Visual question answering

Ahmed Ashraf Abdelkarim - Amr Abdelsamee Yousef - Hazem Abdallah Mohammed

June 25, 2023

Contents

0.1 Introduction

Visual Question Answering (VQA) is the task of answering open-ended questions based on an image. VQA has many applications: Medical VQA, Education purposes, for surveillance and numerous other applications. In this assignment we

will use Viz-Wiz data-set, this data-set was constructed to train models to help visually impaired people. In the words of creators of Viz-Wiz: "we introduce the visual question answering (VQA) data-set coming from this population, which we call Viz-Wiz-VQA. It originates from a natural visual question answering setting where blind people each took an image and recorded a spoken question about it, together with 10 crowd sourced answers per visual question.

0.2 Data

0.2.1 What is Viz-Wiz?

The Viz-Wiz data-set is a widely-used benchmark data-set for visual question answering (VQA) tasks. It is designed to evaluate the ability of machine learning models to answer questions about visual content, such as images and videos.

The dataset was created by researchers at the University of Rochester in collaboration with the VizWiz project, which aims to improve access to visual information for people with visual impairments. The dataset contains over 31,000 images, each with a corresponding question and answer pair collected from real-world scenarios.

The Viz-Wiz data-set is unique in that it contains questions and answers that are generated by non-expert users, making it more representative of the types of questions that people might ask in real-world situations. The data-set also includes additional information such as image captions and textual context, which can help models better understand the visual content and answer questions more accurately.

The VizWiz dataset has been used in a variety of research studies and competitions, and has led to significant advances in VQA technology. It remains an important resource for researchers and developers working in this field.

0.2.2 How to load the data-set?

To load data into Google Colab using Kaggle API, follow these steps:

- Download the Kaggle API credentials from your Kaggle account and upload them to your Google Drive.
- Install the Kaggle API in your Google Colab notebook.
- Use the Kaggle API to download data-sets directly to your notebook.
- Unzip the data

That's it! You can now use the downloaded data in your machine learning models on Google Colab.

0.2.3 Selecting the best answer

As stated above each question is supplied with 10 answers along with an answer confidence for each answer. We encoded the confidence as follows:

- YES maps to 3
- Maybe maps to 2
- No maps to 1

Then we repeat each answer a number of instances equivalent to the confidence encoded value. Then if there is only 1 answer repeated maximum number of times then it is the chosen answer for this question. In case 2 or more are repeated equally we check which one of them is repeated maximum number of times in the data-set to check if the answer is not an out-liner and if more than one is equal we use the **Levenshtein distance** to see which one is more representative of the data

0.2.4 Random samples visualisation

Here are some random samples along with their questions and other data visualized:

questions : What Is this?

answerable :1

answer type :other

answers:[android phone, cell phone, cell phone, cellphone, samsung phone, unanswerable, phone, smart phone, white smartphone, phone]

answer confidence[maybe, maybe, maybe, maybe, yes, yes, yes, yes, yes, yes]

questions:Oh, what's this?

answerable:1

answer type:other

answers:[unanswerable, door, door, door locks, door, door, door, door, door, door]

answer confidence:[yes, yes, yes, yes, yes, yes, yes, yes, yes, maybe]

questions:What flavor and make of coffee is this, please?

answerable:1

answer type:other

answers:[french roast tullys, tullys french roast, tullys french roast, tolls, unanswerable, unsuitable, french roast, tullys french roast, unsuitable, tullys french roast]

answer confidence:[yes, yes, yes, maybe, no, yes, maybe, yes, yes, yes]

questions: What is the direction of the curtain loop and why?

answerable:0

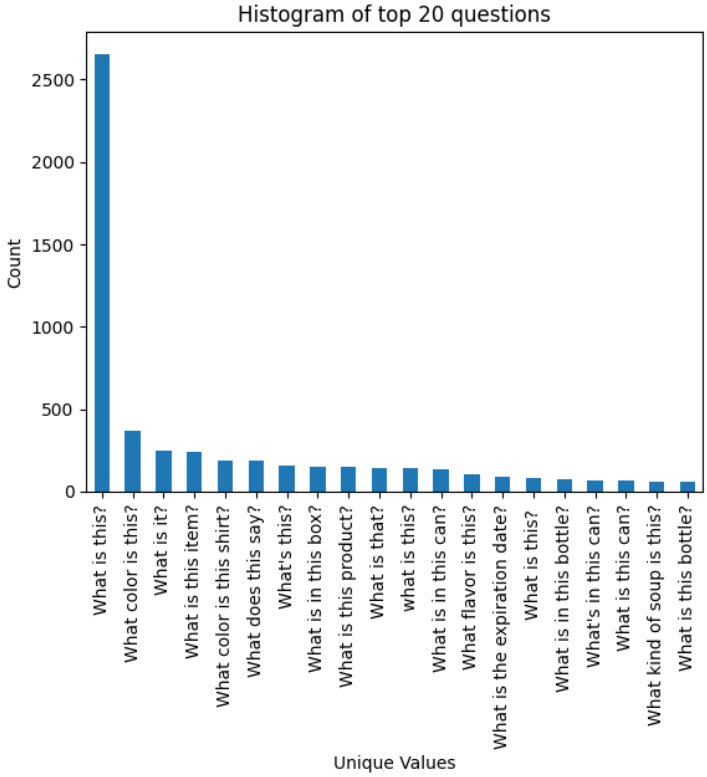
answer type:unanswerable

answers:[unanswerable, unanswerable, unanswerable, unanswerable, unanswerable, unanswerable, unanswerable, unanswerable, unanswerable, unanswerable]

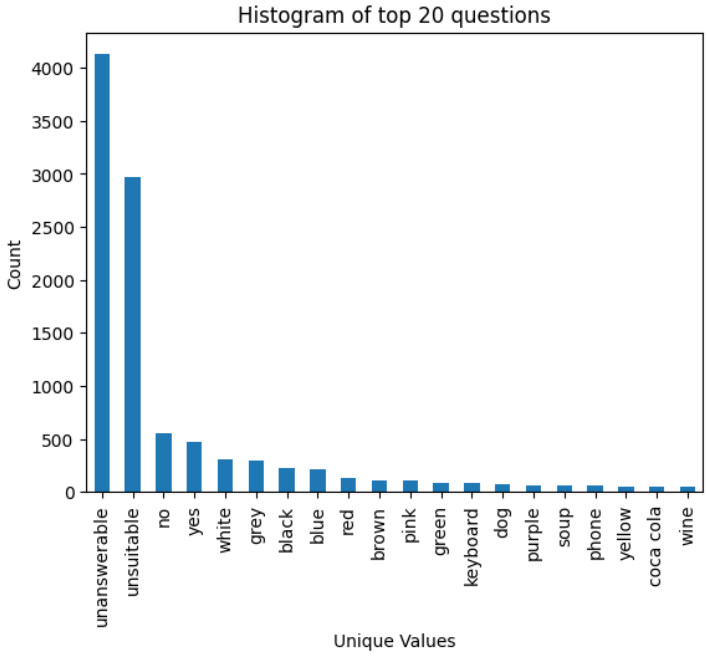
answer confidence:[no, no, yes, yes, yes, yes, no, yes, yes, yes]

0.2.5 Some finer details about the data

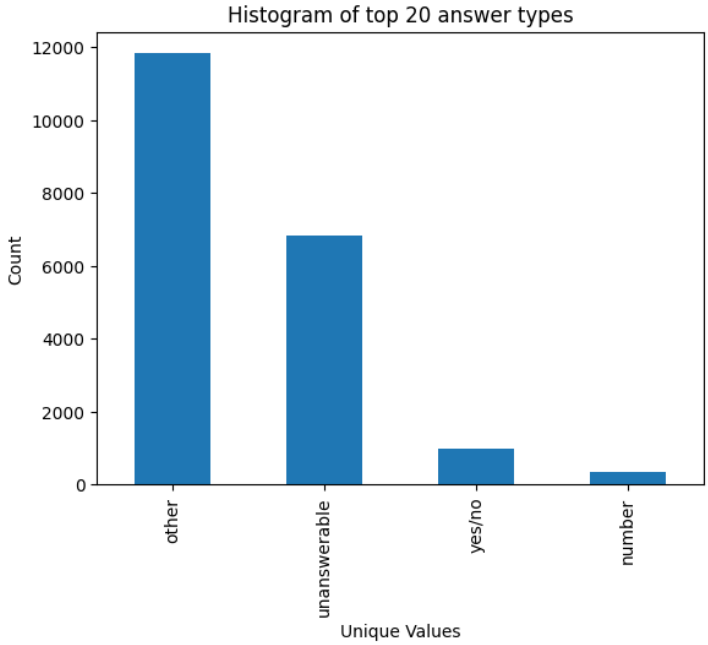
Some questions are repeated many times in the model as shown in the histogram below



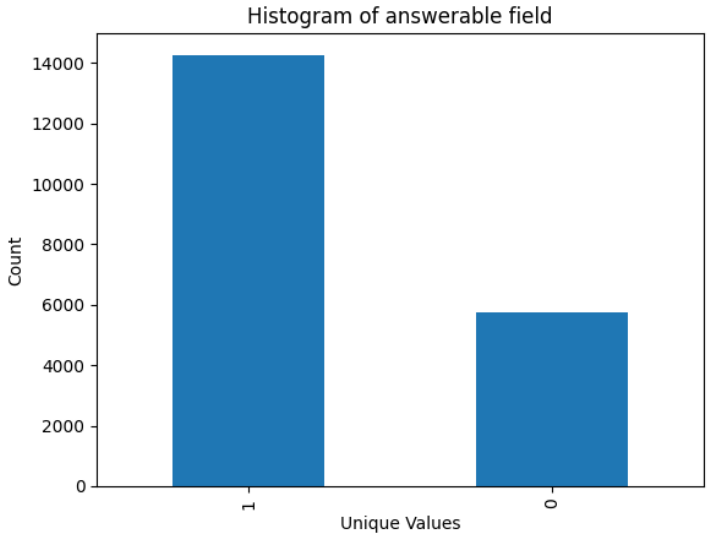
Also some answers are repeated more than others as shown in the histogram below



Also shown below the top 20 answer types



And also shown the count of answerable vs unanswerable questions



The last step before applying the data to our model was to encode the answers and answer types using one hot encoding

0.3 Model

We will discuss here each part of the code in a seprate subsection

0.3.1 EncoderDataset class

The EncoderDataset class is defined, which inherits from the PyTorch Dataset class. It takes in a datadf Pandas DataFrame, a filesdir directory containing the image files, a preprocess function for preprocessing the images, a tokenizer function for tokenizing the questions, and a clipmodel pre-trained CLIP model for encoding the images and questions.

The init method initializes the dataset with the input parameters.

The len method returns the length of the dataset.

The getitem method reads a row from the datadf DataFrame at index idx, opens the corresponding image file using Image.open(), applies the preprocess function to preprocess the image, and applies the tokenizer function to tokenize the question. The image and question tokens are then passed through the clipmodel using clipmodel.encodeimage() and clipmodel.encodetext() to obtain the image and question features.

The image and question features are returned as a tuple.

The clipencoder function takes in the data DataFrame and filesdir directory as inputs.

The CLIP model and preprocessing function are loaded using clip.load() and assigned to clipmodel and preprocess, respectively:

A EncoderDataset object is created with the input parameters and a PyTorch DataLoader object is created using the dataset object with a batch size of 1.

A loop is executed over each batch of data in the dataloader, where the image and question features are reshaped and concatenated into a single encoding, and the encoding is appended to the encoding list.

The encoding list is then added as a new column to the data DataFrame with the name 'encoding'.

Overall, this code encodes image-question pairs using the CLIP model and returns a new DataFrame with an additional 'encoding' column containing the encoded features for each image-question pair.

0.3.2 VQA model class

This part defines a PyTorch model VQAModel that takes in an encoded image-question pair and outputs predicted answers and answer types. The model is a feedforward neural network with three branches - one for the answer prediction, one for the answer type prediction, and one for combining the two to produce the final answer prediction.

Here's a detailed explanation of what is happening in the code:

The VQAModel class is defined, which inherits from the PyTorch nn.Module class. It takes in input size, num classes, and num answers types as inputs. input size is the size of the input feature vector, num classes is the number of answer classes, and num answers types is the number of answer types.

The init method initializes the model with the input parameters.

The model consists of several layers:

- nn.LayerNorm is used for normalization of the input feature vector.

- nn.Dropout is used to prevent overfitting.

- nn.Linear is used to perform a linear transformation on the input feature vector.

Three branches are defined:

- The linear2 layer is used to predict the answer.

- The linear2 aux layer is used to predict the answer type.

- The linear output and sigmoid output layers are used to combine the answer and answer type predictions into a single output.

The forward method defines the forward pass of the model. The input feature vector is passed through the normalization, dropout, and linear layers to obtain the combined features. The combined features are then passed through the answer and answer type branches to obtain the answer and answer type predictions. The answer type prediction is used to weight the answer prediction using the linear output and sigmoid output layers, and the final answer prediction is obtained by multiplying the weighted answer prediction with the answer type prediction.

The final answer prediction and answer type prediction are returned as a tuple.

Overall, this part defines a PyTorch model that takes in an encoded image-question pair, predicts the answer and answer type, and combines them to produce a final answer prediction. The model consists of several layers, including normalization, dropout, and linear layers, as well as several branches for predicting the answer and answer type.

0.3.3 VQA dataset class

This part defines a PyTorch dataset VQADataset that takes encoded image-question pairs, answers, and answer types as input and returns them in a format that can be used to train a model.

Here's a detailed explanation of what is happening in the part:

The VQADataset class is defined, which inherits from the PyTorch Dataset class. It takes in img qust encoded, answers, and answers types as inputs. img qust encoded is a list of encoded image-question pairs, answers is a list of answers, and answers types is a list of answer types.

The init method initializes the dataset with the input parameters.

The len method returns the length of the dataset.

The getitem method reads the encoded data, answer, and answer type from the img qust encoded, answers, and answers types lists at index idx.

The encoded data, answer, and answer type are returned as a tuple.

Overall, this part defines a dataset that can be used to train a model to predict answers and answer types given encoded image-question pairs. The dataset takes in lists of encoded image-question pairs, answers, and answer types as input and returns them in a format that can be fed into a PyTorch model for training.

0.3.4 data training

The data train procedure is done as follows:

Initialize some variables to keep track of the totalloss and accuracy for the answers and answer types.

Set the model to training mode using model.train().

Loop through each batch of data in the data loader.

Move the data to the device (e.g. GPU) specified in the device variable.

Zero out the gradients in the optimizer using optimizer.zerograd().

Pass the encoded data through the model using output, aux = model(encoded data), which returns the model's output for both the answer and answer type.

Squeeze the output tensors to remove any extra dimensions using torch.squeeze().

Compute the answer loss and answer type loss separately using loss.fn(output, answers) and loss.fn(aux, answers types), respectively.

Compute the total loss as the sum of the answer loss and answer type loss.

Back propagate the total loss through the model using loss.backward().

Update the model's parameters using the optimizer with optimizer.step().

Calculate the accuracy for the answers by comparing the model's predicted answer to the true answer using (predicted = answers).sum().item().

Calculate the accuracy for the answer types in the same way.

Calculate the total accuracy as the average of the answer accuracy and answer type accuracy.

Calculate the total loss as the average of the per-batch loss.

Return the total loss, total accuracy, answer accuracy, and answer type accuracy.

0.3.5 Model eval

The model evaluation is done as follows:

Initialize some variables to keep track of the total loss and accuracy for the answers and answer types.

Set the model to evaluation mode using model.eval().

Disable gradient calculation using with torch.nograd(): to speed up evaluation and save memory.

Loop through each batch of data in the dataloader.

Move the data to the device (e.g. GPU) specified in the device variable.

Pass the encoded data through the model using output, aux = model(encodeddata), which returns the model's output for both the answer and answer type.

Squeeze the output tensors to remove any extra dimensions using torch.squeeze().

Compute the answer loss and answer type loss separately using lossfn(output, answers) and lossfn(aux, answerstypes), respectively.

Compute the total loss as the sum of the answer loss and answer type loss.

Calculate the accuracy for the answers by comparing the model's predicted answer to the true answer using (predicted == answers).sum().item().

Calculate the accuracy for the answer types in the same way as step 10.

Calculate the total accuracy as the average of the answer accuracy and answer type accuracy.

Calculate the total loss as the average of the per-batch loss.

Return the total loss, total accuracy, answer accuracy, and answer type accuracy.

0.3.6 Predictions

The prediction procdeure is done as follows: Disable gradient calculation using with torch.nograd(): to speed up prediction and save memory.

Pass the encoded data through the model using output, aux = model(encodeddata), which returns the model's output for both the answer and answer type.

Squeeze the output tensors to remove any extra dimensions using torch.squeeze().

Compute the predicted answer by finding the index of the maximum value in output using torch.max(output.data, 1).

Compute the predicted answer type in the same way as step 4 but using aux instead of output.

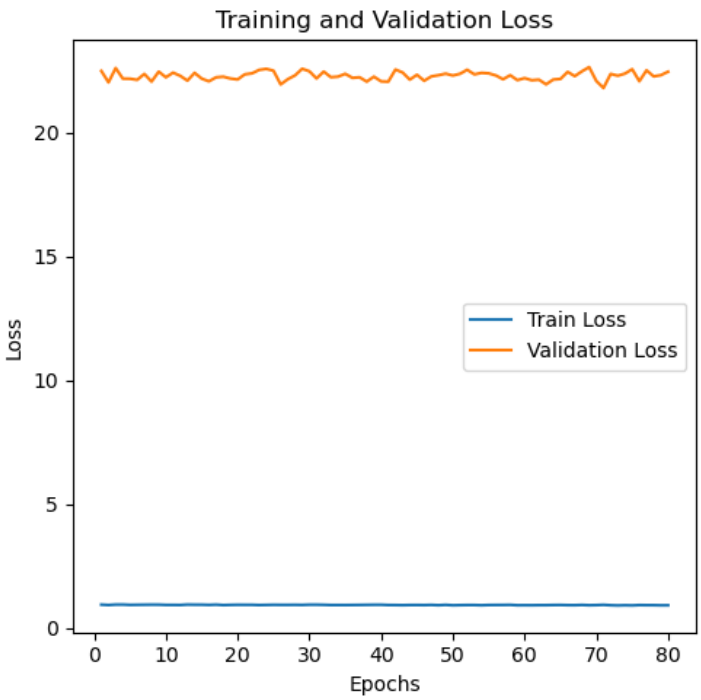
Decode the predicted answer and answer type using the respective inverse transformers answersencoder.inversetransform() and answerstypesencoder.inversetransform().

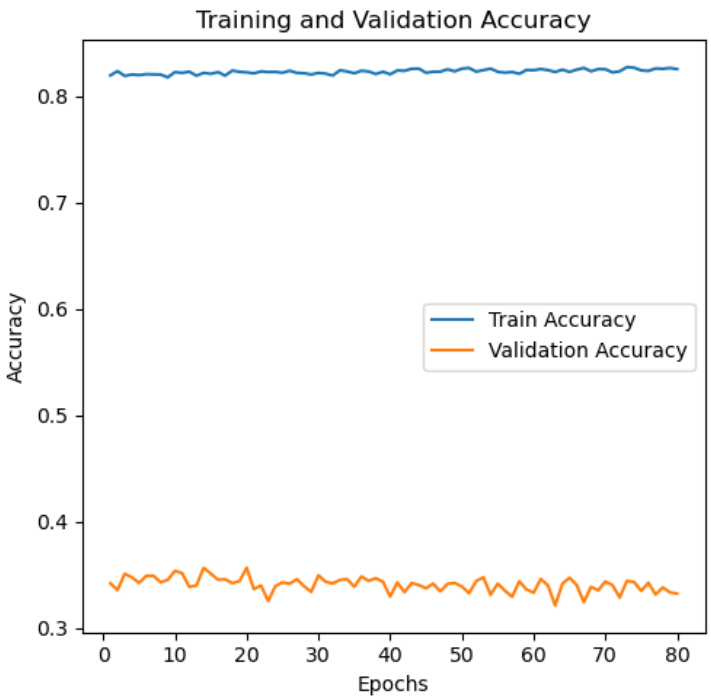
Return the predicted answer and answer type as strings.

0.4 Results

0.4.1 Training results

```
Epoch: 1/80 - Learning Rate:  0.001000
 loss:  0.9367 - accuracy:  0.8192 - val-loss:  22.4993 - val-accuracy:  0.3422
 ans-acc:  0.8005 - ans-type-acc:  0.8380 - val-ans-acc:  0.0000 - val-ans-type-acc:  0.6844
-----------------------------------
Epoch: 2/80 - Learning Rate:  0.001000
 loss:  0.9212 - accuracy:  0.8232 - val-loss:  22.0382 - val-accuracy:  0.3360
 ans-acc:  0.8052 - ans-type-acc:  0.8412 - val-ans-acc:  0.0000 - val-ans-type-acc:  0.6719
............
............
............
Epoch: 79/80 - Learning Rate:  0.001000
 loss:  0.9083 - accuracy:  0.8261 - val-loss:  22.3289 - val-accuracy:  0.3338
 ans-acc:  0.8033 - ans-type-acc:  0.8489 - val-ans-acc:  0.0000 - val-ans-type-acc:  0.6675
-----------------------------------
Epoch: 80/80 - Learning Rate:  0.001000
 loss:  0.9093 - accuracy:  0.8251 - val-loss:  22.4685 - val-accuracy:  0.3326
 ans-acc:  0.8018 - ans-type-acc:  0.8485 - val-ans-acc:  0.0000 - val-ans-type-acc:  0.6652
-----------------------------------
```





0.4.2 Testing results

loss: 7.3584 - accuracy: 0.5487

answer-accuracy: 0.4041 - answer-type-accuracy: 0.8485

0.4.3 Test samples

what is this? predicted answer:unanswerable, and the type: other

what is the type of this vehicle?

predicted answer:unanswerable, and the type: unanswerable

what is this?

predicted answer:unanswerable, and the type: other

what is this?

predicted answer:unanswerable, and the type: unanswerable

what is the color?

predicted answer:black, and the type: other