ISSM REPORT, POLICE ALERT SYSTEM

NG SIEW PHENG, TEA LEE SENG, YANG XIAOYAN

Institute of Systems Science, National University of Singapore, Singapore 119615

ABSTRACT

Singapore is a safe country. However, any gun shots become more dangerous in neighbourhood area, due to less awareness in public. It is critical to maintain safety here and automate alerting mechanism to Police will be time and life saving. We propose machine learning mechanism to understand background sound in Urban setup to identify gun shots. We apply techniques like 1D conv neural network, auto-encoder, LSTM, neural network on MFCC to understand which is good for recognizing gun shots in URBANSOUND8K DATASET.

Index Terms— URBANSOUND8K DATASET, long short-term memory, auto encoder, mfcc, conv1D, MaxPooling1D, deep neural network, deep learning

1 Introduction

Dangerous weapons can be a source of dangerous in a safe country. Gun shots become more dangerous in neighbourhood area, due to less awareness in public. It is critical to maintain safety here and automate alerting mechanism to Police will be time and life saving. It is easpecially the case that police station may be far away from the incident area and they do not get alerted unless civilians reported it to them. Even with reports, the actual sound about the incident is lost and therefore losing original data.

2 Related work

We download URBANSOUND8K DATASET from [1] for analysis and classification models training. URBANSOUND8K DATASET was processed for [2] research.

We also refer to Ricky Kim's articles about UrbanSound Classification Part 1 [3] and Part 2 [4] for initial understanding of datasets.

Mike's work [5] on 2d convolution deep neural network on MFCC also share us another direction on utilizing frequency domain representation for analysis.

3 Proposed approach

Based on the nature that all sounds wave files are 1 dimensional signals, sampling at different frequencies, we retrieved wave data via librosa library at default 22khz sampling rate.

We concatenates signals itself to 4 seconds long as signal feature.

We feed the signals to conv1d, auto-encoder, and LSTM networks for classification.

we also convert signals to mfcc representation, apply mean on each mfcc stream and 3 layers dense neural network for classification.

Please refers experimental results section for details.

4 Data preprocessing and analysis

URBANSOUND8K DATASET[1] contains 8732 labeled sounds of 10 classes: air_conditioner, car_horn, children_playing, dog_bark, drilling, enginge_idling, gun_shot, jackhammer, siren, and street_music. The sounds are of various sampling rate as well as sampling rate. We utilize librose to convert sampling rate to 22kHz and mono channel.

```
## uncomment following lines only if data.pk is never generated.
cpuCount=8 # your system cpu count.

def path_class(filename):
    d = data[data['slice_file_name']==filename]
    path_name = os.path.join('data','UrbanSound8K','audio','fold'+str(d.fold.values[0]), filename)
    return path_name, d['class'].values[0]

def loadWave(pn):
    ok=True
    try:
        aWave =  librosa.core.load(pn)
    except TypeError as error:
        print(error)
        ok=False

    return aWave

def parallelize(data, func, num_of_processes=3):
    data_split = np.array_split(data, num_of_processes)
    pool = Pool(num_of_processes)
    data = pd.concat(pool.map(func, data_split))
    pool.close()
    pool.join()
    return data

def run_on_subset(func, data_subset):
    return data_subset.apply(func, axis=1)

def loadWaveRow(x):
    return loadWave(path_class(x['slice_file_name'])[0])

data = pd.read_csv('data/UrbanSound8K/metadata/UrbanSound8K.csv')

data['rawWave']=parallelize(data, partial(run_on_subset, loadWaveRow), cpuCount-1)
data['wave']=data['rawWave'].apply(lambda x :  x[0])
data['len_wave']=data['wave'].apply(lambda x : len(x))

f = open('data.pk', 'wb')
pickle.dump(data, f)
f.close()
```

Fig. 1. sound data loading, multiprocessing.

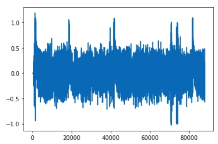
As for sound duration that's less than 4 seconds, we concatenates the sound itself until it fills up the 4 seconds duration.

```python
# reconstruct all sounds into 4 seconds. repeat if shorter than 4(88200 samples) seconds.
#
def repeatTo4SecondsWave(row):
#     print(row[0])

    sampleWave = row
    unit = sampleWave
    while len(sampleWave)<88200:
        sampleWave=np.append(sampleWave,unit)

    return sampleWave[:88200]

data['wave4']=data['wave'].apply(repeatTo4SecondsWave)
data['len_wave4']=data['wave4'].apply(len)

f = open('data-w4.pk', 'wb')
pickle.dump(data, f)
f.close()
```

Fig. 2. self repeating to fill up to 4 seconds.

We also plot and listen to a few gun shots and other sound files. There are ten classes in the data. They have different

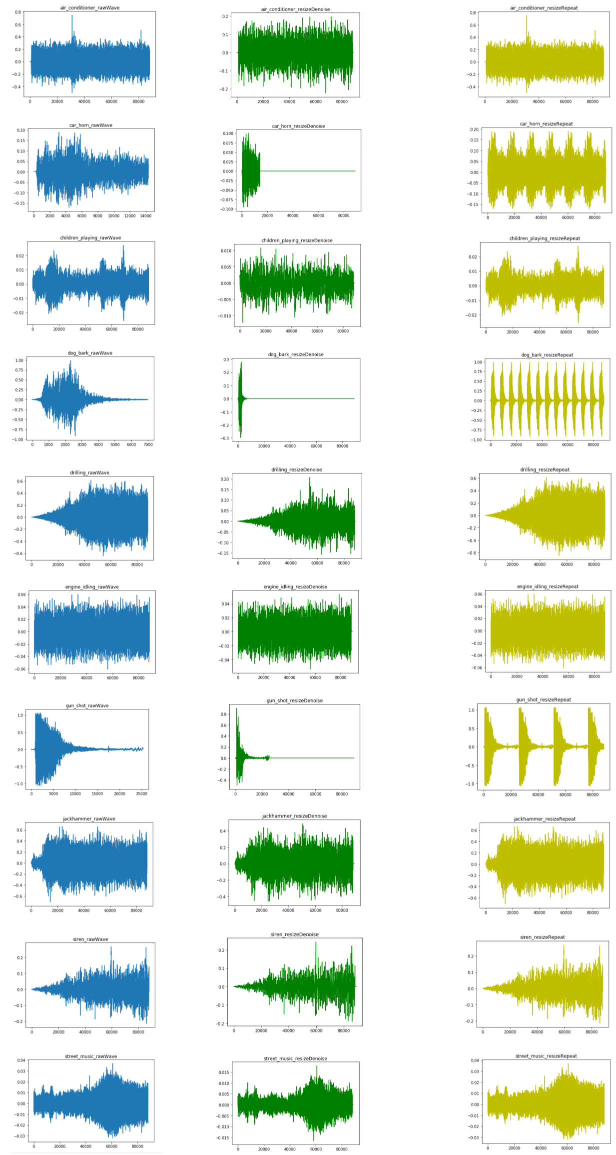
```
In [6]:  # pick 135527-6-14-0.wav and see the signal data. librosa tried to converted it to -1 to 1 range.
         plt.plot(data[data['slice_file_name']=='135527-6-14-0.wav']['wave'].values[0])

Out[6]:  [<matplotlib.lines.Line2D at 0x7f78e42941d0>]
```



```
In [7]:  # try to listen a few other gun shot sounds as well.
         import IPython.display as ipd
         ipd.Audio(path_class('35800-6-0-0.wav')[0])

Out[7]:  0:00 / 0:04
```

Fig. 3. Plot and listen some sound files.

length of waves. After data extraction,need to ensure all the data has same size. We have use two methods to resize the waveform, one method is padding zero and another method is repeat the wave forms until the max. below figures show the plots before and after resizing use the two method.Blue color plots are the raw wave forms, green plots applied zero padding and yellow applied repeating.



(raw wave) (resize with zeros) (resize with repeating)

Fig. 4. sounds types, and 2 resizing schemes to 4 seconds.

5 Experimental results

5.1 Conv1D deep neural network classification

We constructed a four layers of Convolutional Neural Networks model for the wave data. the input data is 1D array which has 89007 features.

```python
def createModel():
    model = Sequential()
    model.add(Conv1D(32, 3, activation='relu', input_shape = (numCols,1)))

    model.add(MaxPooling1D(pool_size= 2))
    model.add(Dropout(0.2))

    model.add(Conv1D(128, 2,activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.2))

    model.add(Conv1D(256, 2,activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.25))

    model.add(Conv1D(256, 2,activation='relu'))

    model.add(Flatten())
    model.add(Dense(32, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    return model
```

Fig. 5. model layers.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_4 (Conv1D)            (None, 89007, 32)         128
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 44503, 32)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 44503, 32)         0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 44502, 128)        8320
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 22251, 128)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 22251, 128)        0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 22250, 256)        65792
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 11125, 256)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 11125, 256)        0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 11124, 256)        131328
_________________________________________________________________
flatten_1 (Flatten)          (None, 2847744)           0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                91127840
_________________________________________________________________
dense_3 (Dense)              (None, 10)                330
=================================================================
Total params: 91,333,738
Trainable params: 91,333,738
Non-trainable params: 0
```

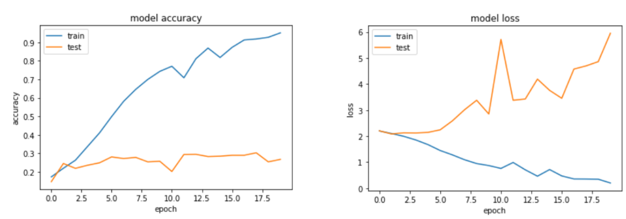
Fig. 6. Model summary.

After data training and validation, the accuracy and the loss function showed training has good result but validation not.

```
Best accuracy (on test data set): 24.37%
                  precision    recall  f1-score   support

 air_conditioner     0.3433    0.2347    0.2788        98
        car_horn     0.1212    0.2162    0.1553        37
children_playing     0.1810    0.2100    0.1944       100
        dog_bark     0.2857    0.3232    0.3033        99
        drilling     0.2000    0.1111    0.1429       108
   engine_idling     0.5714    0.1250    0.2051        96
        gun_shot     0.6875    0.6471    0.6667        34
      jackhammer     0.2189    0.4190    0.2876       105
           siren     0.1605    0.1512    0.1557        86
    street_music     0.2203    0.2342    0.2271       111

        accuracy                         0.2437       874
       macro avg     0.2990    0.2672    0.2617       874
    weighted avg     0.2810    0.2437    0.2393       874

[[23  3 18  6  9  1  0 27  5  6]
 [ 2  8  4  2  2  1  5  8  1  4]
 [ 6  5 21 15  8  0  1 14 17 13]
 [ 1 10 15 32 11  0  2  6 11 11]
 [ 1 11 17 16 12  0  0 24 10 17]
 [18  7 14  4  2 12  1 20  3 15]
 [ 1  2  0  3  1  2 22  3  0  0]
 [ 3  8  6  9  6  0  1 44  9 19]
 [ 5  4 14 19  5  1  0 18 13  7]
 [ 7  8  7  6  4  4  0 37 12 26]]
```

Fig. 7. CNN accuracy score and confusion matrix

We used a test data set did a prediction for the sounds classes, the result is 24.37%. The gun shot prediction is 66.67%.



(model accuracy) (model loss)

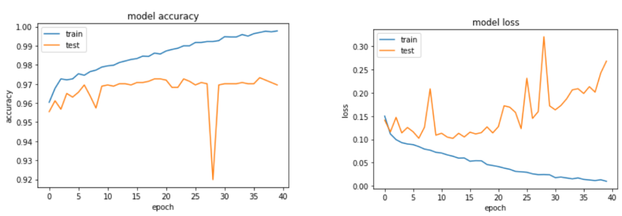
Fig. 8. Model accuracy and model loss

Since our system is to check the dangerous sound (gun shot), we grouped non gun shot sounds as one class and gun shot as one class. Did another training for the data.

We used the test data set did a prediction for the sounds classes, the result is 97.14%, but the gun shot prediction is 57.63%.

```
Best accuracy (on test data set): 97.14%
              precision    recall  f1-score   support

      normal     0.9800    0.9905    0.9852       840
    gun_shot     0.6800    0.5000    0.5763        34

    accuracy                         0.9714       874
   macro avg     0.8300    0.7452    0.7807       874
weighted avg     0.9683    0.9714    0.9693       874

[[832   8]
 [ 17  17]]
```

Fig. 9. CNN accuracy score and confusion matrix



(model accuracy) (model loss)

Fig. 10. Model accuracy and model loss

The accuracy of CNN is not good, we switch to other models.

5.2 Autoencoder anomaly detection

Based on 4 seconds self-repeat sound, we construct autoencoder and train with sounds sample without gun shots. Below is the code snipet for the autoencoder network.

```
In [62]:  # Autoencoder model

          from tensorflow.keras.utils import plot_model
          from tensorflow.keras.datasets import fashion_mnist
          from tensorflow.keras.models import Model
          from tensorflow.keras.models import Sequential
          from tensorflow.keras.layers import Dense, Input,Conv2D

          # slow memory growth to monitor GPU RAM usage for model.
          os.environ["TF_FORCE_GPU_ALLOW_GROWTH"]="true"


          # Dimension of input signal, determined by your test data
          signal_dimension = 88200

          # Dimension of encoded features, can be adjusted by user
          encoder_dimension = signal_dimension/10000
          decoder_dimension = signal_dimension/1000


          # training setup
          EPOCHS = 10
          BATCH_SIZE = 64

          # Configuration on autoencoder model
          def AE_method0(x_train):
              data_dim = x_train.shape[1]
              autoencoder = Sequential()
              autoencoder.add(Dense(encoder_dimension, activation='relu', input_shape=(signal_dimension, ), name='layer1'
              autoencoder.add(Dense(signal_dimension, activation='relu',name='layer2_1'))
              autoencoder.compile(optimizer='adam', loss='mse')
              autoencoder.fit(x_train, x_train, epochs=EPOCHS, batch_size=BATCH_SIZE, shuffle=True)

              encoder = Model(inputs=autoencoder.input, outputs=autoencoder.get_layer('layer1').output)
              return encoder, autoencoder



          model_encoder, model_autoencoder = AE_method0(x_train=data_train)
          model_encoder.summary()
          model_autoencoder.summary()
```

Fig. 11. Autoencoder network.

```
Epoch 1/10
8358/8358 [==============================] - 2s 282us/sample - loss: 0.0101
Epoch 2/10
8358/8358 [==============================] - 2s 248us/sample - loss: 0.0101
Epoch 3/10
8358/8358 [==============================] - 2s 246us/sample - loss: 0.0101
Epoch 4/10
8358/8358 [==============================] - 2s 248us/sample - loss: 0.0101
Epoch 5/10
8358/8358 [==============================] - 2s 247us/sample - loss: 0.0101
Epoch 6/10
8358/8358 [==============================] - 2s 248us/sample - loss: 0.0101
Epoch 7/10
8358/8358 [==============================] - 2s 247us/sample - loss: 0.0101
Epoch 8/10
8358/8358 [==============================] - 2s 247us/sample - loss: 0.0101
Epoch 9/10
8358/8358 [==============================] - 2s 241us/sample - loss: 0.0100
Epoch 10/10
8358/8358 [==============================] - 2s 237us/sample - loss: 0.0100
Model: "model_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer1_input (InputLayer)    [(None, 88200)]           0
_________________________________________________________________
layer1 (Dense)               (None, 8)                 705608
=================================================================
Total params: 705,608
Trainable params: 705,608
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer1 (Dense)               (None, 8)                 705608
_________________________________________________________________
layer2_1 (Dense)             (None, 88200)             793800
=================================================================
Total params: 1,499,408
Trainable params: 1,499,408
Non-trainable params: 0
_________________________________________________________________
```

Fig. 12. Autoencoder network training and its structure

By training autoencoder network with non gun shots sound data, we hope that gun shots sound will be recognized as anomaly to the network. Due to 4 seconds and sampling rate of 22kHz, our data will be 1D array of 88.200k samples. From Fig. 12, layer1, which is the innermost layer of undercomplete autoencoder neural network, it has only 8 neurons.

The code to analyse performance of training data and gun shot(noise, test) anomaly detection is as below.

Based on the errors collected from network prediction on training data and gun shot data, we calculate mean and standard deviation, std. With 95% confidence level as guideline, we set the threshold to mean+2*std, which is 62.17.

```python
In [38]:
# Perform prediction using the trained autoencoder model
noise_err=[]

for i in range(len(data_test)):
    test_rec = model_autoencoder.predict(data_test[i:i+1,:])
    noise_err.append(np.linalg.norm(data_test[i:i+1,:]-test_rec, axis=-1))

train_err=[]
for i in range(len(data_train)):
    train_rec = model_autoencoder.predict(data_train[i:i+1,:])
    train_err.append(np.linalg.norm(data_train[i:i+1,:]-train_rec, axis=-1))

plt.style.use('ggplot')
plt.plot(noise_err, color='red')
plt.plot(train_err, color='blue')

print('max of test, ', np.max(noise_err), ' max of train along with test:', np.max(train_err[:len(noise_err)]),

print('mean of test,', np.mean(noise_err), 'mean of train:', np.mean(train_err))
print('std of test,', np.std(noise_err), 'std of train:', np.std(train_err))

print('threshold is 2 STD of training mean:',np.mean(train_err)+2*np.std(train_err),'\n')

train_anomaly = np.sum(np.array(train_err)>np.mean(train_err)+2*np.std(train_err))
noise_anomaly = np.sum(np.array(noise_err)>np.mean(train_err)+2*np.std(train_err))

print('case of training anomaly based on 95% confidence level training model:', train_anomaly)

print('case of gun_shot anomaly based on 95% confidence level training model:', noise_anomaly)

print('percentage of training anomaly : ',train_anomaly/len(train_err)*100, '%') # 4 percent
print('percentage of gunshot anomaly : ', noise_anomaly/len(noise_err)*100, '%') # noise has 31 percent error th
```

Fig. 13. Auto encoder performance analysis code snipet

```
max of test,  242.28867  max of train along with test: 91.33413  max of train 204.6156
mean of test, 44.728745 mean of train: 21.946918
std of test, 32.00736 std of train: 20.110588
threshold is 2 STD of training mean: 62.168094635009766

case of training anomaly based on 95% confidence level training model: 352
case of gun_shot anomaly based on 95% confidence level training model: 114
percentage of training anomaly :  4.211533859775066 %
percentage of gunshot anomaly :  30.481283422459892 %
```

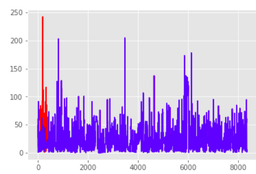


Fig. 14. Autoencoder performance analysis

With threshold of 62.17 in Fig. 14, we found out that only 4.2% of training data is anomaly, whereas gun shot data anomaly rate is about 30.48% at 95% confidence level.

Though the performance is not amazing compared to other methods in later parts of experiements. It is interesting that autoencoder can detect anomaly even within its own training data. It is useful that when we don't have much labelled data but unsupervised learning like this, helps to point out anomaly.

We also notice another counter intuitive perspective about autoencoder. For neural network, more neurons tends to performance better. However, this is not the case for autoencoder. We configure 100x more neurons for innermost layer, and spend more times on compute. And here is the performance of the network,

```
Epoch 1/10
8358/8358 [==============================] - 6s 686us/sample - loss: 0.0101
Epoch 2/10
8358/8358 [==============================] - 5s 554us/sample - loss: 0.0102
Epoch 3/10
8358/8358 [==============================] - 5s 550us/sample - loss: 0.0101
Epoch 4/10
8358/8358 [==============================] - 5s 540us/sample - loss: 0.0104
Epoch 5/10
8358/8358 [==============================] - 5s 539us/sample - loss: 0.0105
Epoch 6/10
8358/8358 [==============================] - ETA: 0s - loss: 0.011 - 5s 545us/sample - loss: 0.0117
Epoch 7/10
8358/8358 [==============================] - 5s 540us/sample - loss: 0.0116
Epoch 8/10
8358/8358 [==============================] - 5s 550us/sample - loss: 0.0141
Epoch 9/10
8358/8358 [==============================] - 5s 541us/sample - loss: 0.0127
Epoch 10/10
8358/8358 [==============================] - 5s 541us/sample - loss: 0.0133
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer1_input (InputLayer)    [(None, 88200)]           0
_________________________________________________________________
layer1 (Dense)               (None, 882)               77793282
=================================================================
Total params: 77,793,282
Trainable params: 77,793,282
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer1 (Dense)               (None, 882)               77793282
_________________________________________________________________
layer2_1 (Dense)             (None, 88200)             77880600
=================================================================
Total params: 155,673,882
Trainable params: 155,673,882
Non-trainable params: 0
_________________________________________________________________
```

Fig. 15. Autoencoder with 800 innermost layer

```
max of test,  242.27917  max of train along with test: 91.30074  max of train 1623.5248
mean of test, 44.730824 mean of train: 22.357073
std of test, 32.01389 std of train: 30.583294
threshold is 2 STD of training mean: 83.52366065979004

case of training anomaly based on 95% confidence level training model: 123
case of gun_shot anomaly based on 95% confidence level training model: 18
percentage of training anomaly :  1.4716439339554916 %
percentage of gunshot anomaly :  4.81283422459893 %
```

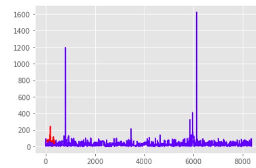


Fig. 16. Performance of Autoencoder with 800 innermost layer

The performance on gunshot anomaly detection is only 4.8%, which is only 16% of 8 neurons network. Therefore, more under-complete autoencoder network likely pressing the neural network to perform better.

5.3 LSTM neural network classifcation

Long short-term memory(LSTM) neural network is another typical tool to handle 1D time series data. Below is the network structure we deployed for classification model training. Due to slow training performance in LSTM, we deployed conv1D layers and multiple MaxPooling1D(16) to reduce data while keeping peak values to LSTM layer.

```
seed = 29
np.random.seed(seed)

def createLstmModel(x_train):
#     FINAL_DIM = 900
    data_dim = x_train.shape[1]
    inputs = Input(shape=(x_train.shape[1],1))
    y = Conv1D(256, 11, activation='relu') (inputs)
    y = MaxPooling1D(16)(y)
    y = Conv1D(128, 5, activation='relu') (y)
    y = Dropout(0.25)(y)
    y = MaxPooling1D(16)(y)
    y = Conv1D(64, 3, activation='relu') (y)
    y = MaxPooling1D(8)(y)
    y = Conv1D(32, 3, activation='relu') (y)

    y = LSTM(32,
        return_sequences=True,
        dropout=0.5,
        recurrent_dropout=0.5)(y)
    y = LSTM(32)(y)
    y = Dense(10, activation='sigmoid')(y)

    model = Model(inputs=inputs, outputs = y)
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
#     model.summary()

    return model



lstmModel = createLstmModel(x_train=data_train)
lstmModel.summary()
```

Fig. 17. LSTM network structures

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/init_ops.py:1251: calling \
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 88200, 1)]        0
_________________________________________________________________
conv1d (Conv1D)              (None, 88190, 256)        3072
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 5511, 256)         0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 5507, 128)         163968
_________________________________________________________________
dropout (Dropout)            (None, 5507, 128)         0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 344, 128)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 342, 64)           24640
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 42, 64)            0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 40, 32)            6176
_________________________________________________________________
lstm (LSTM)                  (None, 40, 32)            8320
_________________________________________________________________
lstm_1 (LSTM)                (None, 32)                8320
_________________________________________________________________
dense (Dense)                (None, 10)                330
=================================================================
Total params: 214,826
Trainable params: 214,826
Non-trainable params: 0
_________________________________________________________________
```

Fig. 18. LSTM network layers

From the initial training, even with maxPooling1D(16), we roughly estimated that LSTM training takes more epochs to improve.

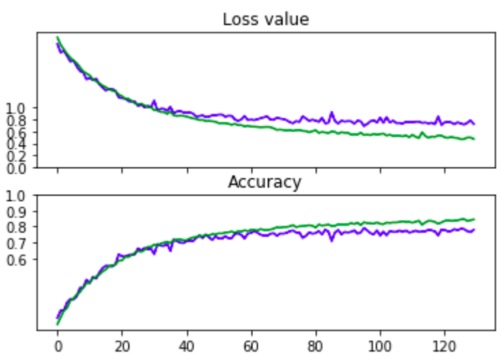
```
Train on 6069 samples, validate on 1776 samples
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_grad.py:1250: add_disp
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Epoch 1/130
6069/6069 [==============================] - 124s 21ms/sample - loss: 2.1572 - acc: 0.1883 - val_loss: 2.0491 -
Epoch 2/130
6069/6069 [==============================] - 120s 20ms/sample - loss: 2.0371 - acc: 0.2297 - val_loss: 1.9071 -
Epoch 3/130
6069/6069 [==============================] - 120s 20ms/sample - loss: 1.9578 - acc: 0.2730 - val_loss: 1.9394 -
Epoch 4/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.8823 - acc: 0.2905 - val_loss: 1.8526 -
Epoch 5/130
6069/6069 [==============================] - 120s 20ms/sample - loss: 1.8222 - acc: 0.3358 - val_loss: 1.7568 -
Epoch 6/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.7727 - acc: 0.3498 - val_loss: 1.7750 -
Epoch 7/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.7306 - acc: 0.3727 - val_loss: 1.6786 -
Epoch 8/130
6069/6069 [==============================] - 120s 20ms/sample - loss: 1.6604 - acc: 0.3903 - val_loss: 1.5993 -
Epoch 9/130
6069/6069 [==============================] - 120s 20ms/sample - loss: 1.5935 - acc: 0.4182 - val_loss: 1.5785 -
Epoch 10/130
6069/6069 [==============================] - 120s 20ms/sample - loss: 1.5599 - acc: 0.4446 - val_loss: 1.4625 -
Epoch 11/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.5283 - acc: 0.4503 - val_loss: 1.4840 -
Epoch 12/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.4660 - acc: 0.4749 - val_loss: 1.4528 -
Epoch 13/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.4291 - acc: 0.4853 - val_loss: 1.4721 -
Epoch 14/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.3973 - acc: 0.4986 - val_loss: 1.3823 -
Epoch 15/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.3580 - acc: 0.5175 - val_loss: 1.3260 -
Epoch 16/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.3315 - acc: 0.5307 - val_loss: 1.2777 -
Epoch 17/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.2937 - acc: 0.5528 - val_loss: 1.3047 -
Epoch 18/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.2764 - acc: 0.5548 - val_loss: 1.3022 -
Epoch 19/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.2648 - acc: 0.5622 - val_loss: 1.2703 -
Epoch 20/130
6069/6069 [==============================] - 119s 20ms/sample - loss: 1.2092 - acc: 0.5859 - val_loss: 1.1624 -
Epoch 21/130
6069/6069 [==============================] - 120s 20ms/sample - loss: 1.1828 - acc: 0.5869 - val_loss: 1.1499 -
```

Fig. 19. LSTM training, beginging phase

```
========================] - 119s 20ms/sample - loss: 0.4860 - acc: 0.8356 - val_loss: 0.7603 - val_acc: 0.7658

========================] - 119s 20ms/sample - loss: 0.5851 - acc: 0.8107 - val_loss: 0.7392 - val_acc: 0.7675

========================] - 120s 20ms/sample - loss: 0.5210 - acc: 0.8250 - val_loss: 0.7448 - val_acc: 0.7691

========================] - 120s 20ms/sample - loss: 0.4917 - acc: 0.8354 - val_loss: 0.7320 - val_acc: 0.7793

========================] - 120s 20ms/sample - loss: 0.5127 - acc: 0.8337 - val_loss: 0.7424 - val_acc: 0.7748

========================] - 119s 20ms/sample - loss: 0.5073 - acc: 0.8276 - val_loss: 0.7168 - val_acc: 0.7686

========================] - 119s 20ms/sample - loss: 0.5306 - acc: 0.8161 - val_loss: 0.8461 - val_acc: 0.7438

========================] - 119s 20ms/sample - loss: 0.5280 - acc: 0.8248 - val_loss: 0.7064 - val_acc: 0.7804

========================] - 119s 20ms/sample - loss: 0.5003 - acc: 0.8349 - val_loss: 0.7476 - val_acc: 0.7782

========================] - 120s 20ms/sample - loss: 0.4905 - acc: 0.8379 - val_loss: 0.7555 - val_acc: 0.7691

========================] - 120s 20ms/sample - loss: 0.5068 - acc: 0.8349 - val_loss: 0.7460 - val_acc: 0.7708

========================] - 120s 20ms/sample - loss: 0.4930 - acc: 0.8359 - val_loss: 0.7175 - val_acc: 0.7832

========================] - 119s 20ms/sample - loss: 0.4881 - acc: 0.8375 - val_loss: 0.7486 - val_acc: 0.7759

========================] - 120s 20ms/sample - loss: 0.4686 - acc: 0.8438 - val_loss: 0.7295 - val_acc: 0.7855

========================] - 119s 20ms/sample - loss: 0.4719 - acc: 0.8463 - val_loss: 0.7124 - val_acc: 0.7832

========================] - 120s 20ms/sample - loss: 0.4958 - acc: 0.8337 - val_loss: 0.7390 - val_acc: 0.7703

========================] - 119s 20ms/sample - loss: 0.4968 - acc: 0.8385 - val_loss: 0.7799 - val_acc: 0.7641

========================] - 119s 20ms/sample - loss: 0.4732 - acc: 0.8435 - val_loss: 0.7210 - val_acc: 0.7793
```

Fig. 20. LSTM training, ending phase

And based on following chart, LSTM training indeed taking more time to reach plateu by around 100 epochs, which is way more epochs than autoencoder as well as Conv1D network. This is likely due to slow propagation of feedback via memory unit cell in LSTM.



(green color is validation)

Fig. 21. LSTM training accuracy and loss

From figure below, we observe that our LSTM model achieve 78.69% accuracy on various sound classification. For gun shot sound, the F1 score is 0.8732, which is 2nd best classifiable sounds. Car_horn and drilling sound classification tasks dragged down the overall model performance.

```
Best accuracy (on testing dataset): 78.69%
                  precision    recall  f1-score   support

        dog_bark     0.6923    0.8889    0.7784        81
children_playing     0.7674    0.6471    0.7021        51
        car_horn     0.6701    0.6500    0.6599       100
 air_conditioner     0.8700    0.8286    0.8488       105
    street_music     0.8462    0.7624    0.8021       101
        gun_shot     0.8942    0.8532    0.8732       109
           siren     1.0000    0.8974    0.9459        39
    engine_idling    0.8365    0.8788    0.8571        99
      jackhammer     0.8476    0.8558    0.8517       104
        drilling     0.5769    0.6122    0.5941        98

        accuracy                         0.7869       887
       macro avg     0.8001    0.7874    0.7913       887
    weighted avg     0.7926    0.7869    0.7877       887

[[72  0  1  0  1  0  0  2  1  4]
 [ 5 33  3  1  3  0  0  0  0  6]
 [ 2  0 65  6  3  2  0  0  5 17]
 [ 0  1  9 87  1  1  0  1  2  3]
 [ 5  0  0  2 77  1  0  8  2  6]
 [ 4  2  1  0  1 93  0  5  0  3]
 [ 0  0  0  0  1  2 35  0  0  1]
 [ 5  0  0  1  3  2  0 87  0  1]
 [ 2  0  6  1  0  3  0  0 89  3]
 [ 9  7 12  2  1  0  0  1  6 60]]
```

Fig. 22. LSTM classification performance

5.4 MFCC representation and neural network classification

In the earlier model using CNN, we were extracting each audio file as a floating point time series to train our model. However, this method does not capture envelope of the short time power spectrum characteristics of the audio. As such, we decided to use MFCC feature extraction function in librosa as per [5]. However, we are different in that we MFCC our audio file and load the mean of each MFCC bin, into our Neural network (NN) model to see if model accuracy improved.

```
num_epochs = 100
num_batch_size = 32
seed = 29
np.random.seed(seed)
```

Based on the code that we referenced from , we started our training by extracting MFCC feature of our audio file into 40 ”bins”.

```
def extract_audio_features_from(audio):

    try:
        sample_rate=22050
        mfccs = librosa.feature.mfcc(y=audio,
                                     sr=sample_rate,
                                     n_mfcc=n_mfcc)
        mfccsscaled = np.mean(mfccs.T,axis=0)

    except Exception as e:
        print("Error encountered while parsing file: ",
              file_name)
        print(e)
        return None

    return mfccsscaled
```

Let’s take a look at MFCC spectrogram of 2 audio files, namely Children Playing and Gun Shot.

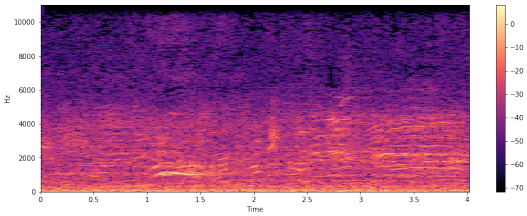


Fig. 23. Spectrogram of Children Playing audio

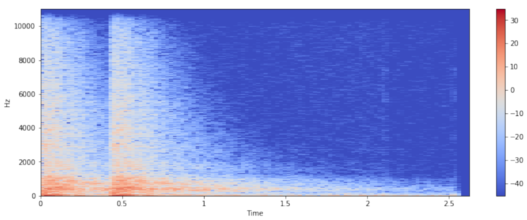


Fig. 24. Spectrogram of Gun Shot audio

You can see these 2 audio files have very different spectrogram where gunshot has a high pitch at the beginning of the audio file. With that, we will start training our model.

As our class label is categorical value, we need to do a one time label encoding to cover them to numeric for our model to work.

```
le = LabelEncoder()
yy = to_categorical(le.fit_transform(y))
```

We split the dataset to 70%, 20% and 10% for training, validation and testing purpose respectively.

```
mask = np.random.rand(len(data))
train_mask = mask ¡ 0.7
validation_mask = np.logical_and(mask ¿ 0.7, mask ¡ 0.9)
test_mask = mask ¿ 0.9
```

For the training, we are using a CNN model. As the model is to predict and classify to 10 audio classes, our model final output layer will be 10.

```
model = Sequential()
model.add(Dense(256, input_shape=(n_mfcc,)))
model.add(Activation(”relu”))
model.add(Dropout(0.25))
model.add(Dense(64))
model.add(Activation(”relu”))
model.add(Dropout(0.25))
model.add(Dense(128))
model.add(Activation(”relu”))
model.add(Dropout(0.25))
model.add(Dense(256))
model.add(Activation(”relu”))
model.add(Dropout(0.25))
model.add(Dense(512))
model.add(Activation(řelu”))
model.add(Dropout(0.5))
model.add(Dense(yy.shape[1]))
model.add(Activation(”softmax”))
model.compile(loss=”categorical_crossentropy”,
metrics=[”accuracy”], optimizer=”adam”)
model.summary()

history= model.fit(x_train, y_train,
batch_size=num_batch_size,
epochs=num_epochs,
validation_data=(x_valid, y_valid),
callbacks=[checkpointer],
verbose=1)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               10496
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                16448
_________________________________________________________________
activation_1 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               8320
_________________________________________________________________
activation_2 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               33024
_________________________________________________________________
activation_3 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               131584
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 205,002
Trainable params: 205,002
Non-trainable params: 0
```

Fig. 25. CNN Model for n_mfcc=40

```
score=model.evaluate(x_train,y_train,verbose=0)
score=model.evaluate(x_test,y_test,verbose=0)

Training Accuracy: 0.92509854
Testing Accuracy: 0.8630435
```

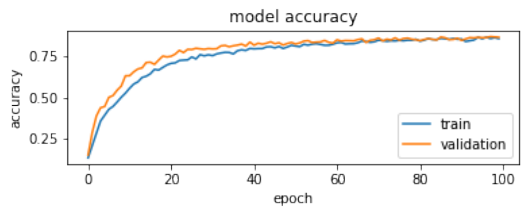


Fig. 26. Accuracy for CNN Model with n_mfcc=40

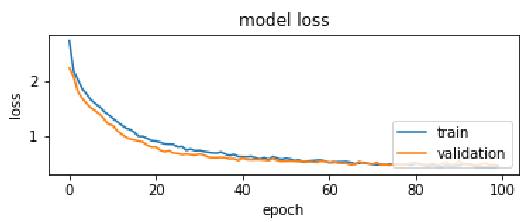


Fig. 27. Loss for CNN Model with n_mfcc=40

```
Best accuracy (on testing dataset): 86.30%
                  precision    recall  f1-score   support

        dog_bark     0.8750    0.9722    0.9211       108
children_playing     0.9000    0.7500    0.8182        48
        car_horn     0.6806    0.8305    0.7481       118
 air_conditioner     0.8304    0.8455    0.8378       110
    street_music     0.9394    0.8942    0.9163       104
        gun_shot     0.9444    0.9341    0.9392        91
           siren     0.8929    0.5435    0.6757        46
    engine_idling    0.9565    0.9565    0.9565        92
     jackhammer     0.9623    0.9273    0.9444       110
        drilling     0.7753    0.7419    0.7582        93

        accuracy                         0.8630       920
       macro avg     0.8757    0.8396    0.8516       920
    weighted avg     0.8696    0.8630    0.8624       920
```

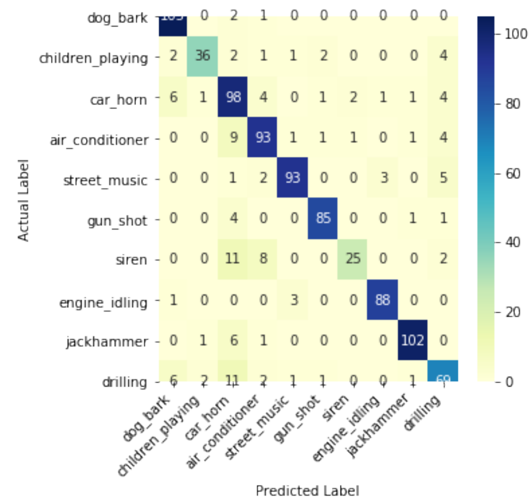


Fig. 28. Confusion Matrix for CNN Model with n_mfcc=40

Using MFCC features of the audio files for training, has improved the model overall accuracy, from 24.26% to 86.30%. And looking at the confusion matrix, we can see a small number of audio files were missed classified. Let's take a quick look some of them.

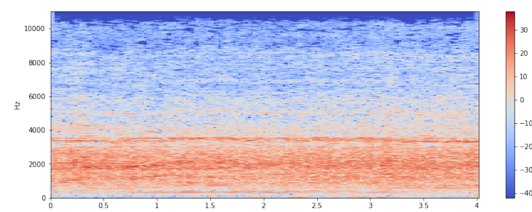


Fig. 29. Spectrogram for a Drilling audio

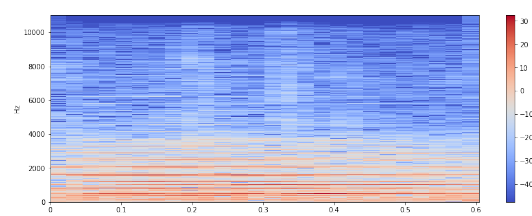


Fig. 30. Spectrogram for a Drilling audio

With the classifier model, we are going to test it using a audio file where we created by overlaying a gunshot audio on top of a children playing audio at 6.5 seconds. Here is what the MFCC spectrogram looks like:

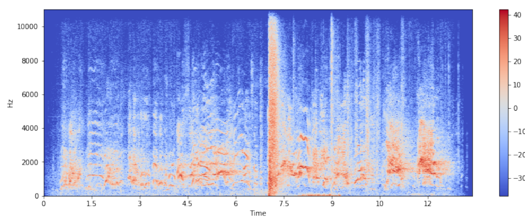


Fig. 31. Spectrogram for a mixed audio

The predicted class is: children_playing

air_conditioner	0.03%
car_horn	0.02%
children_playing	69.43%
dog_bark	15.47%
drilling	0.48%
engine_idling	0.45%
gun_shot	11.01%
jackhammer	0.00%
siren	2.32%
street_music	0.80%

Using this current model (with overall accuracy of 86.30%), we were about to identify the presence of gunshot sound within the children playing. But we wonder if we can improve the model accuracy further by increasing the number of MFCC bins. Thus next, we will explore this parameter.

Our approach is to increase the MFCC bin size from 40 to 60 to extract more features from the audio file. And due to the increase in features, we needed more epochs to train the model.

n_mfcc = 60
num_epochs = 200
num_batch_size = 64

Training Accuracy: 0.97848225
Testing Accuracy: 0.90434784

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               15616
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                16448
_________________________________________________________________
activation_1 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               8320
_________________________________________________________________
activation_2 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               33024
_________________________________________________________________
activation_3 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               131584
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 210,122
Trainable params: 210,122
Non-trainable params: 0
```

Fig. 32. CNN Model for n_mfcc=60

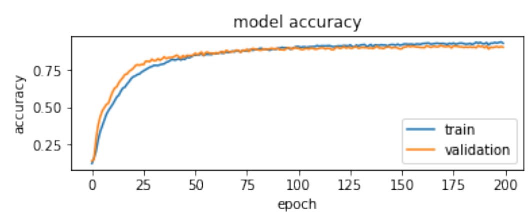


Fig. 33. Accuracy for CNN Model with n_mfcc=60

With the increase in the number of MFCC bins, we were able to improve the overall model accuracy from 86.30% to 90.40%. An improvement in accuracy by 7.23% with 2.50% increase in total params in the model.

The predicted class is: children_playing

air_conditioner	0.08%
car_horn	0.00%
children_playing	98.15%
dog_bark	0.52%
drilling	0.11%
engine_idling	0.14%
gun_shot	0.85%
jackhammer	0.01%
siren	0.06%
street_music	0.07%

We observed that the improvement in model accuracy has increase the prediction probability for the children playing

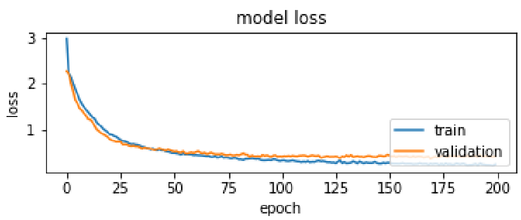


Fig. 34. Loss for CNN Model for n_mfcc=60

```
Best accuracy (on testing dataset): 90.43%
                  precision    recall  f1-score   support

        dog_bark     0.8992    0.9907    0.9427       108
children_playing     0.9286    0.8125    0.8667        48
        car_horn     0.8835    0.7712    0.8235       118
 air_conditioner     0.9192    0.8273    0.8708       110
    street_music     0.9515    0.9423    0.9469       104
        gun_shot     0.9263    0.9670    0.9462        91
           siren     0.9512    0.8478    0.8966        46
    engine_idling    0.9684    1.0000    0.9840        92
      jackhammer     0.9722    0.9545    0.9633       110
        drilling     0.7130    0.8817    0.7885        93

        accuracy                         0.9043       920
       macro avg     0.9113    0.8995    0.9029       920
    weighted avg     0.9091    0.9043    0.9044       920
```

class. And the second highest probability is the gun shot sound that we have overlay-ed on top of children playing.

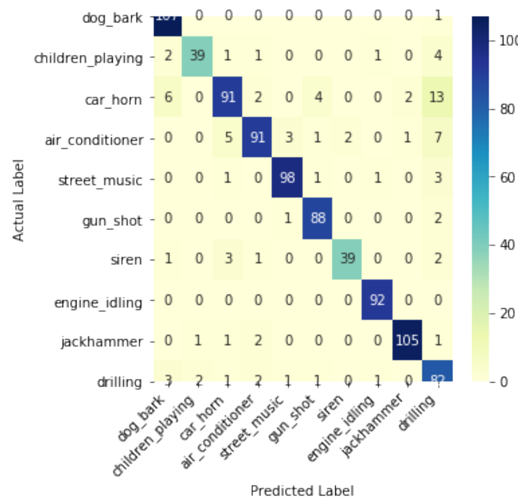


Fig. 35. Confusion Matrix for model using n_mfcc=60

6 Conclusions

With findings above, we summarize our classification performance as following table:

Table 1. The performance comparison.

Approaches	Classification Accuracy
Conv1D	0.2437
Autoencoder	0.3048
LSTM	0.7869
MFCC NN	0.9043

From experiments, we can conclude following ideas: 1. MFCC converts time series signal to frequency domain, which in this case, well suit for classification. 2. But if time series signal isn't well suit, as maybe it doesn't have clear frequency bands within the signal, then LSTM probably can handle those time series signals well enough. At least, it is able to handle time series signal without special handling in this case. 3. autoencoder is interesting to use especially when no clear data labelling. 4. conv1d isn't performed well, if compared to LSTM, which means that memory unit designed in LSTM does have advantage over normal convolution network.

7 References

[1] "Urbansound8k dataset," .

[2] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *22nd ACM International Conference on Multimedia (ACM-MM'14)*, Orlando, FL, USA, Nov. 2014, pp. 1041–1044.

[3] Ricky Kim, "Urban sound classification - part 1: sound wave, digital audio signal," .

[4] Ricky Kim, "Urban sound classification - part 2: sample rate conversion, librosa," .

[5] Mike Smales, "Sound classification using deep learning," .