Task 1: Understanding VQA (20 points)

Based on the above description:

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?

Since we posed this as a classification problem, the problem of using all the classes as the number of possible classification is that you will end up with number classes equal to the largest possible number of answers in any given training instance. Not only is that much less computationally efficient, it creates a very sparse output for most other instances when they are mostly non-frequent.

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.

[code implemented]

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?

It should be equal to the number of image-questions-answer pairs, which is also the same as the number of answers, since we have a question to answer pairs, and multiple possible questions for the same question.

1.4 Complete the __getitem__ method. You need to (1) figure out what is the idx-th item of the dataset; (2) load the question string. To make sure that your results are comparable to the TAs' (and your peers') implementation, we have included the image preprocessing function. The __getitem__ method returns the question and list of answers, the preprocessed (normalized) image as well as the original image for visualization purposes.

[code implemented]

Additionally, the VQA dataset contains 10 answers per question. These are usually synonyms/paraphrases (or 'other', if they're very rare). Our dataset class considers all of them and returns a binary vector indicating the validity of each answer (field answers).

Task 2: Building a pipeline for VQA (50 points)

To build a learning pipeline we need i) a dataset class (we have already built it!), ii) a model that absorbs the data and returns some predictions, iii) a training pipeline that optimizes these predictions and iv) metrics that evaluate the predictions.

Model

As you've already figured out, a VQA model for this assignment is simply a classifier over 5217 classes. The first baseline is very simple. We featurize the image using a frozen pre-trained ResNet18. For that, we feed the image to ResNet18 except for the last layer which gives the classification logits. This gives us a 1-d feature vector for the image. We also featurize the question into an 1-d vector using a frozen pre-trained RoBERTa, a Transformer-based language model. Finally, we concatenate the two representations and feed to a linear layer. The parameters of this linear layer are trainable. The code for loading the pre-trained models is given. The forward pass of the language model is also given.

Based on the above description:

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.

The output dimension should be 5216+1 including an additional class for all the others, thus a total of 5217 classes.

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

[code implemented]

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

[code implemented]

Training loop

We provide skeleton code in main.py. We use Adam optimizer, although you can experiment with different optimizers and hyperparameter values. As mentioned earlier, the VQA dataset contains 10 synonym/paraphrased/duplicate answers per question. A standard softmax classification would create competition among such answers. To avoid that, we treat all of them as correct.

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

The loss is (binary) cross entropy loss where the activation in the last layer is sigmoidal as multiple answers are possible (we don’t restrict). Therefore, we can use the class Binary Cross Entropy with logit loss.

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

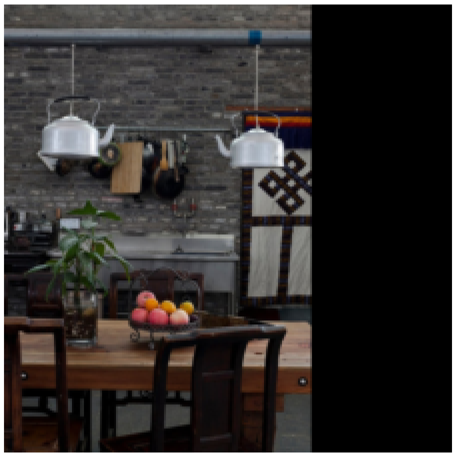
[code implemented]

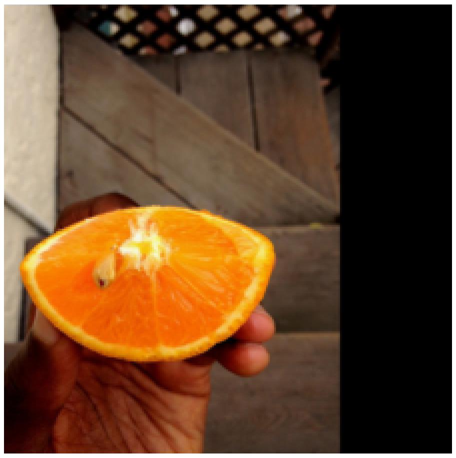
Evaluation

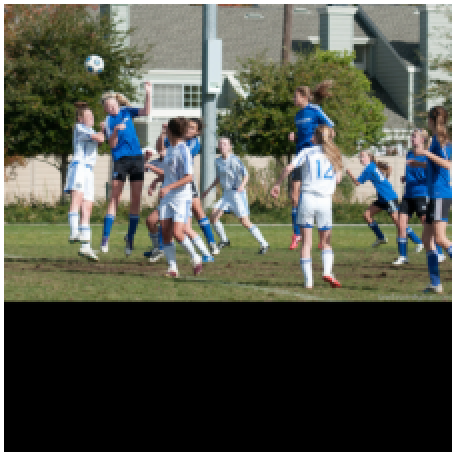
The prediction is considered accurate if the most confident answer is correct, i.e. if it is one of the multiple correct answers. We provide the code to compute accuracy.

2.6 Complete the train_test_loop method to monitor the performance in Tensorboard. Plot the loss, accuracy and multiple images (at least 3) with the respective questions and answers (predicted and ground-truth). Include all plots in your report.

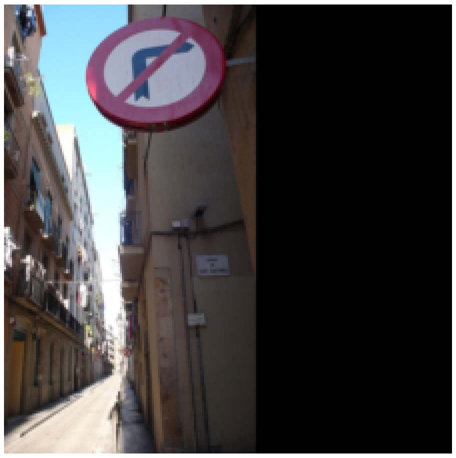
[Image 1]


[Correctly Predicted Question-answer pairs]
Question: Is the food napping on the table? Correct Answers:yes
Question:What has been upcycled to make lights? Correct Answers:Other

[Image 2]

[Correctly Predicted Question-answer pairs]
Question:Does this orange have seeds? Correct Answers:yes
Question:IS this a whole orange? Correct Answers:yes
[Image 3]

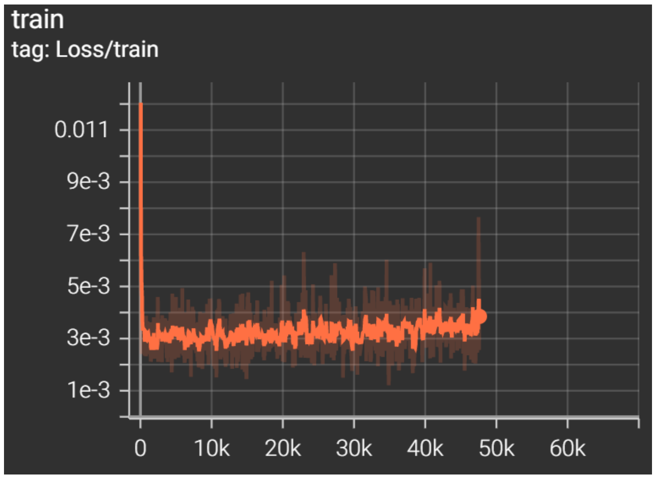


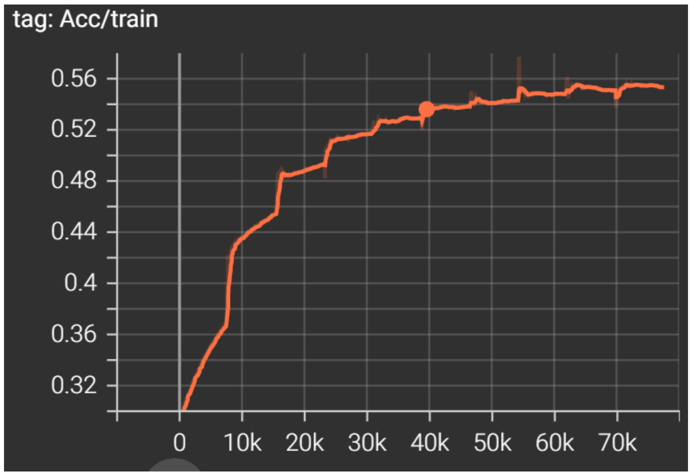
[Correctly Predicted Question-answer pairs]
Question:What body part has most recently contacted the ball? Correct Answers:Other
Question:Are these people social? Correct Answers:yes
[Image Extra]



[Correctly Predicted Question-answer pairs]
Question:Are there balconies? Correct Answers:yes

Training





Now, we are ready to train/test the model! Aim for a validation accuracy of 50% (or near that). For reference, the TAs were able to get 52% accuracy with the default hyperparameters in 10 epochs. You're free to use different hyperparameters but not to change the architecture at this point.

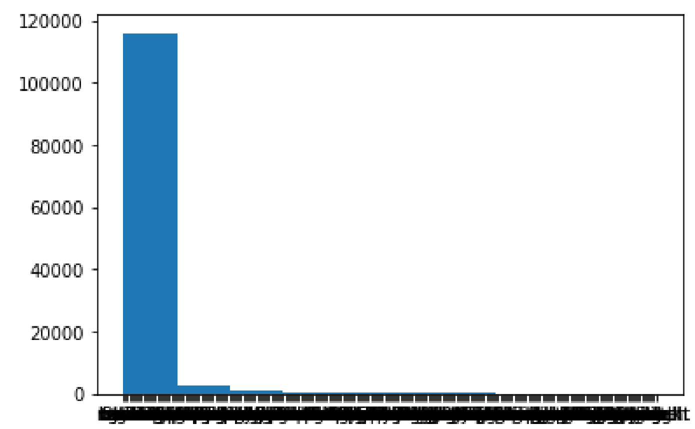
End Validation Accuracy: 0.5573

Analysis

We want to get an idea of why our performance is so low. You may already have an idea actually from what you visualized.

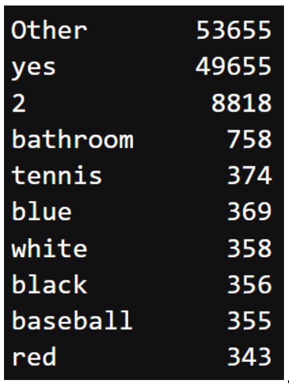
2.7 Make a histogram of the frequencies of the predicted answers (all 5217) without labels. Specifically, gather all the predicted answers of your model through the whole dataset. For each sample, keep only the argmax (most confident) prediction. Then compute the frequency of each answer, sort in decreasing order of frequency and plot the histogram.

No labels: all

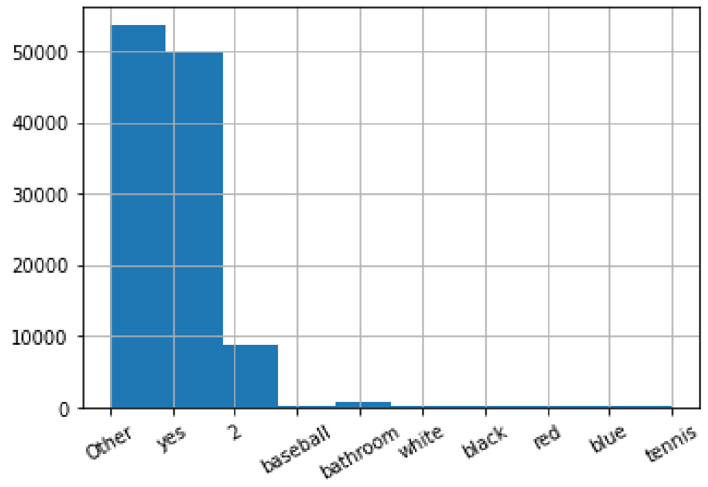


2.8 What are the 10 most frequently predicted classes? Why do you think this happens? You can optionally visualize the frequency of ground-truth answers in the training set.

Counts



```
Other        53655
yes          49655
2             8818
bathroom       758
tennis         374
blue           369
white          358
black          356
baseball       355
red            343
```



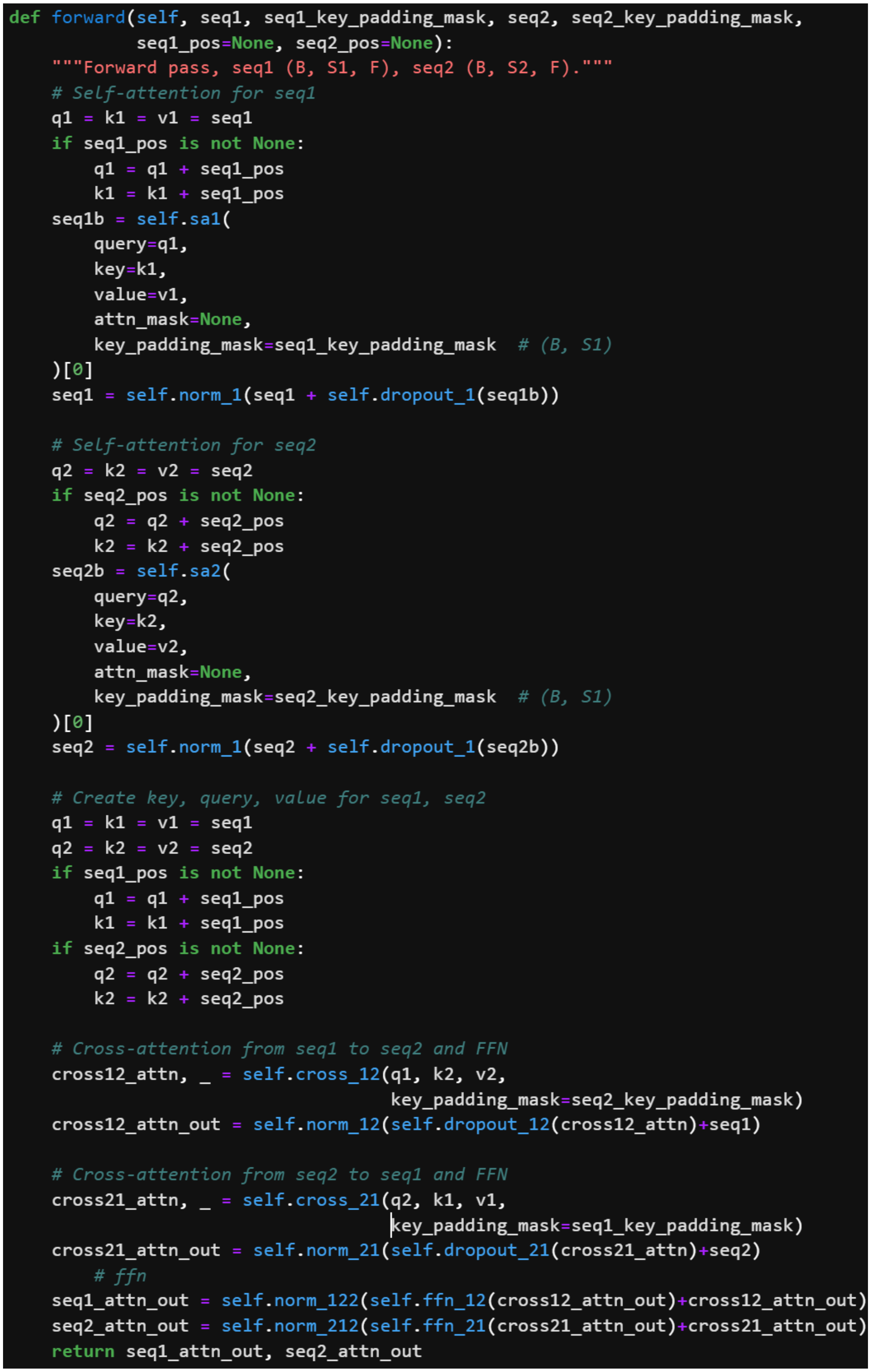
I think it happened because the model has converged to an local optimum where it learnt to take a simple shortcut to achieve higher accuracy but am unable to capture complexity between features in other labels in minor classes. We could say the features are also not well engineered to the model for it to learn from.

Take-away message: when your network is not expressive enough to capture the causal factors of your data, it will try to capture the output distribution. Representing the true data distribution rather than the label one is an important and open problem in Machine Learning!

Task 3: Transformer Network (30 points)

3.1 Complete the CrossAttentionLayer. Include a screenshot of the forward method in your report.

[Forward pass image]



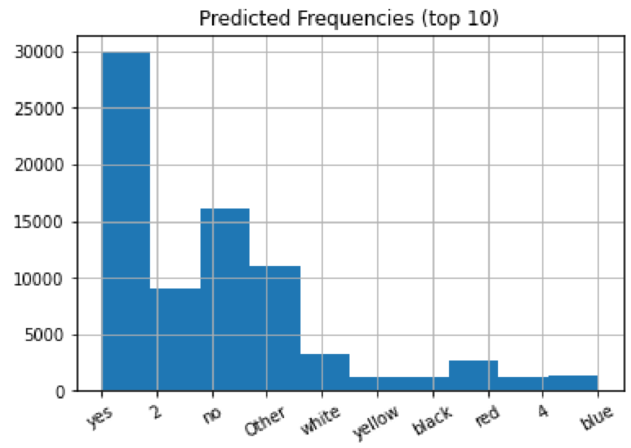
```python
def forward(self, seq1, seq1_key_padding_mask, seq2, seq2_key_padding_mask,
            seq1_pos=None, seq2_pos=None):
    """Forward pass, seq1 (B, S1, F), seq2 (B, S2, F)."""
    # Self-attention for seq1
    q1 = k1 = v1 = seq1
    if seq1_pos is not None:
        q1 = q1 + seq1_pos
        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, S1)
    )[0]
    seq2 = self.norm_1(seq2 + self.dropout_1(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
    cross12_attn, _ = self.cross_12(q1, k2, v2,
                                    key_padding_mask=seq2_key_padding_mask)
    cross12_attn_out = self.norm_12(self.dropout_12(cross12_attn)+seq1)

    # Cross-attention from seq2 to seq1 and FFN
    cross21_attn, _ = self.cross_21(q2, k1, v1,
                                    key_padding_mask=seq1_key_padding_mask)
    cross21_attn_out = self.norm_21(self.dropout_21(cross21_attn)+seq2)
        # ffn
    seq1_attn_out = self.norm_122(self.ffn_12(cross12_attn_out)+cross12_attn_out)
    seq2_attn_out = self.norm_212(self.ffn_21(cross21_attn_out)+cross21_attn_out)
    return seq1_attn_out, seq2_attn_out
```

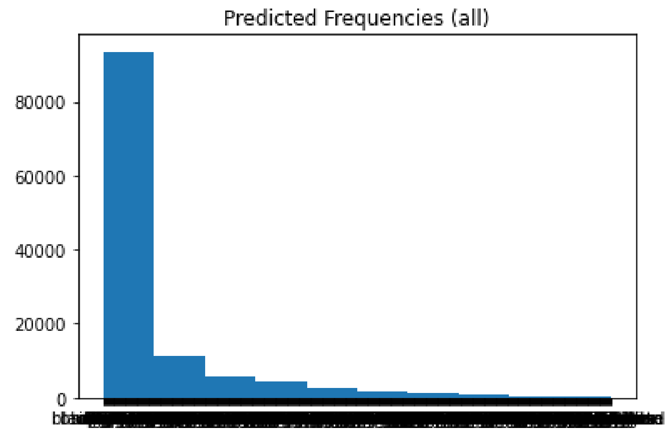
Hints: do not change any layer's name. The current __init__ method contains the names of all layers that you will need. We've implemented one of the self-attention layers for reference. Build the others based on that and the comments. nn.MultiheadAttention directly implements multi-head attention. Make sure to initialize it and call it with appropriate arguments. This is a common source of bugs.

3.2 Load the trained weights and reproduce our result.

Accuracy: 0.6772
Reproduced successfully

3.3 How does the histogram of answers look now?





Take-away message: the success of a representation model highly depends on the distributional properties of the data and the labels. A non-expressive model learns the biases of the frequent classes. An expressive model can learn more accurate classifiers for the frequent classes but is still biased towards them. In such cases, more careful loss design is required.

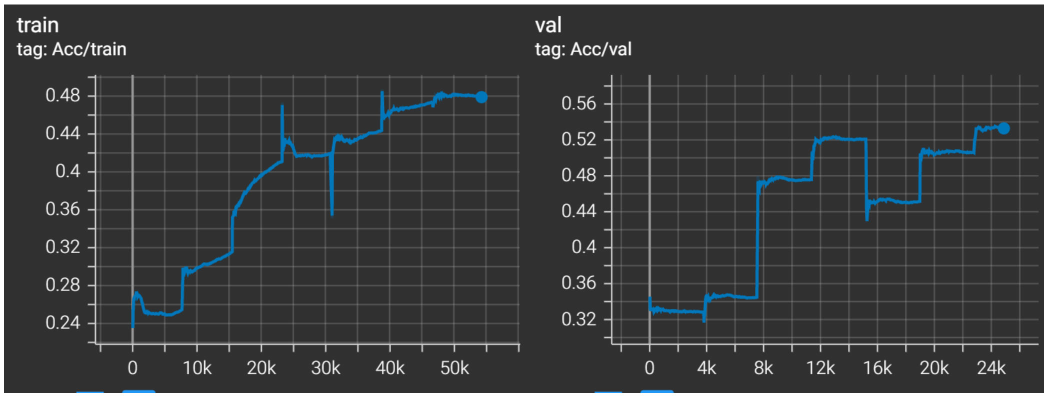
Task 4: Bells and Whistles (30 bonus points)

4.1 Is it a matter of capacity? In the BaselineNet replace the linear classifier with a deeper MLP. Report the accuracy plot of this model.

[code implemented]
What is done:

- Added 2 hidden layers to the FC classifier

It is not a matter of capacity. In fact, it performs worse in validation due to potentially more overfitting.



Final val accuracy: 50.64%

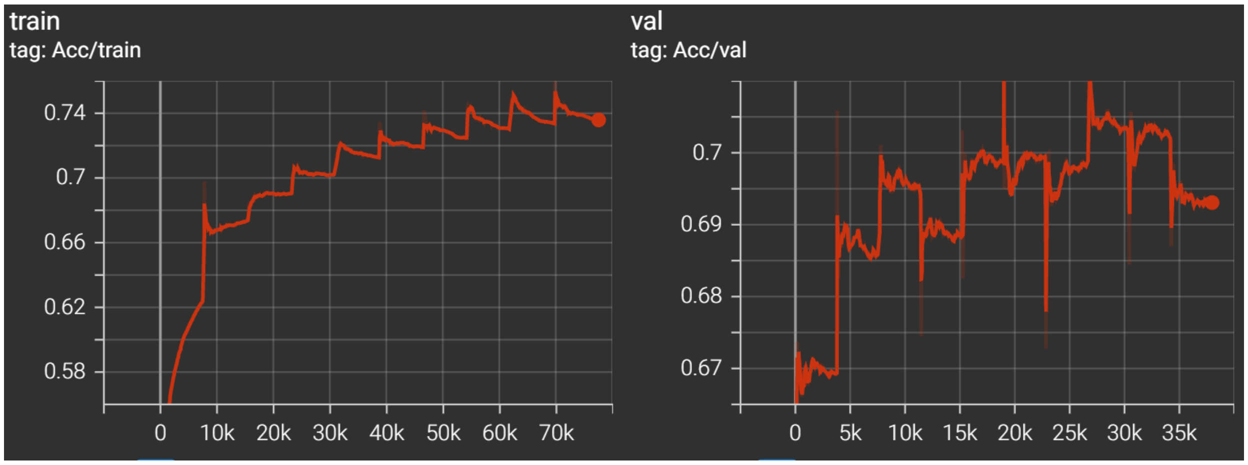
4.2 Is it a matter of context? Featurize the image with ResNet18 and pool an 1-d representation. Then use this vector to condition (provide it as initial hidden state) an LSTM that reads the language word-by-word. Gather the hidden representations of this LSTM and classify the answer.

[code implemented]
What is done:

- Featurize with the ResNet18 (as in the template ***compute_vis_feats***), pooled features into the hidden_dimension of LSTM so that it can be used as initial hidden state
- Pass text encoded feature (as given by the template ***compute_text_feats***) into the LSTM

4.3 Better than 67.62? Try to improve the Transformer-based model. Some ideas include adding more encoder layers or (partially) unfreezing the backbones' weights. Note that this can take a significant amount of GPU time.

[code implemented]
Apart from the above, a 2 layer bi-directional LSTM is used to extract the features. There is also the idea of adding cross-attention across image and visual features, however, since these training use a lot of time and compute it was not implemented.
However, the performance from the previous LSTM model improved upon the transformer net.
For this implementation, we use LSTM to encode sequence/visual features , and 1D convolution is applied to image features. (As described above with some modification)
The model is able to easily outperform the base transformer net and achieve a final validation accuracy of: 0.695
Training plot attached.



4.4 Propose and implement any other idea you have! This does not have to include implementing/training a new model. It can be an interesting analysis or a better loss.

Idea from 4.3
Mixing LSTM encoder, which improved sequence feature extraction (with the image context) with cross attention to learn to further attend to features from both domains.

Idea 2: Generative domain adaptation
We can add contrastive loss terms between the mapping of latent representation from image and text, text to image because we can generally guess certain aspects about the image from the text prompt itself and vice versa.

Idea 3: Regularization for attention weights (inspired by Neural Image Captioning Paper)
The idea is to regularize using attention weights to help the attention mechanism to attend to more diverse parts of the encoded features, that may help the model to be less overfitted as well as learn better patterns.