Question 1:

Task 1. Create a CNN-LSTM model. Use the same configuration of CNN and LSTM as you can see in 3.py and 1.py respectively.
Task 2. To do task 1 correctly, did you have to remove or add any additional layer? If yes, then indicate the layer and give the reason for it.
Task 3. Now, implement the entire CNN-LSTM model at the character level. Instead of using pre-trained Glove word embeddings, use random character embedding of size 50. The rest of the model should remain the same. Hint: you should use character level tokenizer in Keras. See details of the tokenizer class in Keras and pass the appropriate argument.

Optional question:

Implement the self-attention mechanism on top of the LSTM as given in 1.py. The self-attention mechanism can be implemented using the following steps -

Suppose we have L input sentences each having M words. Each of these words have N number of features. Hence, the input matrix X has dimension of L*M*N. Now say, we want to classify sentiment of each sentence.
Can we automatically learn a function that learns and tells us the role of each word in the classification?

We call this self-attention mechanism.

The idea is simple.

1) We apply a dense layer or Fully Connected (FC) network on the input matrix X. Lets call this FC layer as W. W is a trainable layer or weights of dimension N*1. We get X’ = XW of dimension L*M*1

2) We add some non-linearity to X’ so we get X’’ = tanh(X’)

3) Now, apply softmax to the last dimension to get the importance probability of each word. Say, we store this to X’’’= softmax(X’’, axes=-1) of dimension L*M*1.

4) We multiply X’’’ to the input X to find an abstract representation of our inputs that encode the role of each word along with the original semantic of the words. Hence we have X’’’’ = X’’’*X to get the abstract features of dimension L*N. So, we represent each sentence in N dimensional space.

5) You can feed X’’’’ to a logistic regression classifier to classify.