CNN_3-BNDO

April 26, 2017

1 CS109B Project Group 26 - Deep Learning

Main Specifications - Ver5
** Data preprocessing:**

- Channel rearrangement
- 14 labels multi label classification
- data augmentation by shift and zoom for 8000 training samples and 2000 test samples
- Centered features

** Main Architecture **

- Multi layer CNN with Batch Normalization Layers and Dropout :

- Conv2D - Relu depth 32 and 7x7 kernel - He uniform initialization - 30 % Dropout - Batch Norm - MaxPool 2x2 - Conv2D - Relu depth 32 and 5x5 kernel - He uniform initialization - 30 % Dropout - Batch Norm - MaxPool 2x2 - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization - 30 % Dropout - Batch Norm - MaxPool 2x2 - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization - 30 % Dropout - Batch Norm - FC 128 , Relu , He uniform initialization - 50 % Dropout - Batch Norm - FC 64 , Relu , He uniform initialization - 50 % Dropout - Batch Norm

- SGD optimzer with 1e-4 learning rate and 0.99 momentum - Binary cross entropy loss function - Batch size 128 - Training convergence after 100 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, Adam
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
import keras.initializers as init
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
from keras.models import load_model
```

```
Using TensorFlow backend.
```

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

In [3]: train_split = 8000

In [4]: x_train_raw = x_train_dict['images']

        x_train = np.array(x_train_raw)[:train_split,: ,: , :]
        x_test  = np.array(x_train_raw)[train_split:,: ,: , :]

        print x_train.shape
```

```
(8000, 128, 85, 3)
```

```
In [5]: img_rows = x_train.shape[1]

        img_cols = x_train.shape[2]

        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)

        else:
            x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)
            input_shape = (img_rows, img_cols, 3)
```

```
In [6]: x_train = x_train.astype('float32')
        x_test  = x_test.astype('float32')

        x_train /= 255.0
        x_test  /= 255.0

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

        print 'x_test shape:', x_test.shape
        print  x_test.shape[0], 'test samples'
```

```
x_train shape: (8000, 128, 85, 3)
8000 train samples
x_test shape: (1988, 128, 85, 3)
1988 test samples
```

```
In [7]: y_raw = pd.read_csv('Genres_labels_All_cleaned.csv')

        y_train = y_raw.iloc[:, 1:-1].values[:train_split, :]
        y_test  = y_raw.iloc[:, 1:-1].values[train_split:, :]

        num_classes = y_train.shape[1]

        print 'number of classes:  ' , num_classes
```

```
number of classes:   14
```

```
In [8]: datagen = ImageDataGenerator(
            featurewise_center=True,
            featurewise_std_normalization=True,
            width_shift_range=0.2,
            height_shift_range=0.2,
            zoom_range = 0.5,
            fill_mode = 'wrap')

        datagen.fit(x_train)

        datagen.fit(x_test)
```

```
In [34]: # create an empty network model
         modelb = Sequential()

         # --- input layer ---
         modelb.add(Conv2D(32, kernel_size=(7, 7), activation='relu', input_shape=input_shape
                          kernel_initializer = init.he_normal(109)))
```

```
# ------Dropout -------
modelb.add(Dropout(0.3))

# -----Batch Normalization -----
modelb.add(BatchNormalization())

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


# ----------------------------------------------------------------------------------

# ---- Conv Layer ---
modelb.add(Conv2D(32, kernel_size=(5, 5), activation='relu',
                kernel_initializer = init.he_normal(109)))

# ------Dropout -------
modelb.add(Dropout(0.3))

# -----Batch Normalization -----
modelb.add(BatchNormalization())

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


# ----------------------------------------------------------------------------------

# --- Conv layer ---
modelb.add(Conv2D(64, kernel_size=(3, 3), activation='relu',
               kernel_initializer = init.he_normal(109)))

# ------Dropout -------
modelb.add(Dropout(0.3))

# -----Batch Normalization -----
modelb.add(BatchNormalization())

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

# ----------------------------------------------------------------------------------

# --- Conv layer ---
modelb.add(Conv2D(64, kernel_size=(3, 3), activation='relu' ,
               kernel_initializer = init.he_normal(109)))

# ------Dropout -------
```

```python
modelb.add(Dropout(0.3))

# -----Batch Normalization -----
modelb.add(BatchNormalization())

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

#-------------------------------------------------------------------------------------

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

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

# ------Dropout -------
modelb.add(Dropout(0.5))

# -----Batch Normalization -----
modelb.add(BatchNormalization())

# ------------------------------------------------------------------------------------

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

# ------Dropout -------
modelb.add(Dropout(0.5))

# -----Batch Normalization -----
modelb.add(BatchNormalization())

# ------------------------------------------------------------------------------------

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 122, 79, 32)       4736
_________________________________________________________________
```

```
dropout_31 (Dropout)         (None, 122, 79, 32)       0
_________________________________________________________________
batch_normalization_31 (Batc (None, 122, 79, 32)       128
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 61, 39, 32)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 57, 35, 32)        25632
_________________________________________________________________
dropout_32 (Dropout)         (None, 57, 35, 32)        0
_________________________________________________________________
batch_normalization_32 (Batc (None, 57, 35, 32)        128
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 28, 17, 32)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 26, 15, 64)        18496
_________________________________________________________________
dropout_33 (Dropout)         (None, 26, 15, 64)        0
_________________________________________________________________
batch_normalization_33 (Batc (None, 26, 15, 64)        256
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 13, 7, 64)         0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 11, 5, 64)         36928
_________________________________________________________________
dropout_34 (Dropout)         (None, 11, 5, 64)         0
_________________________________________________________________
batch_normalization_34 (Batc (None, 11, 5, 64)         256
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 5, 2, 64)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 640)               0
_________________________________________________________________
dense_16 (Dense)             (None, 128)               82048
_________________________________________________________________
dropout_35 (Dropout)         (None, 128)               0
_________________________________________________________________
batch_normalization_35 (Batc (None, 128)               512
_________________________________________________________________
dense_17 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_36 (Dropout)         (None, 64)                0
_________________________________________________________________
batch_normalization_36 (Batc (None, 64)                256
_________________________________________________________________
dense_18 (Dense)             (None, 14)                910
=================================================================
Total params: 178,542.0
Trainable params: 177,774.0
```

CNN_3-BNDOReg

April 26, 2017

1 CS109B Project Group 26 - Deep Learning

Main Specifications - Ver6
** Data preprocessing:**

- Channel rearrangement
- 14 labels multi label classification
- data augmentation by shift and zoom for 8000 training samples and 2000 test samples
- Centered features

** Main Architecture **

- Multi layer CNN with Batch Normalization Layers and Dropout and 0.1 L2 Kernel Regularization :

- Conv2D - Relu depth 32 and 7x7 kernel - He uniform initialization - 0.1 L2 Kernel Regularization - 30 % Dropout - Batch Norm - MaxPool 2x2 - Conv2D - Relu depth 32 and 5x5 kernel - He uniform initialization - 0.1 L2 Kernel Regularization - 30 % Dropout - Batch Norm - MaxPool 2x2 - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization - 0.1 L2 Kernel Regularization - 30 % Dropout - Batch Norm - MaxPool 2x2 - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization - 0.1 L2 Kernel Regularization - 30 % Dropout - Batch Norm - FC 128 , Relu , He uniform initialization - 0.1 L2 Kernel Regularization - 50 % Dropout - Batch Norm - FC 64 , Relu , He uniform initialization - 0.1 L2 Kernel Regularization - 50 % Dropout - Batch Norm

- SGD optimzer with 1e-4 learning rate and 0.99 momentum - Binary cross entropy loss function - Batch size 128 - Training convergence after 100 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 import regularizers as reg
from keras.optimizers import SGD, Adam
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
import keras.initializers as init
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
from keras.models import load_model
```

```
Using TensorFlow backend.
```

Open the Preprocessed Poster Data

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

Specify the training/test split

```
In [3]: train_split = 8000
```

```
In [4]: x_train_raw = x_train_dict['images']

        x_train = np.array(x_train_raw)[:train_split,: ,: , :]
        x_test  = np.array(x_train_raw)[train_split:,: ,: , :]

        print x_train.shape
```

```
(8000, 128, 85, 3)
```

```
In [5]: img_rows = x_train.shape[1]

        img_cols = x_train.shape[2]
```

```
        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)

        else:
            x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)
            input_shape = (img_rows, img_cols, 3)
```

```
In [6]: x_train = x_train.astype('float32')
        x_test  = x_test.astype('float32')

        x_train /= 255.0
        x_test  /= 255.0

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

        print 'x_test shape:', x_test.shape
        print  x_test.shape[0], 'test samples'
```

```
x_train shape: (8000, 128, 85, 3)
8000 train samples
x_test shape: (1988, 128, 85, 3)
1988 test samples
```

```
In [7]: y_raw = pd.read_csv('Genres_labels_All_cleaned.csv')

        y_train = y_raw.iloc[:, 1:-1].values[:train_split, :]
        y_test  = y_raw.iloc[:, 1:-1].values[train_split:, :]

        num_classes = y_train.shape[1]

        print 'number of classes:   ' , num_classes
```

```
number of classes:   14
```

```
In [8]: datagen = ImageDataGenerator(
            featurewise_center=True,
            featurewise_std_normalization=True,
            width_shift_range=0.2,
            height_shift_range=0.2,
            zoom_range = 0.5,
            fill_mode = 'wrap')

        datagen.fit(x_train)

        datagen.fit(x_test)
```

In [9]:
```
# Specify regularization parameter
reg_par = 0.1
```

In [10]:
```
# create an empty network model
modelc = Sequential()

# --- input layer ---
modelc.add(Conv2D(32, kernel_size=(7, 7), activation='relu', input_shape=input_shape
                kernel_initializer = init.he_normal(109),
                kernel_regularizer = reg.l2(reg_par)))

# ------Dropout -------
modelc.add(Dropout(0.3))

# -----Batch Normalization -----
modelc.add(BatchNormalization())

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


# ---------------------------------------------------------------------------------

# ---- Conv Layer ---
modelc.add(Conv2D(32, kernel_size=(5, 5), activation='relu',
                kernel_initializer = init.he_normal(109),
                kernel_regularizer = reg.l2(reg_par)))

# ------Dropout -------
modelc.add(Dropout(0.3))

# -----Batch Normalization -----
modelc.add(BatchNormalization())

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


# ---------------------------------------------------------------------------------

# --- Conv layer ---
modelc.add(Conv2D(64, kernel_size=(3, 3), activation='relu',
                kernel_initializer = init.he_normal(109),
                kernel_regularizer = reg.l2(reg_par)))

# ------Dropout -------
modelc.add(Dropout(0.3))
```

```
# -----Batch Normalization -----
modelc.add(BatchNormalization())

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

# ------------------------------------------------------------------------------------

# --- Conv layer ---
modelc.add(Conv2D(64, kernel_size=(3, 3), activation='relu' ,
                 kernel_initializer = init.he_normal(109),
                 kernel_regularizer = reg.l2(reg_par)))

# ------Dropout -------
modelc.add(Dropout(0.3))

# -----Batch Normalization -----
modelc.add(BatchNormalization())

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

#-------------------------------------------------------------------------------------

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

# --- fully connected layer ---
modelc.add(Dense(128, activation='relu',
                kernel_initializer = init.he_normal(109),
                kernel_regularizer = reg.l2(reg_par)))

# ------Dropout -------
modelc.add(Dropout(0.5))

# -----Batch Normalization -----
modelc.add(BatchNormalization())

# ------------------------------------------------------------------------------------

# --- fully connected layer ---
modelc.add(Dense(64, activation='relu' ,
                kernel_initializer = init.he_normal(109),
                kernel_regularizer = reg.l2(reg_par)))

# ------Dropout -------
modelc.add(Dropout(0.5))
```

```python
# -----Batch Normalization -----
modelc.add(BatchNormalization())

# ---------------------------------------------------------------------------------

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 122, 79, 32)       4736
_________________________________________________________________
dropout_1 (Dropout)          (None, 122, 79, 32)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 122, 79, 32)       128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 61, 39, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 57, 35, 32)        25632
_________________________________________________________________
dropout_2 (Dropout)          (None, 57, 35, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 57, 35, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 17, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 15, 64)        18496
_________________________________________________________________
dropout_3 (Dropout)          (None, 26, 15, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 26, 15, 64)        256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 7, 64)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 5, 64)         36928
_________________________________________________________________
dropout_4 (Dropout)          (None, 11, 5, 64)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 11, 5, 64)         256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 2, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 640)               0
_________________________________________________________________
```

CNN_3

April 26, 2017

1 CS109B Project Group 26 - Deep Learning

Main Specifications - Ver2
** Data preprocessing:**

- Channel rearrangement
- 14 labels multi label classification
- data augmentation by shift and zoom for 8000 training samples and 2000 test samples
- 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 64 and 3x3 kernel - He uniform initialization - MaxPool 2x2 - Conv2D - Relu depth 64 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 30 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, Adam
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
import keras.initializers as init
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
from keras.models import load_model
```

```
Using TensorFlow backend.
```

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

In [3]: train_split = 8000

In [4]: x_train_raw = x_train_dict['images']

        x_train = np.array(x_train_raw)[:train_split,: ,: , :]
        x_test  = np.array(x_train_raw)[train_split:,: ,: , :]

        print x_train.shape
```

```
(8000, 128, 85, 3)
```

```
In [5]: img_rows = x_train.shape[1]

        img_cols = x_train.shape[2]

        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)

        else:
            x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)
            input_shape = (img_rows, img_cols, 3)

In [6]: x_train = x_train.astype('float32')
        x_test  = x_test.astype('float32')
```

```
        x_train /= 255.0
        x_test  /= 255.0

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

        print 'x_test shape:', x_test.shape
        print  x_test.shape[0], 'test samples'
```

```
x_train shape: (8000, 128, 85, 3)
8000 train samples
x_test shape: (1988, 128, 85, 3)
1988 test samples
```

```
In [7]: y_raw = pd.read_csv('Genres_labels_All_cleaned.csv')

        y_train = y_raw.iloc[:, 1:-1].values[:train_split, :]
        y_test  = y_raw.iloc[:, 1:-1].values[train_split:, :]

        num_classes = y_train.shape[1]
```

```
In [30]: datagen = ImageDataGenerator(
             featurewise_center=True,
             featurewise_std_normalization=True,
             width_shift_range=0.2,
             height_shift_range=0.2,
             zoom_range = 0.5,
             fill_mode = 'wrap')

         datagen.fit(x_train)

         datagen.fit(x_test)
```

```
In [31]: # create an empty network model
         model2 = Sequential()

         # --- input layer ---
         model2.add(Conv2D(32, kernel_size=(7, 7), activation='relu', input_shape=input_shape
                          kernel_initializer = init.he_normal(109)))

         # --- max pool ---
         model2.add(MaxPooling2D(pool_size=(2, 2)))


         # ---- Conv Layer ---
         model2.add(Conv2D(32, kernel_size=(5, 5), activation='relu',
                         kernel_initializer = init.he_normal(109)))
```

```
    # --- max pool ---
    model2.add(MaxPooling2D(pool_size=(2, 2)))

    # --- Conv layer ---
    model2.add(Conv2D(64, kernel_size=(3, 3), activation='relu',
                      kernel_initializer = init.he_normal(109)))

    # --- max pool ---
    model2.add(MaxPooling2D(pool_size=(2, 2)))

    # --- Conv layer ---
    model2.add(Conv2D(64, kernel_size=(3, 3), activation='relu' ,
                      kernel_initializer = init.he_normal(109)))

    # --- max pool ---
    model2.add(MaxPooling2D(pool_size=(2, 2)))


    # flatten for fully connected classification layer
    model2.add(Flatten())

    # --- fully connected layer ---
    model2.add(Dense(128, activation='relu',
                     kernel_initializer = init.he_normal(109)))

    # --- fully connected layer ---
    model2.add(Dense(64, activation='relu' ,
                      kernel_initializer = init.he_normal(109)))

    # --- classification ---
    model2.add(Dense(num_classes, activation='sigmoid'))

    # prints out a summary of the model architecture
    model2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 122, 79, 32)       4736
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 61, 39, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 57, 35, 32)        25632
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 28, 17, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 26, 15, 64)        18496
```

CNN_4

April 26, 2017

1 CS109B Project Group 26 - Deep Learning

Main Specifications - Ver3
** Data preprocessing:**

- Channel rearrangement
- 14 labels multi label classification
- Data augmentation by shift and zoom for 8000 training sample and 2000 test sample
- 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 64 and 3x3 kernel - He uniform initialization - MaxPool 2x2 - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization - FC 128 , Relu , He uniform initialization - FC 64 , Relu , He uniform initialization

- Adam optimzer with 1e-6 learning rate - Binary cross entropy loss function - Batch size 64 - Training convergence after 30 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, Adam
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
import keras.initializers as init
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
from keras.models import load_model
```

```
Using TensorFlow backend.
```

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

In [3]: train_split = 8000

In [4]: x_train_raw = x_train_dict['images']

        x_train = np.array(x_train_raw)[:train_split,: ,: , :]
        x_test  = np.array(x_train_raw)[train_split:,: ,: , :]

        print x_train.shape
```

```
(8000, 128, 85, 3)
```

```
In [5]: img_rows = x_train.shape[1]

        img_cols = x_train.shape[2]

        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)

        else:
            x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)
            input_shape = (img_rows, img_cols, 3)

In [6]: x_train = x_train.astype('float32')
        x_test  = x_test.astype('float32')
```

```
        x_train /= 255.0
        x_test  /= 255.0

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

        print 'x_test shape:', x_test.shape
        print  x_test.shape[0], 'test samples'
```

```
x_train shape: (8000, 128, 85, 3)
8000 train samples
x_test shape: (1988, 128, 85, 3)
1988 test samples
```

```
In [7]: y_raw = pd.read_csv('Genres_labels_All_cleaned.csv')

        y_train = y_raw.iloc[:, 1:-1].values[:train_split, :]
        y_test  = y_raw.iloc[:, 1:-1].values[train_split:, :]

        num_classes = y_train.shape[1]
```

```
In [8]: datagen = ImageDataGenerator(
            featurewise_center=True,
            featurewise_std_normalization=True,
            width_shift_range=0.2,
            height_shift_range=0.2,
            zoom_range = 0.5,
            fill_mode = 'wrap')

        datagen.fit(x_train)

        datagen.fit(x_test)
```

```
In [10]: # create an empty network model
         model3 = Sequential()

         # --- input layer ---
         model3.add(Conv2D(32, kernel_size=(7, 7), activation='relu', input_shape=input_shape
                          kernel_initializer = init.he_normal(109)))

         # --- max pool ---
         model3.add(MaxPooling2D(pool_size=(2, 2)))


         # ---- Conv Layer ---
         model3.add(Conv2D(32, kernel_size=(5, 5), activation='relu',
                         kernel_initializer = init.he_normal(109)))
```

```python
# --- max pool ---
model3.add(MaxPooling2D(pool_size=(2, 2)))

# --- Conv layer ---
model3.add(Conv2D(64, kernel_size=(3, 3), activation='relu',
               kernel_initializer = init.he_normal(109)))

# --- max pool ---
model3.add(MaxPooling2D(pool_size=(2, 2)))

# --- Conv layer ---
model3.add(Conv2D(64, kernel_size=(3, 3), activation='relu' ,
               kernel_initializer = init.he_normal(109)))

# --- max pool ---
model3.add(MaxPooling2D(pool_size=(2, 2)))


# flatten for fully connected classification layer
model3.add(Flatten())

# --- fully connected layer ---
model3.add(Dense(128, activation='relu',
              kernel_initializer = init.he_normal(109)))

# --- fully connected layer ---
model3.add(Dense(64, activation='relu' ,
               kernel_initializer = init.he_normal(109)))

# --- classification ---
model3.add(Dense(num_classes, activation='sigmoid'))

# prints out a summary of the model architecture
model3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 122, 79, 32)       4736
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 61, 39, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 57, 35, 32)        25632
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 28, 17, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 26, 15, 64)        18496
```

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)))
```

```python
# --- 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
```

CNN_BN

April 26, 2017

1 CS109B Project Group 26 - Deep Learning

Main Specifications - Ver4
** Data preprocessing:**

- Channel rearrangement
- 14 labels multi label classification
- data augmentation by shift and zoom for 8000 training samples and 2000 test samples
- Centered features

** Main Architecture **

- Multi layer CNN with Batch Normalization Layers :

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

- SGD optimzer with 1e-4 learning rate and 0.99 momentum - Binary cross entropy loss function - Batch size 128 - 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, Adam
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
import keras.initializers as init
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
from keras.models import load_model
```

```
Using TensorFlow backend.
```

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

In [3]: train_split = 8000

In [4]: x_train_raw = x_train_dict['images']

        x_train = np.array(x_train_raw)[:train_split,: ,: , :]
        x_test  = np.array(x_train_raw)[train_split:,: ,: , :]

        print x_train.shape
```

```
(8000, 128, 85, 3)
```

```
In [5]: img_rows = x_train.shape[1]

        img_cols = x_train.shape[2]

        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)

        else:
            x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)
            input_shape = (img_rows, img_cols, 3)
```

```
In [6]: x_train = x_train.astype('float32')
        x_test  = x_test.astype('float32')

        x_train /= 255.0
        x_test  /= 255.0

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

        print 'x_test shape:', x_test.shape
        print  x_test.shape[0], 'test samples'
```

```
x_train shape: (8000, 128, 85, 3)
8000 train samples
x_test shape: (1988, 128, 85, 3)
1988 test samples
```

```
In [59]: y_raw = pd.read_csv('Genres_labels_All_cleaned.csv')

         y_train = y_raw.iloc[:, 1:-1].values[:train_split, :]
         y_test  = y_raw.iloc[:, 1:-1].values[train_split:, :]

         num_classes = y_train.shape[1]

         print 'number of classes:  ' , num_classes
```

```
number of classes:   14
```

```
In [60]: datagen = ImageDataGenerator(
             featurewise_center=True,
             featurewise_std_normalization=True,
             width_shift_range=0.2,
             height_shift_range=0.2,
             zoom_range = 0.5,
             fill_mode = 'wrap')

         datagen.fit(x_train)

         datagen.fit(x_test)
```

```
In [74]: # create an empty network model
         modela = Sequential()

         # --- input layer ---
         modela.add(Conv2D(32, kernel_size=(7, 7), activation='relu', input_shape=input_shape
                           kernel_initializer = init.he_normal(109)))
```

```
# -----Batch Normalization -----
modela.add(BatchNormalization())

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

# ---------------------------------------------------------------------------------

# ---- Conv Layer ---
modela.add(Conv2D(32, kernel_size=(5, 5), activation='relu',
                kernel_initializer = init.he_normal(109)))

# -----Batch Normalization -----
modela.add(BatchNormalization())

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

# ---------------------------------------------------------------------------------

# --- Conv layer ---
modela.add(Conv2D(64, kernel_size=(3, 3), activation='relu',
               kernel_initializer = init.he_normal(109)))

# -----Batch Normalization -----
modela.add(BatchNormalization())

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

# ---------------------------------------------------------------------------------

# --- Conv layer ---
modela.add(Conv2D(64, kernel_size=(3, 3), activation='relu' ,
               kernel_initializer = init.he_normal(109)))

# -----Batch Normalization -----
modela.add(BatchNormalization())

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

#----------------------------------------------------------------------------------

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

# --- fully connected layer ---
```

```python
modela.add(Dense(128, activation='relu',
              kernel_initializer = init.he_normal(109)))

# -----Batch Normalization -----
modela.add(BatchNormalization())

# ----------------------------------------------------------------------------------

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

# -----Batch Normalization -----
modela.add(BatchNormalization())

# ----------------------------------------------------------------------------------

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_48 (Conv2D)           (None, 122, 79, 32)       4736
_________________________________________________________________
batch_normalization_63 (Batc (None, 122, 79, 32)       128
_________________________________________________________________
max_pooling2d_48 (MaxPooling (None, 61, 39, 32)        0
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 57, 35, 32)        25632
_________________________________________________________________
batch_normalization_64 (Batc (None, 57, 35, 32)        128
_________________________________________________________________
max_pooling2d_49 (MaxPooling (None, 28, 17, 32)        0
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 26, 15, 64)        18496
_________________________________________________________________
batch_normalization_65 (Batc (None, 26, 15, 64)        256
_________________________________________________________________
max_pooling2d_50 (MaxPooling (None, 13, 7, 64)         0
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 11, 5, 64)         36928
_________________________________________________________________
batch_normalization_66 (Batc (None, 11, 5, 64)         256
_________________________________________________________________
```

In [89]:
```
from __future__ import print_function

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD

import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')

import numpy as np
```

First we load in the data. There are two unique datasets in this analysis because one includes only the RGB color data and the other utilizes all of the quantitative metadata we scraped.

In [98]:
```
# getting RGB data for MLP

x_train = np.genfromtxt('train1_RGB.csv', delimiter=',', skip_header =
1)
y_train = np.genfromtxt('train1_y.csv', delimiter=',', skip_header = 1)

x_train_full = np.genfromtxt('train1_x.csv', delimiter=',', skip_header
= 1)
# print(x_train_full)

x_test = np.genfromtxt('test1_RGB.csv', delimiter=',', skip_header = 1)
y_test = np.genfromtxt('test1_y.csv', delimiter=',', skip_header = 1)

x_test2 = np.genfromtxt('test2_x.csv', delimiter=',', skip_header = 1)
y_test2 = np.genfromtxt('test2_y.csv', delimiter=',', skip_header = 1)
```

In **model** we only utilize the color data (RGB means and sd for posters). Here, we utilized the Sequential model in Keras in order to create an MLP model for multi-level softmax classification. Instead of using the "relu" activation parameter, we chose to utilize the "sigmoid" activation because it was better across the board. We experimented with the learning rate by hand to examine the process along the epochs. Experimenting with (1e-5,1e-4,1e-3,.01,.1) we concluded that .01 was the best learning rate. We decided on the .01 rate by looking at the training data even though when evaluated with the test data the total accuracy was roughly the same. We also experimented with some momentum values (.5 - .99), but .9 seemed to be the best. They were often rougly the same, but other values seemed to have more volatile output accuracies. So, .9 seemed to be a pretty typical value that was reliable.

Overall, the accuracy for the color vectors is not very good. On the test set it comes out to about 20% accuracy. However, this is reasonable because our color analysis is extremely elementary. We scraped the mean value for each of the three RGB vectors and also recorded its standard deviation. You can imgaine many scenarios where this kind of analysis fails to pick up differences between genres or the character of the movie, but it may have some predictive power when added to a larger deep learning model for the posters themselves.

In [91]:

```
model = Sequential()


# try relu or sigmoid
model.add(Dense(64, activation = 'sigmoid', input_dim = 6))
model.add(Dropout(.5))
model.add(Dense(64, activation = 'sigmoid'))
model.add(Dropout(.5))
model.add(Dense(18, activation = 'softmax'))
```

In [131]:

```
learn = .01
decay_rate = 1e-6
mom = .9
sgd = SGD(lr = learn, decay = decay_rate, momentum = mom , nesterov = True)
model.compile(loss = 'categorical_crossentropy', optimizer = sgd, metrics = ['accuracy'])
```

In [ ]:
```
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, Adam
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
import keras.initializers as init
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
from keras.models import load_model
```

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

In [ ]:
```
x_train_raw = x_train_dict['images']

x_train = np.array(x_train_raw)

print x_train.shape
```

In [ ]:
```
img_rows = x_train.shape[1]

img_cols = x_train.shape[2]

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)

else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)
    input_shape = (img_rows, img_cols, 3)
```

In [ ]:
```
x_train = x_train.astype('float32')

x_train /= 255

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

In [ ]:
```
y_raw = pd.read_csv('Genres_labels_All_cleaned.csv')

y_train = y_raw.iloc[:, 1:-1].values

num_classes = y_train.shape[1]
```

In [ ]:
```
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range = 0.5,
    fill_mode = 'wrap')

datagen.fit(x_train)
```

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

model.add(Dense(64, activation='relu', input_shape=input_shape))
# this is our hidden layer

model.add(Dense(64, activation='relu'))
# and an output layer

model.add(Dense(8, activation='softmax'))

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

In [ ]:
```
ada = Adam(lr=0.01)
model.compile(loss='binary_crossentropy',
              optimizer=ada,
              metrics=['accuracy'])
```

In [ ]:
```
batch_size = 64
epochs = 20
```

In [ ]:
```
history = model.fit_generator(datagen.flow(x_train, y_train,
batch_size=batch_size),
                              steps_per_epoch=len(x_train) / batch_size,

                              epochs=epochs)
```

In [ ]:
```
plt.plot(history.history['acc'])
plt.xlabel("epoch")
plt.ylabel("accuracy")
```

In [ ]:
```
import h5py as h5py
```

In [ ]:
```
model.save('mlp_var1.h5')
```

In [ ]:
```
Acc_mlp_var1 = pd.DataFrame(history.history['acc'] , columns = ['Accurac
y'])
```

In [ ]:
```
Acc_mlp_var1.to_csv('mlp_v1.csv')
```