HW5

1.preprocessing

data extraction
I am using librosa.load(osp.join(language, file), sr=16000, mono=True) to load data.(I delete the english_0128.wav file because Icannot even read this file by sox).
Then I use librosa.effects.split(y, top_db=30) to delete the silence intervals
and concatenate these non silence intervals as one. librosa.feature.mfcc(y=y_no_sil, sr=sr, n_mfcc=64, n_fft=int(sr*0.025), hop_length=int(sr*0.010))
to transfer the sudio file to MFCC features.
And then I choose 10s as a sequence length which reshape data to (batch_size, sequence_length=1000, num_feature=64)
Also set each label to (batch_size, sequence_length=1000, 1)
English labels to 0, hindi labels to 1, mandarin labels to 2.

```
def extract_features(language):
    list = os.listdir(language)

    for file in tqdm(list):
        y, sr = librosa.load(osp.join(language, file), sr=16000, mono=True)
        intervals = librosa.effects.split(y, top_db=30)
        for interval in intervals:
            if (interval == intervals[0]).all():
                y_no_sil = y[interval[0]: interval[1]]
            else:
                y_no_sil = np.concatenate((y_no_sil, y[interval[0]: interval[1]]))
        mat = librosa.feature.mfcc(y=y_no_sil, sr=sr, n_mfcc=64, n_fft=int(sr*0.025), hop_length=int(sr*0.010))
        mat = mat.T
        if file == list[0]:
            mfcc = mat
        else:
            mfcc = np.concatenate([mfcc, mat], axis=0)
    return mfcc
```

:
```
seq_len = 1000
N_english = eng_mfcc.shape[0] // seq_len
N_english = N_english // seq_len * seq_len
eng_mfcc = eng_mfcc[:N_english * seq_len]
eng_mfcc = eng_mfcc.reshape((N_english, seq_len, 64))
print(eng_mfcc.shape)
```

```
(6000, 1000, 64)
```

:
```
eng_label = np.zeros((N_english, seq_len, 1))
print(eng_label.shape)
```

```
(6000, 1000, 1)
```

From above example, we can change English train as other languages and change the train files to test files.

Combining the feature and label as (batch_size, sequence_length=1000, 65)
Combining all three language training data as the data_all_train
Use the same step, Combining all three language testing data as the data_all_test
split data and label:
train_label = train_set[:,:,-1]
train_data = train_set[:,:,:64]
test_label = test_set[:,:,-1]
test_data = test_set[:,:,:64]
finally, save them into hw5.hdf5.

2. dataloader

I split train and test as 0.85:0.15, then split the rest of train as train and valid as 0.8 : 0.2
X_train, X_val, y_train, y_val = train_test_split(train_data, train_label, test_size=0.2, random_state=10)
X_test = test_data
y_test = test_label

3. Model

```
----------------------------------------------------------------
      Layer (type)                         Output Shape
Param #     Tr. Param #
================================================================
              GRU-1     [64, 1000, 128], [2, 64, 128]
173,568         173,568
           Linear-2                      [64, 1000, 3]
387             387
================================================================
Total params: 173,955
Trainable params: 173,955
Non-trainable params: 0
```

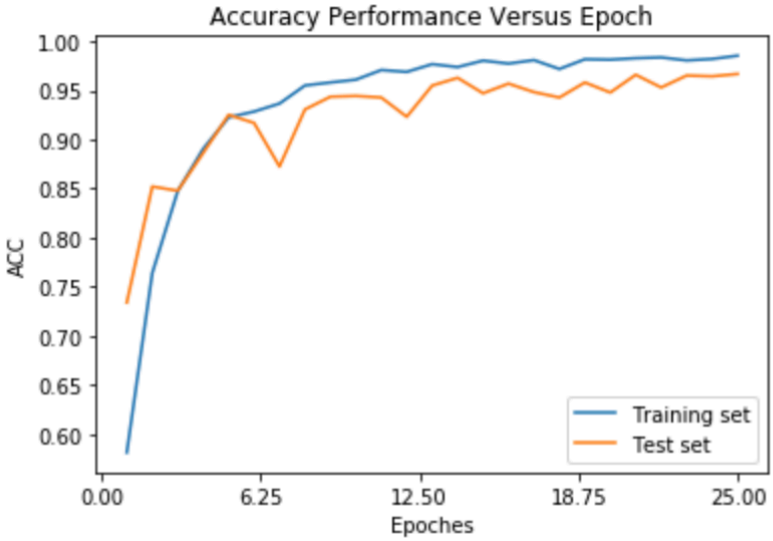
```python
hidden_size = 128
class Model(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(Model, self).__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.rnn = nn.GRU(self.input_size, self.hidden_size, num_layers=2, dropout=0.4, batch_first=True).to(device)
        self.fc = nn.Linear(self.hidden_size, self.output_size).to(device)


    # create function to init state
    def init_hidden(self, batch_size):
        return torch.zeros(2, batch_size, self.hidden_size)


    def forward(self, x):
        batch_size = x.size(0)
        h = self.init_hidden(batch_size).to(device)

        out, h = self.rnn(x, h)
        h = h.to(device)
        out = self.fc(out)

        #return out, h
        return out


model = Model(input_size=64, hidden_size=hidden_size, output_size=classes)
```

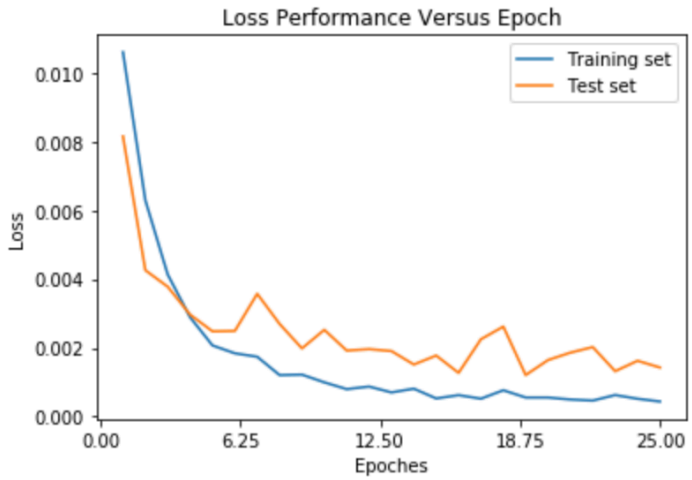
I use 128 hidden layers, 2 layers in GRU and drop out 0.4. From this model I take input as (batch_size, 1000, 64) and get output(batch_size, 1000, 3) to see the result is which language.

4. Training Model

Due to the different number of language file, I set the cross entropy weights as [0.55, 0.09, 0.36] to fit the language. Also use Adam optimizer with 0.001 learning rate.

After 25 epochs, I got a pretty well model result both in validation and training set.



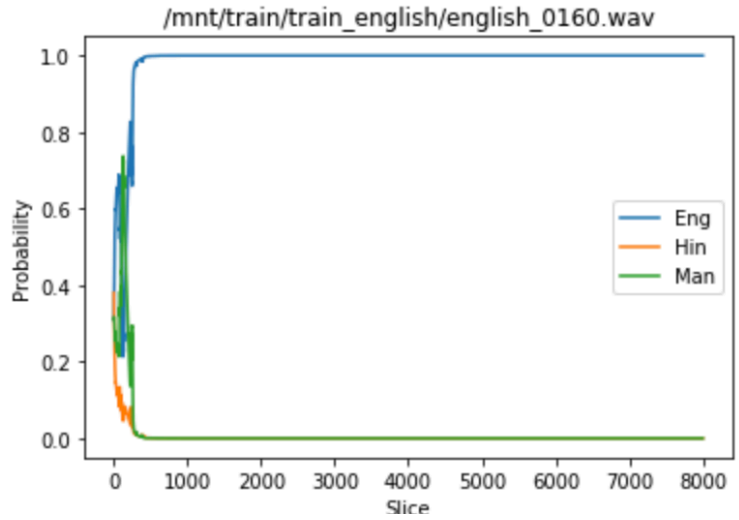


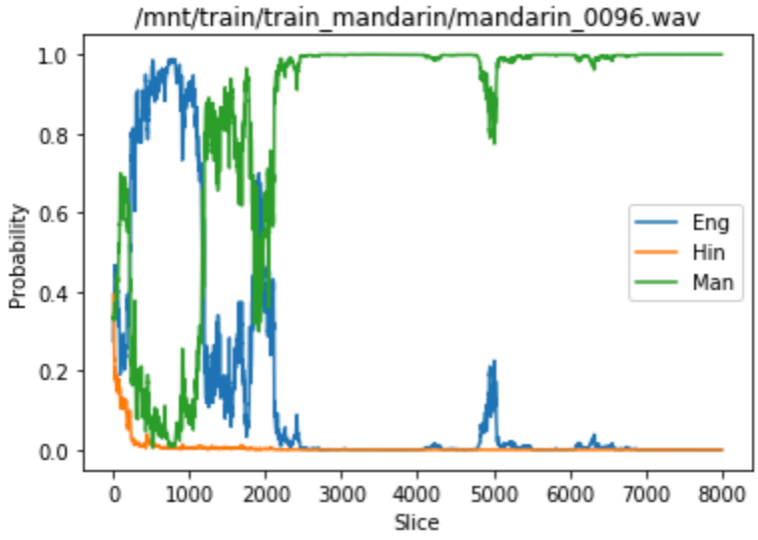
I use the entirely different audio files with preprocessing to test the model in sequenced mode and get a kind of good result.

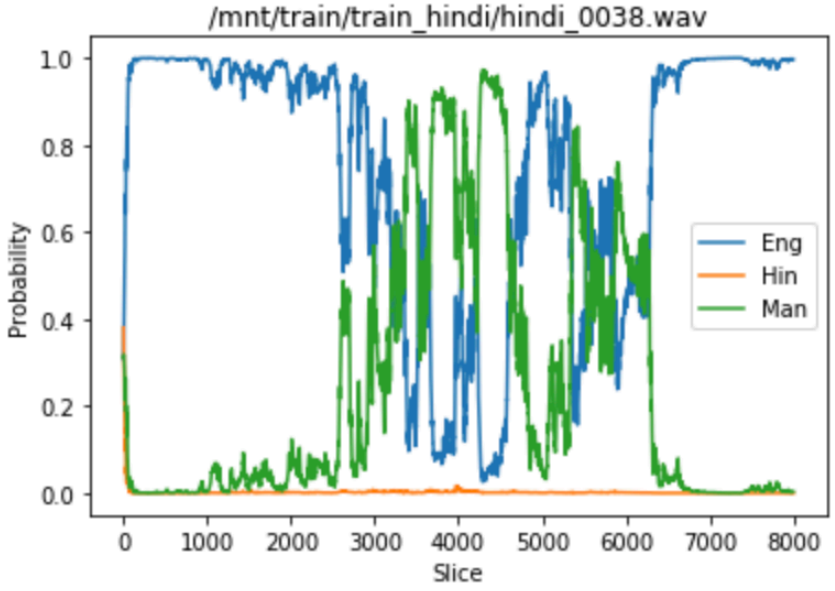
```
Test Loss: 0.0254 Acc: 0.7206
```

4. Streaming Model Test

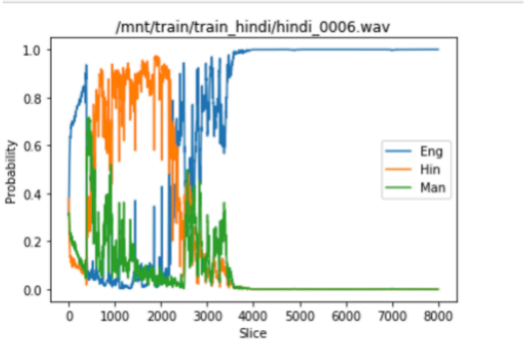
To get each frame performance, I reset the dataloader and reshape some test files data as (1, 8000, 64). Also add a softmax layer to know the probability of these 3 languages.
I randomly choose one of each languages test sample to test the model.

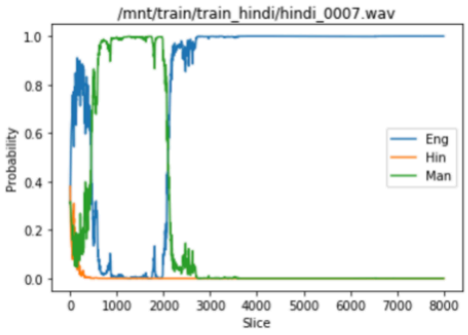


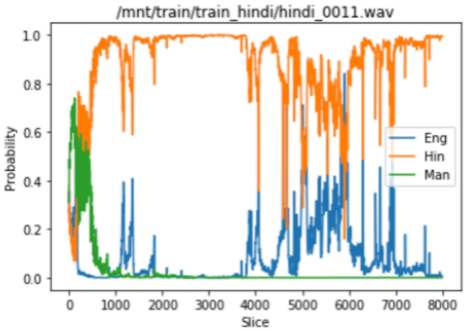


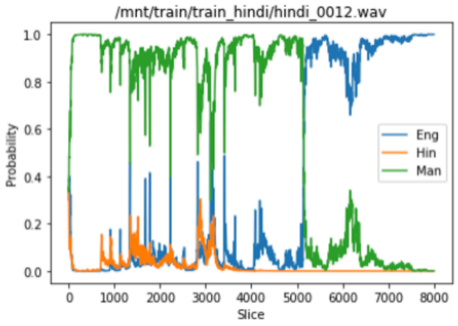


After some experiments, It is obvious that the model can easily identify the English and Mandarin but the ability to identify the Hindi is very low. To figure out, I also test some trained hindi audio to see if the model has some problem or the hindi itself is hard to identify.









We can see that even in training data, it is still hard to identify correctly. Some times to be identified as English, some times Mandarin, some times Hindi. It may caused by the number of hindi example is very small compared to other languages. Also, I cutdown the voice under 30db may lead this result as well. Maybe the frequency of Hindi is very low.