Learning Rank of Evaluated Songs for Classifying Unknown Music by Convolutional Neural Network

First Author B.K.LEE, Hanyang University, Korea

Abstract

This paper shows ranking unevaluated songs, new songs that is not ranked. Various Songs' data and its ranking data is based on an official and popular formal site for music. This paper takes Short-Term-Fourier-Transformation (STFT) with music data for pre-processing. And data with STFT is processed with down sampling for resolution. After its processing, songs data with preprocessed are taken deep learning. While they are considered as input in three layers convolutional neural network (CNN), its ranking data is output data for target. Reducing overfitting is so hard that when having a time comparing many overfitting technics, this paper choose batch-normalization(BN), excluding dropout. It helps overfitting for accuracy in test set batch. In result, this research gets higher accuracy , which is 0.7, than ever.

I. INTRODUCTION

When new songs are coming, Can we predict what ranks the songs have? or Can we evaluate songs whether these are hitting to public? This is why this paper is researching music processing and deep learning structure. Therefore it needs algorithm with more complex deep layer structure for CNN, using Tensorflow, the most famous library for deep learning. Then it learns rankings of various songs by CNN. It is critical role to try to make an fancy model for evaluating song's ranks and test whether it shows good precision.

Based on a formal site specifically in music, music ranking is properly classified with four groups that $1^{st} \sim 19^{th}$ is first group, $20^{th} \sim 49^{th}$ is second group, $50^{th} \sim 79^{th}$ is third group, and $80^{th} \sim 100^{th}$ is fourth group. The following figures are mathematical expression vector for each group in deep learning output label.

$$\text{First class : } \hat{y}_1=(1,0,0,0)$$
$$\text{Second class : } \hat{y}_2=(0,1,0,0)$$
$$\text{Third class : } \hat{y}_3=(0,0,1,0)$$
$$\text{Fourth class : } \hat{y}_4=(0,0,0,1)$$

This is because when doing calculation in deep layer, it can much lesson time in terms of computer memory than each class vector.

Music data is processed with STFT. While Fast Fourier Transformation (FFT) gives frequency histogram information in total song interval, STFT can show it in small time interval.

$$STFT\left(x[n]\right)=X(m,w)=\sum_{n=-\infty}^{\infty} x[n]\,w[n-m]\,e^{-jwn}$$

$x[n]$: music signals
$w[n]$: window
m : window shifting value
w : frequency

And each two dimensional data processed with STFT should be resized for resolution. If its procedure is not done, it is hard to control overfitting.

To reduce it, this paper takes BN method. Histogram of estimated values in deep learning gives mean, variance, skewness, kurtosis and so on. Although training set can be fit each other by learning itself, histogram of test set values are naturally different shape with that of training set, because of data only interested in high accuracy. Then getting standardized data in each deep layer is better idea so that error between histogram of training set and that of test set is more alleviated. And this is reason BN is needed not only in this paper, but also in others. It is introduction for mathematical expression of BN, as below.

$$\hat{x}^{(i)}=\frac{x^{(i)}-E\left(x^{(i)}\right)}{\sqrt{\sigma^2\left(x^{(i)}\right)+\epsilon}}$$

$$\widehat{X}^{(i)}=\gamma^{(i)}\hat{x}^{(i)}+\beta^{(i)}$$

$x^{(i)}$: i th minibatch containing $x_1^{(i)}, x_2^{(i)}, \cdots$ element
$\hat{x}^{(i)}$: i th standardized minibatch set
$\widehat{X}^{(i)}$: new input of activation function in a layer with BN
$\gamma^{(i)}$: scaling factor with respect to i th minibatch
$\beta^{(i)}$: shifting factor with respect to i th minibatch

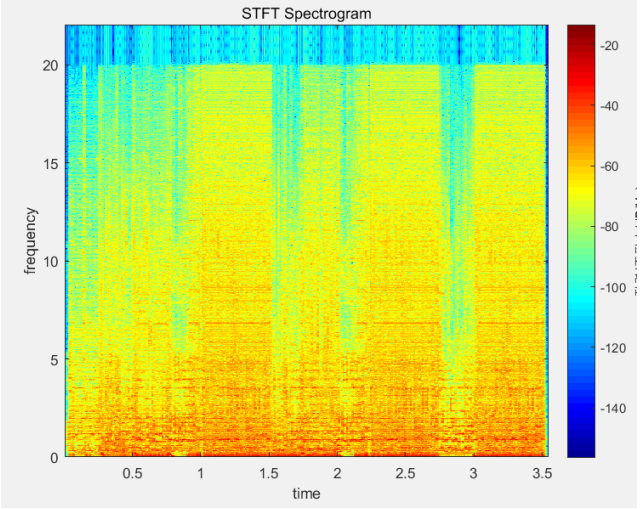
While controlling $\gamma^{(i)}$, $\beta^{(i)}$ in each feature map, by doing back-propagation for lessoning cross entropy with softmax, it is point to finding optimal values where those of cost are going to be small.

It is advantage of BN that it is more efficient than dropout in terms of calculation. And it makes designer ignore to control learning rate. Furthermore bias constant in each layer is not more needed at least with BN applied layer.

After resizing resolution of data by bilinear interpolation, it is put as input layer for CNN as stochastic minibatch in total batch shuffled. This deep model consists of 3 convolution layers and fully connected 4 layers. Adam-optimizer is chosen that is gradient method updating weights. scaling factor and shifting factor. In hidden layer, activation function is ReLU and Classification function in output layer is Softmax which are all commonly used.

II. Preprocessing For STFT

The number of music data set is 600 which is made up of 6 months songs set where each month songs set consists of 100 numbers songs that are ranked orderly. Following figure is a STFT-ed song data colormap which shows impact frequency by color in each sample time. X-axis means time and Y-axis means frequency. Then when color converges red, it means that an area where it is red has a high probability to have impact frequency color.

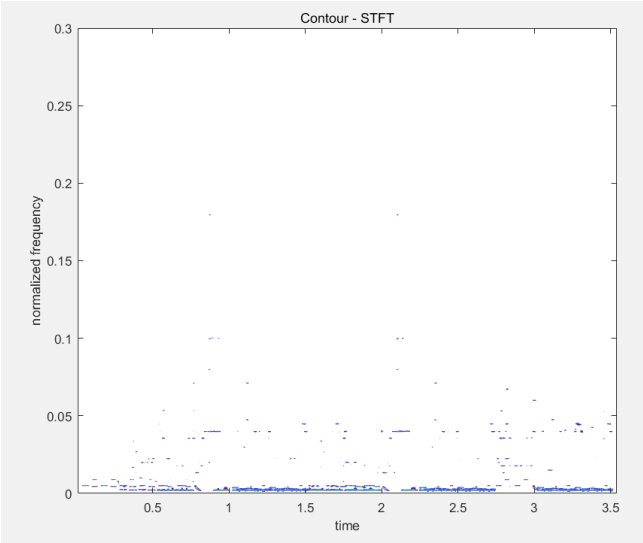


[Matlab code]
- spectrogram(mono-music, ones(1,88200), 44100, [], 'yaxis')

Each music has sample frequency 44100 Hz. Then this paper chooses ideal window whose length is 88200, which means analyze 2 seconds interval. But window shifting value is 44100. That is, overlapping interval is 1second, 44100 sample. If it is more larger, time – dependent accuracy will increase. But, vice versa, frequency – dependent accuracy or resolution will decrease. So it is hard to increase window shifting value for having accurate information of frequency.

After STFT, we normalize and standardize many STFT to resize 100 by 100for same pixed. This is because it it procedure for input size of deep learning to be same.

Then it is crucial to be understanding what resizing means. If it is not handled about input size problem, it should be overfit because deep model has to control rapidly changing values as input size is large, which means it becomes so sensitive model.



[Matlab code]
-[s,w,t]=spectrogram(mono-music, ones(1,88200), 44100, [], 'yaxis')
-contour(t, w, abs(s))

As above, it is contour STFT graph which shows where the peak's locations are in original resolution. But as you can see, this data looks no pattern and no arranged. Therefore before it is considered as deep learning input, it is important to use other technical method to be less sensitive to easily analyze. The key to this problem is just image smoothing by bilinear interpolation.

When image are needed to be resized, there is many method neighborhood, bilinear, bi cubic interpolation. And they are commonly used by backward mapping. Because if it is not, there can be many holes and cracks in the images. It is math expression as below.

<Notation>
T : Linear Transformation matrix
(x, y) : original image's coordinate expression
(x', y') : coordinate expression of it being transformed.
$f(x, y)$: image intensity value of (x, y) coordinate

Then it is satisfied that if what we want is resizing, then T is just this formation

$$T=\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

Because

$$x'=ax$$
$$y'=by$$

Then it can be expressed

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = T \begin{pmatrix} x \\ y \end{pmatrix}$$

So because of focusing (x', y') , it can be expressed to be estimating what original coordinate is.

$$T^{-1} \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

Then

$$\frac{1}{a} x' = x$$
$$\frac{1}{b} y' = y$$

In this paper, resized resolution is 100 by 100. So a equals 100 and b equals 100 too. And possible value of each x' , y' is zero to 99 which means pixels 100 by 100. By this expression, it can be estimated what original image 's coordinate is, which fills up with cracks and holes in resized image.

$$\frac{1}{100} x' = x$$
$$\frac{1}{100} y' = y$$

However, if original coordinate (x, y) is not integer, it is wondered how it can control. It is bilinear interpolation.

$$T^{-1} \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x_{est} \\ y_{est} \end{pmatrix}$$

$$\frac{1}{100} x' = x_{est}$$
$$\frac{1}{100} y' = y_{est}$$

And estimate how (x_{est}, y_{est}) represents integer values.

Let

$$x_0 = [x_{est}] \qquad x_1 = [x_{est}] + 1$$
$$y_0 = [y_{est}] \qquad y_1 = [y_{est}] + 1$$

$$f(x_{est}, y_{est}) = \frac{(x - x_0)(y - y_0)}{(x_1 - x_0)(y_1 - y_0)} \times f(x_{1,} y_1) + \frac{(x - x_0)(y_1 - y)}{(x_1 - x_0)(y_1 - y_0)} \times f(x_{1,} y_0) + \frac{(x_1 - x)(y - y_0)}{(x_1 - x_0)(y_1 - y_0)} \times f(x_{0,} y_1) + \frac{(x_1 - x)(y_1 - y)}{(x_1 - x_0)(y_1 - y_0)} \times f(x_{0,} y_0)$$

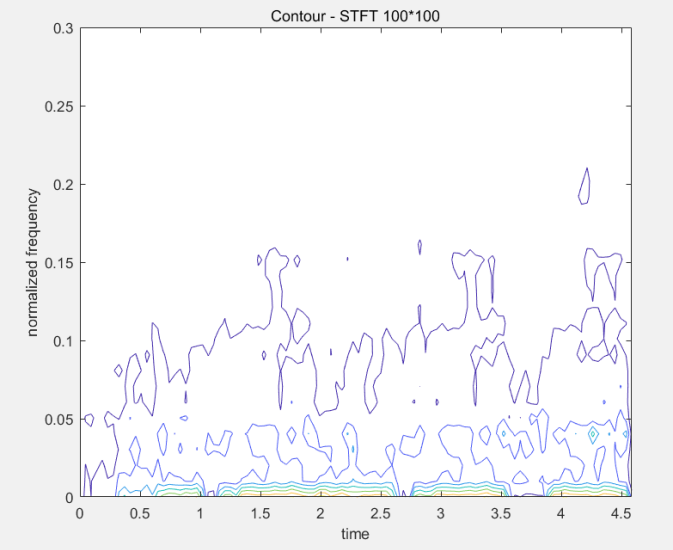
Because of

$$x_1 - x_0 = 1$$
$$y_1 - y_0 = 1$$

Then

$$f(x_{est}, y_{est}) = (x - x_0)(y - y_0) f(x_{1,} y_1) + (x - x_0)(y_1 - y) f(x_{1,} y_0) + (x_1 - x)(y - y_0) f(x_{0,} y_1) + (x_1 - x)(y_1 - y) f(x_{0,} y_0)$$

By this bilinear interpolation method, original STFT image can be resized as below graph.



[Matlab code]
-imresize(STFT image, [100 100], 'bilinear')

And it is figure that expresses impact frequency by Contour function in Matlab. Besides, resized STFT data compresses information of frequency. Then it is more easy to check what histogram shape is. However It is not accurate than before. But despite of it, small accuracy has more potential than so accurate output value in terms of reducing overfitting. Because in deep layer, there are many procedures for increasing accuracy. But data which has been already fitted means that conversely, it has no more potential of increasing accuracy.

III. Deep learning Code - CNN

This paper constructs deep model has CNN and fully connected neural network(FCNN) layer. CNN includes one input layer and 2 hidden layer. Then second hidden layer is also considered as FCNN input layer. And hidden network of FCNN is 2 layers and output classification layer is one.

[Defining function to learn data, feed forward propagation]

```
# Create model
def conv_net(x, weights, biases):
    # Reshape input picture
    x = tf.reshape(x, shape=[-1, 100, 100, 1])

    # Convolution Layer with BN
    conv1 = conv2d_batch(x, weights['wc1'], biases['bc1'],
            weights['offset1'], weights['scale1'], strides=3)

    # Max Pooling (down-sampling)
    conv1 = maxpool2d(conv1, k=2)

    # Convolution Layer2 with BN
    conv2 = conv2d_batch(conv1, weights['wc2'], biases['bc2'],
                weights['offset2'], weights['scale2'], strides=1)

    # Max Pooling (down-sampling)
    conv2 = maxpool2d(conv2, k=2)

    # Fully connected layer
    # Reshape conv2 output
    # to fit fully connected layer input
    fc1 = tf.reshape(conv2,
                    [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1=batch_perceptron(fc1, weights['wd1'], biases['bd1'],
                                        weights['offset3'],
weights['scale3'],)

    # Fully connected layer
    fc2 = batch_perceptron(fc1, weights['wd2'], biases['bd2'],

                                weights['offset4'], weights['scale4'],)
    # Output, class prediction
    out = tf.add(tf.matmul(fc2, weights['out']), biases['out'])
    return out
```

[Defining function how to calculate CNN FCNN]

```
# Create some wrappers for simplicity
def conv2d_batch(x, W, offset, scale, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1],
                    padding='SAME')
    mean, var=tf.nn.moments(x, axes=[0,1,2])
    x = tf.nn.batch_normalization(x, mean=mean,
                    variance=var, offset=offset, scale=scale,
                                variance_epsilon = 0.001)
    return tf.nn.relu(x)

def maxpool2d(x, k=2):
    # MaxPool2D wrapper
    return tf.nn.max_pool(x, ksize=[1, k, k, 1],
                    strides=[1, k, k, 1], padding='SAME')
def batch_perceptron(x, weights, offset=1, scale=0):

    # Hidden layer with RELU activation
    x = tf.matmul(x, weights)
    mean, var = tf.nn.moments(x, axes=[0])
    x = tf.nn.batch_normalization(x, mean=mean,
                    variance=var, offset=offset, scale=scale,
                    variance_epsilon=0.001)
    out = tf.nn.relu(x)

    return out
```

And CNN layer channel number and FCNN layer node number are all different. Its information is as below algorithm code.

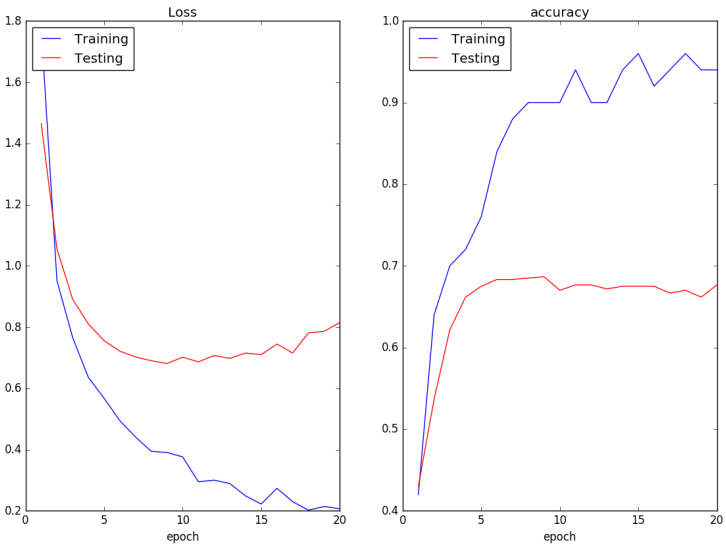
```
#first hidden layer channel number
n_wc1=15
#seocnd hidden layer channel number
n_wc2=30
#first hidden layer node number
n_wd1=100
#first hidden layer node number
n_wd2=25
#output classification number
n_classes=4

# Store layers weight & bias
weights = {
    # 7x7 conv, 1 input,  n_wc1 outputs
    'wc1': tf.Variable(tf.random_normal([7, 7, 1, n_wc1])),
    # 5x5 conv, n_wc1 inputs, n_wc2 outputs
    'wc2': tf.Variable(tf.random_normal([5, 5, n_wc1, n_wc2])),

    # fully connected, 9*9* n_wc2 inputs, n_wd1 outputs
    'wd1': tf.Variable(tf.random_normal([9*9*n_wc2, n_wd1])),
    # fully connected, n_wd1 inputs, n_wd2 outputs
    'wd2': tf.Variable(tf.random_normal([n_wd1, n_wd2])),
    # n_wd1 inputs, n_classes outputs (class prediction)
    'out': tf.Variable(tf.random_normal([n_wd2, n_classes])),

    'offset1' : tf.Variable(tf.zeros([n_wc1])),
    'scale1' : tf.Variable(tf.ones([n_wc1])),
    'offset2': tf.Variable(tf.zeros([n_wc2])),
    'scale2': tf.Variable(tf.ones([n_wc2])),
    'offset3': tf.Variable(tf.zeros([n_wd1])),
    'scale3': tf.Variable(tf.ones([n_wd1])),
    'offset4': tf.Variable(tf.zeros([n_wd2])),
    'scale4': tf.Variable(tf.ones([n_wd2]))
}

biases = {
    'bc1': tf.Variable(tf.random_normal([n_wc1])),
    'bc2': tf.Variable(tf.random_normal([n_wc2])),
    'bd1': tf.Variable(tf.random_normal([n_wd1])),
    'bd2': tf.Variable(tf.random_normal([n_wd2])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}
```

It is hyper-parameter and Network parameter ; input's node and output's node number(class number, n_class) and memory placeholder of input and output. Additionally, Constructing code with cross entropy, gradient decent method (Adam), checking and evaluating accuracy model.

```
#Hyper-Parameters
learning_rate = 0.01
training_epochs = 20
batch_size = 50 #minibatch size

# Network Parameters
n_input = 100*100 # MNIST data input (img shape: 100*100)
n_classes = 4 # MNIST total classes (0-9 digits)

# tf Graph input
x = tf.placeholder(tf.float32, [None, n_input])
y = tf.placeholder(tf.float32, [None, n_classes])

# Construct model
pred = conv_net(x, weights, biases)

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y))

optimizer =
tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# Evaluate model
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

IV. Conclusion

After learning and optimizing 600 training set, test accuracy should be calculated. Test batch of this paper has one hundred number of music songs. As below figure, It is graph comparing loss and accuracy between training set and testing set.



Test set is already overfit. Its reasons are so various that this paper can not consider and control them. However accuracy of training set becomes higher than 0.9, and that of testing set higher than 0.65. Then its possibility of reducing overfitting increase. Therefore if this paper has larger number of songs about one hundred thousands training sets and one hundred testing set, overfitting of this deep model will be more reduced then ever before model.

V. References

https://www.tensorflow.org/api_docs/ : tensorflow opensource for google

https://www.tensorflow.org/get_started/mnist/pros : Mnist for experts , tutorial sources

https://arxiv.org/abs/1502.03167 : batch normalization paper

https://en.wikipedia.org/wiki/Short-time_Fourier_transform : STFT definition

https://en.wikipedia.org/wiki/Bilinear_interpolation : bilinear interpolation definition

https://en.wikipedia.org/wiki/Backward_design : backward designing method