Assignment 7: Deep Learning

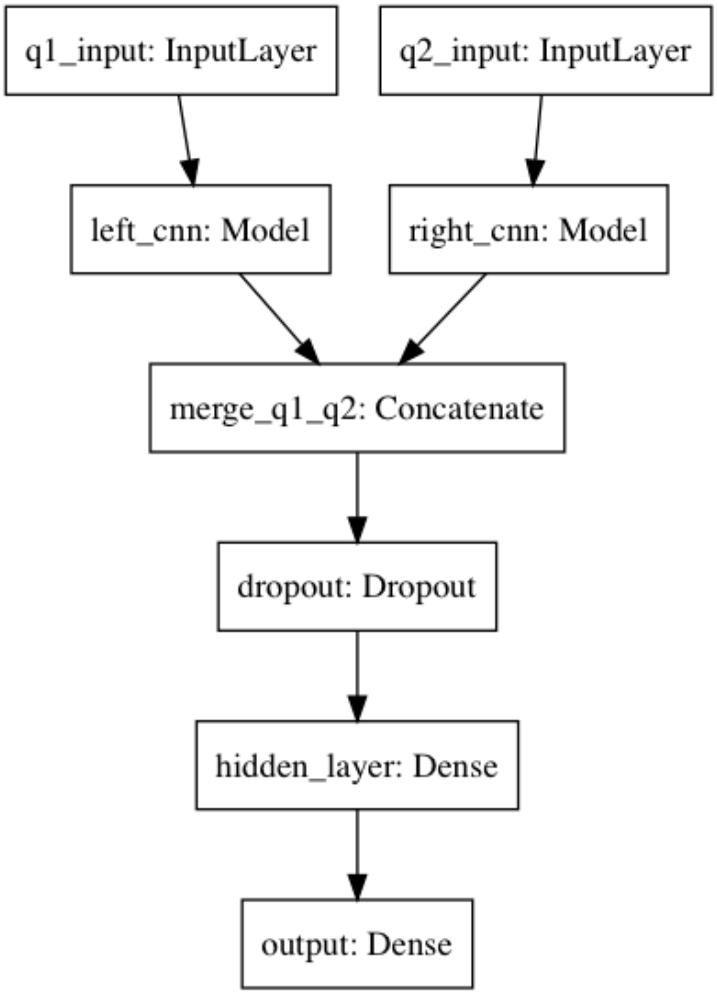
In Assignment 5, Q3 (bonus question), you were asked to create a classification model for to detect duplicate questons. Now let's try the same problem using a deep learning approach.

You'll need 'quora_duplicate_question_500.csv' for this assignment. This dataset is in the following format

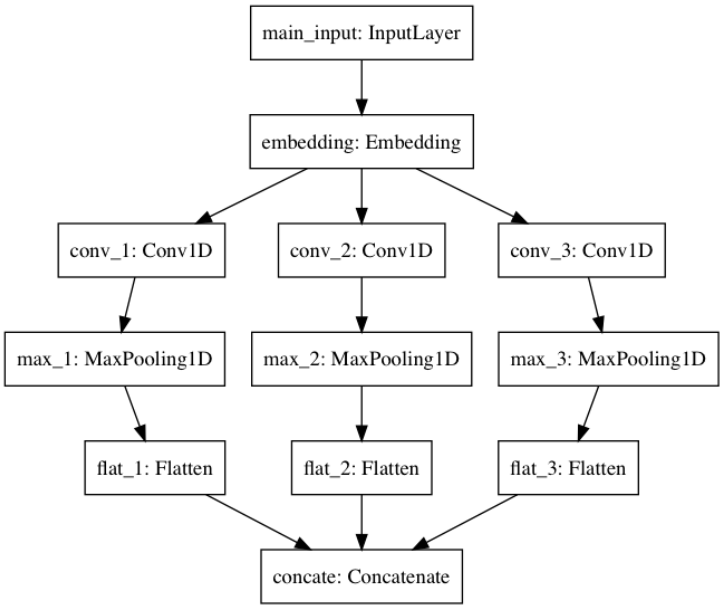
q1	**q2**	**is_duplicate**
How do you take a screenshot on a Mac laptop?	How do I take a screenshot on my MacBook Pro? ...	1
Is the US election rigged?	Was the US election rigged?	1
How scary is it to drive on the road to Hana g...	Do I need a four-wheel-drive car to drive all ...	0
...	...	...

- Create a function **detect_duplicate( )** to detect sentiment as follows:
  - the input parameter is the full filename path to quora_duplicate_question_500.csv
  - convert $q1$ and $q2$ into padded sequences of numbers (see Exercise 5.2)
  - **hold 20% of the data for testing**
  - **carefully select hyperparameters**, in particular, input sentence length, filters, the number of filters, batch size, and epoch etc.
  - create a CNN model with the training data. Some hints:
    - Since you have a small dataset, consider to use **pre-trained word vectors**
    - In your model, you use CNN to extract features from q1 and q2, and then predict if they are duplicates based on these features
    - Your model may have a structure shown below.
  - print out accuracy, precision, recall, and auc calculated from testing data.
    - Your **average precision, recall, accurracy, and auc should be all about 70%**.
    - If your result is lower than that (e.g. below 70%), you need to tune the hyperparameters
- This function has no return. Besides your code, also provide a pdf document showing the following
  - How you choose the hyperparameters
  - model summary
  - Screenshots of model trainning history
  - Testing accuracy, precision, recall, and auc
- A few more notes about this assignment:
  - Due to small sample size, the performance may vary in each round of training. Also, you may see the performance does not improve much from the result of Assignment 5. Don't worry about this for now. We just use this example to practice how to build the deep learning model.
  - If you use pretrained word vectors, please describe which pretrained word vector you choose. You don't need to submit pretrained word vector files.

Hint: Possible structure of model:



Where the left_cnn or right_cnn is shown below:



In [179]:
```
from keras.layers import Embedding, Dense, Conv1D, MaxPooling1D, \
Dropout, Activation, Input, Flatten, Concatenate
# add import
```

In [195]:
```
def detect_duplicate(datafile):

   # add your code
```

In [196]:
```
if __name__ == "__main__":

    detect_duplicate("../../dataset/quora_duplicate_question_500.csv")
```

Overall Model: **This is just a referenfce structure. You don't have to use the same structure**

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #     C
onnected to
==================================================================
==================================
q1_input (InputLayer)        (None, 35)                0
_________________________________________________________________
_________________________________________
q2_input (InputLayer)        (None, 35)                0
_________________________________________________________________
_________________________________________
left_cnn (Model)             (None, 192)               877692      q
1_input[0][0]
_________________________________________________________________
_________________________________________
right_cnn (Model)            (None, 192)               877692      q
2_input[0][0]
_________________________________________________________________
_________________________________________
merge_q1_q2 (Concatenate)    (None, 384)               0           l
eft_cnn[1][0]

right_cnn[1][0]
_________________________________________________________________
_________________________________________
dropout (Dropout)            (None, 384)               0           m
erge_q1_q2[0][0]
_________________________________________________________________
_________________________________________
hidden_layer (Dense)         (None, 64)                24640       d
ropout[0][0]
_________________________________________________________________
_________________________________________
output (Dense)               (None, 1)                 65          h
idden_layer[0][0]
==================================================================
==================================
Total params: 1,780,089
```

Trainable params: 255,489
Non-trainable params: 1,524,600

sub CNN model for left or right CNN:

Layer (type)	Output Shape	Param #	Connected to
main_input (InputLayer)	(None, 35)	0	
embedding (Embedding)	(None, 35, 300)	762300	main_input[0][0]
conv_1 (Conv1D)	(None, 35, 64)	19264	embedding[0][0]
conv_2 (Conv1D)	(None, 34, 64)	38464	embedding[0][0]
conv_3 (Conv1D)	(None, 33, 64)	57664	embedding[0][0]
max_1 (MaxPooling1D)	(None, 1, 64)	0	conv_1[0][0]
max_2 (MaxPooling1D)	(None, 1, 64)	0	conv_2[0][0]
max_3 (MaxPooling1D)	(None, 1, 64)	0	conv_3[0][0]
flat_1 (Flatten)	(None, 64)	0	max_1[0][0]
flat_2 (Flatten)	(None, 64)	0	max_2[0][0]
flat_3 (Flatten)	(None, 64)	0	max_3[0][0]

```
_____________________________________
concate (Concatenate)            (None, 192)           0           f
lat_1[0][0]

flat_2[0][0]

flat_3[0][0]
==================================================================
================================
Total params: 877,692
Trainable params: 115,392
Non-trainable params: 762,300

_____________________________________
Train on 400 samples, validate on 100 samples
Epoch 1/100
Epoch 00000: val_acc improved from -inf to 0.68000, saving model to
best_model
11s - loss: 0.8028 - acc: 0.5950 - val_loss: 0.7682 - val_acc: 0.680
0
Epoch 2/100
Epoch 00001: val_acc did not improve
0s - loss: 0.7252 - acc: 0.6725 - val_loss: 0.7201 - val_acc: 0.6700
Epoch 3/100
Epoch 00002: val_acc improved from 0.68000 to 0.69000, saving model
to best_model
0s - loss: 0.7005 - acc: 0.6575 - val_loss: 0.7446 - val_acc: 0.6900
Epoch 4/100
Epoch 00003: val_acc did not improve
0s - loss: 0.6407 - acc: 0.7675 - val_loss: 0.6793 - val_acc: 0.6800
Epoch 5/100
Epoch 00004: val_acc improved from 0.69000 to 0.70000, saving model
to best_model
0s - loss: 0.5488 - acc: 0.8350 - val_loss: 0.6725 - val_acc: 0.7000
Epoch 6/100
Epoch 00005: val_acc improved from 0.70000 to 0.71000, saving model
to best_model
0s - loss: 0.4717 - acc: 0.8675 - val_loss: 0.6860 - val_acc: 0.7100
Epoch 7/100
Epoch 00006: val_acc did not improve
0s - loss: 0.4090 - acc: 0.9225 - val_loss: 0.6693 - val_acc: 0.6700
Epoch 8/100
Epoch 00007: val_acc improved from 0.71000 to 0.76000, saving model
to best_model
0s - loss: 0.3352 - acc: 0.9425 - val_loss: 0.6492 - val_acc: 0.7600
Epoch 9/100
Epoch 00008: val_acc did not improve
0s - loss: 0.2628 - acc: 0.9675 - val_loss: 0.6512 - val_acc: 0.7600
Epoch 10/100
Epoch 00009: val_acc did not improve
0s - loss: 0.2210 - acc: 0.9750 - val_loss: 0.6662 - val_acc: 0.7300
Epoch 11/100
Epoch 00010: val_acc did not improve
```

```
0s - loss: 0.1783 - acc: 0.9950 - val_loss: 0.7010 - val_acc: 0.7300
Epoch 12/100
Epoch 00011: val_acc did not improve
0s - loss: 0.1687 - acc: 0.9925 - val_loss: 0.6838 - val_acc: 0.7600
Epoch 13/100
Epoch 00012: val_acc did not improve
0s - loss: 0.1376 - acc: 1.0000 - val_loss: 0.6915 - val_acc: 0.7300
Epoch 14/100
Epoch 00013: val_acc did not improve
0s - loss: 0.1340 - acc: 0.9925 - val_loss: 0.7275 - val_acc: 0.7100
Epoch 00013: early stopping
             precision    recall  f1-score   support

        0.0       0.79      0.87      0.83        67
        1.0       0.67      0.55      0.60        33

avg / total       0.75      0.76      0.75       100

('auc', 0.7403889642695614)
```

In [ ]: