# Assignment 4

**Visual question answering**

*****Press here to join the report with the LaTeX code*****

Sally Kamel Soliman Armanyous ... 19015762

Salma Saaed Mahmoud Ghareb ... 19015779

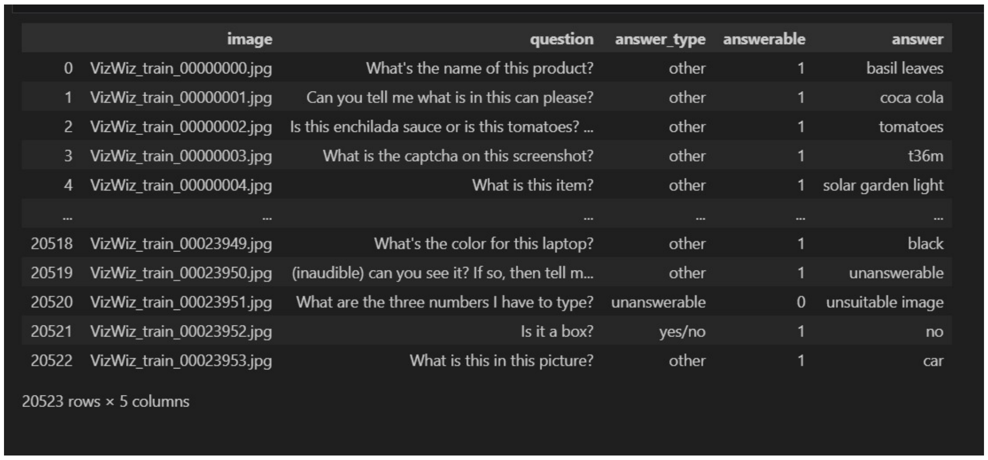
Esraa Hassan Mokhtar Aboshady ... 19015407

## 0.1 Problem Statement

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 VizWiz dataset, this dataset was constructed to train models to help visually impaired people.
In the words of creators of VizWiz: "we introduce the visual question answering (VQA) dataset coming from this population, which we call VizWiz-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 crowdsourced answers per visual question."

## 0.2 Download the Dataset and Data analysis

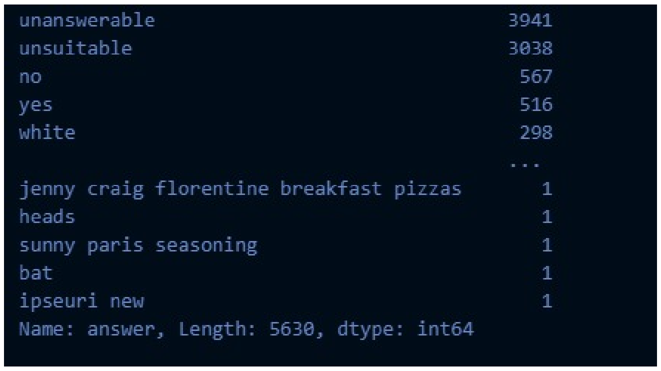
- We download the json file of train data.



| | image | question | answer_type | answerable | answer |
|---|---|---|---|---|---|
| 0 | VizWiz_train_00000000.jpg | What's the name of this product? | other | 1 | basil leaves |
| 1 | VizWiz_train_00000001.jpg | Can you tell me what is in this can please? | other | 1 | coca cola |
| 2 | VizWiz_train_00000002.jpg | Is this enchilada sauce or is this tomatoes? ... | other | 1 | tomatoes |
| 3 | VizWiz_train_00000003.jpg | What is the captcha on this screenshot? | other | 1 | t36m |
| 4 | VizWiz_train_00000004.jpg | What is this item? | other | 1 | solar garden light |
| ... | ... | ... | ... | ... | ... |
| 20518 | VizWiz_train_00023949.jpg | What's the color for this laptop? | other | 1 | black |
| 20519 | VizWiz_train_00023950.jpg | (inaudible) can you see it? If so, then tell m... | other | 1 | unanswerable |
| 20520 | VizWiz_train_00023951.jpg | What are the three numbers I have to type? | unanswerable | 0 | unsuitable image |
| 20521 | VizWiz_train_00023952.jpg | Is it a box? | yes/no | 1 | no |
| 20522 | VizWiz_train_00023953.jpg | What is this in this picture? | other | 1 | car |

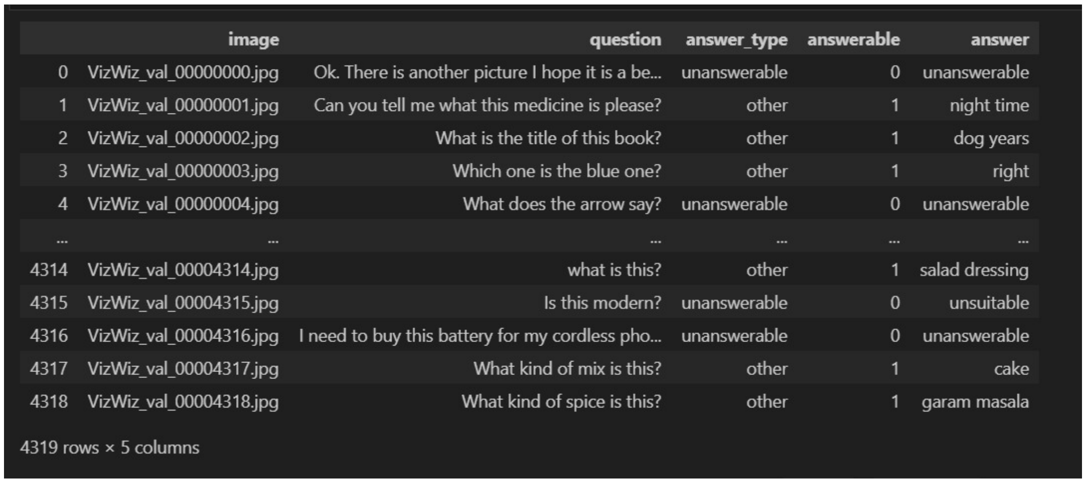
20523 rows × 5 columns

- Then count the occurrences of each unique value in the 'answer' column of the DataFrame train-df.



```
unanswerable                           3941
unsuitable                             3038
no                                      567
yes                                     516
white                                   298
                                       ...
jenny craig florentine breakfast pizzas   1
heads                                     1
sunny paris seasoning                     1
bat                                       1
ipseuri new                               1
Name: answer, Length: 5630, dtype: int64
```

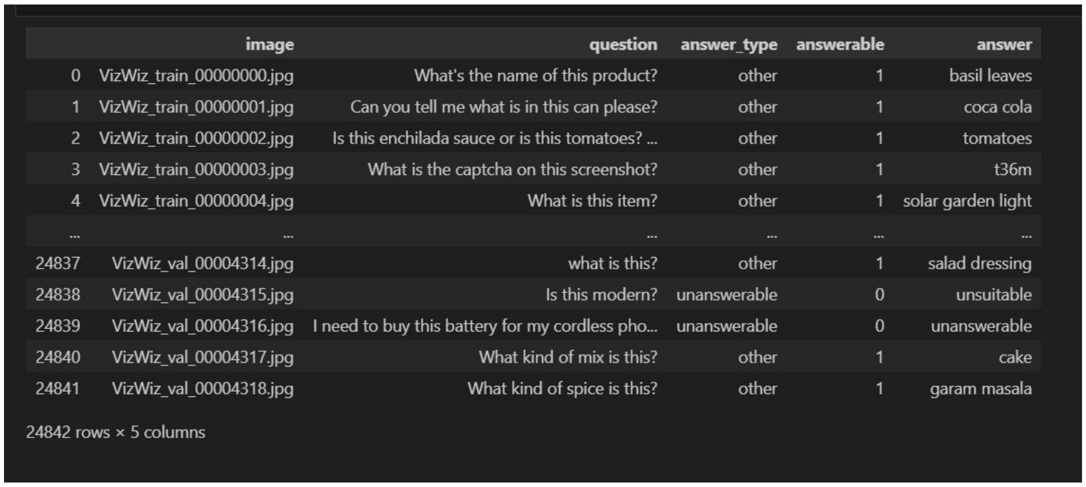
- We download the json file of validation data.
- Then calculate the number of unique values in the 'answer' column of the DataFrame val-df.



| | image | question | answer_type | answerable | answer |
|---|---|---|---|---|---|
| 0 | VizWiz_val_00000000.jpg | Ok. There is another picture I hope it is a be... | unanswerable | 0 | unanswerable |
| 1 | VizWiz_val_00000001.jpg | Can you tell me what this medicine is please? | other | 1 | night time |
| 2 | VizWiz_val_00000002.jpg | What is the title of this book? | other | 1 | dog years |
| 3 | VizWiz_val_00000003.jpg | Which one is the blue one? | other | 1 | right |
| 4 | VizWiz_val_00000004.jpg | What does the arrow say? | unanswerable | 0 | unanswerable |
| ... | ... | ... | ... | ... | ... |
| 4314 | VizWiz_val_00004314.jpg | what is this? | other | 1 | salad dressing |
| 4315 | VizWiz_val_00004315.jpg | Is this modern? | unanswerable | 0 | unsuitable |
| 4316 | VizWiz_val_00004316.jpg | I need to buy this battery for my cordless pho... | unanswerable | 0 | unanswerable |
| 4317 | VizWiz_val_00004317.jpg | What kind of mix is this? | other | 1 | cake |
| 4318 | VizWiz_val_00004318.jpg | What kind of spice is this? | other | 1 | garam masala |

4319 rows × 5 columns

- Then concatenate two DataFrames vertically along the rows. It combines the DataFrames train-df and val-df into a single DataFrame.data-df.



| | image | question | answer_type | answerable | answer |
|---|---|---|---|---|---|
| 0 | VizWiz_train_00000000.jpg | What's the name of this product? | other | 1 | basil leaves |
| 1 | VizWiz_train_00000001.jpg | Can you tell me what is in this can please? | other | 1 | coca cola |
| 2 | VizWiz_train_00000002.jpg | Is this enchilada sauce or is this tomatoes? ... | other | 1 | tomatoes |
| 3 | VizWiz_train_00000003.jpg | What is the captcha on this screenshot? | other | 1 | t36m |
| 4 | VizWiz_train_00000004.jpg | What is this item? | other | 1 | solar garden light |
| ... | ... | ... | ... | ... | ... |
| 24837 | VizWiz_val_00004314.jpg | what is this? | other | 1 | salad dressing |
| 24838 | VizWiz_val_00004315.jpg | Is this modern? | unanswerable | 0 | unsuitable |
| 24839 | VizWiz_val_00004316.jpg | I need to buy this battery for my cordless pho... | unanswerable | 0 | unanswerable |
| 24840 | VizWiz_val_00004317.jpg | What kind of mix is this? | other | 1 | cake |
| 24841 | VizWiz_val_00004318.jpg | What kind of spice is this? | other | 1 | garam masala |

24842 rows × 5 columns

### 0.2.1 some used functions:

- def find-most-common-answer(answers):
The function uses the Counter class from the collections module to count the occurrences of each answer in the list. It then calls the most-common() method of the Counter object to obtain a list of answer-count pairs, sorted in descending order based on the count. The first element of this list (most-common-answers[0]) represents the most common answer and its count. The function extracts the most common answer and returns it.

- def select-most-common-answers(df):
The function iterates over each row of the DataFrame. For each row, it extracts the list of answers from the "answers" column and calls the function to obtain the most common answer. The selected answer is then appended to a list of dictionaries called selected-answers.
After iterating over all rows, the function updates the "answer" column in the DataFrame with the selected answers. Finally, it removes the "answers" column from the DataFrame and returns the updated DataFrame.

- def plot-img(path):
The function takes a single argument path, which represents the file path of the JPEG image to be plotted. First, the function load the JPEG image and assigns it to the image variable. Then, it plots the image, The line plt.axis('off') is used to remove the axis ticks and labels from the plot, resulting in a cleaner visualization. Finally, plt.show() is called to display the plot with the image.

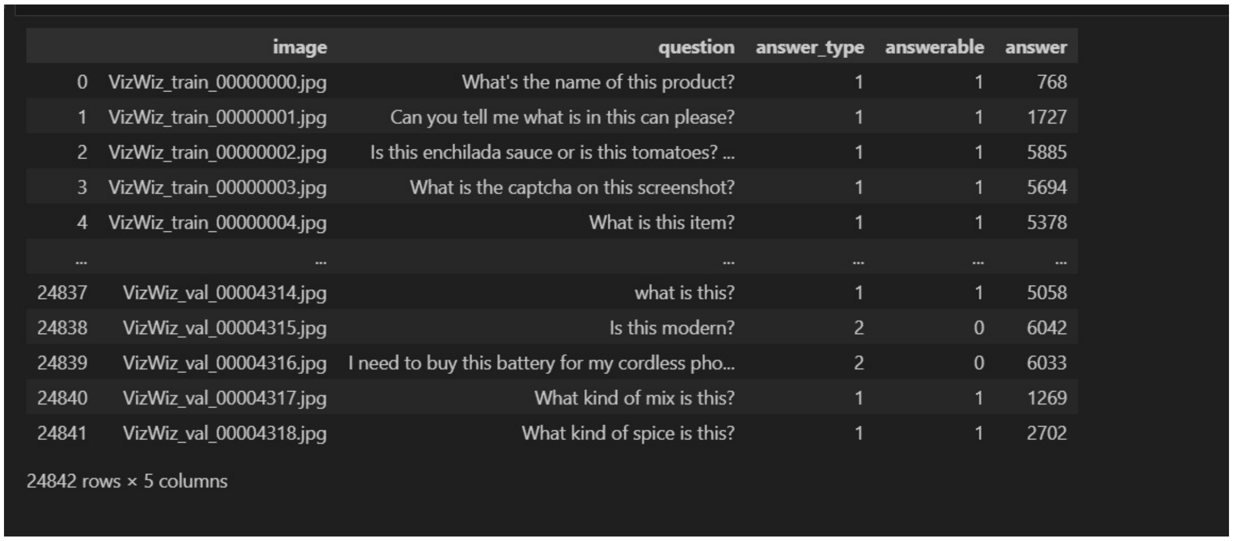
- def dataloader-json(path,test=False):
It opens the JSON file specified by the path argument and reads its contents. then load the JSON data from the file and assigns it to the data variable. It creates a DataFrame df from the loaded JSON data. If the test parameter is set to True, it returns the DataFrame df without further processing. Otherwise, it calls the select-most-common-answers function to select the most common answers for each row in the DataFrame. The resulting DataFrame is then returned.

## 0.3 Building the Model:

### 0.3.1 Label encoding:

- we create an instance of LabelEncoder and assigns it to the variable ans-lb.
- then applies the fit-transform method of ans-lb to the 'answer' column of data-df. It first fits the encoder on the unique values of the 'answer' column and then transforms the values of the column by replacing each value with its encoded label.
- Then creates another instance of LabelEncoder and assigns it to the variable ans-type-lb.
- And applies the fit-transform method of ans-type-lb to the 'answer-type' column of data-df. Similar to step 2, it fits the encoder on the unique values of the 'answer-type' column and trans-

forms the values of the column by replacing each value with its encoded label.

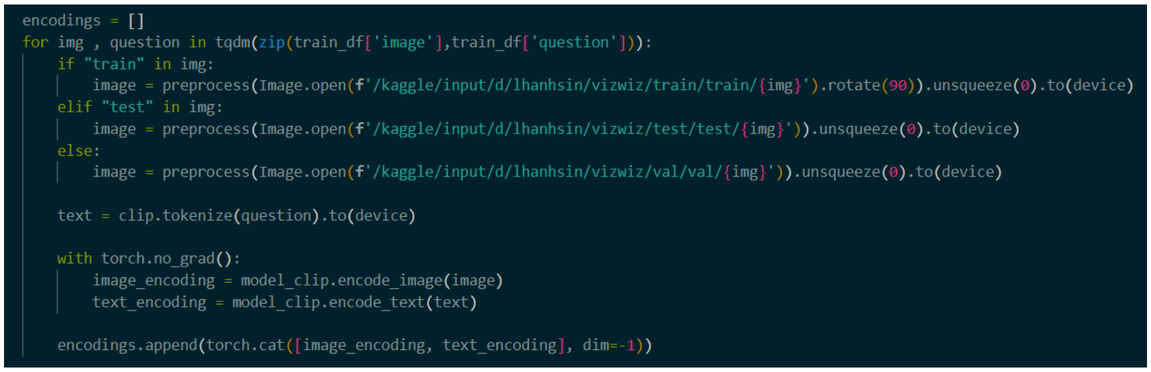


| | image | question | answer_type | answerable | answer |
|---|---|---|---|---|---|
| 0 | VizWiz_train_00000000.jpg | What's the name of this product? | 1 | 1 | 768 |
| 1 | VizWiz_train_00000001.jpg | Can you tell me what is in this can please? | 1 | 1 | 1727 |
| 2 | VizWiz_train_00000002.jpg | Is this enchilada sauce or is this tomatoes? ... | 1 | 1 | 5885 |
| 3 | VizWiz_train_00000003.jpg | What is the captcha on this screenshot? | 1 | 1 | 5694 |
| 4 | VizWiz_train_00000004.jpg | What is this item? | 1 | 1 | 5378 |
| ... | ... | ... | ... | ... | ... |
| 24837 | VizWiz_val_00004314.jpg | what is this? | 1 | 1 | 5058 |
| 24838 | VizWiz_val_00004315.jpg | Is this modern? | 2 | 0 | 6042 |
| 24839 | VizWiz_val_00004316.jpg | I need to buy this battery for my cordless pho... | 2 | 0 | 6033 |
| 24840 | VizWiz_val_00004317.jpg | What kind of mix is this? | 1 | 1 | 1269 |
| 24841 | VizWiz_val_00004318.jpg | What kind of spice is this? | 1 | 1 | 2702 |

24842 rows × 5 columns

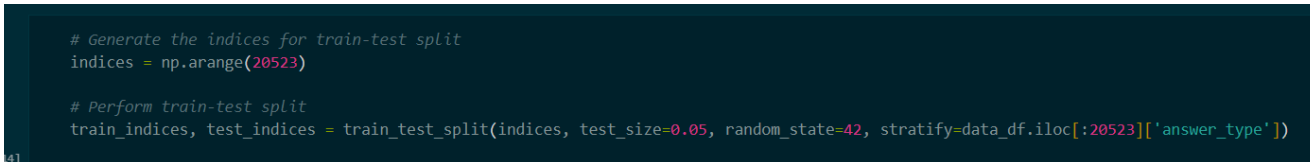
## 0.3.2 Clip encoding:

- We made a loop that iterates over pairs of image and question from the 'image' and 'question' columns of the 'train-df' DataFrame. It performs encoding using a pre-trained CLIP model and appends the resulting encodings to a list called 'encodings'.
- And save the 'encodings' list as a serialized file using the PyTorch torch.save() function.



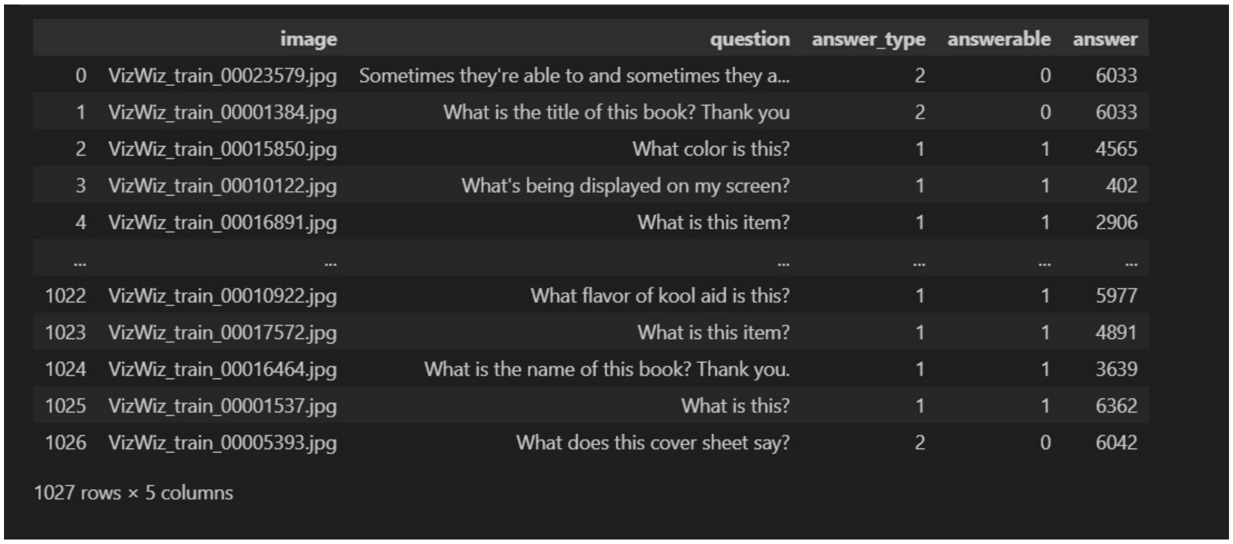
```python
encodings = []
for img , question in tqdm(zip(train_df['image'],train_df['question'])):
    if "train" in img:
        image = preprocess(Image.open(f'/kaggle/input/d/lhanhsin/vizwiz/train/train/{img}').rotate(90)).unsqueeze(0).to(device)
    elif "test" in img:
        image = preprocess(Image.open(f'/kaggle/input/d/lhanhsin/vizwiz/test/test/{img}')).unsqueeze(0).to(device)
    else:
        image = preprocess(Image.open(f'/kaggle/input/d/lhanhsin/vizwiz/val/val/{img}')).unsqueeze(0).to(device)

    text = clip.tokenize(question).to(device)

    with torch.no_grad():
        image_encoding = model_clip.encode_image(image)
        text_encoding = model_clip.encode_text(text)

    encodings.append(torch.cat([image_encoding, text_encoding], dim=-1))
```

## 0.3.3 Train and test splitting:



```python
# Generate the indices for train-test split
indices = np.arange(20523)

# Perform train-test split
train_indices, test_indices = train_test_split(indices, test_size=0.05, random_state=42, stratify=data_df.iloc[:20523]['answer_type'])
```

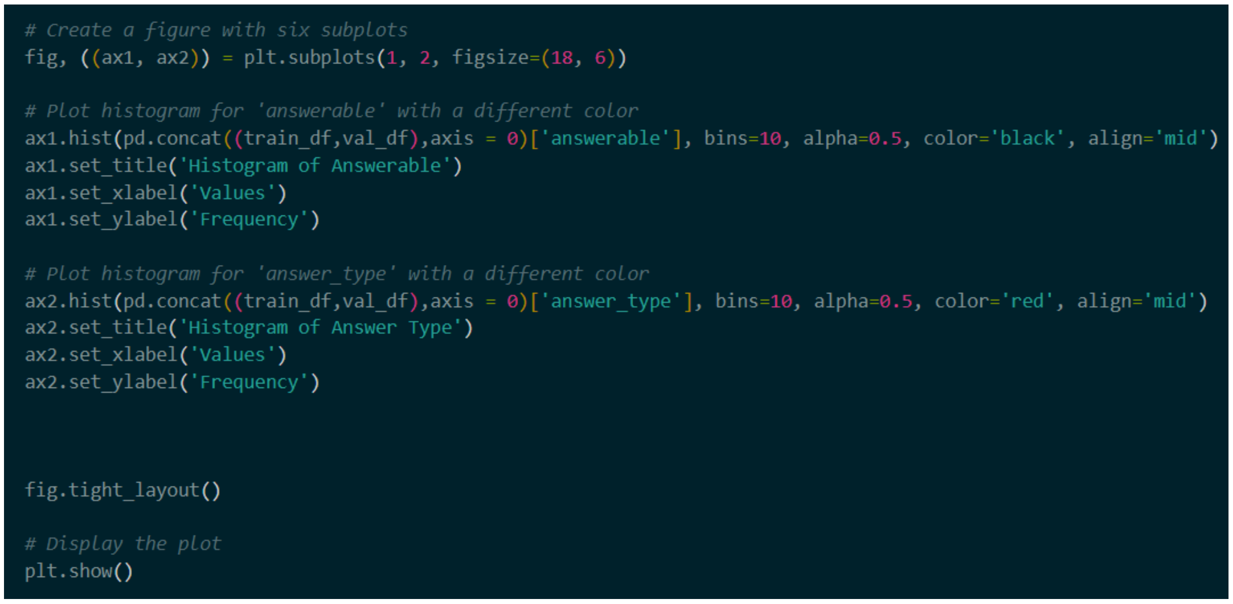
- indices = np.arange(20523) generates an array of indices ranging from 0 to 20522. This assumes that your dataset has a total of 20523 samples.
- train-test-split splits the indices into train and test sets. test-size=0.05 of the total dataset. And random-state=42 which ensures that the same split is generated each time you run the code.
- The target variable used for stratified sampling. It ensures that the distribution of the 'answer-type' variable is preserved in both the train and test sets. In this case, it assumes that the 'answer-type' column is present in the DataFrame data-df and contains the target variable values for the first 20523 samples.
- The resulting train indices are assigned to train-indices, and the test indices are assigned to test-indices.



```
                          image                                      question  answer_type  answerable  answer
0     VizWiz_train_00023579.jpg  Sometimes they're able to and sometimes they a...            2           0    6033
1     VizWiz_train_00001384.jpg          What is the title of this book? Thank you            2           0    6033
2     VizWiz_train_00015850.jpg                               What color is this?            1           1    4565
3     VizWiz_train_00010122.jpg               What's being displayed on my screen?            1           1     402
4     VizWiz_train_00016891.jpg                               What is this item?            1           1    2906
...                         ...                                               ...          ...         ...     ...
1022  VizWiz_train_00010922.jpg                 What flavor of kool aid is this?            1           1    5977
1023  VizWiz_train_00017572.jpg                               What is this item?            1           1    4891
1024  VizWiz_train_00016464.jpg         What is the name of this book? Thank you.            1           1    3639
1025  VizWiz_train_00001537.jpg                                    What is this?            1           1    6362
1026  VizWiz_train_00005393.jpg                 What does this cover sheet say?            2           0    6042

1027 rows × 5 columns
```

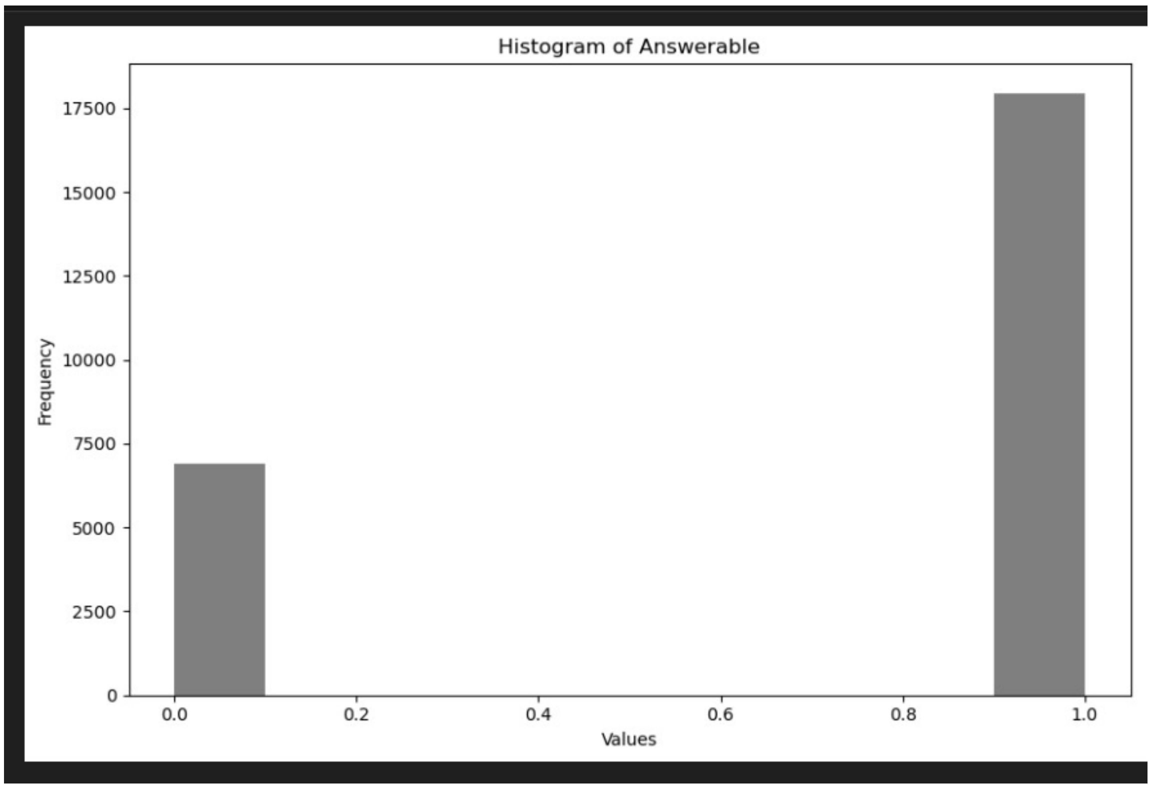
- Train-indices-aug = train-indices + 20523: adds the value 20523 to each element in the train-indices array, creating a new array called train-indices-aug. This operation is performed to shift the indices by 20523, assuming that the original indices corresponded to a subset of the dataset, and you want to create an augmented version of the train indices that refers to the complete dataset.
- The train-indices array contains the indices of the rows you want to select. The selected subset of rows is assigned to the DataFrame train.
- After that, the code train = train.reset-index(drop=True) resets the index of the train DataFrame, dropping the original index and assigning a new sequential index starting from 0.
- The test-indices array contains the indices of the rows you want to select. The selected subset of rows is assigned to the DataFrame test.
- After that, the code test = test.reset-index(drop=True) resets the index of the test DataFrame, dropping the original index and assigning a new sequential index starting from 0.

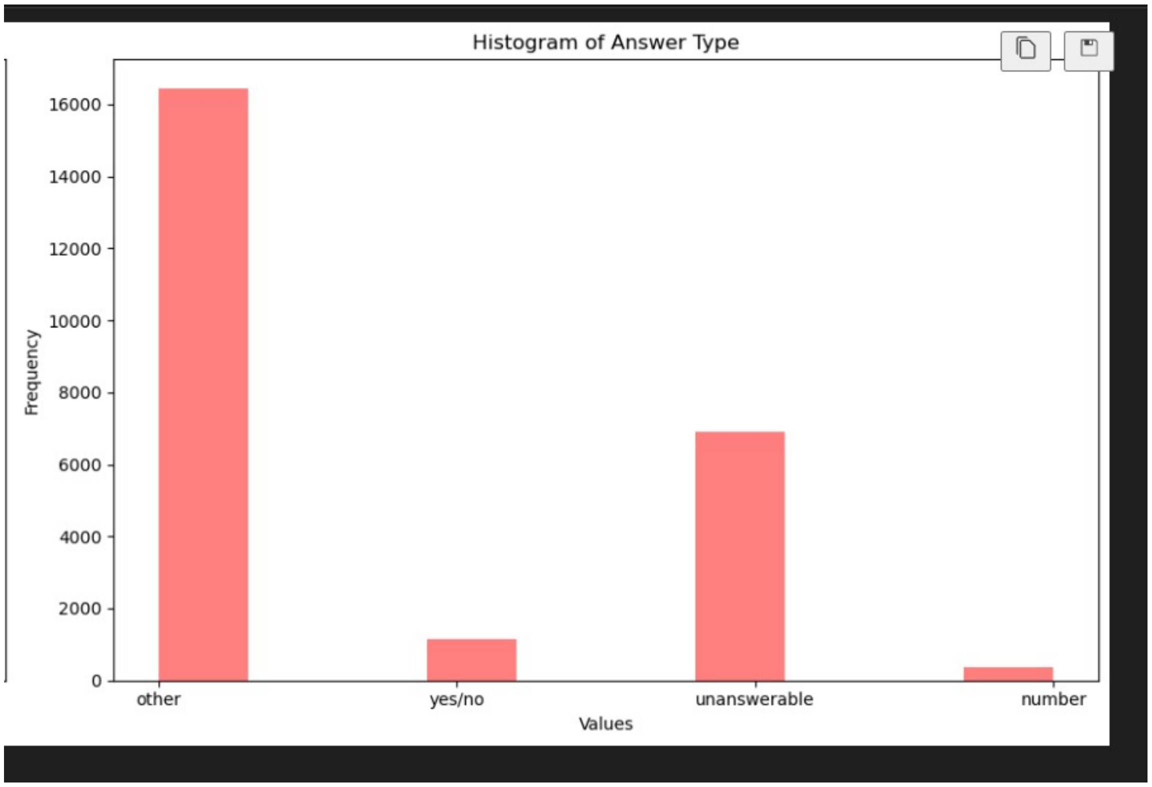
### 0.3.4 Histograms:



```
# Create a figure with six subplots
fig, ((ax1, ax2)) = plt.subplots(1, 2, figsize=(18, 6))

# Plot histogram for 'answerable' with a different color
ax1.hist(pd.concat((train_df,val_df),axis = 0)['answerable'], bins=10, alpha=0.5, color='black', align='mid')
ax1.set_title('Histogram of Answerable')
ax1.set_xlabel('Values')
ax1.set_ylabel('Frequency')

# Plot histogram for 'answer_type' with a different color
ax2.hist(pd.concat((train_df,val_df),axis = 0)['answer_type'], bins=10, alpha=0.5, color='red', align='mid')
ax2.set_title('Histogram of Answer Type')
ax2.set_xlabel('Values')
ax2.set_ylabel('Frequency')



fig.tight_layout()

# Display the plot
plt.show()
```

- creates a figure with two subplots using plt.subplots(). Each subplot represents a histogram of a different variable from the concatenated DataFrame pd.concat((train-df,val-df),axis=0).

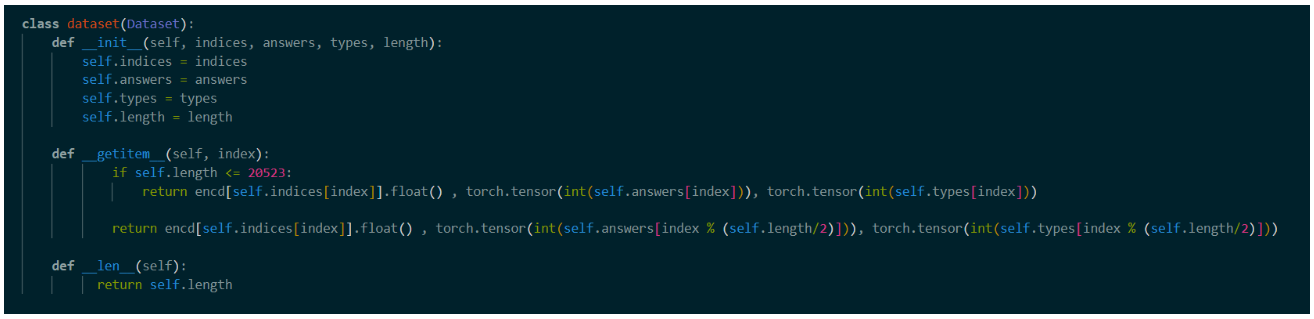




### 0.3.5 The Model:

- For the dimensions:
embedding-size = 768.
classes = len(np.unique(ans-lb.classes-)): This variable determines the number of classes in a classification problem.
aux-classes = len(np.unique(train['answer-type'])): This variable computes the number of auxiliary classes in a classification problem.
BATCH-SIZE = 64: It indicates that the model will process 64 samples in each training batch or during inference.

- For the encoded data: load a PyTorch tensor from a file using the torch.load function and concatenates it with itself using torch.cat



```python
class dataset(Dataset):
    def __init__(self, indices, answers, types, length):
        self.indices = indices
        self.answers = answers
        self.types = types
        self.length = length

    def __getitem__(self, index):
        if self.length <= 20523:
            return encd[self.indices[index]].float() , torch.tensor(int(self.answers[index])), torch.tensor(int(self.types[index]))

        return encd[self.indices[index]].float() , torch.tensor(int(self.answers[index % (self.length/2)])), torch.tensor(int(self.types[index % (self.length/2)]))

    def __len__(self):
        return self.length
```

- for this function the constructor method of the dataset class takes four arguments: indices, answers, types, and length. These arguments represent the indices, answers, answer types, and length of the dataset, respectively. These values are stored as attributes of the dataset object.

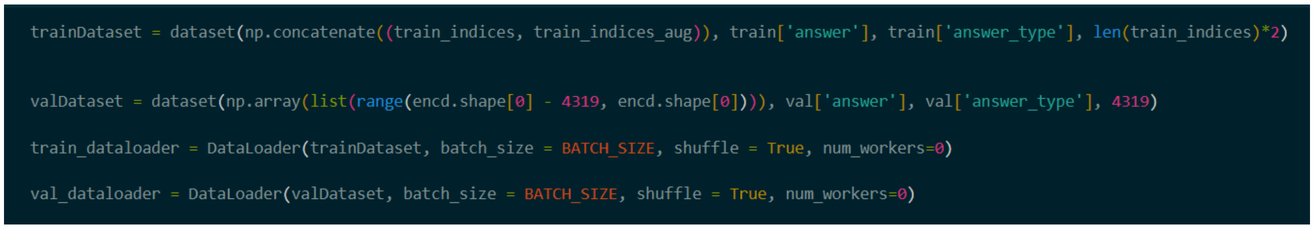
- getitem-(self, index): This method is used to retrieve a specific item from the dataset at the given index. It returns a tuple containing three elements. The implementation includes an if-else condition:

- If the self.length is less than or equal to 20523, it returns the tensor at the given index from the encd tensor. The second and third elements of the tuple convert the corresponding elements from answers and types to tensors of integer type.

- If the self.length is greater than 20523, it returns encd[self.indices[index]].float(), similar to the previous case. However, the second and third elements of the tuple are obtained from answers and types using modulo operations based on the given index. This is likely done to handle cases where the index exceeds the length of answers and types by wrapping around.

- len-(self): This method returns the length of the dataset, which is defined by the self.length attribute.

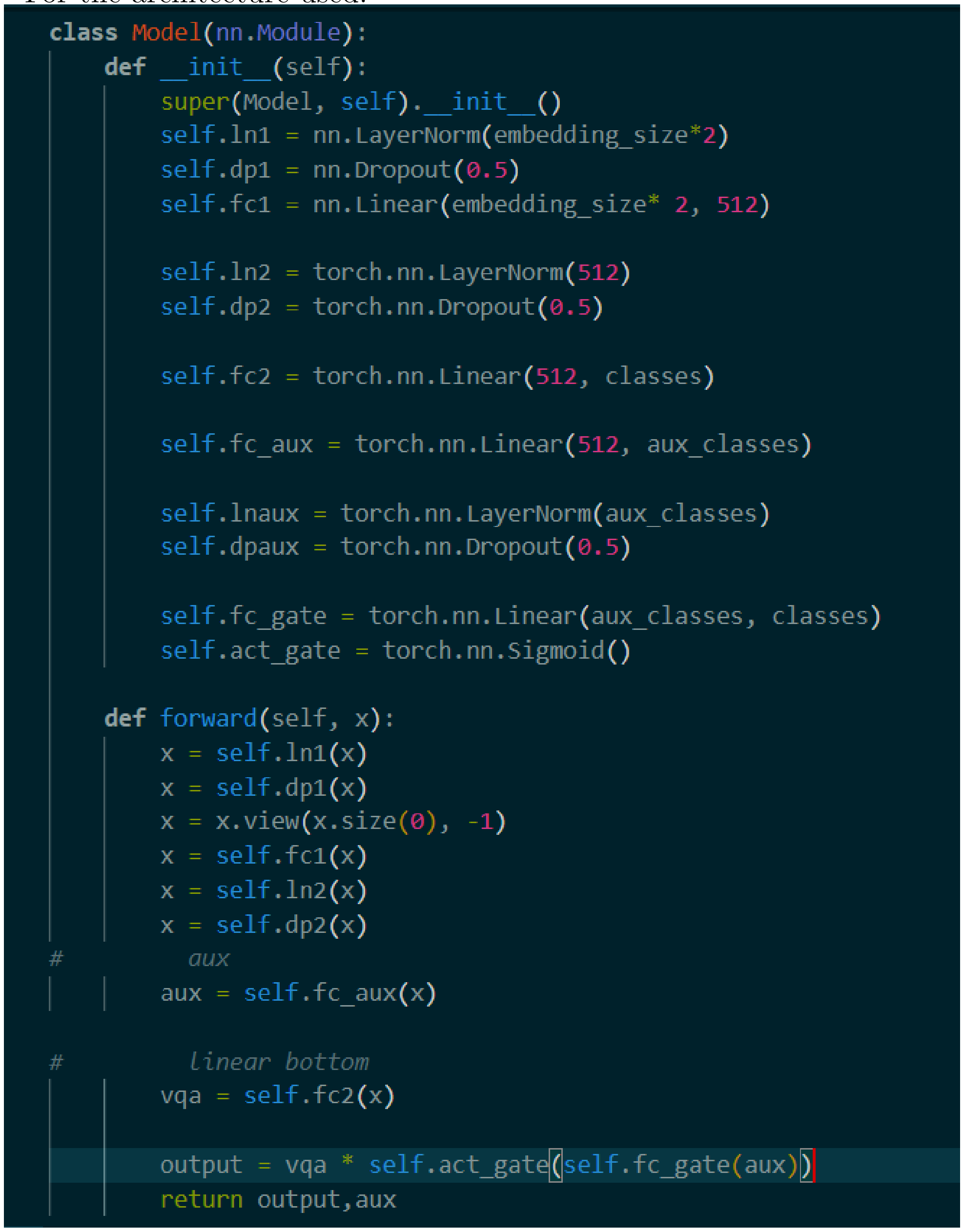
-This custom dataset class allows you to access individual samples from the dataset by index and obtain the corresponding inputs (encd tensor), answers, and answer types. Ensure that the encd, indices, answers, and types variables are properly defined before using this dataset class.



```python
trainDataset = dataset(np.concatenate((train_indices, train_indices_aug)), train['answer'], train['answer_type'], len(train_indices)*2)

valDataset = dataset(np.array(list(range(encd.shape[0] - 4319, encd.shape[0]))), val['answer'], val['answer_type'], 4319)

train_dataloader = DataLoader(trainDataset, batch_size = BATCH_SIZE, shuffle = True, num_workers=0)

val_dataloader = DataLoader(valDataset, batch_size = BATCH_SIZE, shuffle = True, num_workers=0)
```

- Then demonstrates the usage of the dataset class to create training and validation datasets, as well as the corresponding data loaders using PyTorch's DataLoader.

- For the architecture used:

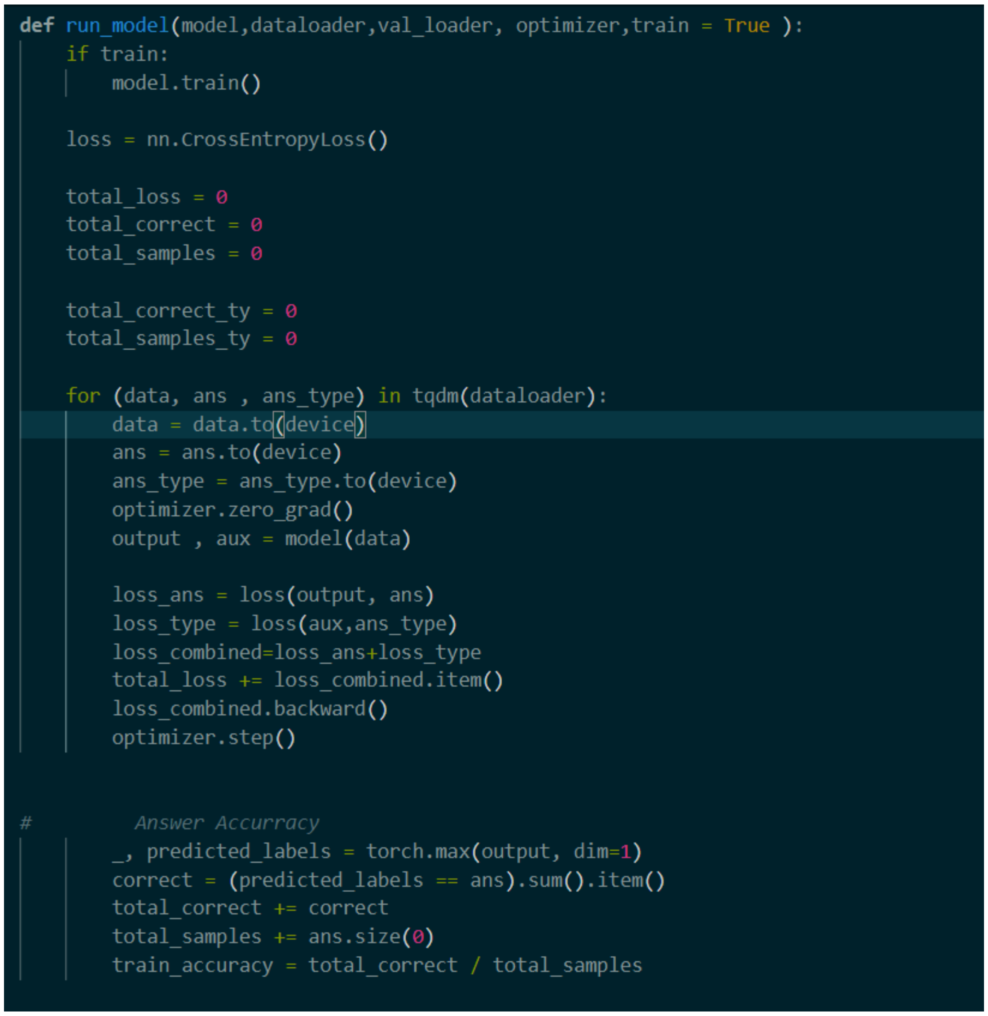


```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.ln1 = nn.LayerNorm(embedding_size*2)
        self.dp1 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(embedding_size* 2, 512)

        self.ln2 = torch.nn.LayerNorm(512)
        self.dp2 = torch.nn.Dropout(0.5)

        self.fc2 = torch.nn.Linear(512, classes)

        self.fc_aux = torch.nn.Linear(512, aux_classes)

        self.lnaux = torch.nn.LayerNorm(aux_classes)
        self.dpaux = torch.nn.Dropout(0.5)

        self.fc_gate = torch.nn.Linear(aux_classes, classes)
        self.act_gate = torch.nn.Sigmoid()

    def forward(self, x):
        x = self.ln1(x)
        x = self.dp1(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.ln2(x)
        x = self.dp2(x)
#           aux
        aux = self.fc_aux(x)

#           linear bottom
        vqa = self.fc2(x)

        output = vqa * self.act_gate(self.fc_gate(aux))
        return output,aux
```

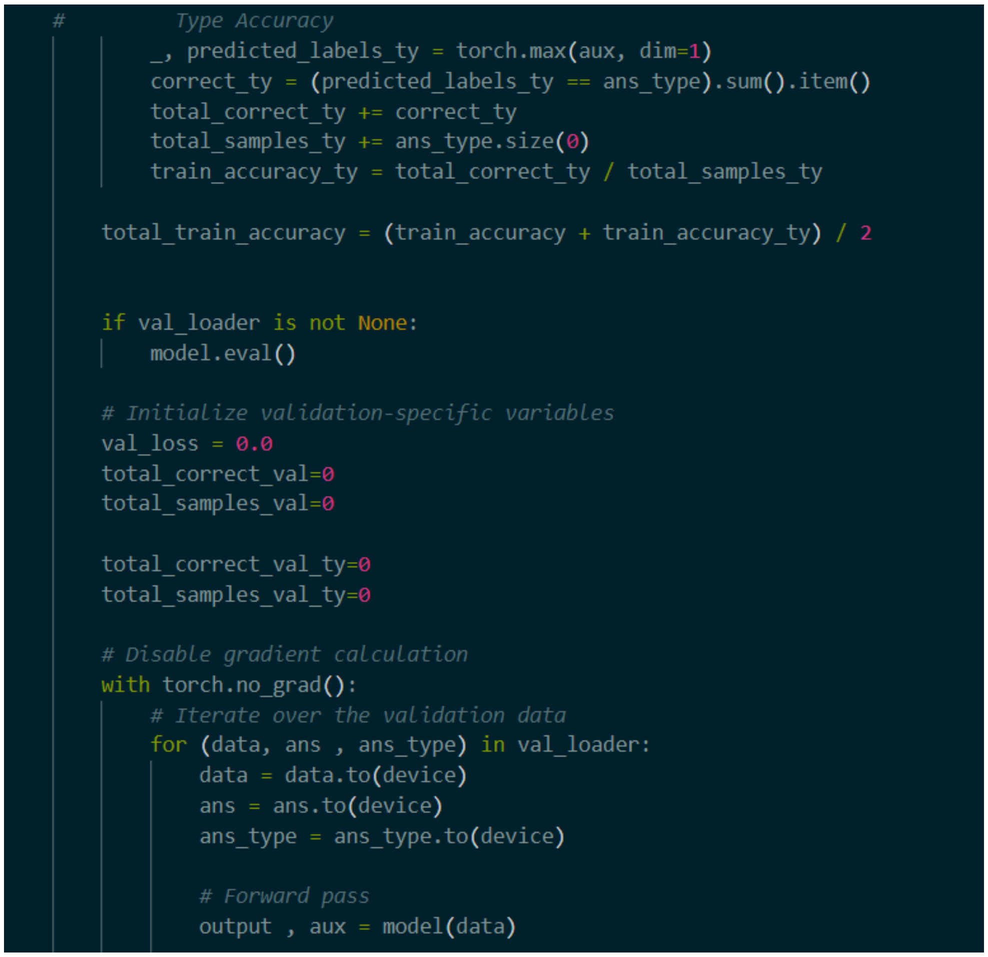
- def-init-(self): The constructor method initializes the Model class.
- self.ln1 = nn.LayerNorm(embedding-size*2): This line creates an instance of the LayerNorm layer from the nn module. It normalizes the input tensor of size embedding-size*2 along the last dimension.
- self.dp1 = nn.Dropout(0.5): This line creates an instance of the Dropout layer from the nn module. It randomly zeros some elements of the input tensor with a probability of 0.5, which helps prevent overfitting during training.
- self.fc1 = nn.Linear(embedding-size*2, 512): This line creates a fully connected (linear) layer with an input size of embedding-size*2 and an output size of 512. It performs a linear transfor-

mation on the input tensor.
- self.ln2 = torch.nn.LayerNorm(512): This line creates another instance of the LayerNorm layer, but this time with an input size of 512.
- self.dp2 = torch.nn.Dropout(0.5): This line creates another instance of the Dropout layer with a dropout probability of 0.5.
- self.fc2 = torch.nn.Linear(512, classes): This line creates another fully connected layer with an input size of 512 and an output size of classes. It performs a linear transformation on the input tensor.
- self.fc-aux = torch.nn.Linear(512, aux-classes): This line creates a fully connected layer with an input size of 512 and an output size of aux-classes.
- self.lnaux = torch.nn.LayerNorm(aux-classes): This line creates a LayerNorm layer with an input size of aux-classes.
- self.dpaux = torch.nn.Dropout(0.5): This line creates a Dropout layer with a dropout probability of 0.5.
- self.fc-gate = torch.nn.Linear(aux-classes, classes): This line creates another fully connected layer with an input size of aux-classes and an output size of classes.
- self.act-gate = torch.nn.Sigmoid(): This line creates an instance of the sigmoid activation function.
- def forward(self, x):: The forward method defines the forward pass of the model.
- x = self.ln1(x): This line applies layer normalization to the input tensor x.
- x = self.dp1(x): This line applies dropout to the normalized tensor x.
- x = x.view(x.size(0), -1): This line reshapes the tensor x by flattening it, keeping the batch size intact.
- x = self.fc1(x): This line applies the fully connected layer fc1 to the flattened tensor x.
- x = self.ln2(x): This line applies layer normalization to the tensor x.
- x = self.dp2(x): This line applies dropout to the normalized tensor x.]]
- aux = self.fc-aux(x): This line applies the fully connected layer fc-aux to the tensor x,

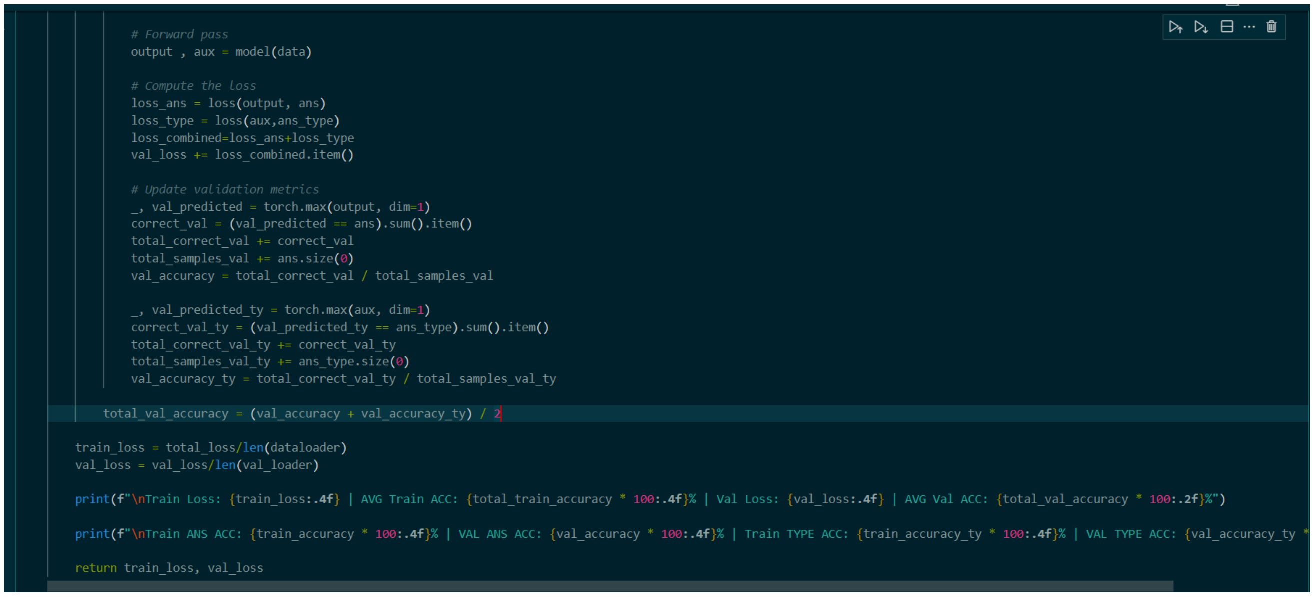
### 0.3.6 Training and validation:



```
def run_model(model,dataloader,val_loader, optimizer,train = True ):
    if train:
        model.train()

    loss = nn.CrossEntropyLoss()

    total_loss = 0
    total_correct = 0
    total_samples = 0

    total_correct_ty = 0
    total_samples_ty = 0

    for (data, ans , ans_type) in tqdm(dataloader):
        data = data.to(device)
        ans = ans.to(device)
        ans_type = ans_type.to(device)
        optimizer.zero_grad()
        output , aux = model(data)

        loss_ans = loss(output, ans)
        loss_type = loss(aux,ans_type)
        loss_combined=loss_ans+loss_type
        total_loss += loss_combined.item()
        loss_combined.backward()
        optimizer.step()


#          Answer Accurracy
        _, predicted_labels = torch.max(output, dim=1)
        correct = (predicted_labels == ans).sum().item()
        total_correct += correct
        total_samples += ans.size(0)
        train_accuracy = total_correct / total_samples
```



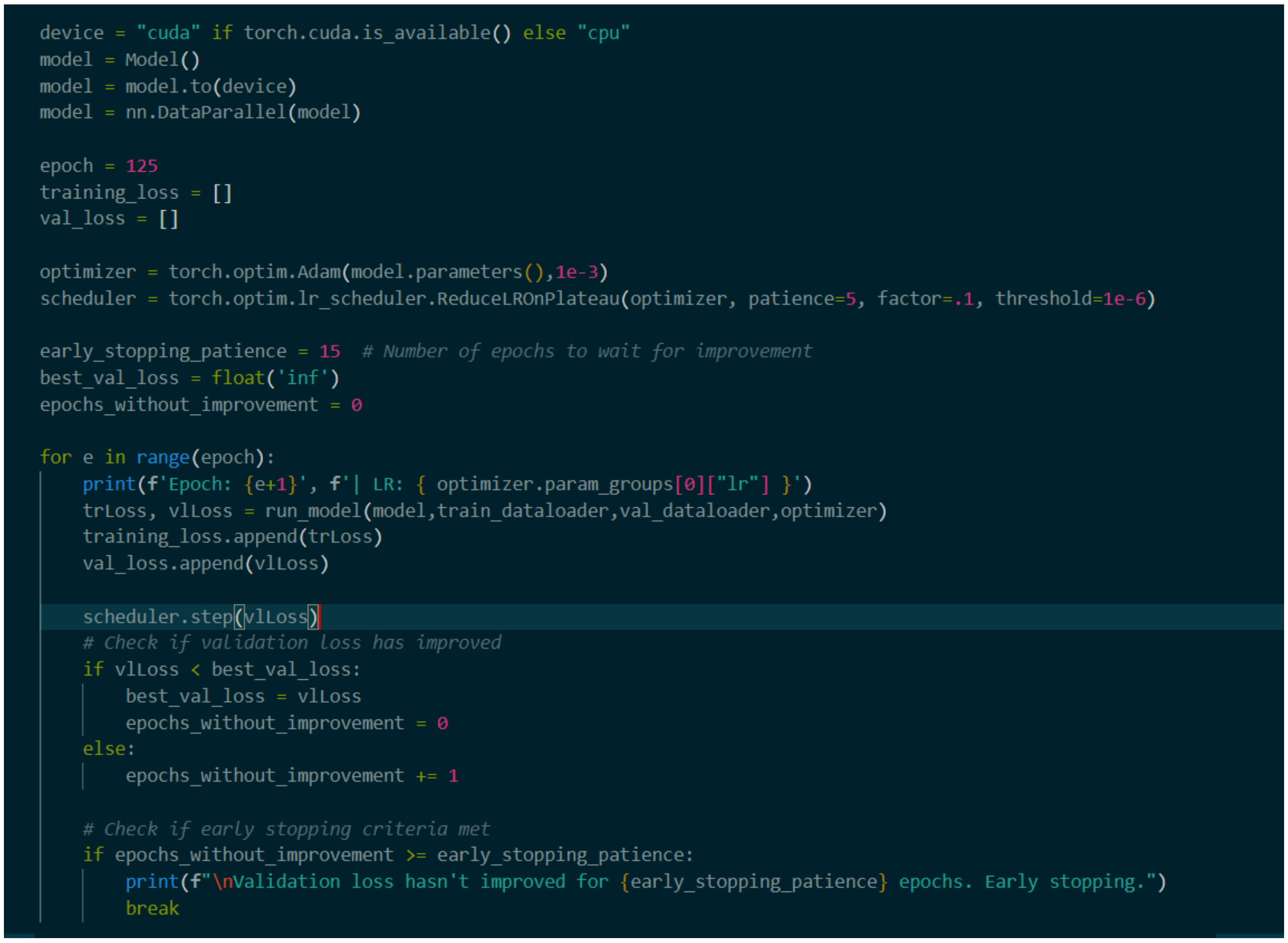
```
#         Type Accuracy
        _, predicted_labels_ty = torch.max(aux, dim=1)
        correct_ty = (predicted_labels_ty == ans_type).sum().item()
        total_correct_ty += correct_ty
        total_samples_ty += ans_type.size(0)
        train_accuracy_ty = total_correct_ty / total_samples_ty

    total_train_accuracy = (train_accuracy + train_accuracy_ty) / 2


    if val_loader is not None:
        model.eval()

    # Initialize validation-specific variables
    val_loss = 0.0
    total_correct_val=0
    total_samples_val=0

    total_correct_val_ty=0
    total_samples_val_ty=0

    # Disable gradient calculation
    with torch.no_grad():
        # Iterate over the validation data
        for (data, ans , ans_type) in val_loader:
            data = data.to(device)
            ans = ans.to(device)
            ans_type = ans_type.to(device)

            # Forward pass
            output , aux = model(data)
```



```
            # Forward pass
            output , aux = model(data)

            # Compute the loss
            loss_ans = loss(output, ans)
            loss_type = loss(aux,ans_type)
            loss_combined=loss_ans+loss_type
            val_loss += loss_combined.item()

            # Update validation metrics
            _, val_predicted = torch.max(output, dim=1)
            correct_val = (val_predicted == ans).sum().item()
            total_correct_val += correct_val
            total_samples_val += ans.size(0)
            val_accuracy = total_correct_val / total_samples_val

            _, val_predicted_ty = torch.max(aux, dim=1)
            correct_val_ty = (val_predicted_ty == ans_type).sum().item()
            total_correct_val_ty += correct_val_ty
            total_samples_val_ty += ans_type.size(0)
            val_accuracy_ty = total_correct_val_ty / total_samples_val_ty

        total_val_accuracy = (val_accuracy + val_accuracy_ty) / 2

    train_loss = total_loss/len(dataloader)
    val_loss = val_loss/len(val_loader)

    print(f"\nTrain Loss: {train_loss:.4f} | AVG Train ACC: {total_train_accuracy * 100:.4f}% | Val Loss: {val_loss:.4f} | AVG Val ACC: {total_val_accuracy * 100:.2f}%")

    print(f"\nTrain ANS ACC: {train_accuracy * 100:.4f}% | VAL ANS ACC: {val_accuracy * 100:.4f}% | Train TYPE ACC: {train_accuracy_ty * 100:.4f}% | VAL TYPE ACC: {val_accuracy_ty *

    return train_loss, val_loss
```

- This defines a function run-model that is responsible for training and evaluating the model using the given data loaders.

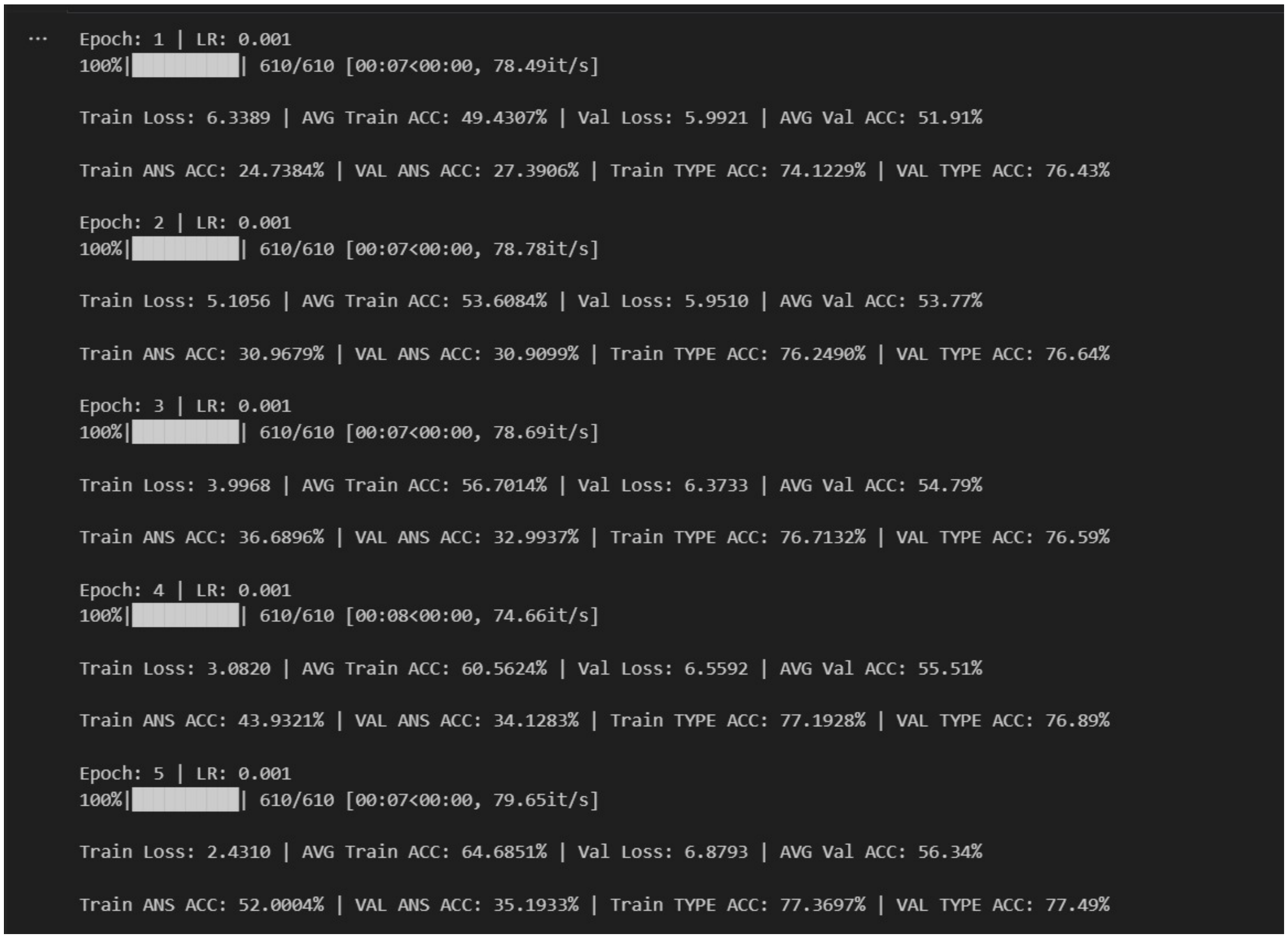
- if train:: the model is set to train mode using model.train(). This ensures that the model's parameters are set for training and enables operations such as dropout.
- loss = nn.CrossEntropyLoss(): it defines the cross-entropy loss function, which is commonly used for multi-class classification problems.
- total-loss, total-correct, total-samples, total-correct-ty, total-samples-ty: These variables are used to keep track of the loss, correct predictions, and total samples for the training phase.
- for (data, ans, ans-type) in tqdm(dataloader): This loop iterates over the batches in the given data loader (dataloader). Each batch consists of data, ans, and ans-type tensors.
- Then move the data, answer, and answer type tensors to the device for efficient computation, assuming device has been defined earlier.
- optimizer.zero-grad(): clears the gradients of all optimized parameters before performing a backward pass.
- output, aux = model(data): performs the forward pass of the model, producing the output logits and auxiliary output.
- Then compute the individual losses for the answer and answer type predictions, as well as the combined loss. And accumulates the total loss by adding the current batch loss to it.
- loss-combined.backward(): it performs backpropagation by computing gradients of the loss with respect to the model's parameters.
- optimizer.step(): it updates the model's parameters using the computed gradients and the chosen optimizer.
- calculate the number of correct predictions and update the total-correct and total-samples variables for answer accuracy.And calculate the number of correct predictions and update the total-correct-ty and total-samples-ty variables for answer type accuracy.
- Then computes the average of answer accuracy and answer type accuracy to obtain the total training accuracy.
- if val-loader is not None:: This condition checks if a validation data loader is provided.
- model.eval(): This line sets the model to evaluation mode.This disables operations like dropout.



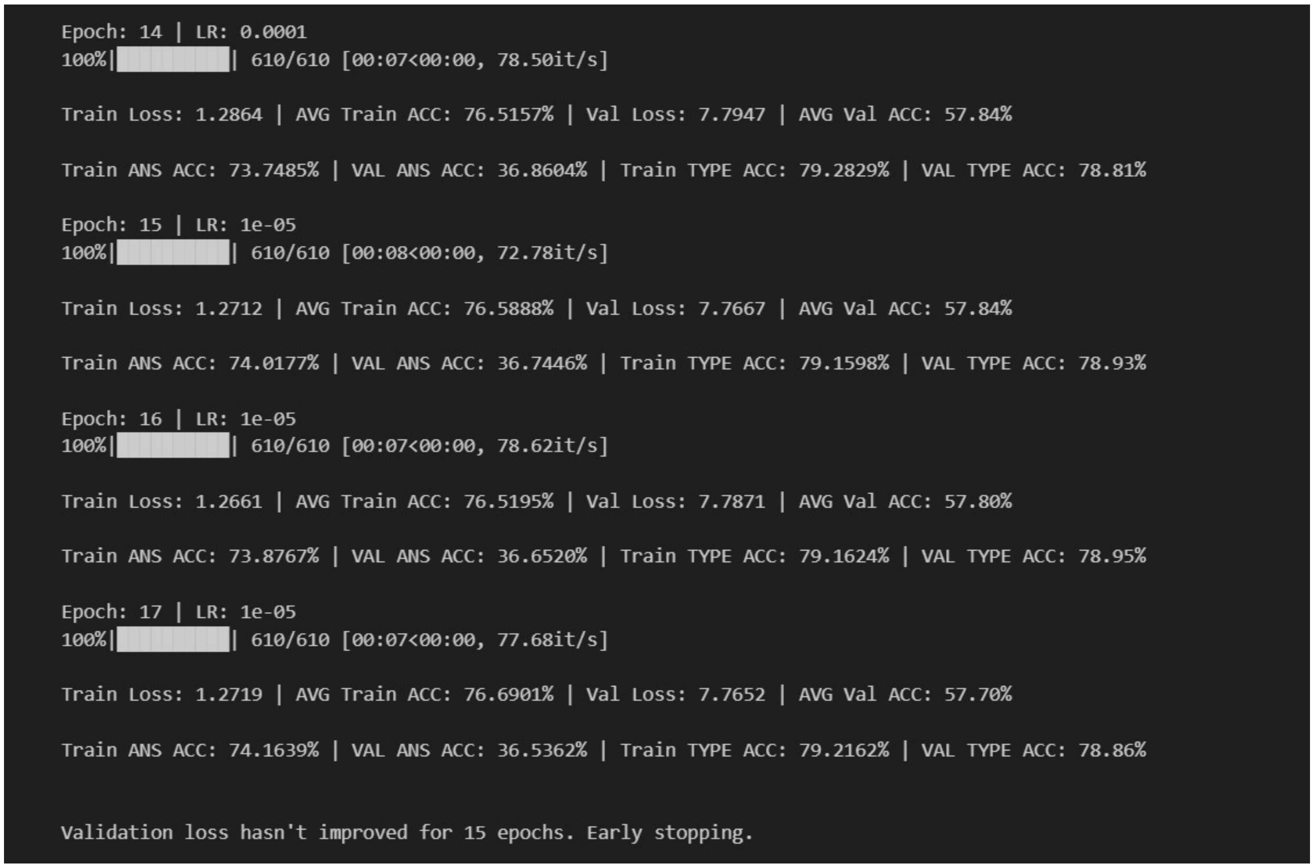
```python
device = "cuda" if torch.cuda.is_available() else "cpu"
model = Model()
model = model.to(device)
model = nn.DataParallel(model)

epoch = 125
training_loss = []
val_loss = []

optimizer = torch.optim.Adam(model.parameters(),1e-3)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=5, factor=.1, threshold=1e-6)

early_stopping_patience = 15  # Number of epochs to wait for improvement
best_val_loss = float('inf')
epochs_without_improvement = 0

for e in range(epoch):
    print(f'Epoch: {e+1}', f'| LR: { optimizer.param_groups[0]["lr"] }')
    trLoss, vlLoss = run_model(model,train_dataloader,val_dataloader,optimizer)
    training_loss.append(trLoss)
    val_loss.append(vlLoss)

    scheduler.step(vlLoss)
    # Check if validation loss has improved
    if vlLoss < best_val_loss:
        best_val_loss = vlLoss
        epochs_without_improvement = 0
    else:
        epochs_without_improvement += 1

    # Check if early stopping criteria met
    if epochs_without_improvement >= early_stopping_patience:
        print(f"\nValidation loss hasn't improved for {early_stopping_patience} epochs. Early stopping.")
        break
```

- device = ”cuda” if torch.cuda.is-available() else ”cpu”: This line checks if a CUDA-enabled GPU is available and assigns the appropriate device (cuda) or falls back to CPU (cpu) if not.
- then instantiate the Model, move it to the specified device, and wrap it with nn.DataParallel for parallel computation if multiple GPUs are available.
- epoch = 125: the total number of epochs for training.
- training-loss, val-loss: These lists are initialized to store the training and validation losses for each epoch.
- we define the Adam optimizer for optimizing the model’s parameters, with a learning rate of 1e-3.
- Then sets up a learning rate scheduler that reduces the learning rate by a factor of 0.1 if the validation loss plateaus for 5 consecutive epochs.
- early-stopping-patience = 15: the number of epochs to wait for improvement in validation loss before triggering early stopping.
- The for loop iterates over the specified number of epochs.
- Then calls the run-model function to train the model using the training data loader and evaluate it using the validation data loader. The returned training and validation losses are assigned to trLoss and vlLoss, respectively.

- And append the training and validation losses to the corresponding lists.
- scheduler.step(vlLoss): updates the learning rate scheduler with the current validation loss to possibly adjust the learning rate.
- Then check if the validation loss has improved compared to the best validation loss so far and update the respective variables accordingly. If the validation loss hasn't improved for a specified number of epochs, early stopping is triggered.
- If early stopping is triggered, a message is printed indicating that the validation loss hasn't improved for the specified number of epochs, and the loop is exited.
- After the loop finishes or early stopping is triggered, the training and validation losses can be accessed from the training-loss and val-loss lists, respectively.

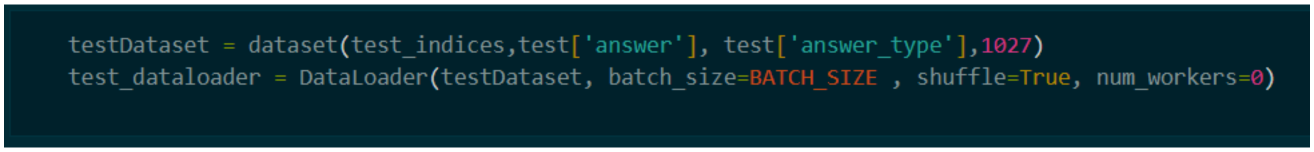


```
Epoch: 1 | LR: 0.001
100%|██████████| 610/610 [00:07<00:00, 78.49it/s]

Train Loss: 6.3389 | AVG Train ACC: 49.4307% | Val Loss: 5.9921 | AVG Val ACC: 51.91%

Train ANS ACC: 24.7384% | VAL ANS ACC: 27.3906% | Train TYPE ACC: 74.1229% | VAL TYPE ACC: 76.43%

Epoch: 2 | LR: 0.001
100%|██████████| 610/610 [00:07<00:00, 78.78it/s]

Train Loss: 5.1056 | AVG Train ACC: 53.6084% | Val Loss: 5.9510 | AVG Val ACC: 53.77%

Train ANS ACC: 30.9679% | VAL ANS ACC: 30.9099% | Train TYPE ACC: 76.2490% | VAL TYPE ACC: 76.64%

Epoch: 3 | LR: 0.001
100%|██████████| 610/610 [00:07<00:00, 78.69it/s]

Train Loss: 3.9968 | AVG Train ACC: 56.7014% | Val Loss: 6.3733 | AVG Val ACC: 54.79%

Train ANS ACC: 36.6896% | VAL ANS ACC: 32.9937% | Train TYPE ACC: 76.7132% | VAL TYPE ACC: 76.59%

Epoch: 4 | LR: 0.001
100%|██████████| 610/610 [00:08<00:00, 74.66it/s]

Train Loss: 3.0820 | AVG Train ACC: 60.5624% | Val Loss: 6.5592 | AVG Val ACC: 55.51%

Train ANS ACC: 43.9321% | VAL ANS ACC: 34.1283% | Train TYPE ACC: 77.1928% | VAL TYPE ACC: 76.89%

Epoch: 5 | LR: 0.001
100%|██████████| 610/610 [00:07<00:00, 79.65it/s]

Train Loss: 2.4310 | AVG Train ACC: 64.6851% | Val Loss: 6.8793 | AVG Val ACC: 56.34%

Train ANS ACC: 52.0004% | VAL ANS ACC: 35.1933% | Train TYPE ACC: 77.3697% | VAL TYPE ACC: 77.49%
```



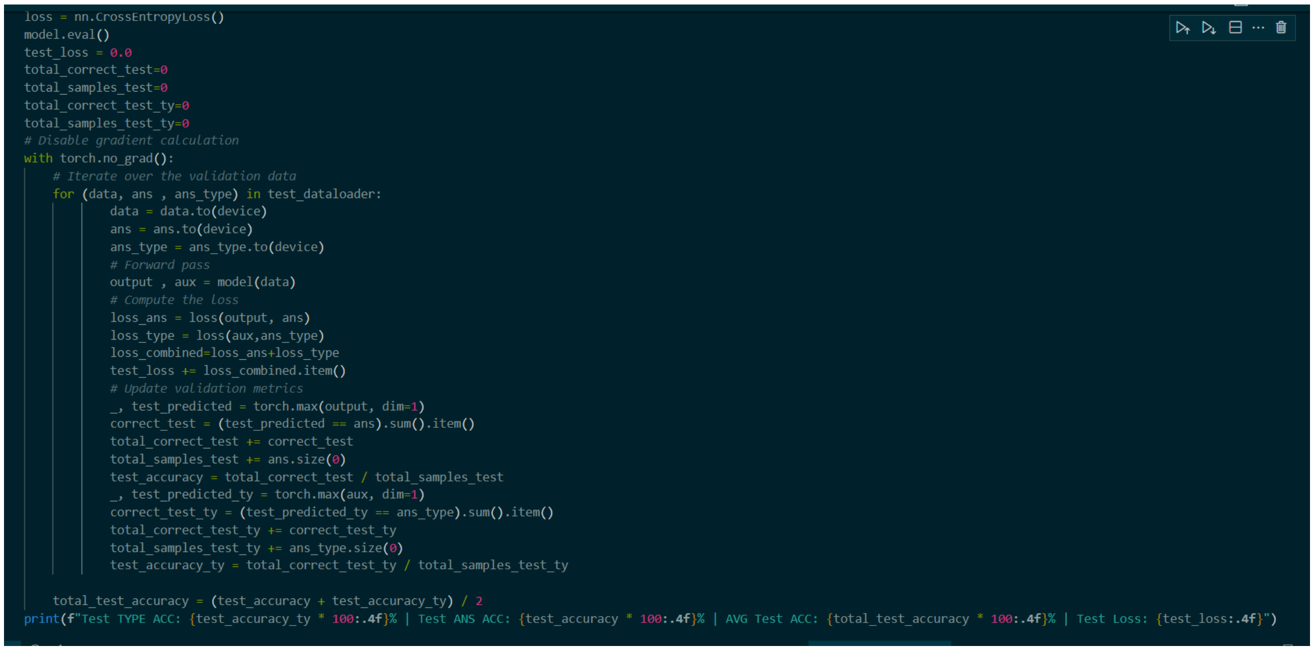
```
Epoch: 14 | LR: 0.0001
100%|██████████| 610/610 [00:07<00:00, 78.50it/s]

Train Loss: 1.2864 | AVG Train ACC: 76.5157% | Val Loss: 7.7947 | AVG Val ACC: 57.84%

Train ANS ACC: 73.7485% | VAL ANS ACC: 36.8604% | Train TYPE ACC: 79.2829% | VAL TYPE ACC: 78.81%

Epoch: 15 | LR: 1e-05
100%|██████████| 610/610 [00:08<00:00, 72.78it/s]

Train Loss: 1.2712 | AVG Train ACC: 76.5888% | Val Loss: 7.7667 | AVG Val ACC: 57.84%

Train ANS ACC: 74.0177% | VAL ANS ACC: 36.7446% | Train TYPE ACC: 79.1598% | VAL TYPE ACC: 78.93%

Epoch: 16 | LR: 1e-05
100%|██████████| 610/610 [00:07<00:00, 78.62it/s]

Train Loss: 1.2661 | AVG Train ACC: 76.5195% | Val Loss: 7.7871 | AVG Val ACC: 57.80%

Train ANS ACC: 73.8767% | VAL ANS ACC: 36.6520% | Train TYPE ACC: 79.1624% | VAL TYPE ACC: 78.95%

Epoch: 17 | LR: 1e-05
100%|██████████| 610/610 [00:07<00:00, 77.68it/s]

Train Loss: 1.2719 | AVG Train ACC: 76.6901% | Val Loss: 7.7652 | AVG Val ACC: 57.70%

Train ANS ACC: 74.1639% | VAL ANS ACC: 36.5362% | Train TYPE ACC: 79.2162% | VAL TYPE ACC: 78.86%


Validation loss hasn't improved for 15 epochs. Early stopping.
```

### 0.3.7 Testing:



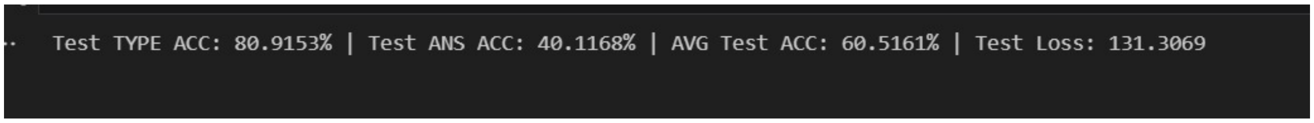
```
testDataset = dataset(test_indices,test['answer'], test['answer_type'],1027)
test_dataloader = DataLoader(testDataset, batch_size=BATCH_SIZE , shuffle=True, num_workers=0)
```

- We create an instance of the dataset class, The length of the dataset is set to 1027.
- Then create a data loader test-dataloader for the test dataset. The testDataset is passed as the first argument, and the batch size is set to BATCH-SIZE. The data is shuffled (shuffle=True), and num-workers is set to 0 for no additional subprocesses to be used for data loading.



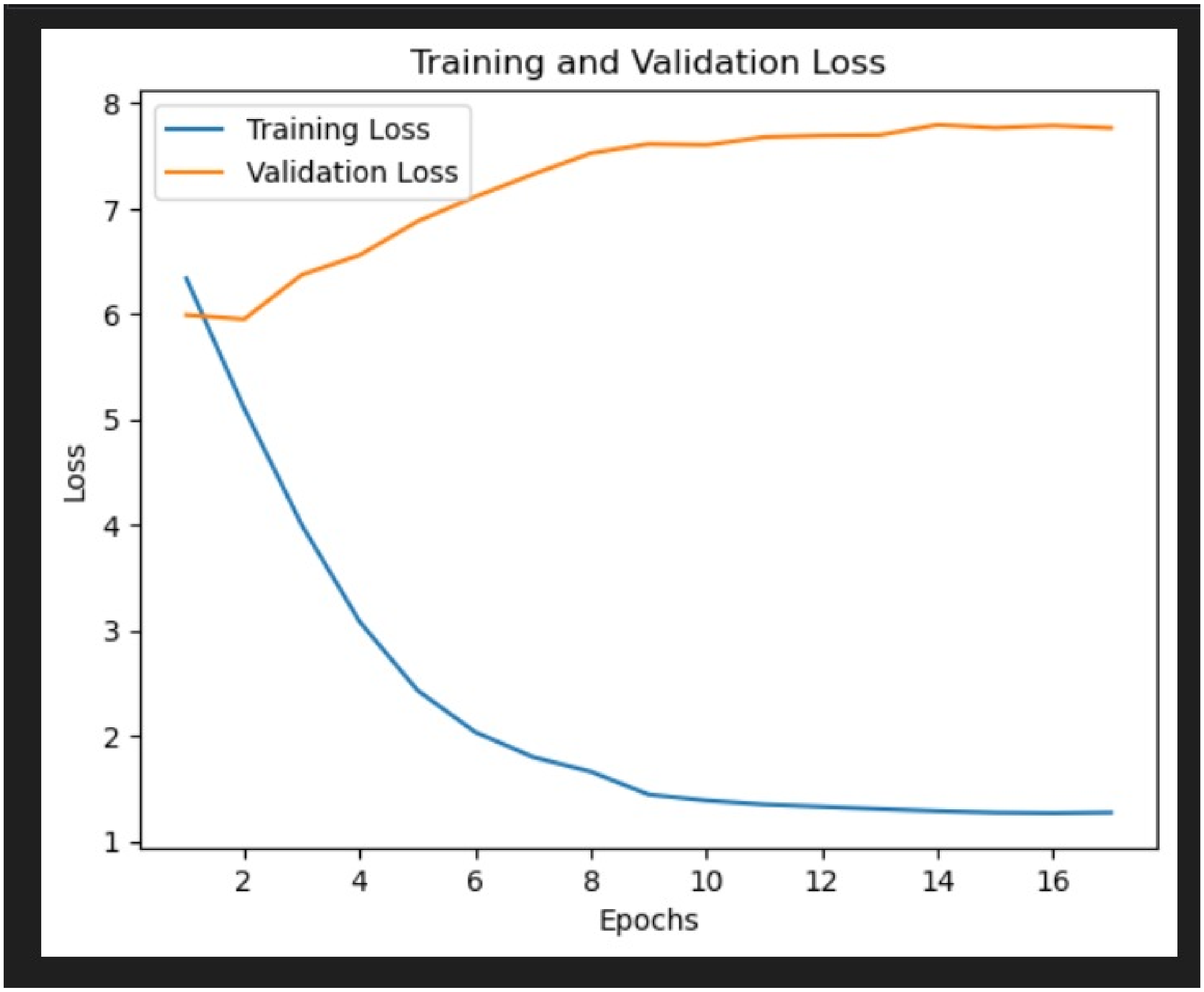
```python
loss = nn.CrossEntropyLoss()
model.eval()
test_loss = 0.0
total_correct_test=0
total_samples_test=0
total_correct_test_ty=0
total_samples_test_ty=0
# Disable gradient calculation
with torch.no_grad():
    # Iterate over the validation data
    for (data, ans , ans_type) in test_dataloader:
            data = data.to(device)
            ans = ans.to(device)
            ans_type = ans_type.to(device)
            # Forward pass
            output , aux = model(data)
            # Compute the loss
            loss_ans = loss(output, ans)
            loss_type = loss(aux,ans_type)
            loss_combined=loss_ans+loss_type
            test_loss += loss_combined.item()
            # Update validation metrics
            _, test_predicted = torch.max(output, dim=1)
            correct_test = (test_predicted == ans).sum().item()
            total_correct_test += correct_test
            total_samples_test += ans.size(0)
            test_accuracy = total_correct_test / total_samples_test
            _, test_predicted_ty = torch.max(aux, dim=1)
            correct_test_ty = (test_predicted_ty == ans_type).sum().item()
            total_correct_test_ty += correct_test_ty
            total_samples_test_ty += ans_type.size(0)
            test_accuracy_ty = total_correct_test_ty / total_samples_test_ty

    total_test_accuracy = (test_accuracy + test_accuracy_ty) / 2
print(f"Test TYPE ACC: {test_accuracy_ty * 100:.4f}% | Test ANS ACC: {test_accuracy * 100:.4f}% | AVG Test ACC: {total_test_accuracy * 100:.4f}% | Test Loss: {test_loss:.4f}")
```

- We initialize the cross-entropy loss function, which will be used to calculate the loss during evaluation.
- model.eval(): sets the model to evaluation mode, which disables certain operations like dropout and batch normalization.
- The torch.no-grad() block is used to disable gradient calculation during evaluation.
- The for loop iterates over the test data loader to evaluate the model on the test set.
- Inside the loop, the input data, answers, and answer types are moved to the appropriate device. And The forward pass is performed through the model to obtain the output and auxiliary predictions.
- The loss is computed by applying the cross-entropy loss to the output and auxiliary predictions.
- The number of correct predictions and total samples are updated based on the predicted labels and ground truth answers.
- The same process is repeated for the answer types to calculate the type accuracy.
- The average test accuracy is calculated as the average of answer accuracy and type accuracy.
- Finally, the test accuracy, type accuracy, average test accuracy, and test loss are printed.

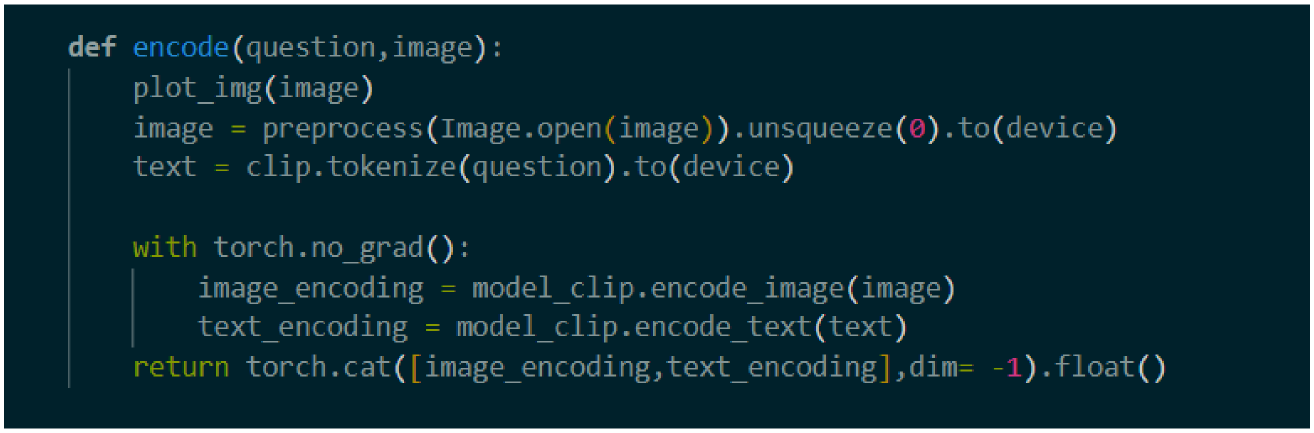


```
Test TYPE ACC: 80.9153% | Test ANS ACC: 40.1168% | AVG Test ACC: 60.5161% | Test Loss: 131.3069
```

- plot the loss:

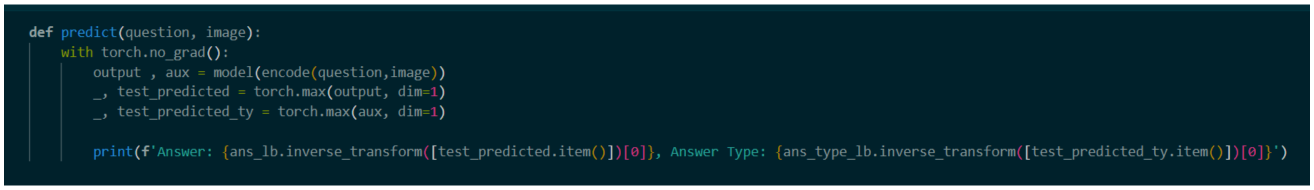


### 0.3.8 Test with unseen images:



```
def encode(question,image):
    plot_img(image)
    image = preprocess(Image.open(image)).unsqueeze(0).to(device)
    text = clip.tokenize(question).to(device)

    with torch.no_grad():
        image_encoding = model_clip.encode_image(image)
        text_encoding = model_clip.encode_text(text)
    return torch.cat([image_encoding,text_encoding],dim= -1).float()
```

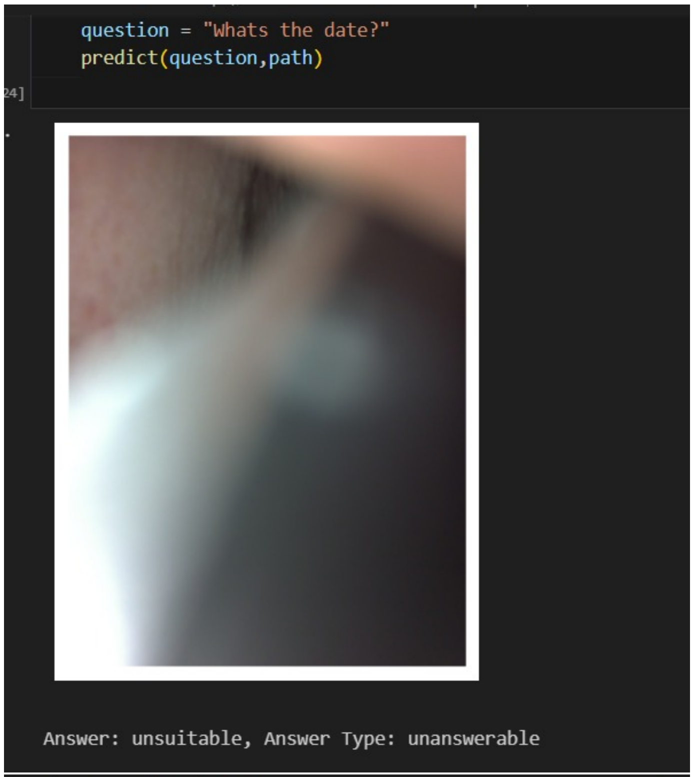
- first we open the image using PIL's Image.open function, preprocesses it, and converts it to a tensor. The image tensor is then unsqueezed to add a batch dimension and moved to the appropriate device.
- Then tokenizes the question using the tokenize method of the CLIP model's tokenizer. The resulting tokens are moved to the appropriate device.
- The with torch.no-grad() block is used to disable gradient calculation during the encoding process.
- Then passe the preprocessed image tensor through the CLIP model's encode-image method to obtain the encoded representation of the image. And passes the tokenized question through the CLIP model's encode-text method to obtain the encoded representation of the text.
- Finally, concatenates the image and text encodings along the last dimension (-1) and converts the result to float. The concatenated encoding is then returned.



```
def predict(question, image):
    with torch.no_grad():
        output , aux = model(encode(question,image))
        _, test_predicted = torch.max(output, dim=1)
        _, test_predicted_ty = torch.max(aux, dim=1)

        print(f'Answer: {ans_lb.inverse_transform([test_predicted.item()])[0]}, Answer Type: {ans_type_lb.inverse_transform([test_predicted_ty.item()])[0]}')
```

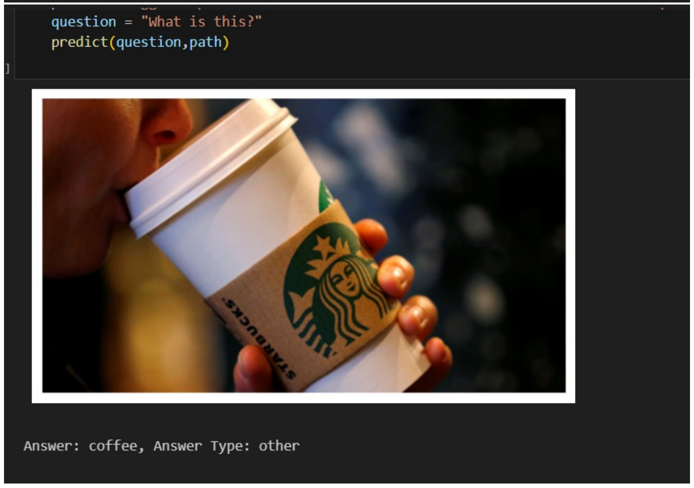
- output, aux = model(encode(question, image)): This line encodes the question and image using the encode function and passes the encoded representation through the trained model to obtain the output and auxiliary predictions.
- Then extract the predicted labels from the output predictions by taking the maximum value along the appropriate dimension.
- And extract the predicted labels for the answer types from the auxiliary predictions by taking the maximum value along the appropriate dimension.
- Finally, prints the predicted answer and answer type by inverse transforming the predicted labels using the inverse-transform method of the corresponding label binarizer.

## 0.4 Results:



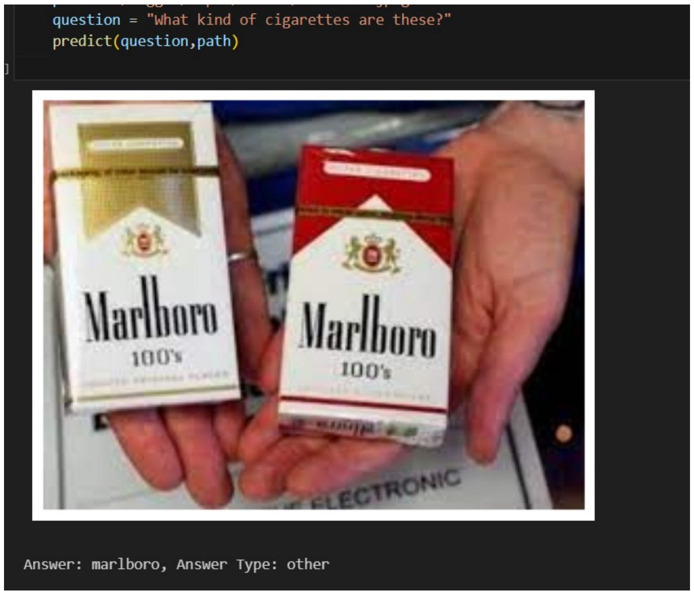
```
question = "Whats the date?"
predict(question,path)
```

```
Answer: unsuitable, Answer Type: unanswerable
```



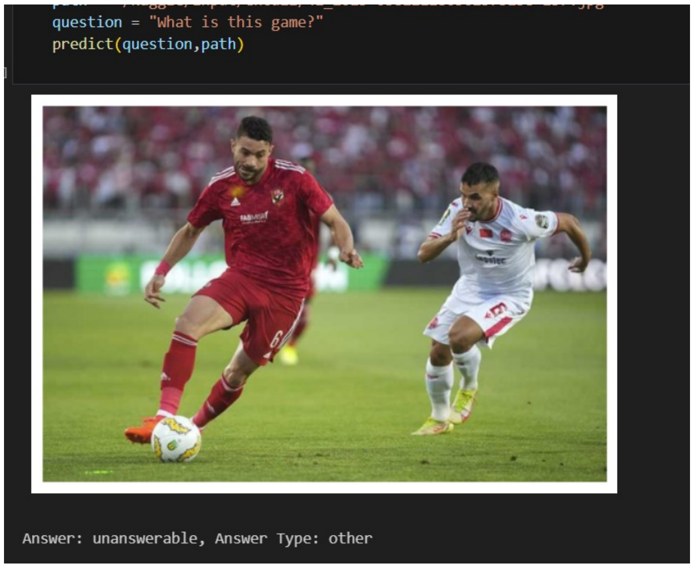
```
question = "What is this?"
predict(question,path)
```

```
Answer: coffee, Answer Type: other
```



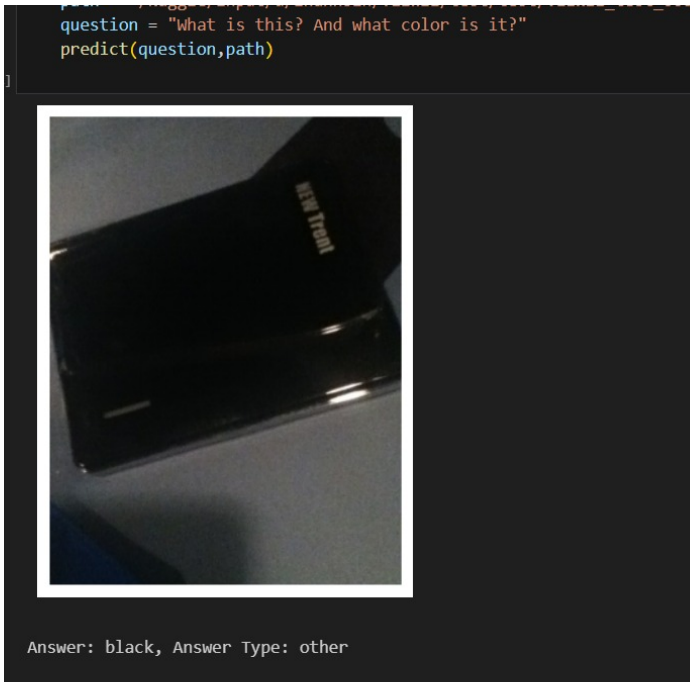
```
    question = "What kind of cigarettes are these?"
    predict(question,path)
```

```
Answer: marlboro, Answer Type: other
```



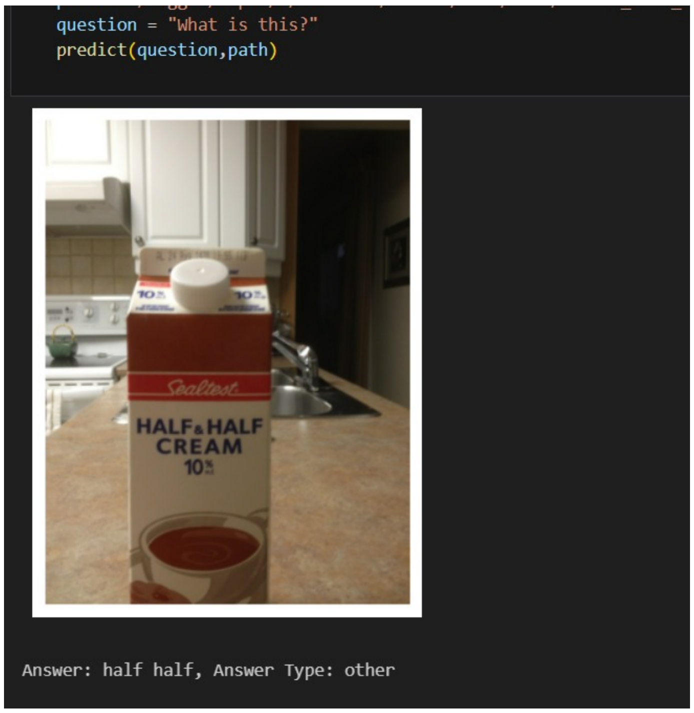
```
    question = "What is this game?"
    predict(question,path)
```

```
Answer: unanswerable, Answer Type: other
```



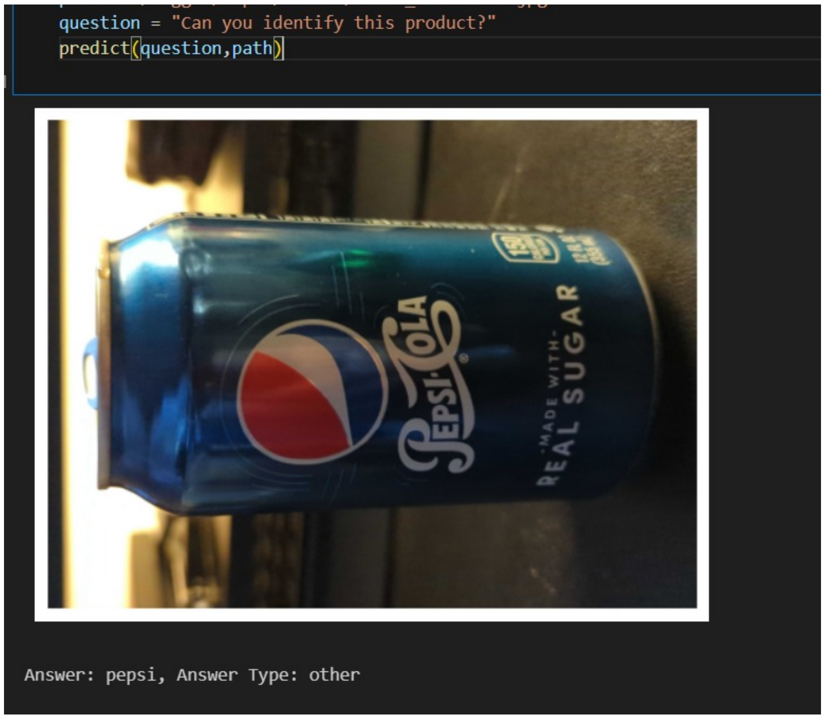
```
question = "What is this? And what color is it?"
predict(question,path)
```

```
Answer: black, Answer Type: other
```



```
question = "What is this?"
predict(question,path)
```

```
Answer: half half, Answer Type: other
```



```
question = "Can you identify this product?"
predict(question,path)
```

```
Answer: pepsi, Answer Type: other
```