the forward and backward pass and the required optimizer operations.

```
def train_loop(model, optimizer, criterion, train_loader):
    model.train()
    total_loss, total = 0, 1e-6
    num_correct = 0

    for img, qs, ans in trainloader:
        img, qs, ans =  img.to(device), qs.to(device), ans.to(device, dtype=torch.float32)
        # Your code here
```

Second, please implement the validation loop and return the average loss and accuracy (remember: we are not training on the validation set, thus, no gradient calculation).

```
def validate_loop(model, criterion, valid_loader):
    model.eval()
    total_loss, total = 0, 1e-6
    num_correct = 0
    # Your code here
```

Task 2.5: Self-Attention

After we checked out the Recurrent-Neural-Network (RNN) approach via LSTMs, it is time to take a look at Transformers. As the whole setup can be a bit complicated, we set a framework for you. In this setup, you "just" need to implement the scaled dot-product attention as shown in the lecture and Figure 2(left).

```
class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, num_heads):
        super(MultiHeadAttention, self).__init__()
        ...

    def scaled_dot_product_attention(self, Q, K, V):
        # Your code here
```

Task 2.6: Assemble the Transformer Encoder

After implementing the attention, it all needs to be put together. Please implement the forward pass of the Encoder Layer as introduced in the lecture and shown in Figure 2(right). Additionally, add dropout to the output of the attention layer and to the output of the feed-forward network.

```
class EncoderLayer(nn.Module):
    def __init__(self, d_model, num_heads, d_ff, dropout):
        super(EncoderLayer, self).__init__()
        ...

    def forward(self, x):
        # Your code here
```

Remember that combining both modalities is missing, just take the same part of code you implemented for the LSTM based VQA model.

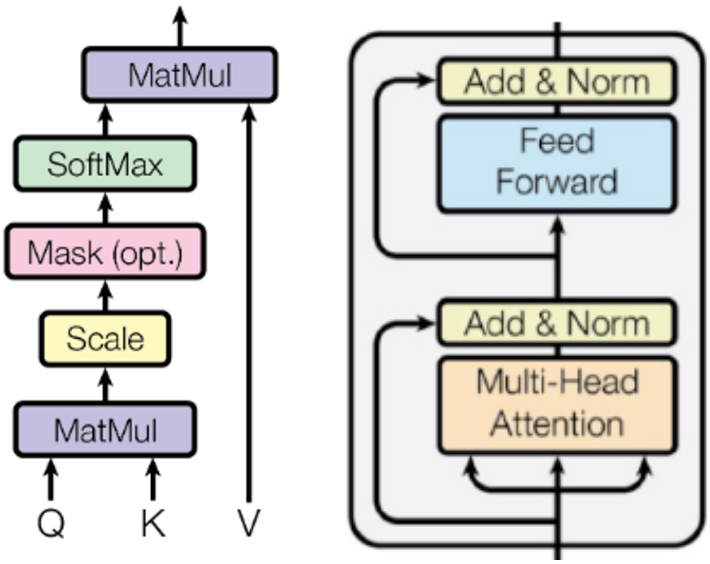


Figure 2: Scaled dot-product attention and the transformer encoder

Task 2.7: Bonus Exercise

Now that your VQA model is working, try to confuse it by, for example:

1. randomizing the question index for each sample,
2. zeroing out all pixels in the images,
3. or permuting the word order in the questions

How does it affect the model performance? What does it say about the data?

Can you verify your assumptions?