CNN_A2

April 26, 2017

1 CS109B Project Group 26 - Deep Learning

## Main Specifications - Ver1

** Data preprocessing:**

- Channel rearrangement
- 14 labels multi label classification
- 20% validation
- Centered features

** Main Architecture **

- Multi layer CNN :

- Conv2D - Relu depth 32 and 7x7 kernel - He uniform initialization - MaxPool 2x2 - Conv2D - Relu depth 32 and 5x5 kernel - He uniform initialization - MaxPool 2x2 - Conv2D - Relu depth 16 and 3x3 kernel - He uniform initialization - MaxPool 2x2 - Conv2D - Relu depth 16 and 3x3 kernel - He uniform initialization - FC 128 , Relu , He uniform initialization - FC 64 , Relu , He uniform initialization

- SGD optimzer with 1e-6 learning rate and 0.99 momentum - Binary cross entropy loss function - Batch size 64 - Training convergence after 50 epochs

1.0.1 Import Modules

```
In [1]: import requests
        import json
        import time
        import itertools
        import wget
        import os
        import pickle
        import numpy as np

        import random
        import matplotlib
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sns
        from sklearn.cluster.bicluster import SpectralCoclustering
        from sklearn.metrics import precision_recall_curve
        import scipy

        sns.set_style('white')
        import tensorflow as tf
        import pandas as pd
        import keras
        from keras.optimizers import SGD
        from keras.models import Sequential
        from keras.layers import Dense,Dropout,Flatten
        from keras.layers import Conv2D, MaxPooling2D
        import keras.initializers as init
        from keras import backend as K
        from keras.models import load_model
```

```
Using TensorFlow backend.
```

1.0.2 Import the Preprocessed Data from the customized AWS Image EBS Storage

```
In [2]: x_train_dict = pickle.load(open('training_num.pik' , 'rb'))
```

```
In [3]: # Extract the list of input tensors
        x_train_raw = x_train_dict['images'][:9988]

        # Transform them into numpy tensor
        x_train = np.array(x_train_raw)

        In shape
        print x_train.shape
```

```
(9988, 128, 85, 3)
```

### Preprocess the training data

```
In [4]: # Extract the image rows
        img_rows = x_train.shape[1]

        # Extract the image columns
        img_cols = x_train.shape[2]

        # Re-arrange according to the configured order of channels
        # If Channels first
        if K.image_data_format() == 'channels_first':
            x_train = x_train.reshape(x_train.shape[0], 3, img_rows, img_cols)
            input_shape = (3, img_rows, img_cols)
        # If channels last
        else:
            x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)
            input_shape = (img_rows, img_cols, 3)
```

Normalize the data

```
In [5]: # transform into float
        x_train = x_train.astype('float32')

        # normalize
        x_train /= 255

        print 'x_train shape:', x_train.shape
        print  x_train.shape[0], 'train samples'
```

```
x_train shape: (9988, 128, 85, 3)
9988 train samples
```

Pre process the label

```
In [6]: # Read From File
        y_raw = pd.read_csv('Genres_labels_All_cleaned.csv')

        # Select Labels from file
        y_train = y_raw.iloc[:, 1:-1].values

        # Define the shape of the labels
        num_classes = y_train.shape[1]
```

1.0.3 Main Architecture Build up

```
In [12]: # create an empty network model
         model1 = Sequential()

         # --- input layer ---
         model1.add(Conv2D(16, kernel_size=(7, 7), activation='relu', input_shape=input_shape
                          kernel_initializer = init.he_normal(109)))

         # --- max pool ---
         model1.add(MaxPooling2D(pool_size=(2, 2)))


         # ---- Conv Layer ---
         model1.add(Conv2D(16, kernel_size=(5, 5), activation='relu',
                          kernel_initializer = init.he_normal(109)))
```

```
# --- max pool ---
model1.add(MaxPooling2D(pool_size=(2, 2)))

# --- Conv layer ---
model1.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                kernel_initializer = init.he_normal(109)))

# --- max pool ---
model1.add(MaxPooling2D(pool_size=(2, 2)))

# --- Conv layer ---
model1.add(Conv2D(32, kernel_size=(3, 3), activation='relu' ,
                kernel_initializer = init.he_normal(109)))

# --- max pool ---
model1.add(MaxPooling2D(pool_size=(2, 2)))


# flatten for fully connected classification layer
model1.add(Flatten())

# --- fully connected layer ---
model1.add(Dense(128, activation='relu',
               kernel_initializer = init.he_normal(109)))

# --- fully connected layer ---
model1.add(Dense(64, activation='relu' ,
               kernel_initializer = init.he_normal(109)))

# --- classification ---
model1.add(Dense(num_classes, activation='sigmoid'))

# prints out a summary of the model architecture
model1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 122, 79, 16)       2368
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 61, 39, 16)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 57, 35, 16)        6416
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 28, 17, 16)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 26, 15, 32)        4640
```