GCT634 HW3. Chord detection

20185262 Sunghee Jung

## 1. Algorithm Description

### a. Convolutional Neural Network (CNN)

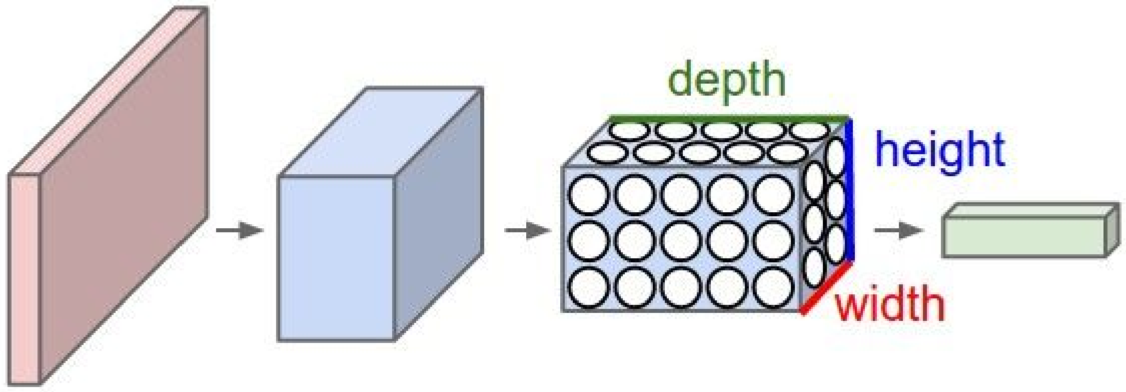


Figure 1. CNN structure

Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).
CNN is largely used specifically in image domain to capture the patterns in the input features hierarchically.

### b. Recurrent Neural Network (RNN)

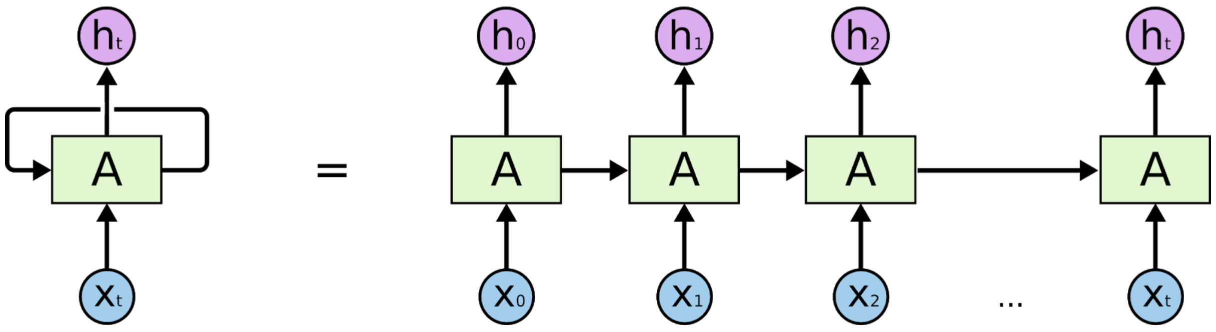


Figure 2. RNN structure

RNN is used in sequential domain. It is assumed that RNN's hidden layer captures the informations from previous states. When compared to Hidden Markov Model(HMM), HMM assumes the hidden state of only one previous time step affects current step. That assumption is very handy in computation wise, but far from real life model. To put my problem into above picture, x0 to xt corresponds to the "sequence length" parameter when implementing the algorithm.

### c. Long Short Term Memory networks (LSTM)

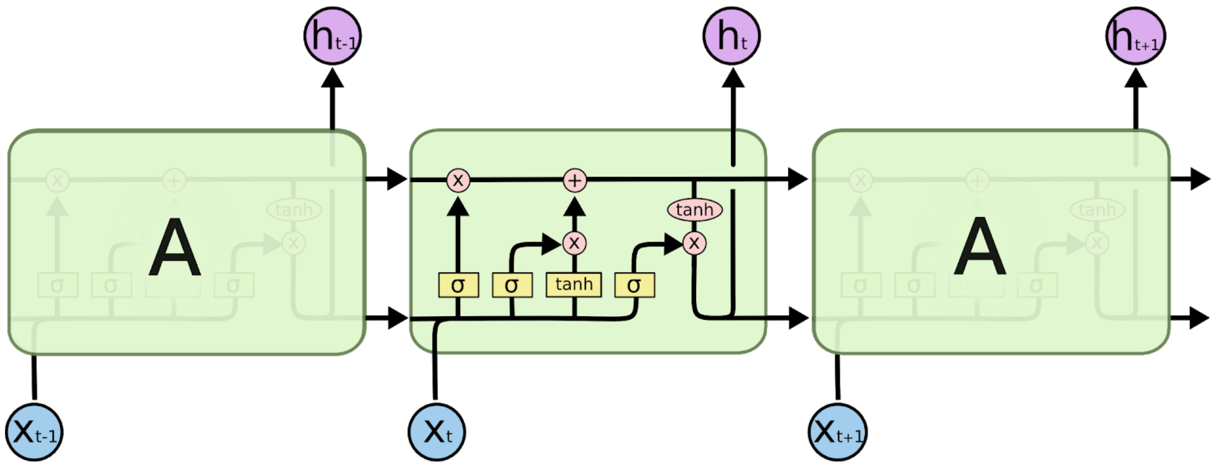
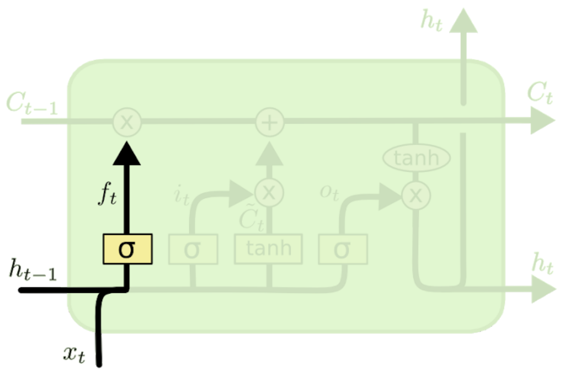


Figure 3. LSTM structure

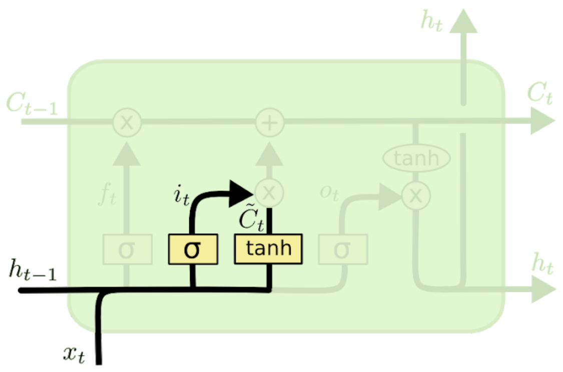
Simple RNN architecture has trouble remembering long term information. That problem is refered to as “Long-term dependency problem”. Long short term memory networks (LSTMs) are devised to overcome such problem. The key to LSTMs is the cell state. It’s very easy for information to just flow along the cell state unchanged(except for some minor linear interactions). The LSTM can remove or add information to the cell state, carefully regulated by structures called gates.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

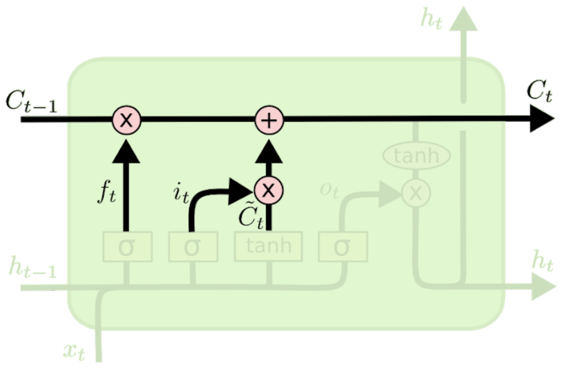
Figure 4. Forget gate of LSTM

Firstly, forget gate is used to decide to what degree it remembers the previous state and the current input. Forexample, in the domain of predicting next word, the subject of previous sentence should be forgotten to make the grammar of current sentence correct.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

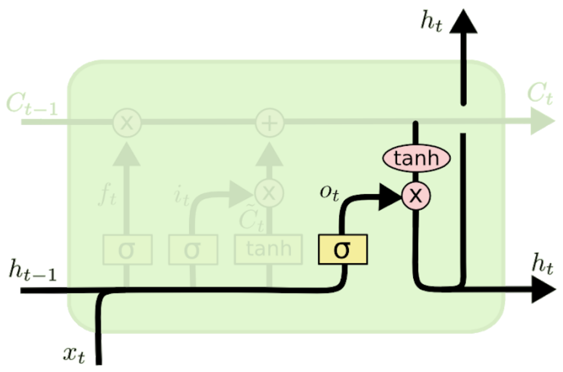
Figure 5. input gate and gate-gate(naming by Andrej Kaparty) of LSTM



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 6. Cell state of LSTM

Secondly, input gate layer decides what new information we’re going to store in the cell state. Next, a tanh layer creates a pseudo-cell state. The multiplication of input gate and pseudo-cell state decides which new information should be included to what amonut for updating current cell state. For example of language modeling, the gender of new subject need to be included.



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 7. output gate and hidden state of LSTM

Finally we decide the output from the cell state. Calculating the output consists of two parts of generating output gate and generating current hidden state from output gate and current cell state. First we run a sigmoid layer which decides what part of cell state we're going to output. Then, we put the cell state through tanh and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

### d. Convolutional Recurrent Neural Network (CRNN)

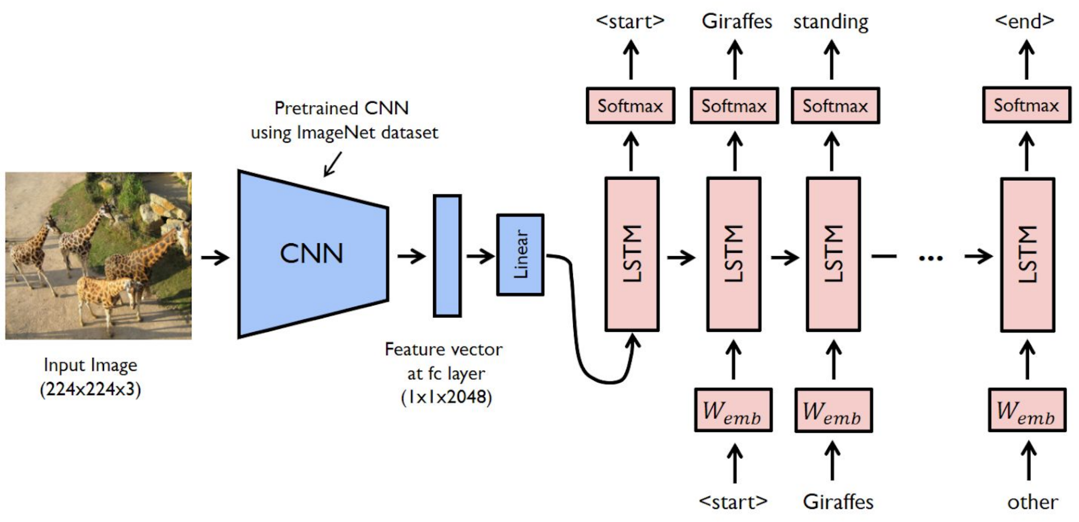


Figure 8. CRNN structure used in image captioning domain

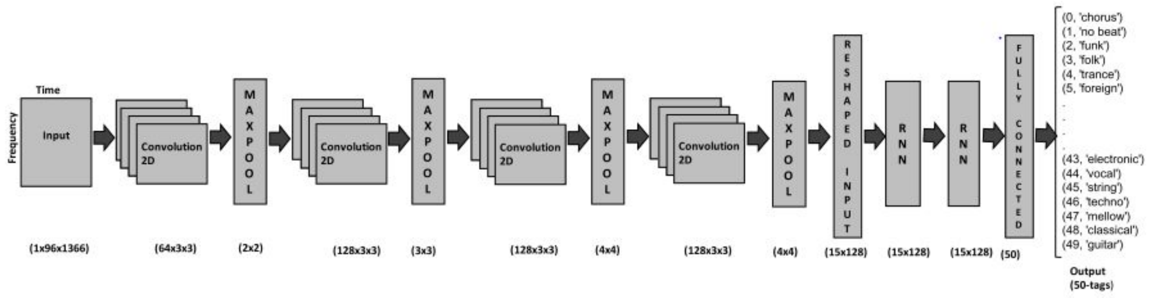
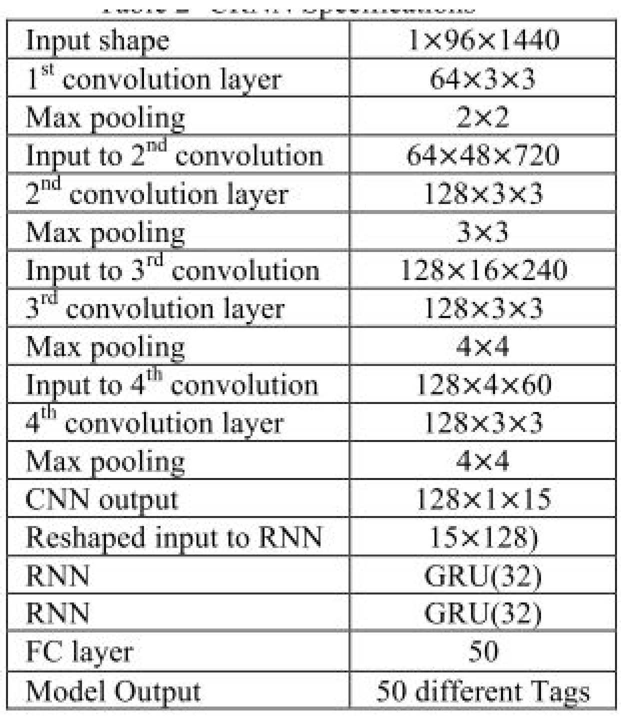


Figure 9. CRNN structure used in music auto tagging domain



| | |
|---|---|
| Input shape | 1×96×1440 |
| 1$^{st}$ convolution layer | 64×3×3 |
| Max pooling | 2×2 |
| Input to 2$^{nd}$ convolution | 64×48×720 |
| 2$^{nd}$ convolution layer | 128×3×3 |
| Max pooling | 3×3 |
| Input to 3$^{rd}$ convolution | 128×16×240 |
| 3$^{rd}$ convolution layer | 128×3×3 |
| Max pooling | 4×4 |
| Input to 4$^{th}$ convolution | 128×4×60 |
| 4$^{th}$ convolution layer | 128×3×3 |
| Max pooling | 4×4 |
| CNN output | 128×1×15 |
| Reshaped input to RNN | 15×128) |
| RNN | GRU(32) |
| RNN | GRU(32) |
| FC layer | 50 |
| Model Output | 50 different Tags |

Table 1. Dimensionality of CRNN structure from Figure 9.

CRNN is a neural net structure where lower CNN layer is connected to upper RNN layer.

It seems CRNN is commonly used in music domain and image captioning domain. Figure describes the use of CRNN in image captioning and Figure describes the use of CRNN in music tagging domain.

For image captioning, CNN ending with linear layer gives summurization of the input image and it is fed to RNN layer which creates a word at each step to make a sentence that describes the picture. Image itself does not have any ‘sequential’ characterisitc but the target sentence has sequential characteristic. The word that the RNN model outputs in the previous step must be reasonable input to calculate the word for the current step. In that sense, using CRNN for the image captioning domain seems reasonable.

For the music tagging domain, RNN alone ending with FC layer seems enough since the input music signal is sequential. But for the case of the paper that I refered above, the input was spectrogram and it had too many dimension. I think CNN here played a role of summarizing the large dimension of spectrogram.

## 2. Experiments and Results

For modeling CRNN, reducing the feature domain to 1 seemed to be necessary. As I assumed that the reason for using CNN prior to RNN was mainly to reduce the dimensionality effectiely. On the other hand, the time dimension should keep some dimensionality to be passed to RNN and be analyzed there. So there were a few dimensionality rules to cover the characteristics of each algorithm and some implementation issues when designing the CRNN model.

A. CNN input needs a channel size: input of this model(12 chroma features extracted from each music frame) does not have nay channel dimension, but CNN assumes that there is channel dimension. Therefore additional dummy dimension of size 1 was introduced to fit the CNN algorithm.
B. Inside CNN: once the input of dimension (batch x channel x feature x sequence_length) was submitted to CNN model, the (feature x sequence_length) is treated as 2D input (similar to spectrogram case, except that in this case, feature domain represents the sum of each note over all octaves)
C. Last layer of CNN has 1 feature dimension: As I mentioned earlier, the role of CNN is expected to summarize the feature domain. And to fit the dimension requirement of RNN without blurring the meaning of each dimension, the CNN layers are designed to have last output with feature dimension of 1.
D. Input to RNN: Input to RNN has interpretation of (sequence_length x batch x feature). The 4D(batch x channel x feature x time) output of CNN is first squeezed to lose the feature dimension of size 1 to become (batch x channel x time) and then permutated in dimension to change order and become (time x batch x channel).

**a. Baseline(beatsync), sequence_length = 1**

Baseline code of beat level chord detection with sequence length for RNN = 1

**b. Baseline(frame), sequence_length = 1**

Baseline code of frame level chord detection with sequence length for RNN = 1

**c. Updated baseline(frame), sequence_length=16**

Updated baseline code of frame level chord detection with sequence length for RNN = 16.

**d. crnn model with 2 layer cnn**

| | |
|---|---|
| Input shape | 32 x 1 x 12 x 128 (b, c, f, t) |
| 1st convolution layer | 64 x 3 x 3 |
| Max pooling | 2 x 2 |
| Input to 2nd convolution | 32 x 64 x 5 x 62 |
| 2nd convolution layer | 128 x 3 x 3 |
| Max pooling | 3 x 3 |

| | |
|---|---|
| CNN output | 32 x128 x 1 x 20 |
| Reshaped input to RNN | 20 x 32 x 128 (t, b, c) |
| RNN | Bi-directional LSTM (64) |
| FC layer | 128 -> 25 |
| Model output | 25 |

Table 2. 2 layer CRNN model

### e. crnn model with 1 layer cnn

crnn1 version seemed to have better starting loss but it continously got stuck on the plateau and could not continue learning for more than 11 epoch.

| | |
|---|---|
| Input shape | 32 x 1 x 12 x 128 (b, c, f, t) |
| 1st convolution layer | 64 x 5 x5 |
| Max pooling | 7 x 7 |
| CNN output | 32 x64 x 1 x 17 |
| Reshaped input to RNN | 17 x 32 x 64 (t, b, c) |
| RNN | Bi-directional LSTM (64) |
| FC layer | 128 -> 25 |
| Model output | 25 |

Table 3. 1 layer CRNN model

### f. RNN only

| | |
|---|---|
| RNN type | bi-directional LSTM |
| Dropout | 0.25 |
| Input | 128 x 32 x 12<br>(sequence_length x batch x chorma feature dim) |
| The number of features in the hidden state | 16 |

| | |
|---|---|
| The number of layers | 1 |
| Number of classes | 25 (sil(1) + major(12) + minor(12)) |
| Learning rate | starting from 0.001 and scheduled to be reduced according to the gradient of loss |
| Optimizer | Adam optimizer |
| Learning speed | 5 min for each epoch on GTX 980 |

Table 4. RNN model with sequence length 128

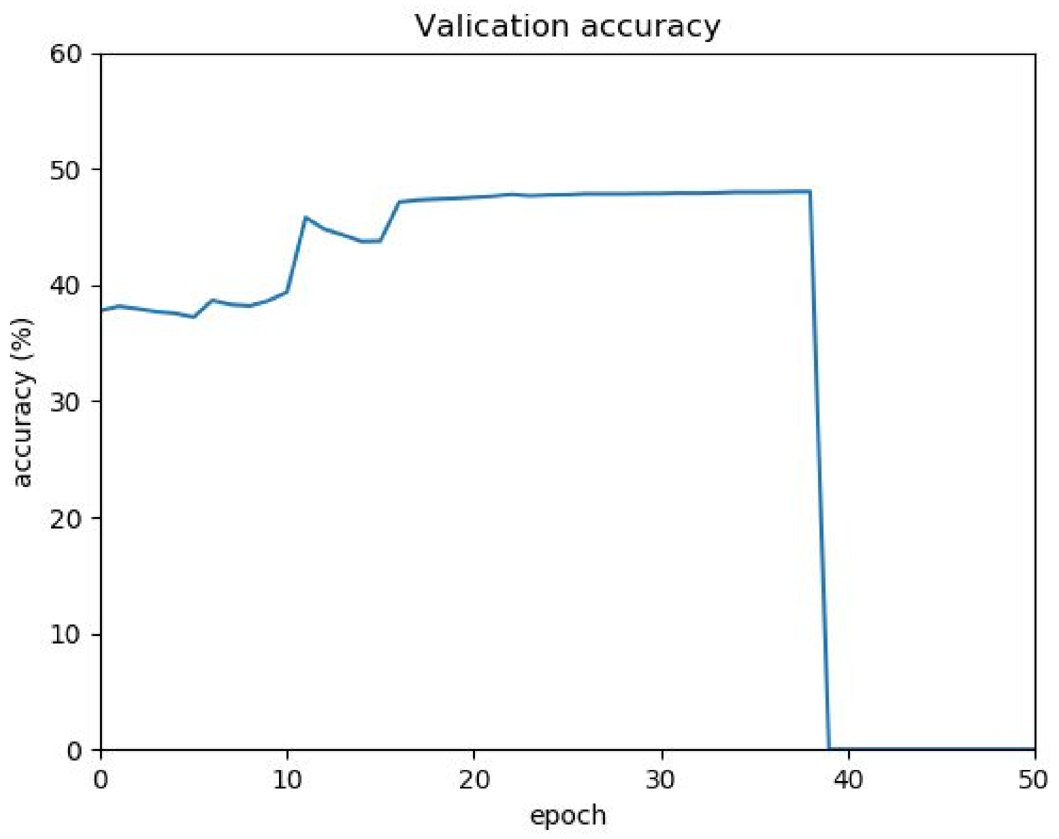


Figure 10. Validation accuracy of RNN only model with 128 sequence length

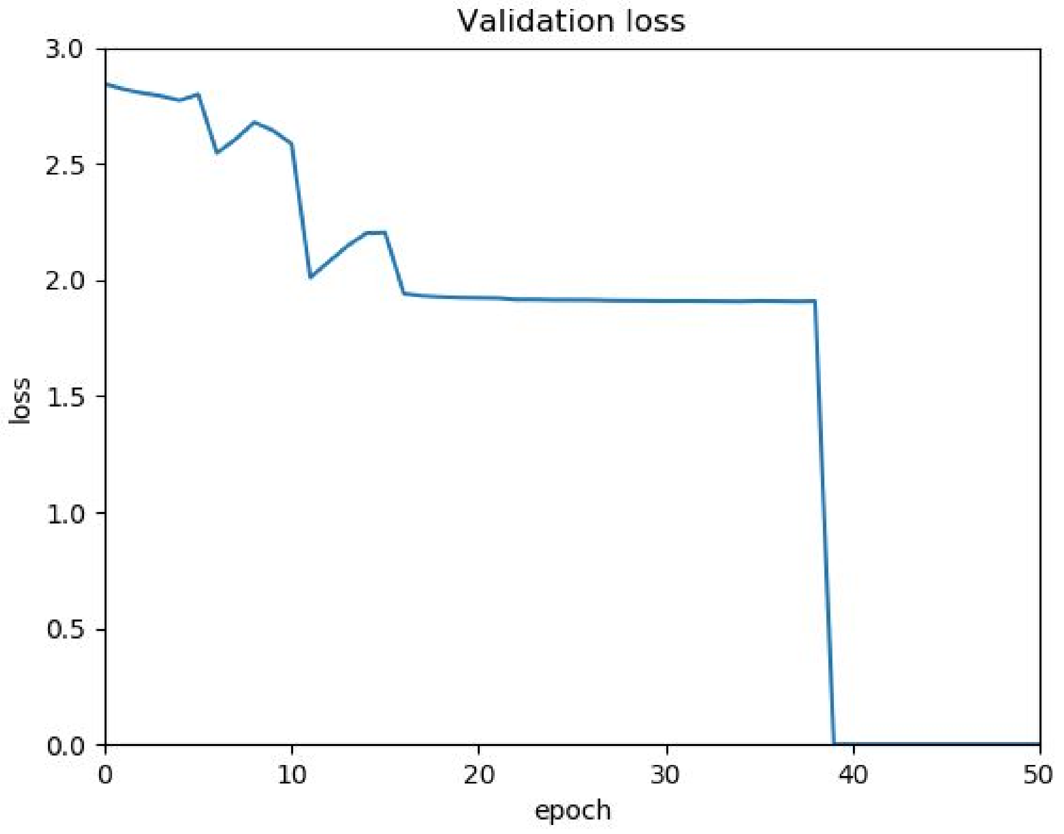


Figure 11. Validadtion loss of RNN only model with 128 sequence length

Figure 10 and Figure 11 are validation accuracy and validation loss of RNN with 128 sequence length model respectively. During the training, annealing(reduction of learning rate by scheduler) happened twice at 15th epoch and 38th epoch. Starting with the learning rate of 0.001, the learning rate decreased to 4.0000e-05 at epoch 15. At epoch 38, the learning rate decreased to 8.0000e-06 and the model stopped at that point because the threshold for ‘early stopping’ was set to 1e-05.  The annealing helped a lot for the model to not pass by the convex point of the cost function(which is cross entropy) and thus converge eventually. I think it was a good idea that I decided to change the model and stop tuing the CRNN model for convergence. The problem does not lie at the optimization when several different trials would all go fail. The early stopping I faced at the CRNN models is not a problem itself. It is a phenomenon that indicates this model does not fit the problem.

- **Experiment Table**

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| Training data size (60%) | 834 | 38,637 | | | | |
| Validation data size (20%) | 323 | 13,668 | | | | |
| Test data (20%) | 284 | 13,339 | | | | |
| Convergence | Yes | Yes | Yes | No | Yes | Yes |
| Validation accuracy | 47.00 | 47.52 | 40.35 | 12.75 | 33.73 | 48.03% |
| Test accuracy (%) | 50.83 | 50.70 | 42.35 | 14.52 | 34.62 | 57.75 |
| Training time | 10 min | N/A | N/A | 3h | 16h | 4h |

Table 5. Experiment result

## 3. Discussion

## a. Frame level vs. beat level

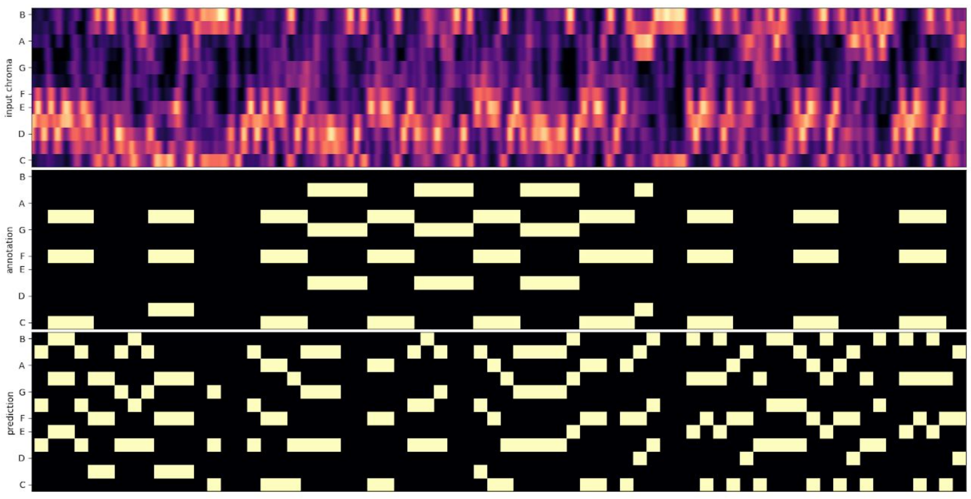


Figure 12. Beat level baseline chord detection result

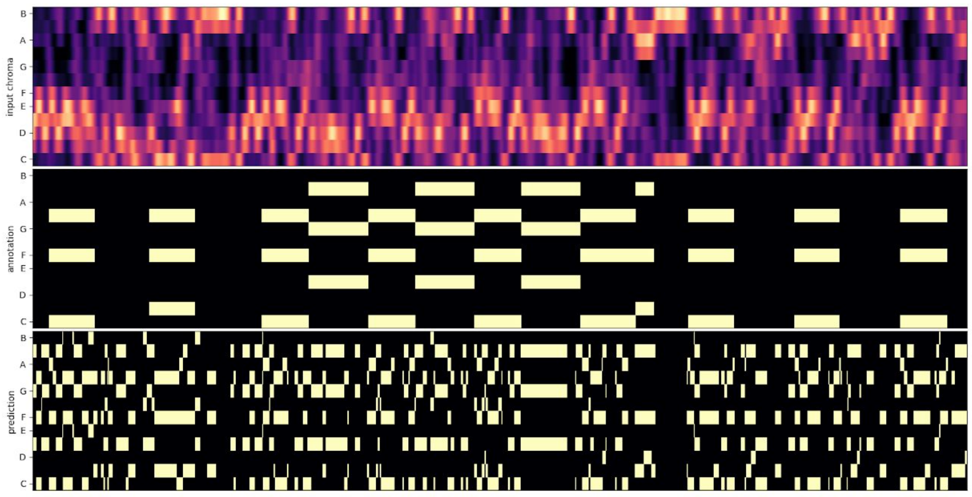


Figure 13. Frame level baseline chord detection result

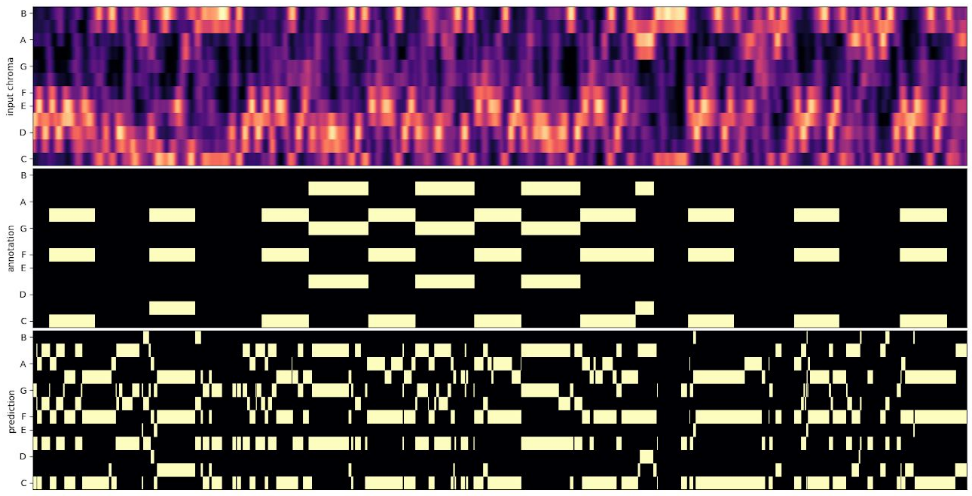


Figure 14. Frame level chord detection using experiemt f

I originally thought I should minimize preprocessing the training data in order to keep its original data size because I belived RNN is a big model and it needs a lot of training data. Also from the visualization of the baseline model as in Figure 12 and Figure 13, it seemed the frame level chord detection gave more precise prediction than the beat level one. Even though the eventual accuracy was about the same for both frame level and beat level cases due to the local jittering of the predicted chord for the frame level prediction case.

### b. CRNN vs. RNN

It was hard to make the CRNN model learn, especially for the beat level detection. I think beat level detection is hard to train effectively because of the size of available training data is a lot smaller for the beat level learning case. For beat level chord detection, it almost always stopped learning too early. I used annealing algorithm implemented on pytorch called 'Reduce LROnPlateau' and it kept stopping before learnin enough. So I encreased the starting learning rate to 0.001 but it still stopped too early and eventually I stopped using 'early stopping' and let it iterate for designated epoch of 200, hoping that it will eventually get out of the plateau in the end. I also tried making the batch size small, hoping that the noise introduced by small batch size would help it get out of the plateau. But small batch size only made the learning slow and it kept beeing stuck at the same point. I also tried different optimizers but only the number of epochs it takes to get stuck on plateau differed.

Therefore, even though beat level chord detection was faster to check the result, I had to let go of it and try frame level one. For beat-level learning, when it stops with early stopping problem, it had learned so little and the test result would show around 12% of accuracy (This experiment is not included in the table).

After a series of different settings to model CRNN, I switched to RNN-only model. RNN was much slower than CRNN models. I think the role of CNN here is to make the summary of the input and make the model learn quickly and it is not for making a better performance.  CRNN model was quick to learn and had worse accuracy, where as RNN model learned awfully slow but got better result.

### c. RNN hyper parameters

For RNN, It seemed 'sequence length' was the most important hyper parameter. When compared to the updated baseline code of experiment c, experiment f had 4 times less hidden layer features and had 8 times larger sequence length. From figure 12 and 13, I assumed that label chord, which is labled by human, takes a lot of frames which probably corresponds to the 'bar' of music. Since I do not have enough musical background to detect 'bar' from the input signal, I just decided to make the sequence frame length as long as the RAM is allowed. For my working environment, the program would give memory error for larger than 128 sequence length and that is why I went with sequence length of 128 for experiment f. In Figure 14, it is observed that with larger sequence length, RNN with smaller hidden features jitters less(comparison of Figure 13 from experiment b)

### d. About data split

According to the TA, the testset is supposed to be more like 'real-world' music. I cannot explain why all my models give better accuracy for testset than the validation set.

### e. About implementation points

**It might not have been the interest of this homework, but I think I have learned a more valuable lesson.** I was a system software engineer making database for past 3 years. For the first homework of this course, I spent about 70% of the time refactoring the code and making the interface of the software easier to log and track the processes of different trials as a professional programmers would do. As the due date came closer, I felt like it was worth nothing because program designing skill was not an evaluation point and I should have spent more time on algorithms. I felt like I should now let go of my obsession as a professional programmer. And for this HW3, I spent almost no time for those program designing factors and as I tried more and more different settings, the logging went to hell. That is why some of the parts in the experiment result table says "N/A". **Now I know that for this area of deep learning, which requires lots of trials and errors and tuning points, those times spent on refactoring the code for better interfaces and productivity is actuallly priceless** and I will always keep that in mind for further studying and researching time in my career.

## 4. References

a. http://cs231n.github.io/convolutional-networks/
b. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
c. "A Deep Learning Approach for Mapping Music Genres", Sharaj Panwar, Arun Das, Mehdi Roopaei, Paul Rad, Department of Electrical and Computer Engineering University of Texas at San Antonio San Antonio, Texas, USA
d. https://www.analyticsvidhya.com/blog/2018/04/solving-an-image-captioning-task-using-deep-learning/