16824 Homework 3

Jiajun Wan
Andrew ID: jiajunw2

Task 1: Understanding VQA

1.1 Why do you think that it's computationally costly to use all answers as separate classes instead of keeping only the most frequent ones?

We have to use a large number of output neurons, that add computations on final sigmoid/softmax calculation, loss computation, and backward propagation. Keeping only the most frequent ones will largely reduce the amount of calculations.

1.2 Complete the __init__ method of the dataset class for VQA in vqa_dataset.py. Specifically, initialize the VQA API and anything you need from that.

```
self._vqa = VQA(annotation_file=annotation_json_file_path,
                question_file=question_json_file_path)
```

1.3 Implement the __len__ method of the VQADataset class. Should the size of the dataset be equal to the number of images, questions or the answers?

```
def __len__(self):
    # total number of questions
    return len(self._vqa.qqa)
```

The size of the dataset be equal to the number of questions.

1.4 Complete the __getitem__ method

```
q_anno = self._vqa.qa[idx]  # load annotation
q_str = self._vqa.qqa[idx]['question']  # question in str format
```

Task 2: Building a pipeline for VQA

2.1 What should be the output dimension of the trainable linear layer? Complete the corresponding part in the __init__ method of BaselineNet in models.py.

```
self.classifier = nn.Linear(
    self.text_encoder.config.hidden_size + 512,
    n_answers,
)
```

It should be n_answers (5127)

2.2 Implement compute_vis_feats that featurizes the image (it can be implemented as an one-liner!).

```
def compute_vis_feats(self, image):
    """Convert image tensors to feature tensors."""
    return torch.nn.functional.adaptive_avg_pool2d(self.vis_encoder(image), 1).
    ↪ squeeze() # feed to vis_encoder and then mean pool on spatial dims
```

2.3 Implement the forward pass of BaselineNet. Make sure to use compute_vis_feats and compute_text_feats.

```
def forward(self, image, question):
    """Forward pass, image (B, 3, 224, 224), qs list of str."""
    text_feats = self.compute_text_feats(question)
    vis_feats = self.compute_vis_feats(image)
    concate_feats = torch.hstack((text_feats, vis_feats))
    logits = self.classifier(concate_feats)
    return logits
```

2.4 What is the loss for multi-label classification? (Hint: in HW1 you also tackled a multi-class classification problem)

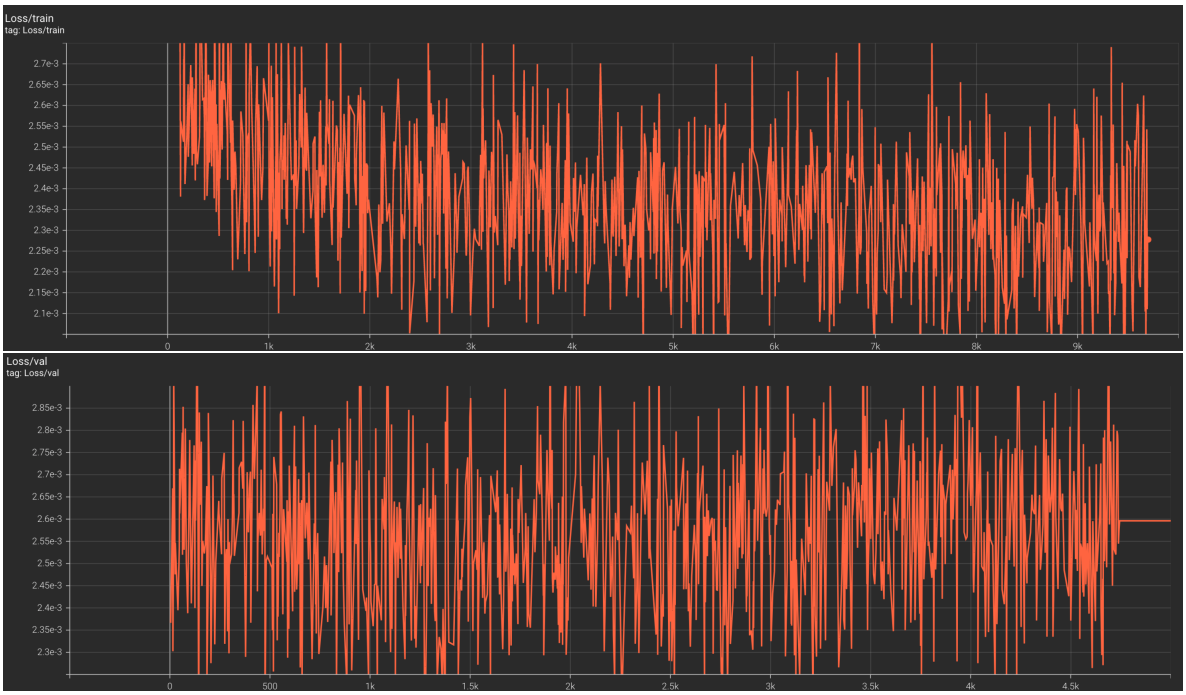
Binary Cross-Entropy Loss

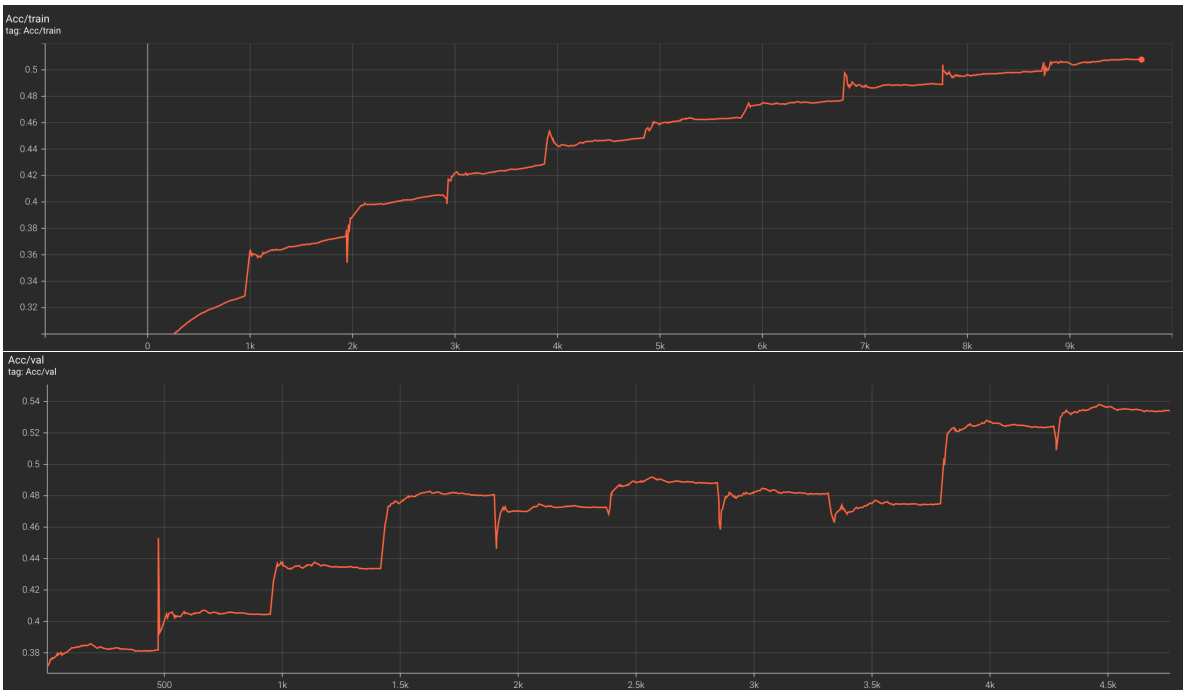
2.5 Implement the loss call and the optimization code in the train_test_loop method.

```
loss = F.binary_cross_entropy_with_logits(scores, answers)

# Update
if mode == 'train':
    # optimize loss
    loss.backward()
    self.optimizer.step()
    self.optimizer.zero_grad()
```

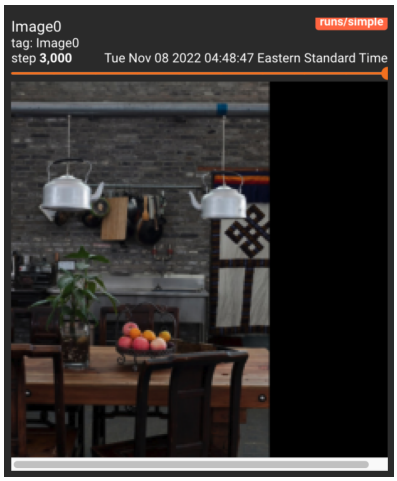
2.6 Complete the train_test_loop method to monitor the performance in Tensorboard. Plot the loss and accuracy and include these in your report. Additionally show multiple image-question pairs (at least 3) with the respective answers (predicted and ground-truth).

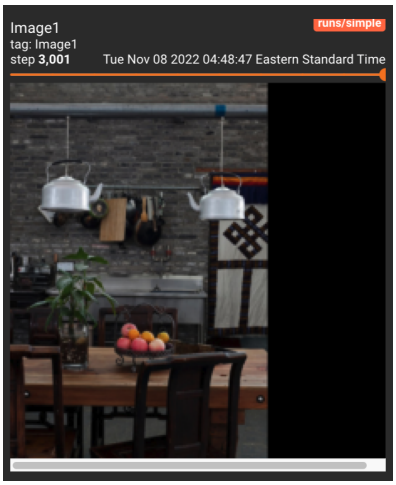
```
    # add code to show the question
    self.writer.add_text(
        'Question%d' % i, data['question'][i],
        epoch * _n_show + i
    )
    # the gt answer
    self.writer.add_text(
        'GT Answer%d' % i, self._id2answer[(data['answers'][i] == 1).nonzero(
    ↪ as_tuple=True)[0][0].item()],
        epoch * _n_show + i
    )
    # and the predicted answer
    self.writer.add_text(
        'Predicted Answer%d' % i, self._id2answer[scores.argmax(1)[i].item()],
        epoch * _n_show + i
    )
# add code to plot the current accuracy
self.writer.add_scalar(
'Acc/' + mode, n_correct / n_samples,
epoch * len(self.data_loaders[mode]) + step
)
```

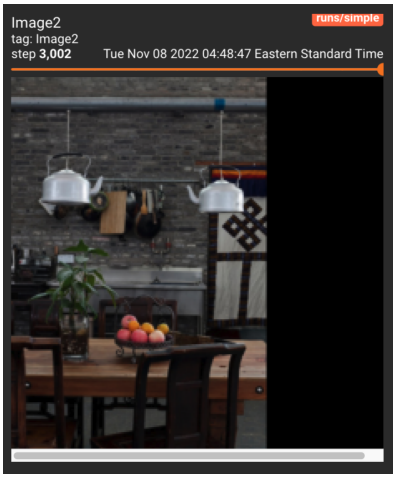




Final Val Acc: 0.5341694647442228







GT Answer0

GT Answer0/text_summary
tag: GT Answer0/text_summary

step 3,000

wood

GT Answer1

GT Answer1/text_summary
tag: GT Answer1/text_summary

step 3,001

yes

GT Answer2

GT Answer2/text_summary
tag: GT Answer2/text_summary

step 3,002

kettle

Predicted Answer0

Predicted Answer0/text_summary
tag: Predicted Answer0/text_summary

step 3,000

Other

Predicted Answer1

Predicted Answer1/text_summary
tag: Predicted Answer1/text_summary

step 3,001

yes

Predicted Answer2

Predicted Answer2/text_summary
tag: Predicted Answer2/text_summary

step 3,002

Other

Question0

Question0/text_summary
tag: Question0/text_summary

step 3,000

What is the table made of?

Question1

Question1/text_summary
tag: Question1/text_summary

step 3,001

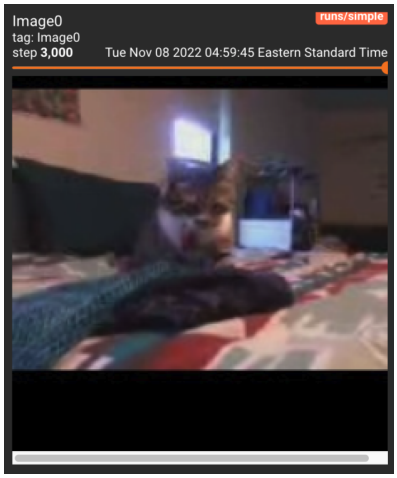
Is the food napping on the table?

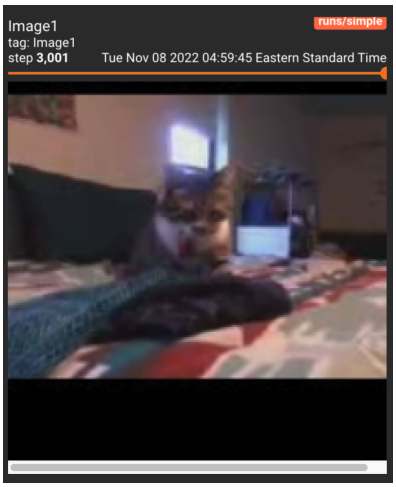
Question2

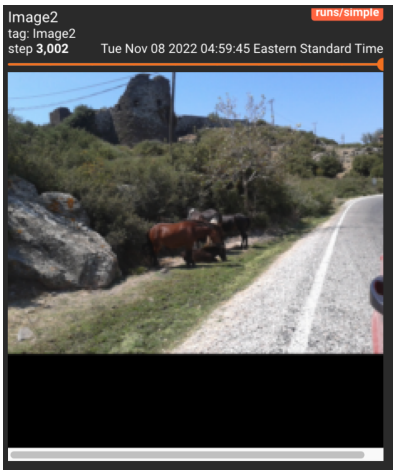
Question2/text_summary
tag: Question2/text_summary

step 3,002

What has been upcycled to make lights?







GT Answer0

GT Answer0/text_summary
tag: GT Answer0/text_summary

step 3,000

camera

GT Answer1

GT Answer1/text_summary
tag: GT Answer1/text_summary

step 3,001

no

GT Answer2

GT Answer2/text_summary
tag: GT Answer2/text_summary

step 3,002

3

Predicted Answer0

Predicted Answer0/text_summary
tag: Predicted Answer0/text_summary

step 3,000

yes

Predicted Answer1

Predicted Answer1/text_summary
tag: Predicted Answer1/text_summary

step 3,001

yes

Predicted Answer2

Predicted Answer2/text_summary
tag: Predicted Answer2/text_summary

step 3,002

2

Question0

Question0/text_summary
tag: Question0/text_summary

step 3,000

What is causing the picture to be blurry?

Question1

Question1/text_summary
tag: Question1/text_summary

step 3,001

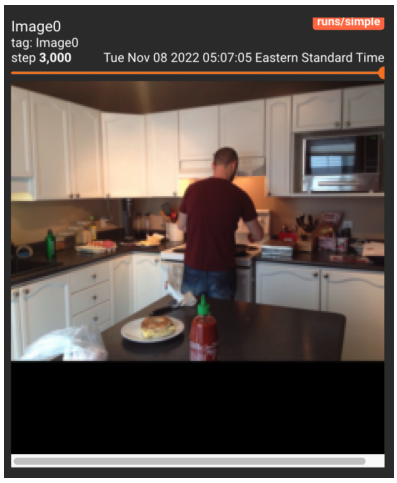
Is it night time?

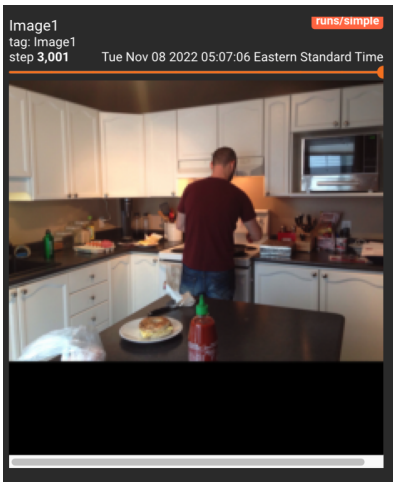
Question2

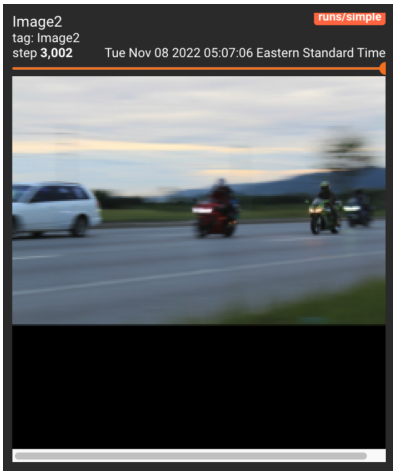
Question2/text_summary
tag: Question2/text_summary

step 3,002

How many horses are shown?







GT Answer0

GT Answer0/text_summary
tag: GT Answer0/text_summary

step 3,000

ketchup

GT Answer1 GT Answer0

GT Answer1/text_summary
tag: GT Answer1/text_summary

step 3,001

yes

GT Answer2

GT Answer2/text_summary
tag: GT Answer2/text_summary

step 3,002

yes

Predicted Answer0

Predicted Answer0/text_summary
tag: Predicted Answer0/text_summary

step 3,000

kitchen

Predicted Answer1

Predicted Answer1/text_summary
tag: Predicted Answer1/text_summary

step 3,001

kitchen

Predicted Answer2

Predicted Answer2/text_summary
tag: Predicted Answer2/text_summary

step 3,002

2

Question0

Question0/text_summary
tag: Question0/text_summary

step 3,000

What is in the bottle on the kitchen Isle?

Question1

Question1/text_summary
tag: Question1/text_summary

step 3,001

Is the sauce spicy?

Question2

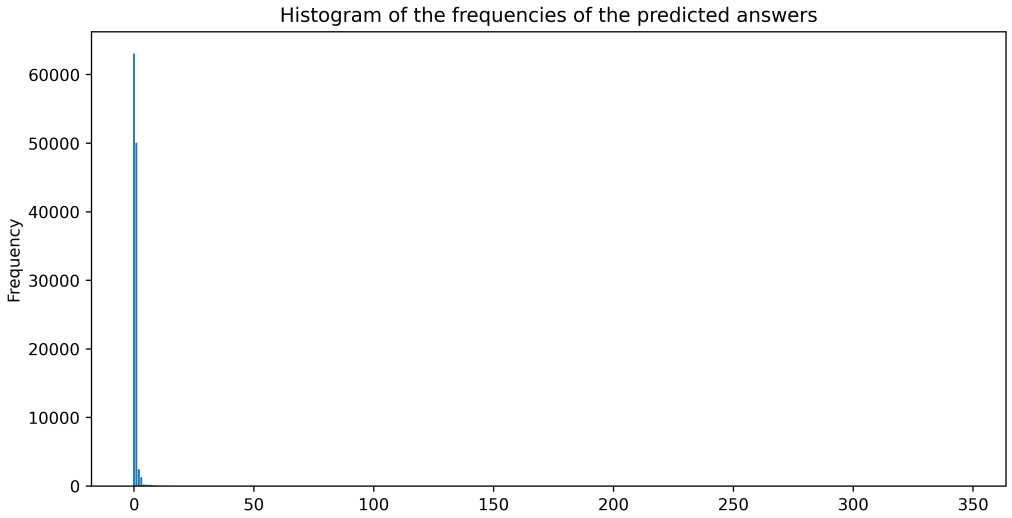
Question2/text_summary
tag: Question2/text_summary

step 3,002

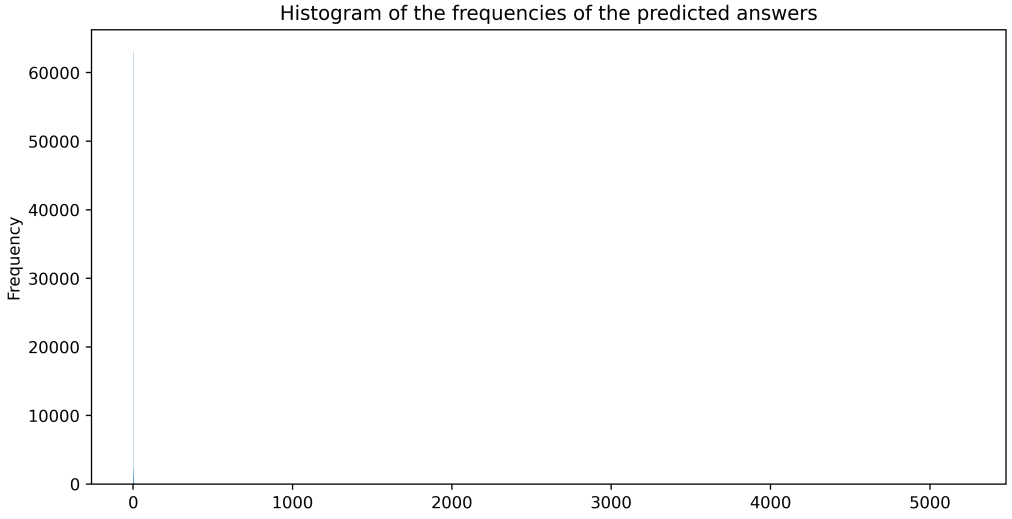
How many headlights are visible?

2.7 Make a histogram of the frequencies of the predicted answers (all 5217) without labels.

Show only the label with predictions



Show all the 5217 labels



2.8 What are the 10 most frequently predicted classes? Why do you think this happens? You can optionally visualize the frequency of ground-truth answers in the training set.

10 most frequent classes are: ['Other', 'yes', '2', 'no', 'baseball', 'broccoli', 'red', 'sheep', '3', 'surfing']
Because network is not expressive enough to capture the causal factors the data, it will try to capture the output distribution. That is, because the output distribution, the distribution of the labels in the

training dataset, is largely skewed, having a majority of "other", "yes", and so on as we can see in the histogram, the network will try to predict these top frequent classes as much as possible to get better accuracy.

Task 3: Transformer Network

3.1 Complete the CrossAttentionLayer

```python
class CrossAttentionLayer(nn.Module):
    """Self-/Cross-attention between two sequences."""

    def __init__(self, d_model=256, dropout=0.1, n_heads=8):
        """Initialize layers, d_model is the encoder dimension."""
        super().__init__()

        # Self-attention for seq1
        self.sa1 = nn.MultiheadAttention(
            d_model, n_heads, dropout=dropout, batch_first=True
        )  # use batch_first=True everywhere!
        self.dropout_1 = nn.Dropout(dropout)
        self.norm_1 = nn.LayerNorm(d_model)

        # Self-attention for seq2
        self.sa2 = nn.MultiheadAttention(
            d_model, n_heads, dropout=dropout, batch_first=True
        )
        self.dropout_2 = nn.Dropout(dropout)
        self.norm_2 = nn.LayerNorm(d_model)

        # Cross attention from seq1 to seq2
        self.cross_12 = nn.MultiheadAttention(
            d_model, n_heads, dropout=dropout, batch_first=True
        )
        self.dropout_12 = nn.Dropout(dropout)
        self.norm_12 = nn.LayerNorm(d_model)

        # FFN for seq1
        self.ffn_12 = nn.Sequential(
            nn.Linear(d_model, 1024),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(1024, d_model),
            nn.Dropout(dropout),
        )
        self.norm_122 = nn.LayerNorm(d_model)

        # Cross attention from seq2 to seq1
        self.cross_21 = nn.MultiheadAttention(
            d_model, n_heads, dropout=dropout, batch_first=True
        )
```

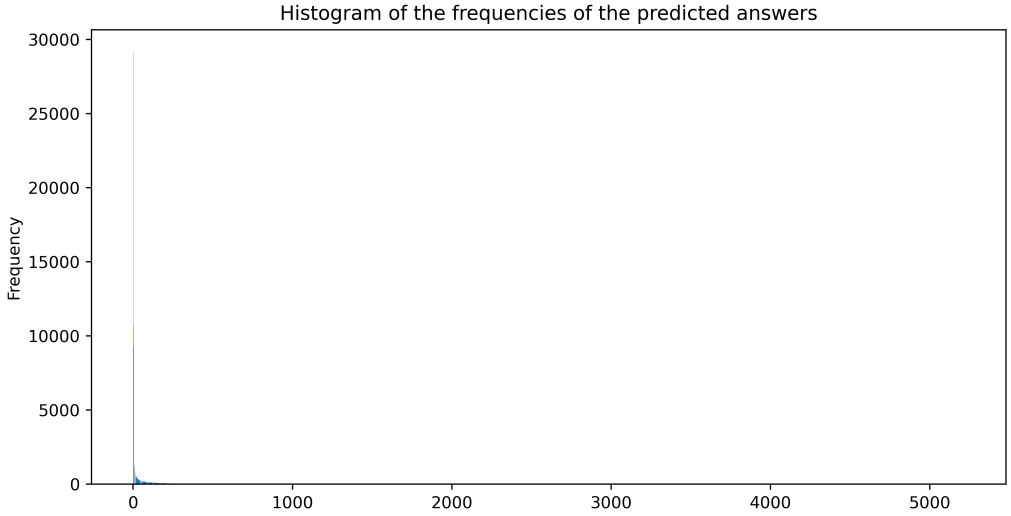
```
        self.dropout_21 = nn.Dropout()
        self.norm_21 = nn.LayerNorm(d_model)

        # FFN for seq2
        self.ffn_21 = nn.Sequential(
            nn.Linear(d_model, 1024),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(1024, d_model),
            nn.Dropout(dropout),
        )
        self.norm_212 = nn.LayerNorm(d_model)

    def forward(self, seq1, seq1_key_padding_mask, seq2,
                seq2_key_padding_mask,
                seq1_pos=None, seq2_pos=None):
        """Forward pass, seq1 (B, S1, F), seq2 (B, S2, F)."""
        # Self-attention for seq1
        q1 = k1 = v1 = seq1
        if seq1_pos is not None:
            q1 = q1 + seq1_pos
            k1 = k1 + seq1_pos
        seq1b = self.sa1(
            query=q1,
            key=k1,
            value=v1,
            attn_mask=None,
            key_padding_mask=seq1_key_padding_mask  # (B, S1)
        )[0]
        seq1 = self.norm_1(seq1 + self.dropout_1(seq1b))

        # Self-attention for seq2
        q2 = k2 = v2 = seq2
        if seq2_pos is not None:
            q2 = q2 + seq2_pos
            k2 = k2 + seq2_pos
        seq2b = self.sa2(
            query=q2,
            key=k2,
            value=v2,
            attn_mask=None,
            key_padding_mask=seq2_key_padding_mask  # (B, S2)
        )[0]
        seq2 = self.norm_2(seq2 + self.dropout_2(seq2b))

        # Create key, query, value for seq1, seq2
        q1 = k1 = v1 = seq1
        q2 = k2 = v2 = seq2
        if seq1_pos is not None:
            q1 = q1 + seq1_pos
            k1 = k1 + seq1_pos
```

```
        if seq2_pos is not None:
            q2 = q2 + seq2_pos
            k2 = k2 + seq2_pos

        # Cross-attention from seq1 to seq2 and FFN
        seq1b = self.cross_12(
            query=q1,
            key=k2,
            value=v2,
            attn_mask=None,
            key_padding_mask=seq2_key_padding_mask  # (B, S2)
        )[0]
        seq1 = self.norm_12(seq1 + self.dropout_12(seq1b))

        # FFN for seq1
        seq1 = self.norm_122(seq1 + self.ffn_12(seq1))

        # Cross-attention from seq2 to seq1 and FFN
        seq2b = self.cross_21(
            query=q2,
            key=k1,
            value=v1,
            attn_mask=None,
            key_padding_mask=seq1_key_padding_mask  # (B, S1)
        )[0]
        seq2 = self.norm_21(seq2 + self.dropout_21(seq2b))

        # FFN for seq2
        seq2 = self.norm_212(seq2 + self.ffn_21(seq2))

        return seq1, seq2
```

3.2 Load the trained weights and reproduce our result.

Final Val Acc: 0.6762130489169794

3.3 How does the histogram of answers look now?



The histogram of answers look much better now with a peak that is much lower, more different sub-peaks, and more less frequent answers (not appearing just once). The top answer is "yes", not "other" in previous task.