Milestone 4: Deep learning, due Wednesday, April 26, 2017

Section 1: Complete description of the deep network you trained from scratch, including parameter settings, performance, features learned, etc.

Before getting into our model, let's talk about the data we train on first. We have two sets of data, one 10000 32*32 poster data we run locally for testing, and another 20000 128*128 poster data we run on AWS for better modeling.

The deep network we trained from scratch is the first model we finished. It is a sequential deep network model with binary_crossentropy loss function, sgd optimizer, 0.1 learning rate, 0.9 momentum, 20 epochs, 512 batch-size, and 7 layers including a fully connected layer with activation "relu" and a classification layer with activation "sigmoid." We used sigmoid instead of the typical softmax because we are running a multi-label classification model and is not compatible with softmax.

We used 4 different metrics to evaluate the performances of the model. They are accuracy (we called it match ratio in our last milestone), precision, recall and f1 score, which is basically an weighted average of precision and recall. We considered other metrics because accuracy rate tends to be deceivingly high (in our case, with average number of genres per movie = about 2, predicting 0 for all would yield 90% accuracy rate).

After 20 epochs, our from-scratch model yields an accuracy rate of 0.8682, a precision of 0.5277, a recall of 0.1057, and a f1 score of 0.1717. From recall, we know our result is not very ideal despite the high accuracy rate, because we only predict around 10% of the true genres correctly.

Section2: Complete description of the pre-trained network that you fine tuned, including parameter settings, performance, features learned, etc.

The pre-trained network we fined tuned is VGG16 with tensor-flow backend. We kept all layers intact except for the top 3 fully connected layers.
We set the top 3 layers to the following:
m = Dense(4096, activation='relu', name='fc1')(m)
m = Dense(4096, activation='relu', name='fc2')(m)
m = Dense(19, activation='sigmoid', name='predictions')(m)

We implemented a baseline model, model1, and subsequently tuned it in different ways.

1. Baseline model named model2: batch size 512, learning rate fixed at 0.1. Performance: Test loss: 0.3208; Test accuracy: 0.8701; **Precision 0.5205; Recall 0.2555; f1 score 0.3411**
2. Model3: same as model2 except that optimizer is changed to Adam with its default values. Performance: Test loss: 0.32637592492; Test accuracy: 0.86956204137; Precision **0.5124; Recall 0.2711; f1 score 0.3539 (similar to baseline)**
3. Model4: Drop-based Learning Rate Schedule, which means learning rate is dropped by half after every group of epochs. Test loss: 0.3264; Test accuracy: **0.8696; Precision 0.5124; Recall 0.2711; f1 score 0.3540 (similar to baseline)**
4. Model5: Time-based Learning Rate Schedule. We set a time-based learning rate schedule for the sgd optimizer. Test loss: 0.326375924921; **Test accuracy: 0.9090; Precision: 0.6623; Recall: 0.3236; f1 score: 0.4339 (better than baseline)**
5. Model6: gridsearchCV tuning batch size and epochs. We weren't able to finish running the code in time but will use it in the future.

Section3: Discussion of the results, how much improvement you gained with fine tuning, etc.
Results and improvements are covered in the last two sections. Improvements are small but the visualizations show that different methods had very different effect on how performance improves. Due to time constraint, we weren't able to increase the number of epochs or tune more hyperparameters.

Section4: Discussion of at least one additional exploratory idea you pursued
We pursued two ideas:

1. Extracting features learned from the trained deep learning model and run SVMs

    For this approach, we would have to extract features from a layer of the deep learning model into vectors. Each layer has an array of weights and filters that we could vectorize, and we could use the first layer. However, we are not sure how to interpret the weights and filters so we decided to pursue the second idea.

2. Incorporating the metadata into the deep learning model

    We used the metadata ('tmdb_metadata.csv) and the PCA text vectors (genre_words_pca.csv) from the last milestone for this deep learning model.

We built this model from scratch. We used sigmoid activation, 100 epochs, and precision, recall, and f1_score as metrics. This model's results:
Test loss: 10.53745; Test accuracy: 0.3396; Test precision: 0.1082; Test recall: 0.5942; Test f1_score: 0.1829
This model has lower accuracy and precision but higher recall than the baseline model.

In [295]:
```
from __future__ import print_function
from sklearn import preprocessing
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatte
n
import json
import urllib
import cStringIO
from keras.optimizers import SGD
from keras.optimizers import Adadelta

from keras import backend as K
import ast

import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')
import numpy as np
from sklearn.model_selection import train_test_split
```

In [273]:
```
# import data

# we have two data one with 128 * 128 posters, one with 32 * 32 posters
# we are running the 128 * 128 on AWS since it takes a long time.
# this notebook shows how run our model and is using 32 * 32 posters for
 faster performance
# data = pd.read_pickle('imgs_20000_128.pkl')
data = pd.read_pickle('imgs.pkl')
data.head()
```

Out[273]:

	RGB	genre_ids
0	[[[15, 36, 71], [13, 33, 68], [14, 34, 70], [1...	[14, 10402, 10749]
1	[[[8, 9, 8], [10, 10, 10], [11, 11, 11], [13, ...	[28, 18, 878]
2	[[[147, 122, 120], [170, 140, 132], [129, 100,...	[16, 35, 18, 10751, 10402]
3	[[[138, 47, 13], [150, 58, 16], [167, 74, 26],...	[28, 12, 14]
4	[[[255, 255, 255], [255, 255, 255], [254, 254,...	[28, 80, 53]

In [274]:
```
# this saves
# data.to_pickle('imgs.pkl')
```

In [275]:
```
# stack RGB values into the right shape
new_RGB = np.stack(data.RGB, axis = 0)
new_RGB.shape
```

Out[275]: (9991, 32, 32, 3)

In [217]:
```
# Drop bad values
data =
data.drop(data.index[[686,1784,2731,3311,5121,5653,8056,8063,9401,11334,1
2760,13628,14071,16186,17271,18552,18997,19659,19690]])
data.RGB.shape
```

Out[217]: (19785,)

In [216]:
```
# test which ones are the bad values, -> delete
res = data.RGB[0]
for i in range(len(data.RGB[1:])):
    try:
        np.stack((res, data.RGB[i]), axis=0)
    except:
        print(i)
```

```
686
1784
2731
3311
5121
5653
8056
8063
9401
11334
12760
13628
14071
16186
17271
18552
18997
19659
19690
```

In [276]:
```
# get genre list -> for getting the correct Y values
genre_list = urllib.urlopen("https://api.themoviedb.org/3/genre/movie/li
st?api_key=2dc6c9f1d17bd39dcbaef83321e1b5a3&language=en-US")

genre_list_json = json.loads(genre_list.read())

genre_lst = {}
for i in genre_list_json['genres']:
    genre_lst[i['id']] = str(i['name'])

labels = []
for i in data.genre_ids:
    label_matrix = np.zeros(len(genre_lst.keys()), dtype=int)
    for j in ast.literal_eval(i):
        if j in genre_lst.keys():
            label_matrix[genre_lst.keys().index(j)] = 1
    labels.append(label_matrix)
data['labels'] = labels
```

In [283]:
```
# input image dimensions - 128 * 128
# img_rows, img_cols = 128, 128
img_rows, img_cols = 32, 32

# smaller batch size means noisier gradient, but more updates per epoch
batch_size = 512
# this is fixed, we have 19 genres in our data set
num_classes = 19
# number of iterations over the complete training data
epochs = 20

# the data, shuffled and split between train and test sets
X = new_RGB
new_labels = np.stack(data['labels'], axis = 0)
Y = new_labels


x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.
3)

input_shape = (img_rows, img_cols, 3)

# normalize image values to [0,1]
# interestingly the keras example code does not center the data
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print (y_test.shape, 'y test samples')
```

```
x_train shape: (6993, 32, 32, 3)
6993 train samples
2998 test samples
(2998, 19) y test samples
```

In [298]:
```
# create an empty network model
model = Sequential()

# --- input layer ---
model.add(Conv2D(16, kernel_size=(5, 5), activation='relu',
input_shape=input_shape))
# --- max pool ---
model.add(MaxPooling2D(pool_size=(2, 2)))

# --- next layer ---
# we could double the number of filters as max pool made the
# feature maps much smaller
# just not doing this to improve runtime
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
# --- max pool ---
model.add(MaxPooling2D(pool_size=(2, 2)))

# flatten for fully connected classification layer
model.add(Flatten())
# note that the 19 is the number of classes we have
# the classes are not mutually exclusive so softmax is not a good choice
 - > we use sigmoid
# --- fully connected layer ---
model.add(Dense(64, activation='relu'))
# --- classification ---
model.add(Dense(19, activation='sigmoid'))

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_31 (Conv2D)           (None, 28, 28, 16)        1216
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 12, 12, 32)        4640
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
flatten_16 (Flatten)         (None, 1152)              0
_________________________________________________________________
dense_31 (Dense)             (None, 64)                73792
_________________________________________________________________
dense_32 (Dense)             (None, 19)                1235
=================================================================
Total params: 80,883
Trainable params: 80,883
Non-trainable params: 0
_________________________________________________________________
```

In [280]:
```
# new metrics function

## all these somehow don't work
from keras import metrics
import keras.backend as K

def precision(y_true, y_pred):
    """Precision metric.
    Only computes a batch-wise average of precision.
    Computes the precision, a metric for multi-label classification of
    how many selected items are relevant.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def recall(y_true, y_pred):
    """Recall metric.
    Only computes a batch-wise average of recall.
    Computes the recall, a metric for multi-label classification of
    how many relevant items are selected.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def f1_score(y_true, y_pred):

    # Count positive samples.
    c1 = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    c2 = K.sum(K.round(K.clip(y_pred, 0, 1)))
    c3 = K.sum(K.round(K.clip(y_true, 0, 1)))

    # If there are no true samples, fix the F1 score at 0.
    if c3 == 0:
        return 0

    # How many selected items are relevant?
    precision = c1 / c2

    # How many relevant items are selected?
    recall = c1 / c3

    # Calculate f1_score
    f1_score = 2 * (precision * recall) / (precision + recall)
    return f1_score
```

In [301]:
```
# this does all necessary compiling. In tensorflow this is much quicker
 than in theano
# the setup is our basic categorical crossentropy with stochastic gradie
nt decent
# we also specify that we want to evaluate our model in terms of accurac
y
sgd = SGD(lr=0.1, momentum=0.9)
model.compile(loss='binary_crossentropy',
              optimizer=sgd,
              metrics=['accuracy', precision, recall, f1_score])
```

In [302]:
```
# this is now the actual training
# in addition to the training data we provide validation data
# this data is used to calculate the performance of the model over all t
he epochs
# this is useful to determine when training should stop
# in our case we just use it to monitor the evolution of the model over
 the training epochs
# if we use the validation data to determine when to stop the training o
r which model to save, we
# should not use the test data, but a separate validation set.
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
#                     epochs=epochs,
                    epochs=20,
                    verbose=1,
                    validation_data=(x_test, y_test))

# once training is complete, let's see how well we have done
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 6993 samples, validate on 2998 samples
Epoch 1/20
6993/6993 [==============================] - 7s - loss: 0.3213 - acc:
 0.8687 - precision: 0.5415 - recall: 0.0893 - f1_score: 0.1521 - val_l
oss: 0.3164 - val_acc: 0.8702 - val_precision: 0.5469 - val_recall: 0.0
787 - val_f1_score: 0.1375
Epoch 2/20
6993/6993 [==============================] - 6s - loss: 0.3207 - acc:
 0.8685 - precision: 0.5361 - recall: 0.0876 - f1_score: 0.1505 - val_l
oss: 0.3160 - val_acc: 0.8710 - val_precision: 0.5397 - val_recall: 0.1
300 - val_f1_score: 0.2095
Epoch 3/20
6993/6993 [==============================] - 6s - loss: 0.3207 - acc:
 0.8691 - precision: 0.5411 - recall: 0.1052 - f1_score: 0.1742 - val_l
oss: 0.3159 - val_acc: 0.8707 - val_precision: 0.5497 - val_recall: 0.0
936 - val_f1_score: 0.1599
Epoch 4/20
6993/6993 [==============================] - 7s - loss: 0.3206 - acc:
 0.8687 - precision: 0.5325 - recall: 0.1033 - f1_score: 0.1723 - val_l
oss: 0.3164 - val_acc: 0.8707 - val_precision: 0.5545 - val_recall: 0.0
857 - val_f1_score: 0.1484
Epoch 5/20
6993/6993 [==============================] - 6s - loss: 0.3206 - acc:
 0.8690 - precision: 0.5466 - recall: 0.1099 - f1_score: 0.1800 - val_l
oss: 0.3161 - val_acc: 0.8703 - val_precision: 0.5600 - val_recall: 0.0
666 - val_f1_score: 0.1191
Epoch 6/20
6993/6993 [==============================] - 7s - loss: 0.3202 - acc:
 0.8689 - precision: 0.5540 - recall: 0.0729 - f1_score: 0.1284 - val_l
oss: 0.3156 - val_acc: 0.8710 - val_precision: 0.5506 - val_recall: 0.1
055 - val_f1_score: 0.1771
Epoch 7/20
6993/6993 [==============================] - 6s - loss: 0.3199 - acc:
 0.8695 - precision: 0.5462 - recall: 0.1117 - f1_score: 0.1842 - val_l
oss: 0.3157 - val_acc: 0.8709 - val_precision: 0.5367 - val_recall: 0.1
357 - val_f1_score: 0.2166
Epoch 8/20
6993/6993 [==============================] - 6s - loss: 0.3196 - acc:
 0.8695 - precision: 0.5474 - recall: 0.1144 - f1_score: 0.1879 - val_l
oss: 0.3157 - val_acc: 0.8715 - val_precision: 0.5573 - val_recall: 0.1
113 - val_f1_score: 0.1856
Epoch 9/20
6993/6993 [==============================] - 6s - loss: 0.3195 - acc:
 0.8693 - precision: 0.5488 - recall: 0.0938 - f1_score: 0.1599 - val_l
oss: 0.3152 - val_acc: 0.8712 - val_precision: 0.5405 - val_recall: 0.1
372 - val_f1_score: 0.2188
Epoch 10/20
6993/6993 [==============================] - 6s - loss: 0.3191 - acc:
 0.8695 - precision: 0.5424 - recall: 0.1164 - f1_score: 0.1896 - val_l
oss: 0.3157 - val_acc: 0.8718 - val_precision: 0.5581 - val_recall: 0.1
220 - val_f1_score: 0.2002
Epoch 11/20
6993/6993 [==============================] - 6s - loss: 0.3194 - acc:
 0.8698 - precision: 0.5498 - recall: 0.1251 - f1_score: 0.2026 - val_l
oss: 0.3152 - val_acc: 0.8707 - val_precision: 0.5737 - val_recall: 0.0
675 - val_f1_score: 0.1208
Epoch 12/20
```

```
6993/6993 [==============================] - 7s - loss: 0.3190 - acc:
 0.8695 - precision: 0.5481 - recall: 0.1316 - f1_score: 0.2069 - val_l
oss: 0.3151 - val_acc: 0.8705 - val_precision: 0.5612 - val_recall: 0.0
723 - val_f1_score: 0.1281
Epoch 13/20
6993/6993 [==============================] - 6s - loss: 0.3185 - acc:
 0.8694 - precision: 0.5452 - recall: 0.1134 - f1_score: 0.1861 - val_l
oss: 0.3145 - val_acc: 0.8714 - val_precision: 0.5417 - val_recall: 0.1
428 - val_f1_score: 0.2260
Epoch 14/20
6993/6993 [==============================] - 6s - loss: 0.3183 - acc:
 0.8699 - precision: 0.5475 - recall: 0.1254 - f1_score: 0.2025 - val_l
oss: 0.3148 - val_acc: 0.8714 - val_precision: 0.5563 - val_recall: 0.1
121 - val_f1_score: 0.1866
Epoch 15/20
6993/6993 [==============================] - 7s - loss: 0.3180 - acc:
 0.8696 - precision: 0.5423 - recall: 0.1356 - f1_score: 0.2149 - val_l
oss: 0.3144 - val_acc: 0.8715 - val_precision: 0.5669 - val_recall: 0.0
988 - val_f1_score: 0.1682
Epoch 16/20
6993/6993 [==============================] - 7s - loss: 0.3178 - acc:
 0.8702 - precision: 0.5501 - recall: 0.1303 - f1_score: 0.2100 - val_l
oss: 0.3148 - val_acc: 0.8716 - val_precision: 0.5578 - val_recall: 0.1
160 - val_f1_score: 0.1920
Epoch 17/20
6993/6993 [==============================] - 6s - loss: 0.3176 - acc:
 0.8698 - precision: 0.5450 - recall: 0.1300 - f1_score: 0.2096 - val_l
oss: 0.3137 - val_acc: 0.8714 - val_precision: 0.5408 - val_recall: 0.1
505 - val_f1_score: 0.2354
Epoch 18/20
6993/6993 [==============================] - 7s - loss: 0.3171 - acc:
 0.8705 - precision: 0.5527 - recall: 0.1427 - f1_score: 0.2263 - val_l
oss: 0.3145 - val_acc: 0.8710 - val_precision: 0.5513 - val_recall: 0.1
070 - val_f1_score: 0.1792
Epoch 19/20
6993/6993 [==============================] - 7s - loss: 0.3168 - acc:
 0.8700 - precision: 0.5439 - recall: 0.1454 - f1_score: 0.2281 - val_l
oss: 0.3141 - val_acc: 0.8715 - val_precision: 0.5765 - val_recall: 0.0
862 - val_f1_score: 0.1500
Epoch 20/20
6993/6993 [==============================] - 7s - loss: 0.3168 - acc:
 0.8701 - precision: 0.5471 - recall: 0.1462 - f1_score: 0.2280 - val_l
oss: 0.3139 - val_acc: 0.8712 - val_precision: 0.5528 - val_recall: 0.1
096 - val_f1_score: 0.1829
Test loss: 0.313871270522
Test accuracy: 0.871194845124
```

In [290]:

```
# # here is a visualization of the training process
# # typically we gain a lot in the beginning and then
# # training slows down
# plt.plot(history.history['acc'])
# plt.xlabel("epoch")
# plt.ylabel("accuracy")
```

Trying a different setting for the model

In [303]:
```
# create an empty network model
model2 = Sequential()

# --- input layer ---
model2.add(Conv2D(16, kernel_size=(5, 5), activation='relu',
input_shape=input_shape))
# --- max pool ---
model2.add(MaxPooling2D(pool_size=(2, 2)))

# --- next layer ---
# we could double the number of filters as max pool made the
# feature maps much smaller
# just not doing this to improve runtime
model2.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
# --- max pool ---
model2.add(MaxPooling2D(pool_size=(2, 2)))

# flatten for fully connected classification layer
model2.add(Flatten())
# note that the 19 is the number of classes we have
# the classes are not mutually exclusive so softmax is not a good choice
# --- fully connected layer ---
model2.add(Dense(64, activation='relu'))
# --- classification ---
model2.add(Dense(19, activation='sigmoid'))

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_33 (Conv2D)           (None, 28, 28, 16)        1216      
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 14, 14, 16)        0         
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 12, 12, 32)        4640      
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 6, 6, 32)          0         
_________________________________________________________________
flatten_17 (Flatten)         (None, 1152)              0         
_________________________________________________________________
dense_33 (Dense)             (None, 64)                73792     
_________________________________________________________________
dense_34 (Dense)             (None, 19)                1235      
=================================================================
Total params: 80,883
Trainable params: 80,883
Non-trainable params: 0
_________________________________________________________________
```

In [304]:
```
# adaptive learning rate
ada = Adadelta(lr=1.0, rho=0.95, epsilon=1e-08, decay=0.0)
model2.compile(loss='binary_crossentropy',
               optimizer=ada,
               metrics=['accuracy', precision, recall, f1_score])
```

In [305]:
```
# this is now the actual training
# in addition to the training data we provide validation data
# this data is used to calculate the performance of the model over all t
he epochs
# this is useful to determine when training should stop
# in our case we just use it to monitor the evolution of the model over
 the training epochs
# if we use the validation data to determine when to stop the training o
r which model to save, we
# should not use the test data, but a separate validation set.
history2 = model2.fit(x_train, y_train,
                    batch_size=batch_size,
#                     epochs=epochs,
                    epochs=20,
                    verbose=1,
                    validation_data=(x_test, y_test))

# once training is complete, let's see how well we have done
score = model2.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 6993 samples, validate on 2998 samples
Epoch 1/20
6993/6993 [==============================] - 8s - loss: 0.5319 - acc:
 0.7693 - precision: 0.2106 - recall: 0.2309 - f1_score: 0.1846 - val_l
oss: 0.3496 - val_acc: 0.8682 - val_precision: 0.4023 - val_recall: 0.0
027 - val_f1_score: 0.0053
Epoch 2/20
6993/6993 [==============================] - 6s - loss: 0.3486 - acc:
 0.8665 - precision: 0.4263 - recall: 0.0269 - f1_score: 0.0498 - val_l
oss: 0.3408 - val_acc: 0.8687 - val_precision: 0.5465 - val_recall: 0.0
146 - val_f1_score: 0.0283
Epoch 3/20
6993/6993 [==============================] - 6s - loss: 0.3453 - acc:
 0.8662 - precision: 0.4574 - recall: 0.0555 - f1_score: 0.0915 - val_l
oss: 0.3394 - val_acc: 0.8686 - val_precision: 0.5088 - val_recall: 0.0
340 - val_f1_score: 0.0638
Epoch 4/20
6993/6993 [==============================] - 6s - loss: 0.3433 - acc:
 0.8666 - precision: 0.4846 - recall: 0.0367 - f1_score: 0.0646 - val_l
oss: 0.3370 - val_acc: 0.8688 - val_precision: 0.5459 - val_recall: 0.0
144 - val_f1_score: 0.0281
Epoch 5/20
6993/6993 [==============================] - 6s - loss: 0.3419 - acc:
 0.8659 - precision: 0.4703 - recall: 0.0431 - f1_score: 0.0725 - val_l
oss: 0.3357 - val_acc: 0.8687 - val_precision: 0.5132 - val_recall: 0.0
383 - val_f1_score: 0.0713
Epoch 6/20
6993/6993 [==============================] - 6s - loss: 0.3397 - acc:
 0.8664 - precision: 0.4968 - recall: 0.0417 - f1_score: 0.0693 - val_l
oss: 0.3333 - val_acc: 0.8685 - val_precision: 0.4984 - val_recall: 0.0
033 - val_f1_score: 0.0066
Epoch 7/20
6993/6993 [==============================] - 7s - loss: 0.3377 - acc:
 0.8671 - precision: 0.5180 - recall: 0.0303 - f1_score: 0.0546 - val_l
oss: 0.3335 - val_acc: 0.8684 - val_precision: 0.4968 - val_recall: 0.0
120 - val_f1_score: 0.0234
Epoch 8/20
6993/6993 [==============================] - 7s - loss: 0.3361 - acc:
 0.8669 - precision: 0.4821 - recall: 0.0363 - f1_score: 0.0625 - val_l
oss: 0.3294 - val_acc: 0.8681 - val_precision: 0.4943 - val_recall: 0.1
064 - val_f1_score: 0.1750
Epoch 9/20
6993/6993 [==============================] - 9s - loss: 0.3331 - acc:
 0.8671 - precision: 0.5085 - recall: 0.0527 - f1_score: 0.0896 - val_l
oss: 0.3266 - val_acc: 0.8687 - val_precision: 0.5391 - val_recall: 0.0
124 - val_f1_score: 0.0242
Epoch 10/20
6993/6993 [==============================] - 8s - loss: 0.3310 - acc:
 0.8670 - precision: 0.4947 - recall: 0.0528 - f1_score: 0.0896 - val_l
oss: 0.3272 - val_acc: 0.8690 - val_precision: 0.5178 - val_recall: 0.0
598 - val_f1_score: 0.1072
Epoch 11/20
6993/6993 [==============================] - 7s - loss: 0.3311 - acc:
 0.8671 - precision: 0.4996 - recall: 0.0756 - f1_score: 0.1273 - val_l
oss: 0.3237 - val_acc: 0.8687 - val_precision: 0.5106 - val_recall: 0.0
434 - val_f1_score: 0.0799
Epoch 12/20
```

```
6993/6993 [==============================] - 8s - loss: 0.3282 - acc:
 0.8667 - precision: 0.4954 - recall: 0.0718 - f1_score: 0.1226 - val_l
oss: 0.3233 - val_acc: 0.8692 - val_precision: 0.5420 - val_recall: 0.0
374 - val_f1_score: 0.0699
Epoch 13/20
6993/6993 [==============================] - 9s - loss: 0.3277 - acc:
 0.8666 - precision: 0.4998 - recall: 0.0788 - f1_score: 0.1313 - val_l
oss: 0.3268 - val_acc: 0.8670 - val_precision: 0.4779 - val_recall: 0.1
187 - val_f1_score: 0.1901
Epoch 14/20
6993/6993 [==============================] - 7s - loss: 0.3273 - acc:
 0.8667 - precision: 0.4916 - recall: 0.0789 - f1_score: 0.1307 - val_l
oss: 0.3209 - val_acc: 0.8682 - val_precision: 0.4962 - val_recall: 0.1
404 - val_f1_score: 0.2188
Epoch 15/20
6993/6993 [==============================] - 7s - loss: 0.3257 - acc:
 0.8675 - precision: 0.5050 - recall: 0.0965 - f1_score: 0.1592 - val_l
oss: 0.3200 - val_acc: 0.8693 - val_precision: 0.5309 - val_recall: 0.0
587 - val_f1_score: 0.1057
Epoch 16/20
6993/6993 [==============================] - 7s - loss: 0.3246 - acc:
 0.8677 - precision: 0.5181 - recall: 0.0908 - f1_score: 0.1509 - val_l
oss: 0.3207 - val_acc: 0.8696 - val_precision: 0.5336 - val_recall: 0.0
736 - val_f1_score: 0.1293
Epoch 17/20
6993/6993 [==============================] - 9s - loss: 0.3250 - acc:
 0.8673 - precision: 0.5089 - recall: 0.0938 - f1_score: 0.1545 - val_l
oss: 0.3198 - val_acc: 0.8697 - val_precision: 0.5328 - val_recall: 0.0
770 - val_f1_score: 0.1346
Epoch 18/20
6993/6993 [==============================] - 7s - loss: 0.3241 - acc:
 0.8681 - precision: 0.5295 - recall: 0.0949 - f1_score: 0.1580 - val_l
oss: 0.3211 - val_acc: 0.8686 - val_precision: 0.5041 - val_recall: 0.0
581 - val_f1_score: 0.1041
Epoch 19/20
6993/6993 [==============================] - 8s - loss: 0.3235 - acc:
 0.8684 - precision: 0.5341 - recall: 0.0943 - f1_score: 0.1578 - val_l
oss: 0.3185 - val_acc: 0.8701 - val_precision: 0.5319 - val_recall: 0.1
025 - val_f1_score: 0.1719
Epoch 20/20
6993/6993 [==============================] - 8s - loss: 0.3236 - acc:
 0.8682 - precision: 0.5277 - recall: 0.1052 - f1_score: 0.1717 - val_l
oss: 0.3206 - val_acc: 0.8689 - val_precision: 0.5163 - val_recall: 0.0
517 - val_f1_score: 0.0940
Test loss: 0.320583623656
Test accuracy: 0.868877511489
```

In [1]:
```
from __future__ import print_function
import keras
# from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatte
n
from keras.optimizers import SGD, Adam
from keras import backend as K

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

import numpy as np
import pandas as pd
from sklearn.cross_validation import train_test_split

from keras.callbacks import TensorBoard

import urllib
import json
import ast
```

```
Using TensorFlow backend.
```

In [2]:
```
data = pd.read_pickle("imgs.pkl")
# get genre list
genre_list = urllib.urlopen("https://api.themoviedb.org/3/genre/movie/li
st?api_key=2dc6c9f1d17bd39dcbaef83321e1b5a3&language=en-US")

genre_list_json = json.loads(genre_list.read())

genre_lst = {}
for i in genre_list_json['genres']:
    genre_lst[i['id']] = str(i['name'])

labels = []
for i in data.genre_ids:
    label_matrix = np.zeros(len(genre_lst.keys()), dtype=int)
    for j in ast.literal_eval(i):
        if j in genre_lst.keys():
            label_matrix[genre_lst.keys().index(j)] = 1
    labels.append(label_matrix)
data['labels'] = labels
data.head()
```

Out[2]:

	RGB	genre_ids	labels
0	[[[15, 36, 71], [13, 33, 68], [14, 34, 70], [1...	[14, 10402, 10749]	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
1	[[[8, 9, 8], [10, 10, 10], [11, 11, 11], [13, ...	[28, 18, 878]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, ...
2	[[[147, 122, 120], [170, 140, 132], [129, 100,...	[16, 35, 18, 10751, 10402]	[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, ...
3	[[[138, 47, 13], [150, 58, 16], [167, 74, 26],...	[28, 12, 14]	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, ...
4	[[[255, 255, 255], [255, 255, 255], [254, 254,...	[28, 80, 53]	[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...

In [3]:
```
# the data, shuffled and split between train and test sets
aset = []
for i in range(0, len(data)):
    if data['RGB'][i].shape != (32, 32, 3):
        aset.append(i)
set(aset) # not all are in the shape 32*32*3
# drop everything not in that shape
data = data.drop(data.index[aset])
len(data)
```

Out[3]: 9991

In [4]:
```
# input image dimensions
img_rows, img_cols = 32, 32

# the data, shuffled and split between train and test sets
new_RGB = np.stack(data['RGB'], axis = 0)
X = new_RGB
Y = np.stack(data['labels'], axis=0)
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.
3)

## X_train is of shape n_samples x 32 x 32
## for a CNN we want to keep the image shape
## need to explicitly tell keras that it is a RGB value image
## so each image is 32x32x3
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 3, img_rows, img_cols)
    x_test = x_test.reshape(x_test.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)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 3)
    input_shape = (img_rows, img_cols, 3)

# normalize image values to [0,1]
# interestingly the keras example code does not center the data
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (6993, 32, 32, 3)
6993 train samples
2998 test samples
```

Use a pre-trained Neural Network

In [5]:
```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.layers import Input, Flatten, Dense
from keras.models import Model
import numpy as np
import h5py
```

In [6]:
```
#Get back the convolutional part of a VGG network trained on ImageNet
model_vgg16_conv = VGG16(weights='imagenet', include_top=False)
model_vgg16_conv.summary()
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/
releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h
5
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, None, None, 3)     0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0         
=================================================================
Total params: 14,714,688.0
Trainable params: 14,714,688.0
Non-trainable params: 0.0
_________________________________________________________________
```

Baseline Model

In [7]:
```
# new metrics function
from keras import metrics

def precision(y_true, y_pred):
    """Precision metric.
    Only computes a batch-wise average of precision.
    Computes the precision, a metric for multi-label classification of
    how many selected items are relevant.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def recall(y_true, y_pred):
    """Recall metric.
    Only computes a batch-wise average of recall.
    Computes the recall, a metric for multi-label classification of
    how many relevant items are selected.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def f1_score(y_true, y_pred):

    # Count positive samples.
    c1 = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    c2 = K.sum(K.round(K.clip(y_pred, 0, 1)))
    c3 = K.sum(K.round(K.clip(y_true, 0, 1)))

    # If there are no true samples, fix the F1 score at 0.
    if c3 == 0:
        return 0
    # How many selected items are relevant?
    precision = c1 / c2

    # How many relevant items are selected?
    recall = c1 / c3

    # Calculate f1_score
    f1_score = 2 * (precision * recall) / (precision + recall)
    return f1_score
```

In [8]:
```
#Create your own input format (here 3x32x32)
input = Input(shape=(32,32,3),name = 'image_input')

#Use the generated model
output_vgg16_conv = model_vgg16_conv(input)

#Add the fully-connected layers
m = Flatten(name='flatten')(output_vgg16_conv)
m = Dense(4096, activation='relu', name='fc1')(m)
m = Dense(4096, activation='relu', name='fc2')(m)
m = Dense(19, activation='sigmoid', name='predictions')(m) #sigmoid inst
ead of softmax

#Create my own model
model2 = Model(input=input, output=m)
model2.summary()

sgd = SGD(lr=0.1, momentum=0.9)

model2.compile(loss='binary_crossentropy',
              optimizer=sgd,
              metrics=['accuracy',precision,recall,f1_score])
```

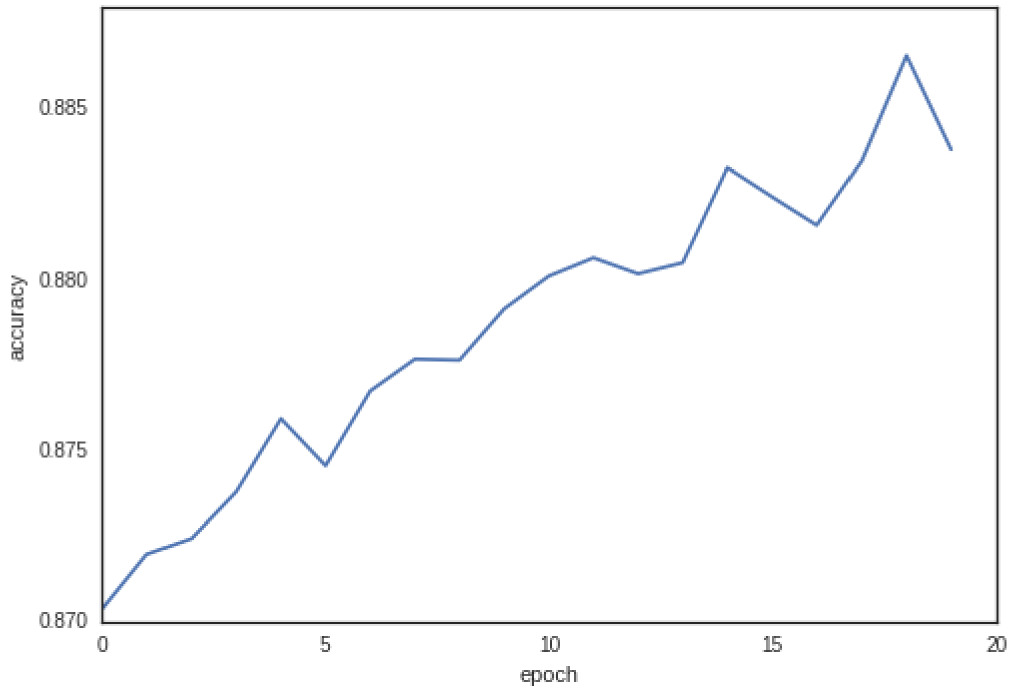
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
image_input (InputLayer)     (None, 32, 32, 3)         0
_________________________________________________________________
vgg16 (Model)                multiple                  14714688
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              2101248
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 19)                77843
=================================================================
Total params: 33,675,091.0
Trainable params: 33,675,091.0
Non-trainable params: 0.0
_________________________________________________________________

/home/ubuntu/.local/lib/python2.7/site-packages/ipykernel/__main__.py:1
4: UserWarning: Update your `Model` call to the Keras 2 API: `Model(out
puts=Tensor("pr..., inputs=Tensor("im...)`
```

In [14]:
```
history2 = model2.fit(x_train, y_train,
                      batch_size=512,
                      epochs=20,
                      verbose=1,
                      validation_data=(x_test, y_test))
```

```
Train on 6993 samples, validate on 2998 samples
Epoch 1/20
6993/6993 [==============================] - 12s - loss: 0.3274 - acc:
 0.8704 - precision: 0.5709 - recall: 0.1000 - f1_score: 0.1663 - val_l
oss: 0.3223 - val_acc: 0.8715 - val_precision: 0.5537 - val_recall: 0.1
469 - val_f1_score: 0.2321
Epoch 2/20
6993/6993 [==============================] - 12s - loss: 0.3189 - acc:
 0.8720 - precision: 0.5608 - recall: 0.1657 - f1_score: 0.2555 - val_l
oss: 0.3156 - val_acc: 0.8717 - val_precision: 0.5486 - val_recall: 0.1
687 - val_f1_score: 0.2580
Epoch 3/20
6993/6993 [==============================] - 12s - loss: 0.3155 - acc:
 0.8724 - precision: 0.5639 - recall: 0.1769 - f1_score: 0.2688 - val_l
oss: 0.3147 - val_acc: 0.8714 - val_precision: 0.5494 - val_recall: 0.1
520 - val_f1_score: 0.2381
Epoch 4/20
6993/6993 [==============================] - 12s - loss: 0.3114 - acc:
 0.8738 - precision: 0.5757 - recall: 0.1948 - f1_score: 0.2889 - val_l
oss: 0.3103 - val_acc: 0.8735 - val_precision: 0.5483 - val_recall: 0.2
450 - val_f1_score: 0.3386
Epoch 5/20
6993/6993 [==============================] - 12s - loss: 0.3063 - acc:
 0.8759 - precision: 0.5854 - recall: 0.2259 - f1_score: 0.3249 - val_l
oss: 0.3054 - val_acc: 0.8764 - val_precision: 0.5925 - val_recall: 0.2
080 - val_f1_score: 0.3078
Epoch 6/20
6993/6993 [==============================] - 12s - loss: 0.3079 - acc:
 0.8746 - precision: 0.5795 - recall: 0.2052 - f1_score: 0.3012 - val_l
oss: 0.3081 - val_acc: 0.8759 - val_precision: 0.6049 - val_recall: 0.1
772 - val_f1_score: 0.2740
Epoch 7/20
6993/6993 [==============================] - 12s - loss: 0.3035 - acc:
 0.8767 - precision: 0.5993 - recall: 0.2215 - f1_score: 0.3212 - val_l
oss: 0.3077 - val_acc: 0.8739 - val_precision: 0.5549 - val_recall: 0.2
318 - val_f1_score: 0.3269
Epoch 8/20
6993/6993 [==============================] - 12s - loss: 0.3011 - acc:
 0.8777 - precision: 0.6030 - recall: 0.2314 - f1_score: 0.3336 - val_l
oss: 0.3049 - val_acc: 0.8769 - val_precision: 0.5990 - val_recall: 0.2
086 - val_f1_score: 0.3093
Epoch 9/20
6993/6993 [==============================] - 12s - loss: 0.2983 - acc:
 0.8777 - precision: 0.6015 - recall: 0.2343 - f1_score: 0.3366 - val_l
oss: 0.3106 - val_acc: 0.8731 - val_precision: 0.5796 - val_recall: 0.1
471 - val_f1_score: 0.2346
Epoch 10/20
6993/6993 [==============================] - 12s - loss: 0.2965 - acc:
 0.8792 - precision: 0.6169 - recall: 0.2395 - f1_score: 0.3431 - val_l
oss: 0.3012 - val_acc: 0.8776 - val_precision: 0.5996 - val_recall: 0.2
247 - val_f1_score: 0.3268
Epoch 11/20
6993/6993 [==============================] - 12s - loss: 0.2937 - acc:
 0.8801 - precision: 0.6154 - recall: 0.2602 - f1_score: 0.3647 - val_l
oss: 0.3033 - val_acc: 0.8770 - val_precision: 0.5798 - val_recall: 0.2
534 - val_f1_score: 0.3525
Epoch 12/20
```

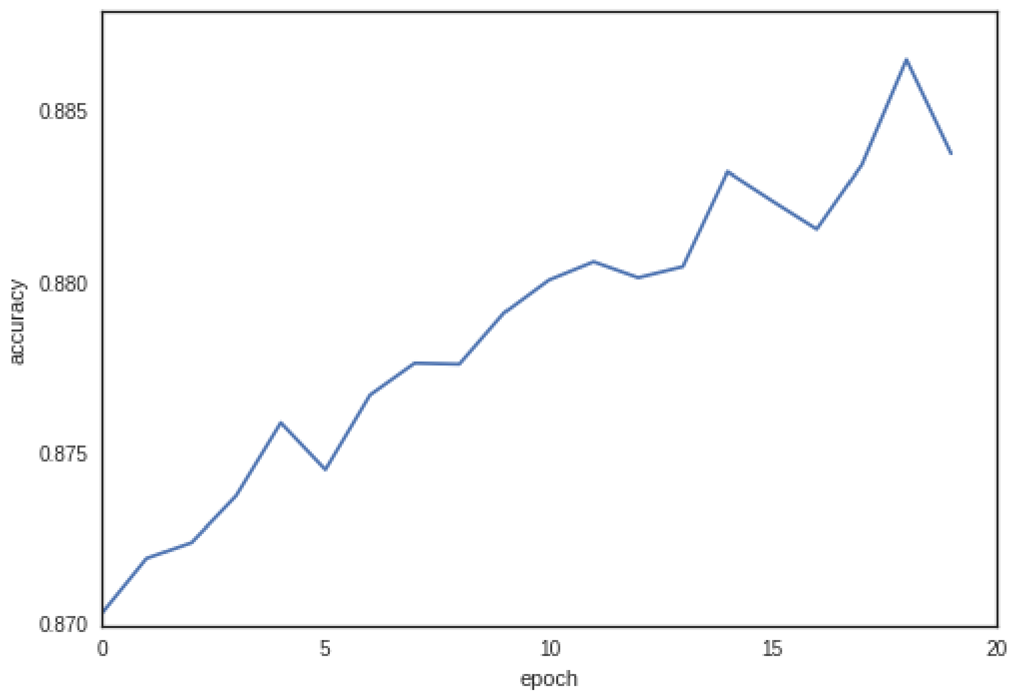
```
6993/6993 [==============================] - 12s - loss: 0.2918 - acc:
 0.8807 - precision: 0.6220 - recall: 0.2594 - f1_score: 0.3652 - val_l
oss: 0.3047 - val_acc: 0.8765 - val_precision: 0.5692 - val_recall: 0.2
720 - val_f1_score: 0.3680
Epoch 13/20
6993/6993 [==============================] - 12s - loss: 0.2920 - acc:
 0.8802 - precision: 0.6165 - recall: 0.2616 - f1_score: 0.3660 - val_l
oss: 0.3060 - val_acc: 0.8762 - val_precision: 0.5613 - val_recall: 0.2
900 - val_f1_score: 0.3823
Epoch 14/20
6993/6993 [==============================] - 12s - loss: 0.2899 - acc:
 0.8805 - precision: 0.6166 - recall: 0.2687 - f1_score: 0.3727 - val_l
oss: 0.3099 - val_acc: 0.8754 - val_precision: 0.5769 - val_recall: 0.2
147 - val_f1_score: 0.3129
Epoch 15/20
6993/6993 [==============================] - 12s - loss: 0.2859 - acc:
 0.8833 - precision: 0.6379 - recall: 0.2801 - f1_score: 0.3886 - val_l
oss: 0.3042 - val_acc: 0.8781 - val_precision: 0.5844 - val_recall: 0.2
698 - val_f1_score: 0.3691
Epoch 16/20
6993/6993 [==============================] - 12s - loss: 0.2887 - acc:
 0.8824 - precision: 0.6242 - recall: 0.2917 - f1_score: 0.3969 - val_l
oss: 0.3140 - val_acc: 0.8733 - val_precision: 0.5474 - val_recall: 0.2
425 - val_f1_score: 0.3359
Epoch 17/20
6993/6993 [==============================] - 12s - loss: 0.2898 - acc:
 0.8816 - precision: 0.6315 - recall: 0.2627 - f1_score: 0.3698 - val_l
oss: 0.3035 - val_acc: 0.8768 - val_precision: 0.5888 - val_recall: 0.2
254 - val_f1_score: 0.3259
Epoch 18/20
6993/6993 [==============================] - 12s - loss: 0.2826 - acc:
 0.8835 - precision: 0.6293 - recall: 0.3019 - f1_score: 0.4069 - val_l
oss: 0.3054 - val_acc: 0.8765 - val_precision: 0.5767 - val_recall: 0.2
497 - val_f1_score: 0.3484
Epoch 19/20
6993/6993 [==============================] - 12s - loss: 0.2761 - acc:
 0.8866 - precision: 0.6584 - recall: 0.3054 - f1_score: 0.4160 - val_l
oss: 0.3036 - val_acc: 0.8783 - val_precision: 0.5891 - val_recall: 0.2
632 - val_f1_score: 0.3638
Epoch 20/20
6993/6993 [==============================] - 12s - loss: 0.2854 - acc:
 0.8838 - precision: 0.6237 - recall: 0.3179 - f1_score: 0.4205 - val_l
oss: 0.3208 - val_acc: 0.8702 - val_precision: 0.5185 - val_recall: 0.2
540 - val_f1_score: 0.3409
['loss', 'acc', 'precision', 'recall', 'f1_score']
Test loss: 0.320800663691
Test accuracy: 0.870176484618
Precision 0.520502572938
Recall 0.255452839811
f1 score 0.341115456728
```

Out[14]:
```
<matplotlib.text.Text at 0x7fbb9c04d710>
```


0.885
0.880
0.875
0.870
accuracy
0
5
10
15
20
epoch

In [17]:
```
# here is a visualization of the training process
plt.plot(history2.history['acc'])
plt.xlabel("epoch")
plt.ylabel("accuracy")

print(model2.metrics_names)
score = model2.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print('Precision', score[2])
print('Recall', score[3])
print('f1 score', score[4])
```

```
['loss', 'acc', 'precision', 'recall', 'f1_score']
Test loss: 0.320800663691
Test accuracy: 0.870176484618
Precision 0.520502572938
Recall 0.255452839811
f1 score 0.341115456728
```



Change optimizer to Adam

In [19]:
```
model3 = Model(input=input, output=m) #keeping settings from model 2
model3.summary()

# change optimizer to adam with default parameter values
adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
decay=0.0)
model3.compile(loss='binary_crossentropy',
              optimizer=adam,
              metrics=['accuracy', precision, recall, f1_score])

batch_size = 512
epochs = 20
history3 = model3.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
/home/ubuntu/.local/lib/python2.7/site-packages/ipykernel/__main__.py:
1: UserWarning: Update your `Model` call to the Keras 2 API: `Model(out
puts=Tensor("pr..., inputs=Tensor("im...)`
  if __name__ == '__main__':
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
image_input (InputLayer)     (None, 32, 32, 3)         0         
_________________________________________________________________
vgg16 (Model)                multiple                  14714688  
_________________________________________________________________
flatten (Flatten)            (None, 512)               0         
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              2101248   
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312  
_________________________________________________________________
predictions (Dense)          (None, 19)                77843     
=================================================================
Total params: 33,675,091.0
Trainable params: 33,675,091.0
Non-trainable params: 0.0
_________________________________________________________________
Train on 6993 samples, validate on 2998 samples
Epoch 1/20
6993/6993 [==============================] - 13s - loss: 0.3513 - acc:
 0.8636 - precision: 0.2403 - recall: 0.0908 - f1_score: nan - val_los
s: 0.3407 - val_acc: 0.8650 - val_precision: 0.4733 - val_recall: 0.188
6 - val_f1_score: 0.2697
Epoch 2/20
6993/6993 [==============================] - 12s - loss: 0.3365 - acc:
 0.8670 - precision: 0.1436 - recall: 0.0454 - f1_score: nan - val_los
s: 0.3342 - val_acc: 0.8678 - val_precision: 0.0000e+00 - val_recall:
 0.0000e+00 - val_f1_score: nan
Epoch 3/20
6993/6993 [==============================] - 12s - loss: 0.3322 - acc:
 0.8668 - precision: 0.2400 - recall: 0.0301 - f1_score: nan - val_los
s: 0.3398 - val_acc: 0.8532 - val_precision: 0.4162 - val_recall: 0.274
0 - val_f1_score: 0.3304
Epoch 4/20
6993/6993 [==============================] - 12s - loss: 0.3326 - acc:
 0.8647 - precision: 0.4078 - recall: 0.1008 - f1_score: nan - val_los
s: 0.3243 - val_acc: 0.8689 - val_precision: 0.6429 - val_recall: 0.018
9 - val_f1_score: 0.0368
Epoch 5/20
6993/6993 [==============================] - 12s - loss: 0.3214 - acc:
 0.8692 - precision: 0.5502 - recall: 0.1150 - f1_score: 0.1814 - val_l
oss: 0.3195 - val_acc: 0.8694 - val_precision: 0.5183 - val_recall: 0.1
675 - val_f1_score: 0.2531
Epoch 6/20
6993/6993 [==============================] - 12s - loss: 0.3155 - acc:
 0.8712 - precision: 0.5470 - recall: 0.1750 - f1_score: 0.2627 - val_l
oss: 0.3134 - val_acc: 0.8707 - val_precision: 0.5307 - val_recall: 0.1
933 - val_f1_score: 0.2834
Epoch 7/20
6993/6993 [==============================] - 12s - loss: 0.3138 - acc:
 0.8714 - precision: 0.5451 - recall: 0.1969 - f1_score: 0.2874 - val_l
oss: 0.3185 - val_acc: 0.8701 - val_precision: 0.5261 - val_recall: 0.1
808 - val_f1_score: 0.2691
Epoch 8/20
6993/6993 [==============================] - 12s - loss: 0.3116 - acc:
```

```
 0.8726 - precision: 0.5597 - recall: 0.1953 - f1_score: 0.2827 - val_l
oss: 0.3135 - val_acc: 0.8724 - val_precision: 0.5377 - val_recall: 0.2
471 - val_f1_score: 0.3385
Epoch 9/20
6993/6993 [==============================] - 12s - loss: 0.3117 - acc:
 0.8726 - precision: 0.5466 - recall: 0.2375 - f1_score: 0.3304 - val_l
oss: 0.3121 - val_acc: 0.8717 - val_precision: 0.5360 - val_recall: 0.2
184 - val_f1_score: 0.3103
Epoch 10/20
6993/6993 [==============================] - 12s - loss: 0.3096 - acc:
 0.8725 - precision: 0.5503 - recall: 0.2119 - f1_score: 0.3039 - val_l
oss: 0.3134 - val_acc: 0.8718 - val_precision: 0.5325 - val_recall: 0.2
497 - val_f1_score: 0.3399
Epoch 11/20
6993/6993 [==============================] - 12s - loss: 0.3054 - acc:
 0.8764 - precision: 0.5807 - recall: 0.2501 - f1_score: 0.3491 - val_l
oss: 0.3124 - val_acc: 0.8730 - val_precision: 0.5438 - val_recall: 0.2
438 - val_f1_score: 0.3366
Epoch 12/20
6993/6993 [==============================] - 12s - loss: 0.3040 - acc:
 0.8768 - precision: 0.5799 - recall: 0.2627 - f1_score: 0.3613 - val_l
oss: 0.3152 - val_acc: 0.8732 - val_precision: 0.5546 - val_recall: 0.2
067 - val_f1_score: 0.3010
Epoch 13/20
6993/6993 [==============================] - 12s - loss: 0.3031 - acc:
 0.8769 - precision: 0.5841 - recall: 0.2530 - f1_score: 0.3522 - val_l
oss: 0.3190 - val_acc: 0.8704 - val_precision: 0.5224 - val_recall: 0.2
302 - val_f1_score: 0.3195
Epoch 14/20
6993/6993 [==============================] - 12s - loss: 0.2984 - acc:
 0.8787 - precision: 0.5947 - recall: 0.2726 - f1_score: 0.3729 - val_l
oss: 0.3158 - val_acc: 0.8725 - val_precision: 0.5398 - val_recall: 0.2
429 - val_f1_score: 0.3349
Epoch 15/20
6993/6993 [==============================] - 12s - loss: 0.2936 - acc:
 0.8822 - precision: 0.6152 - recall: 0.3018 - f1_score: 0.4046 - val_l
oss: 0.3150 - val_acc: 0.8733 - val_precision: 0.5455 - val_recall: 0.2
515 - val_f1_score: 0.3441
Epoch 16/20
6993/6993 [==============================] - 12s - loss: 0.2906 - acc:
 0.8835 - precision: 0.6225 - recall: 0.3123 - f1_score: 0.4156 - val_l
oss: 0.3203 - val_acc: 0.8724 - val_precision: 0.5395 - val_recall: 0.2
343 - val_f1_score: 0.3266
Epoch 17/20
6993/6993 [==============================] - 12s - loss: 0.2847 - acc:
 0.8859 - precision: 0.6396 - recall: 0.3212 - f1_score: 0.4272 - val_l
oss: 0.3214 - val_acc: 0.8730 - val_precision: 0.5389 - val_recall: 0.2
733 - val_f1_score: 0.3625
Epoch 18/20
6993/6993 [==============================] - 12s - loss: 0.2808 - acc:
 0.8870 - precision: 0.6420 - recall: 0.3366 - f1_score: 0.4415 - val_l
oss: 0.3213 - val_acc: 0.8717 - val_precision: 0.5300 - val_recall: 0.2
612 - val_f1_score: 0.3498
Epoch 19/20
6993/6993 [==============================] - 12s - loss: 0.2766 - acc:
 0.8893 - precision: 0.6608 - recall: 0.3409 - f1_score: 0.4496 - val_l
oss: 0.3281 - val_acc: 0.8711 - val_precision: 0.5246 - val_recall: 0.2
```

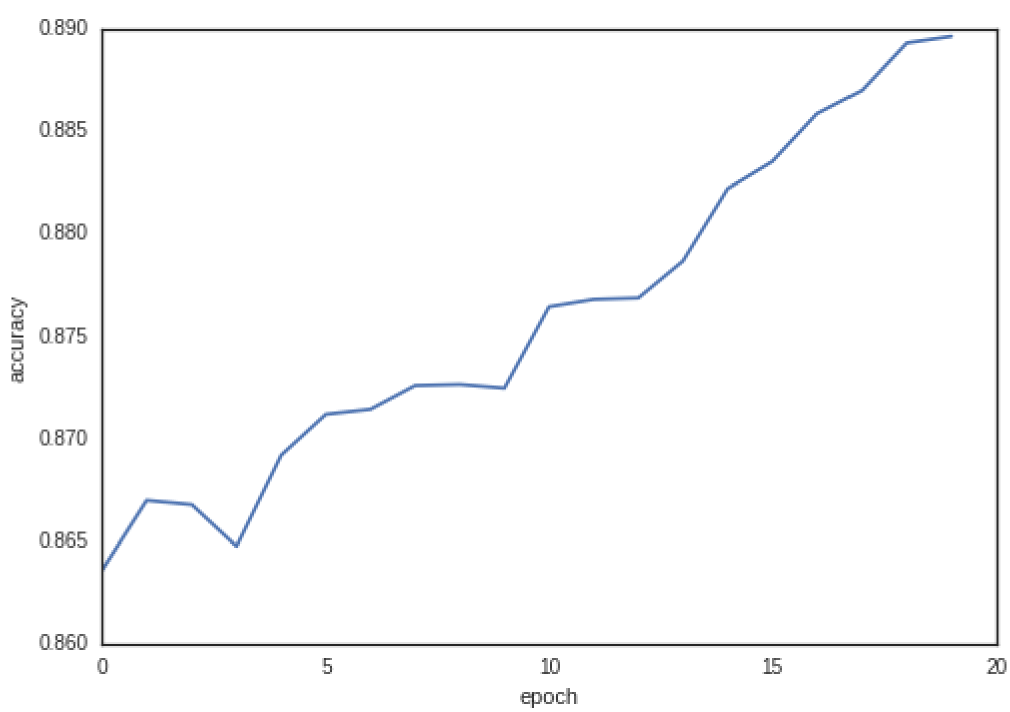
```
684 - val_f1_score: 0.3550
Epoch 20/20
6993/6993 [==============================] - 12s - loss: 0.2743 - acc:
 0.8896 - precision: 0.6628 - recall: 0.3433 - f1_score: 0.4522 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
```

In [20]:
```
# once training is complete, let's see how well we have done
score = model3.evaluate(x_test, y_test, verbose=0)
print(model3.metrics_names)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print('Precision', score[2])
print('Recall', score[3])
print('f1 score', score[4])
```

```
['loss', 'acc', 'precision', 'recall', 'f1_score']
Test loss: 0.326375924921
Test accuracy: 0.86956204137
Precision 0.512464917287
Recall 0.271144805013
f1 score 0.353980610357
```

In [21]:
```
plt.plot(history3.history['acc'])
plt.xlabel("epoch")
plt.ylabel("accuracy")
```

Out[21]: <matplotlib.text.Text at 0x7fbb7bc51090>



Tune: Drop-based Learning Rate Schedule

based on http://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/ (http://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/)

In [25]:
```
from keras.callbacks import LearningRateScheduler

# learning rate schedule
def step_decay(epoch):
    initial_lrate = 0.1
    drop = 0.5
    epochs_drop = 10.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_d
rop))
    return lrate

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)


# Compile model
model4 = Model(input=input, output=m)
sgd = SGD(lr=0.0, momentum=0.9, decay=0.0, nesterov=False)
model4.compile(loss='binary_crossentropy',
               optimizer=sgd,
               metrics=['accuracy',precision,recall,f1_score])


# learning schedule callback
lrate = LearningRateScheduler(step_decay)
callbacks_list = [lrate]

history4 = model4.fit(x_train, y_train,
                    batch_size=512,
                    epochs=20,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
/home/ubuntu/.local/lib/python2.7/site-packages/ipykernel/__main__.py:1
7: UserWarning: Update your `Model` call to the Keras 2 API: `Model(out
puts=Tensor("pr..., inputs=Tensor("im...)`
```

```
Train on 6993 samples, validate on 2998 samples
Epoch 1/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6832 - recall: 0.3695 - f1_score: 0.4795 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 2/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6833 - recall: 0.3694 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 3/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6833 - recall: 0.3693 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 4/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6832 - recall: 0.3692 - f1_score: 0.4793 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 5/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6832 - recall: 0.3693 - f1_score: 0.4793 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 6/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6834 - recall: 0.3692 - f1_score: 0.4793 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 7/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6833 - recall: 0.3693 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 8/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6833 - recall: 0.3692 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 9/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6833 - recall: 0.3694 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 10/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6833 - recall: 0.3692 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 11/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6832 - recall: 0.3693 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 12/20
```

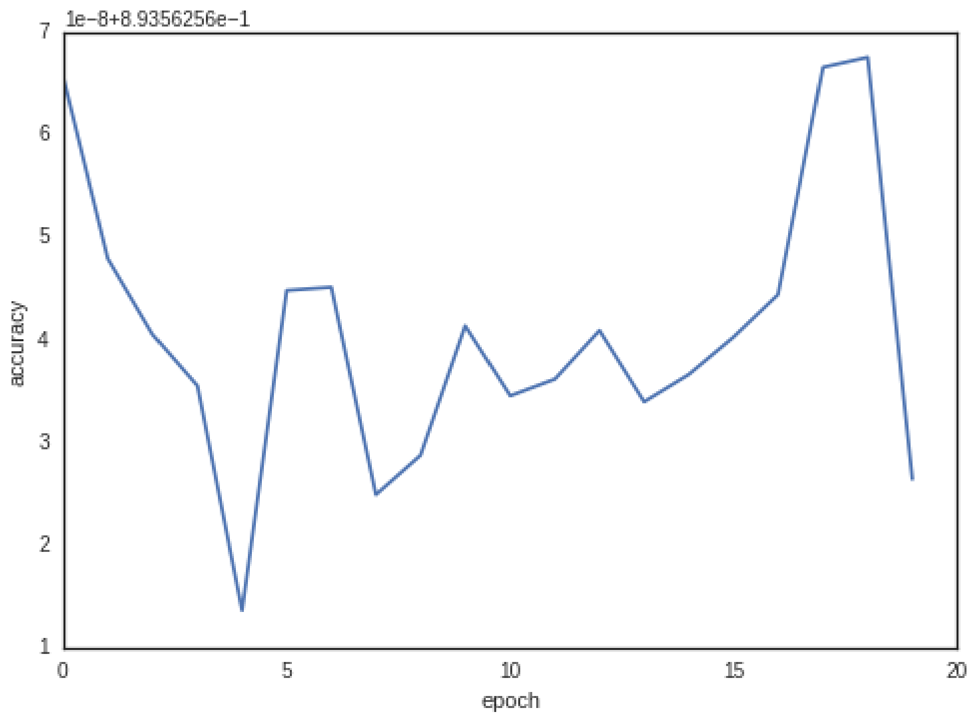
```
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6832 - recall: 0.3692 - f1_score: 0.4793 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 13/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6832 - recall: 0.3692 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 14/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6834 - recall: 0.3693 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 15/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6833 - recall: 0.3694 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 16/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6834 - recall: 0.3693 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 17/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6832 - recall: 0.3695 - f1_score: 0.4795 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 18/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6832 - recall: 0.3692 - f1_score: 0.4793 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 19/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6834 - recall: 0.3694 - f1_score: 0.4795 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
Epoch 20/20
6993/6993 [==============================] - 12s - loss: 0.2660 - acc:
 0.8936 - precision: 0.6833 - recall: 0.3693 - f1_score: 0.4794 - val_l
oss: 0.3264 - val_acc: 0.8696 - val_precision: 0.5128 - val_recall: 0.2
692 - val_f1_score: 0.3530
```

In [26]:
```
score = model4.evaluate(x_test, y_test, verbose=0)
print(model4.metrics_names)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print('Precision', score[2])
print('Recall', score[3])
print('f1 score', score[4])
```

```
['loss', 'acc', 'precision', 'recall', 'f1_score']
Test loss: 0.326375924921
Test accuracy: 0.86956204137
Precision 0.512464917287
Recall 0.271144805013
f1 score 0.353980610357
```

In [27]:
```
plt.plot(history4.history['acc'])
plt.xlabel("epoch")
plt.ylabel("accuracy")
```

Out[27]: <matplotlib.text.Text at 0x7fbb7b5d2ed0>



Tune: Time-based learning rate decay

In [9]:
```
# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# Compile model
epochs = 50
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.8
sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rate, nestero
v=False)

#Create my own model
model5 = Model(input=input, output=m)
model5.summary()

model5.compile(loss='binary_crossentropy',
               optimizer=sgd,
               metrics=['accuracy',precision,recall,f1_score])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
image_input (InputLayer)     (None, 32, 32, 3)         0         
_________________________________________________________________
vgg16 (Model)                multiple                  14714688  
_________________________________________________________________
flatten (Flatten)            (None, 512)               0         
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              2101248   
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312  
_________________________________________________________________
predictions (Dense)          (None, 19)                77843     
=================================================================
Total params: 33,675,091.0
Trainable params: 33,675,091.0
Non-trainable params: 0.0
_________________________________________________________________

/home/ubuntu/.local/lib/python2.7/site-packages/ipykernel/__main__.py:1
3: UserWarning: Update your `Model` call to the Keras 2 API: `Model(out
puts=Tensor("pr..., inputs=Tensor("im...)`
```

In [12]:
```
# Fit the model
history5 = model5.fit(x_train, y_train,
                     batch_size=512,
                     epochs=20,
                     verbose=1,
                     validation_data=(x_test, y_test))
```

```
Train on 6993 samples, validate on 2998 samples
Epoch 1/20
6993/6993 [==============================] - 12s - loss: 0.3274 - acc:
 0.8686 - precision: 0.5478 - recall: 0.0266 - f1_score: nan - val_los
s: 0.3226 - val_acc: 0.8703 - val_precision: 0.6503 - val_recall: 0.041
4 - val_f1_score: 0.0779
Epoch 2/20
6993/6993 [==============================] - 12s - loss: 0.3191 - acc:
 0.8717 - precision: 0.5763 - recall: 0.1464 - f1_score: 0.2285 - val_l
oss: 0.3142 - val_acc: 0.8725 - val_precision: 0.5542 - val_recall: 0.1
846 - val_f1_score: 0.2769
Epoch 3/20
6993/6993 [==============================] - 12s - loss: 0.3123 - acc:
 0.8732 - precision: 0.5743 - recall: 0.1795 - f1_score: 0.2711 - val_l
oss: 0.3100 - val_acc: 0.8749 - val_precision: 0.5867 - val_recall: 0.1
826 - val_f1_score: 0.2785
Epoch 4/20
6993/6993 [==============================] - 12s - loss: 0.3108 - acc:
 0.8739 - precision: 0.5784 - recall: 0.1888 - f1_score: 0.2834 - val_l
oss: 0.3110 - val_acc: 0.8733 - val_precision: 0.6049 - val_recall: 0.1
222 - val_f1_score: 0.2033
Epoch 5/20
6993/6993 [==============================] - 12s - loss: 0.3066 - acc:
 0.8755 - precision: 0.5991 - recall: 0.2006 - f1_score: 0.2966 - val_l
oss: 0.3061 - val_acc: 0.8757 - val_precision: 0.5996 - val_recall: 0.1
802 - val_f1_score: 0.2771
Epoch 6/20
6993/6993 [==============================] - 12s - loss: 0.3025 - acc:
 0.8764 - precision: 0.5942 - recall: 0.2226 - f1_score: 0.3220 - val_l
oss: 0.3058 - val_acc: 0.8757 - val_precision: 0.5737 - val_recall: 0.2
334 - val_f1_score: 0.3318
Epoch 7/20
6993/6993 [==============================] - 12s - loss: 0.3002 - acc:
 0.8781 - precision: 0.6136 - recall: 0.2255 - f1_score: 0.3282 - val_l
oss: 0.3018 - val_acc: 0.8771 - val_precision: 0.6012 - val_recall: 0.2
097 - val_f1_score: 0.3109
Epoch 8/20
6993/6993 [==============================] - 12s - loss: 0.2976 - acc:
 0.8786 - precision: 0.6101 - recall: 0.2387 - f1_score: 0.3421 - val_l
oss: 0.3026 - val_acc: 0.8764 - val_precision: 0.5955 - val_recall: 0.2
044 - val_f1_score: 0.3043
Epoch 9/20
6993/6993 [==============================] - 12s - loss: 0.2957 - acc:
 0.8795 - precision: 0.6171 - recall: 0.2474 - f1_score: 0.3514 - val_l
oss: 0.2994 - val_acc: 0.8783 - val_precision: 0.6021 - val_recall: 0.2
354 - val_f1_score: 0.3384
Epoch 10/20
6993/6993 [==============================] - 12s - loss: 0.2925 - acc:
 0.8805 - precision: 0.6250 - recall: 0.2529 - f1_score: 0.3594 - val_l
oss: 0.3053 - val_acc: 0.8756 - val_precision: 0.5746 - val_recall: 0.2
289 - val_f1_score: 0.3273
Epoch 11/20
6993/6993 [==============================] - 12s - loss: 0.2929 - acc:
 0.8802 - precision: 0.6190 - recall: 0.2559 - f1_score: 0.3611 - val_l
oss: 0.2993 - val_acc: 0.8782 - val_precision: 0.5970 - val_recall: 0.2
438 - val_f1_score: 0.3462
Epoch 12/20
```

```
6993/6993 [==============================] - 12s - loss: 0.2921 - acc:
 0.8808 - precision: 0.6247 - recall: 0.2607 - f1_score: 0.3663 - val_l
oss: 0.3034 - val_acc: 0.8764 - val_precision: 0.5932 - val_recall: 0.2
070 - val_f1_score: 0.3069
Epoch 13/20
6993/6993 [==============================] - 12s - loss: 0.2891 - acc:
 0.8813 - precision: 0.6272 - recall: 0.2590 - f1_score: 0.3658 - val_l
oss: 0.2978 - val_acc: 0.8788 - val_precision: 0.5922 - val_recall: 0.2
698 - val_f1_score: 0.3707
Epoch 14/20
6993/6993 [==============================] - 12s - loss: 0.2849 - acc:
 0.8831 - precision: 0.6320 - recall: 0.2849 - f1_score: 0.3923 - val_l
oss: 0.2963 - val_acc: 0.8791 - val_precision: 0.6098 - val_recall: 0.2
385 - val_f1_score: 0.3428
Epoch 15/20
6993/6993 [==============================] - 12s - loss: 0.2837 - acc:
 0.8840 - precision: 0.6415 - recall: 0.2872 - f1_score: 0.3956 - val_l
oss: 0.2969 - val_acc: 0.8787 - val_precision: 0.5913 - val_recall: 0.2
695 - val_f1_score: 0.3703
Epoch 16/20
6993/6993 [==============================] - 12s - loss: 0.2795 - acc:
 0.8856 - precision: 0.6498 - recall: 0.3009 - f1_score: 0.4110 - val_l
oss: 0.3020 - val_acc: 0.8780 - val_precision: 0.5770 - val_recall: 0.2
915 - val_f1_score: 0.3872
Epoch 17/20
6993/6993 [==============================] - 12s - loss: 0.2837 - acc:
 0.8836 - precision: 0.6388 - recall: 0.2862 - f1_score: 0.3943 - val_l
oss: 0.2994 - val_acc: 0.8775 - val_precision: 0.5840 - val_recall: 0.2
572 - val_f1_score: 0.3571
Epoch 18/20
6993/6993 [==============================] - 12s - loss: 0.2761 - acc:
 0.8864 - precision: 0.6547 - recall: 0.3065 - f1_score: 0.4166 - val_l
oss: 0.2972 - val_acc: 0.8794 - val_precision: 0.5899 - val_recall: 0.2
903 - val_f1_score: 0.3890
Epoch 19/20
6993/6993 [==============================] - 12s - loss: 0.2760 - acc:
 0.8865 - precision: 0.6484 - recall: 0.3193 - f1_score: 0.4271 - val_l
oss: 0.2992 - val_acc: 0.8774 - val_precision: 0.5905 - val_recall: 0.2
381 - val_f1_score: 0.3392
Epoch 20/20
6993/6993 [==============================] - 12s - loss: 0.2720 - acc:
 0.8882 - precision: 0.6623 - recall: 0.3236 - f1_score: 0.4339 - val_l
oss: 0.2996 - val_acc: 0.8788 - val_precision: 0.5856 - val_recall: 0.2
867 - val_f1_score: 0.3849
```

In [ ]:

```
score = model5.evaluate(x_test, y_test, verbose=0)
print(model5.metrics_names)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print('Precision', score[2])
print('Recall', score[3])
print('f1 score', score[4])
plt.plot(history5.history['acc'])
plt.xlabel("epoch")
plt.ylabel("accuracy")
```

Tune: batch size and epochs - takes very long

In [ ]:
```
# # tune
# from sklearn.grid_search import GridSearchCV
# from keras.models import Sequential
# from keras.layers import Dense
# from keras.wrappers.scikit_learn import KerasClassifier

# # Function to create model, required for KerasClassifier
# def create_model():
#     # create model
#     model = Model(input=input, output=m)
#     model.compile(loss='binary_crossentropy',
#                   optimizer='adam',
#                   metrics=['accuracy',precision,recall,f1_score])
#     return model

# # fix random seed for reproducibility
# seed = 7
# np.random.seed(seed)

# # create model
# model = KerasClassifier(build_fn=create_model, verbose=0)

# # define the grid search parameters
# batch_size = [256, 512, 1024] #, 60, 80, 100]
# epochs = [10, 20, 50]

# param_grid = dict(batch_size=batch_size, epochs=epochs)
# grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
# grid_result = grid.fit(x_train,y_train)

# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_, grid_result.best
_params_))
# means = grid_result.cv_results_['mean_test_score']
# stds = grid_result.cv_results_['std_test_score']
# params = grid_result.cv_results_['params']
# for mean, stdev, param in zip(means, stds, params):
#     print("%f (%f) with: %r" % (mean, stdev, param))
```

Milestone 4 Exploration

April 26, 2017

0.0.1 Milestone 4: Deep learning, due Wednesday, April 26, 2017

For this milestone you will (finally) use deep learning to predict movie genres. You will train one small network from scratch on the posters only, and compare this one to a pre-trained network that you fine tune. Here is a description of how to use pretrained models in Keras.

You can try different architectures, initializations, parameter settings, optimization methods, etc. Be adventurous and explore deep learning! It can be fun to combine the features learned by the deep learning model with a SVM, or incorporate meta data into your deep learning model.

Note: Be mindful of the longer training times for deep models. Not only for training time, but also for the parameter tuning efforts. You need time to develop a feel for the different parameters and which settings work, which normalization you want to use, which model architecture you choose, etc.

It is great that we have GPUs via AWS to speed up the actual computation time, but you need to be mindful of your AWS credits. The GPU instances are not cheap and can accumulate costs rather quickly. Think about your model first and do some quick dry runs with a larger learning rate or large batch size on your local machine.

The notebook to submit this week should at least include:

- Complete description of the deep network you trained from scratch, including parameter settings, performance, features learned, etc.
- Complete description of the pre-trained network that you fine tuned, including parameter settings, performance, features learned, etc.
- Discussion of the results, how much improvement you gained with fine tuning, etc.
- Discussion of at least one additional exploratory idea you pursued

```
In [1]: import json
        import urllib
        import cStringIO
        from PIL import Image
        from imdb import IMDb
        import pandas as pd
        import numpy as np
        from pandas import Series, DataFrame
        %matplotlib inline
        import matplotlib.pyplot as plt
        import seaborn as sns
        import time
        import ast
```

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.cross_validation import KFold
import difflib
```

```
/Users/Xincheng/anaconda/lib/python2.7/site-packages/sklearn/cross_validation.py:44
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
# part 3 - top 10 most popular movies of 2016 from TMDb and their genre
top_2016_1 = urllib.urlopen("https://api.themoviedb.org/3/discover/movie?ap
top_2016_1_json = json.loads(top_2016_1.read())

# get genre list
genre_list = urllib.urlopen("https://api.themoviedb.org/3/genre/movie/list?

genre_list_json = json.loads(genre_list.read())

genre_lst = {}
for i in genre_list_json['genres']:
    genre_lst[i['id']] = str(i['name'])

# top most popular movies of 2016
top_2016_1 = urllib.urlopen("https://api.themoviedb.org/3/discover/movie?ap
top_2016_1_json = json.loads(top_2016_1.read())


for i in top_2016_1_json['results']:
    print i['title'], [genre_lst[j] for j in i['genre_ids']]
```

```
Sing ['Animation', 'Comedy', 'Drama', 'Family', 'Music']
Split ['Horror', 'Thriller']
Fantastic Beasts and Where to Find Them ['Action', 'Adventure', 'Fantasy']
Rogue One: A Star Wars Story ['Action', 'Drama', 'Science Fiction', 'War']
Deadpool ['Action', 'Adventure', 'Comedy', 'Romance']
Arrival ['Thriller', 'Drama', 'Science Fiction', 'Mystery']
Boyka: Undisputed IV ['Action']
La La Land ['Comedy', 'Drama', 'Music', 'Romance']
Doctor Strange ['Action', 'Adventure', 'Fantasy', 'Science Fiction']
Tomorrow Everything Starts ['Drama', 'Comedy']
Captain America: Civil War ['Adventure', 'Action', 'Science Fiction']
Finding Dory ['Adventure', 'Animation', 'Comedy', 'Family']
Collateral Beauty ['Drama', 'Romance']
X-Men: Apocalypse ['Action', 'Adventure', 'Fantasy', 'Science Fiction']
Passengers ['Adventure', 'Drama', 'Romance', 'Science Fiction']
```

```
Why Him? ['Comedy']
Underworld: Blood Wars ['Action', 'Horror']
Suicide Squad ['Action', 'Crime', 'Fantasy', 'Science Fiction']
Hacksaw Ridge ['Drama', 'History', 'War']
Assassin's Creed ['Action', 'Adventure', 'Fantasy', 'Science Fiction']
```

```
In [3]: import ast

        movie_2000_df = pd.read_csv('tmdb_metadata.csv')
        movie_2000_df = movie_2000_df.drop('Unnamed: 0', axis=1)

        movie_2000_df = movie_2000_df.dropna()

        labels = []
        for i in movie_2000_df.genre_ids:
            label_matrix = np.zeros(len(genre_lst.keys()), dtype=int)
            for j in ast.literal_eval(i):
                if j in genre_lst.keys():
                    label_matrix[genre_lst.keys().index(j)] = 1
            labels.append(label_matrix)
        movie_2000_df['labels'] = labels

        # convert dates
        import datetime
        def to_integer(dt_time):
            return 10000*dt_time.year + 100*dt_time.month + dt_time.day

        int_dates =[]

        for i in movie_2000_df.release_date:
            f = i.split('-')
            a = datetime.date(int(f[0]), int(f[1]), int(f[2]))
            int_dates.append(to_integer(a))

        movie_2000_df['int_dates'] = int_dates

In [4]: data = movie_2000_df.drop(['genre_ids', 'movie_id', 'poster_path', 'overvie

In [5]: words = pd.read_csv('genre_words_pca.csv').drop('Unnamed: 0', axis = 1)

In [6]: x = pd.concat([data[['popularity', 'vote_average', 'vote_count', 'int_dates
        y = data['labels']
        y = np.asarray(y.tolist())
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3)

In [7]: from __future__ import print_function

        import keras
```

```python
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD

import matplotlib
sns.set_style('white')
```

```
Using TensorFlow backend.
```

In [8]:
```python
# smaller batch size means noisier gradient, but more updates per epoch
batch_size = 512
# this is fixed, we have 10 digits in our data set
num_classes = 10
# number of iterations over the complete training data
epochs = 100

# the data, shuffled and split between train and test sets
# (x_train, y_train), (x_test, y_test) = mnist.load_data()

# x_train = x_train.reshape(60000, 784)
# x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# normalize image values to [0,1]
# interestingly the keras example code does not center the data
# x_train /= 255
# x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
3431 train samples
1471 test samples
```

In [ ]:

In [9]:
```python
# create an empty network model
model = Sequential()
# add an input layer
model.add(Dense(64, activation='relu', input_shape=(304,)))
# this is our hidden layer
model.add(Dense(64, activation='relu'))
# and an output layer
# note that the 10 is the number of classes we have
# the classes are mutually exclusive so softmax is a good choice
model.add(Dense(19, activation='sigmoid'))
```

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 64)                19520     
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_3 (Dense)              (None, 19)                1235      
=================================================================
Total params: 24,915
Trainable params: 24,915
Non-trainable params: 0
_________________________________________________________________
```

In [10]:
```
from keras import metrics
import keras.backend as K

def precision(y_true, y_pred):
    """Precision metric.
    Only computes a batch-wise average of precision.
    Computes the precision, a metric for multi-label classification of
    how many selected items are relevant.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def recall(y_true, y_pred):
    """Recall metric.
    Only computes a batch-wise average of recall.
    Computes the recall, a metric for multi-label classification of
    how many relevant items are selected.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def f1_score(y_true, y_pred):

    # Count positive samples.
    c1 = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
```

```
            c2 = K.sum(K.round(K.clip(y_pred, 0, 1)))
            c3 = K.sum(K.round(K.clip(y_true, 0, 1)))

            # If there are no true samples, fix the F1 score at 0.
            if c3 == 0:
                return 0

            # How many selected items are relevant?
            precision = c1 / c2

            # How many relevant items are selected?
            recall = c1 / c3

            # Calculate f1_score
            f1_score = 2 * (precision * recall) / (precision + recall)
            return f1_score
```

In [11]:
```
sgd = SGD(lr=0.01, momentum=0.9)
model.compile(loss='binary_crossentropy',
              optimizer=sgd,
              metrics=['accuracy', precision, recall, f1_score])
```

In [12]:
```
# this is not the actual training
# in addition to the training data we provide validation data
# this data is used to calculate the performance of the model over all the
# this is useful to determine when training should stop
# in our case we just use it to monitor the evolution of the model over th
# if we use the validation data to determine when to stop the training or
# should not use the test data, but a separate validation set.
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))

# once training is complete, let's see how well we have done
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print('Test precision:', score[2])
print('Test recall:', score[3])
print('Test f1_score:', score[4])
```

```
Train on 3431 samples, validate on 1471 samples
Epoch 1/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 2/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
```

```
Epoch 3/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 4/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 5/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 6/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 7/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 8/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 9/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 10/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 11/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 12/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 13/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 14/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 15/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 16/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 17/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 18/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 19/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 20/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 21/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 22/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 23/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 24/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 25/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 26/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
```

```
Epoch 27/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 28/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 29/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 30/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 31/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 32/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 33/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 34/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 35/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 36/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 37/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 38/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 39/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 40/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 41/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 42/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 43/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 44/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 45/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 46/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 47/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 48/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 49/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 50/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
```

```
Epoch 51/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 52/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 53/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 54/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 55/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 56/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 57/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 58/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 59/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 60/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 61/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 62/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 63/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 64/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 65/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 66/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 67/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 68/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 69/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 70/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 71/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 72/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 73/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 74/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
```

```
Epoch 75/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 76/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 77/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 78/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 79/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 80/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 81/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 82/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 83/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 84/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 85/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 86/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 87/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 88/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 89/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 90/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 91/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 92/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 93/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 94/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 95/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 96/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 97/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 98/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
```

```
Epoch 99/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Epoch 100/100
3431/3431 [==============================] - 0s - loss: 10.4991 - acc: 0.3420 - pre
Test loss: 10.5374518893
Test accuracy: 0.339582815635
Test precision: 0.108194321323
Test recall: 0.594176930647
Test f1_score: 0.182891872203
```

```
In [ ]:
```