```
import os
import pandas as pd
import numpy as np
import librosa
import random
import time
import pickle

from librosa.display import specshow, waveplot

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer

import IPython.display as ipd

import matplotlib
import matplotlib.pyplot as plt

np.random.seed(42)

%matplotlib inline
```

```
from keras.models import Sequential
from keras.layers import Dense, MaxPooling2D, Conv2D, Flatten, Dropout, Input, BatchNor
from keras.models import Model, load_model
from keras.callbacks import Callback, EarlyStopping
from keras.metrics import top_k_categorical_accuracy
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive/My Drive/Colab Notebooks
```

Saved successfully!

```
https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /gdrive
/gdrive/My Drive/Colab Notebooks
```

Making Sense of Genres

The first step is seeing how many tracks per genre we have in our dataset, and simplifiying the output of our neural net to include only the top 5.

Previously, I attempted outputing its confidence for all 161 total genres, and only reached 20% accuracy. You can see that journey in my other notebook, training2_svc.

```
1 genres = pd.read_csv("genres.csv", index_col=0)
2 genres
```

genre_id	#tracks	parent	title	top_level
1	8693	38	Avant-Garde	38
2	5271	0	International	2
3	1752	0	Blues	3
4	4126	0	Jazz	4
5	4106	0	Classical	5
...	...	...	...	...
1032	60	102	Turkish	2
1060	30	46	Tango	2
1156	26	130	Fado	2
1193	72	763	Christmas	38
1235	14938	0	Instrumental	1235

163 rows × 4 columns

```
1 genres = genres.sort_values(by='#tracks', ascending=False)
2 genres.head(5)
```

genre_id	#tracks	parent	title	top_level
[illegible]	[illegible]	[illegible]	xperimental	38
[illegible]	[illegible]	[illegible]	Electronic	15
12	32923	0	Rock	12
1235	14938	0	Instrumental	1235
10	13845	0	Pop	10

Saved successfully!

Adding Echonest Attributes

Whoop, our top genres are: Experimental, Electronic, Rock, Instrumental, and Pop.

Next, since I want to use this classifier for my senior design project as well, I want to incorporate attributes from *echonest*.

Echnoest, now Spotify, includes numerical values for tracks for traits like dancebility, energy, speechiness, etc-- these will be very valuable when teaching a stick figure to dance. (my senior design)

```
echonest = pd.read_csv("echonest.csv", header=[0, 2], skipinitialspace=True, index_col=(
echonest.head()
```

	echonest						
	acousticness	danceability	energy	instrumentalness	liveness	speechiness	t
track_id							
2	0.416675	0.675894	0.634476	0.010628	0.177647	0.159310	1
3	0.374408	0.528643	0.817461	0.001851	0.105880	0.461818	1
5	0.043567	0.745566	0.701470	0.000697	0.373143	0.124595	1
10	0.951670	0.658179	0.924525	0.965427	0.115474	0.032985	1
134	0.452217	0.513238	0.560410	0.019443	0.096567	0.525519	1

5 rows × 249 columns

```
for col in echonest:
    if col[0] == "metadata":
        echonest.drop(col, axis=1, inplace=True)
    elif col[0] == "ranks":
        echonest.drop(col, axis=1, inplace=True)
    elif col[0] == "social features":
                         s=1, inplace=True)
```

Saved successfully!

```
echonest.columns = echonest.columns.droplevel(0)
```

```
echonest_sub = echonest[['acousticness', 'danceability', 'energy', 'instrumentalness',
echonest_sub.head()
```

	acousticness	danceability	energy	instrumentalness	liveness	speechiness	
track_id							
2	0.416675	0.675894	0.634476	0.010628	0.177647	0.159310	1
3	0.374408	0.528643	0.817461	0.001851	0.105880	0.461818	1
5	0.043567	0.745566	0.701470	0.000697	0.373143	0.124595	1
10	0.951670	0.658179	0.924525	0.965427	0.115474	0.032985	1
134	0.452217	0.513238	0.560410	0.019443	0.096567	0.525519	1

Adding Track Data

Now let's incorporating part of the track dataset.

```
tracks = pd.read_csv("tracks.csv", header=[0, 1], skipinitialspace=True, index_col=0)
tracks.columns = tracks.columns.droplevel(0)
tracks.head()
```

	comments	date_created	date_released	engineer	favorites	id	information	li
track_id								
2	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1		
3	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1		
[illegible]	[illegible]	2008-11-26 [illegible]44:45	2009-01-05 00:00:00	NaN	4	1		
10	0	2008-11-26 01:45:08	2008-02-06 00:00:00	NaN	4	6	NaN	
20	0	2008-11-26 01:45:05	2009-01-06 00:00:00	NaN	2	4	"spiritual songs" from Nicky Cook	

Saved successfully!

```
tracks_sub = tracks[['listens', 'name', 'duration', 'genre_top', 'genres', 'title']]
tracks_sub.head()
```

track_id	listens	listens	name	duration	genre_top	genres	title	title
2	6073	1293	AWOL	168	Hip-Hop	[21]	AWOL - A Way Of Life	Food
3	6073	514	AWOL	237	Hip-Hop	[21]	AWOL - A Way Of Life	Electric Ave
5	6073	1151	AWOL	206	Hip-Hop	[21]	AWOL - A Way Of Life	This World
10	47632	50135	Kurt Vile	161	Pop	[10]	Constant Hitmaker	Freeway
20	2710	361	Nicky Cook	311	NaN	[76, 103]	Niris	Spiritual Level

```
1 tracks_sub.columns = ['listens_album', 'listens_track', 'name', 'duration', 'genre_top'
```

```
1 tracks_sub.head()
```

track_id	listens_album	listens_track	name	duration	genre_top	genres	title_album
2	6073	1293	AWOL	168	Hip-Hop	[21]	AWOL - A Way Of Life
3	6073	514	AWOL	237	Hip-Hop	[21]	AWOL - A Way Of Life
5	6073	1151	AWOL	206	Hip-Hop	[21]	AWOL - A Way Of Life
10	47632	50135	Kurt Vile	161	Pop	[10]	Constant Hitmaker
20	2710	361	Nicky Cook	311	NaN	[76, 103]	Niris

Saved successfully!

Merging Tracks, Echonest, and Genres

oh boy

```
1 tracks_echo = pd.merge(tracks_sub, echonest_sub, how="inner", on="track_id")
```

```
1 tracks_echo.head()
```

track_id	listens_album	listens_track	name	duration	genre_top	genres	title_album
2	6073	1293	AWOL	168	Hip-Hop	[21]	AWOL - A Way Of Life
3	6073	514	AWOL	237	Hip-Hop	[21]	AWOL - A Way Of Life
5	6073	1151	AWOL	206	Hip-Hop	[21]	AWOL - A Way Of Life
10	47632	50135	Kurt Vile	161	Pop	[10]	Constant Hitmaker
134	6073	943	AWOL	207	Hip-Hop	[21]	AWOL - A Way Of Life

```
tracks_echo_genres = pd.merge(tracks_echo, genres, how="left", left_on="genre_top", rig
```

```
tracks_echo_genres.head()
```

track_id	listens_album	listens_track	name	duration	genre_top	genres	title_album
2	6073	1293	AWOL	168	Hip-Hop	[21]	AWOL - A Way Of Life
3	6073	514	AWOL	237	Hip-Hop	[21]	AWOL - A Way Of Life
5	6073	1151	AWOL	206	Hip-Hop	[21]	AWOL - A Way Of Life
10	47632	50135	Kurt Vile	161	Pop	[10]	Constant Hitmaker
134	6073	943	AWOL	207	Hip-Hop	[21]	AWOL - A Way Of Life

Saved successfully!

```
tracks_echo_genres.to_pickle("./tracks_echo_genres.pkl")
```

Adding Features

This is the final piece left to merge into our monster dataset. There are a lot of attributes here-- 518-- so I want to do some dimensionality reduction here. I will be using PCA post-merge.

```
features = pd.read_csv("features.csv", header=[0, 1, 2], skipinitialspace=True, index_c
features.head()
```

feature	chroma_cens								
statistics	kurtosis								
number	01	02	03	04	05	06	07	08	09
track_id									
2	7.180653	5.230309	0.249321	1.347620	1.482478	0.531371	1.481593	2.691455	0.8
3	1.888963	0.760539	0.345297	2.295201	1.654031	0.067592	1.366848	1.054094	0.1
5	0.527563	-0.077654	-0.279610	0.685883	1.937570	0.880839	-0.923192	-0.927232	0.6
10	3.702245	-0.291193	2.196742	-0.234449	1.367364	0.998411	1.770694	1.604566	0.5
20	-0.193837	-0.198527	0.201546	0.258556	0.775204	0.084794	-0.289294	-0.816410	0.0

5 rows × 518 columns

```
# MERGING!!!
monster = pd.merge(tracks_echo_genres, features, how="inner", on="track_id")
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/reshape/merge.py:617: UserWarning: me
  warnings.warn(msg, UserWarning)
```

```
monster.head()
```

	listens_album	listens_track	name	duration	genre_top	genres	title_album	t
track_id								
2	6073	1293	AWOL	168	Hip-Hop	[21]	AWOL - A Way Of Life	
3	6073	514	AWOL	237	Hip-Hop	[21]	AWOL - A Way Of Life	
[illegible]	[illegible]	1151	AWOL	206	Hip-Hop	[21]	AWOL - A Way Of Life	
10	47632	50135	Kurt Vile	161	Pop	[10]	Constant Hitmaker	
134	6073	943	AWOL	207	Hip-Hop	[21]	AWOL - A Way Of Life	

Saved successfully!

5 rows × 539 columns

```
monster.to_pickle("./monster.pkl")
```

PCA Shenanigans for Dimensionality Reduction

```
from sklearn.preprocessing import StandardScaler
feats = monster.columns

# Separating out the features
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
x = monster.select_dtypes(include=numerics).values

# x = monster.loc[:, feats].values
# Separating out the target
y = monster.loc[:,['genre_top']].values

# Standardizing the features
X = StandardScaler().fit_transform(x)
```

```
X.shape
```

```
(13129, 532)
```

```
# no nan vals allowed!!!
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(X[:,1:532])
X[:,1:532] = imputer.transform(X[:,1:532])
```

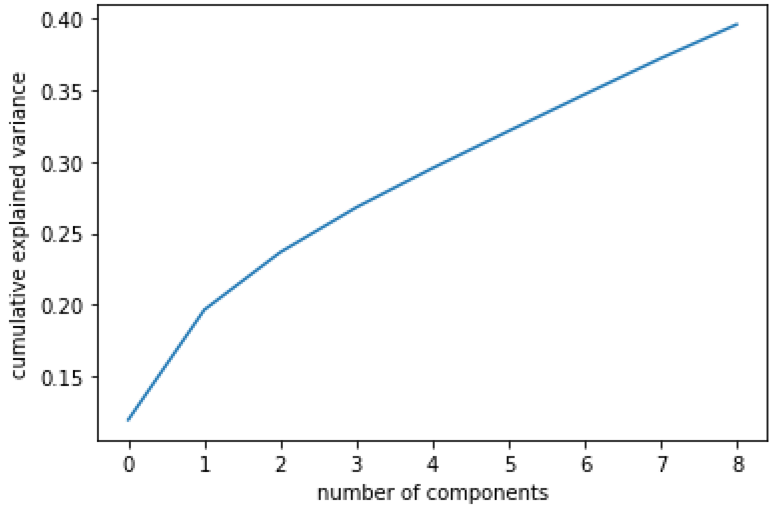
```
from sklearn.decomposition import PCA

pca = PCA(n_components=9) # for number of big attributes?
principalComponents = pca.fit_transform(X)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2', 'principal
```

```
# SCREE PLOT
print(pca.explained_variance_ratio_)
print(np.cumsum(pca.explained_variance_ratio_))
```

Saved successfully!

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```

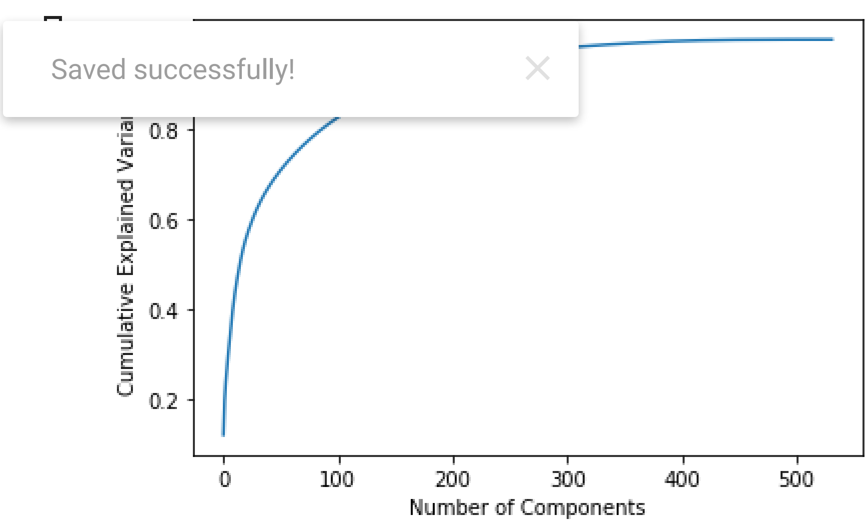
```
[0.11959576 0.07689131 0.04026169 0.03136196 0.02734066 0.0259367
 0.02560827 0.02520922 0.02367956]
[0.11959576 0.19648707 0.23674876 0.26811072 0.29545139 0.32138809
 0.34699636 0.37220558 0.39588514]
```



Ok!!!!!!!! So, this shows that with 9 components, it represents 40% of variance in the data.

Let's make a generic scree plot to see how many components might make more sense:

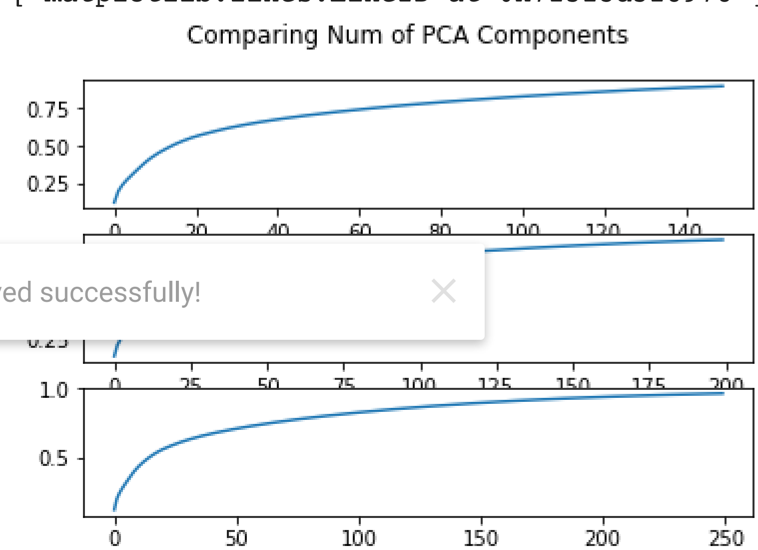
```
# SCREE PLOT

#Explained variance
pca = PCA().fit(X)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')

plt.plot(250, np.cumsum(pca.explained_variance_ratio_)[250], "ob")
plt.plot(200, np.cumsum(pca.explained_variance_ratio_)[200], "or")
plt.plot(150, np.cumsum(pca.explained_variance_ratio_)[150], "og")

plt.show()
```



This plot shows us that around 150-250 components might be a better number to try. Let's try it:

```
#Explained variance

fig, axs = plt.subplots(3)
fig.suptitle('Comparing Num of PCA Components')
# fig.xlabel('Number of Components')
# fig.ylabel('Cumulative Explained Variance')

# PLOT 1: 150 COMPONENTS
pca = PCA(n_components=150)
principalComponents = pca.fit_transform(X)
col_names = [("col_" + str(i)) for i in range(150)]
principalDf = pd.DataFrame(data = principalComponents
             , columns = col_names)
axs[0].plot(np.cumsum(pca.explained_variance_ratio_))

# PLOT 2: 200 COMPONENTS
pca = PCA(n_components=200)
principalComponents = pca.fit_transform(X)
col_names = [("col_" + str(i)) for i in range(200)]
principalDf = pd.DataFrame(data = principalComponents
             , columns = col_names)
axs[1].plot(np.cumsum(pca.explained_variance_ratio_))

# PLOT 3: 250 COMPONENTS
pca = PCA(n_components=250)
principalComponents = pca.fit_transform(X)
col_names = [("col_" + str(i)) for i in range(250)]
principalDf = pd.DataFrame(data = principalComponents
             , columns = col_names)
axs[2].plot(np.cumsum(pca.explained_variance_ratio_))
```

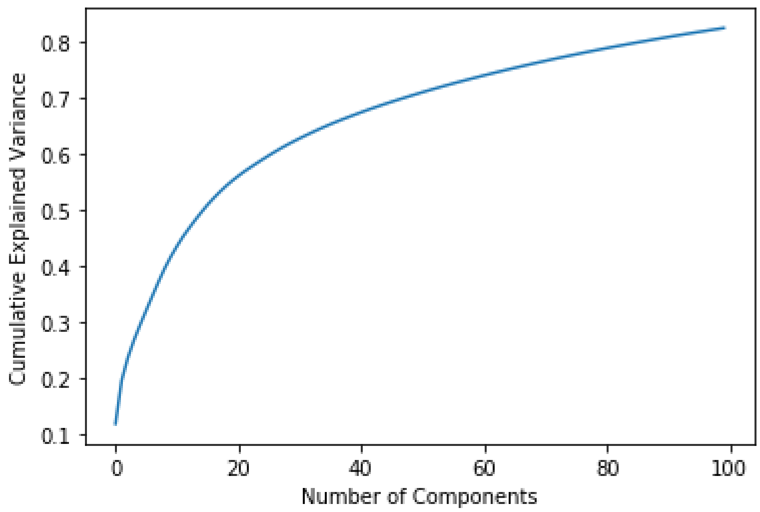
```
[<matplotlib.lines.Line2D at 0x7f5f3d5f8978>]
```



So.... they are looking basically the same! Let's see if we can go smaller than 150 components so that we can be faster while training

```
#Explained variance
pca = PCA(n_components=100) # for number of big attributes?
principalComponents = pca.fit_transform(X)
```

```
col_names = [("col_" + str(i)) for i in range(100)]
principalDf = pd.DataFrame(data = principalComponents
             , columns = col_names)

plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()

print(100, np.cumsum(pca.explained_variance_ratio_)[99])
```




```
100 0.8235167577314023
```

```
def compare_n_comp(n):
  pca = PCA(n_components=n)
  principalComponents = pca.fit_transform(X)
  print(n, str((np.cumsum(pca.explained_variance_ratio_)[n-1])*100), str('%'))

compare_n_comp(150)
compare_n_comp(175)
compare_n_comp(200)
```

```
150 89.51656816113069 %
175 91.96887994360743 %
200 93.89739765751429 %
```

Saved successfully!

ts. That'll be a 62% reduction!

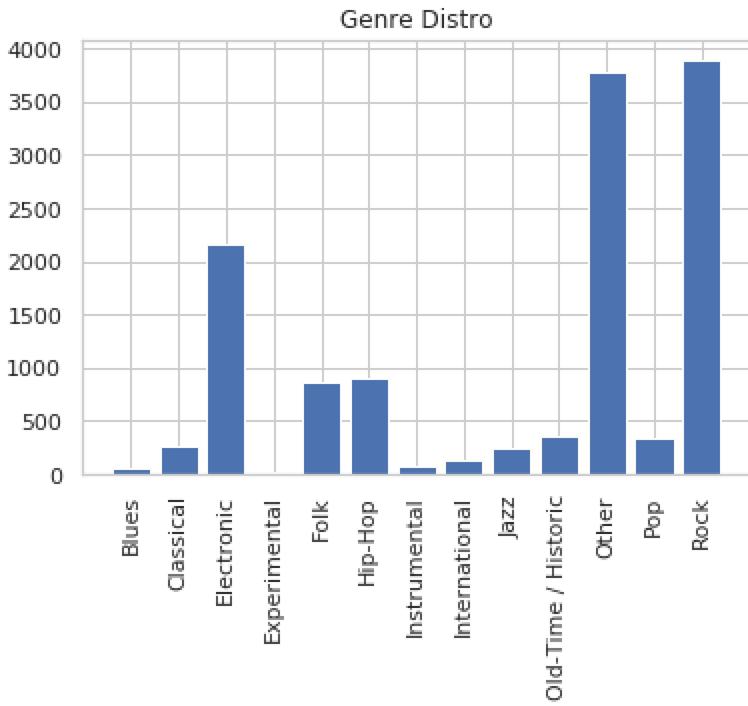
```
#Explained variance
pca = PCA(n_components=200)
principalComponents = pca.fit_transform(X)
col_names = [("col_" + str(i)) for i in range(200)]
principalDf = pd.DataFrame(data = principalComponents
             , columns = col_names)
```

```
print(principalDf.shape)
print(y.shape)
```

I want to visualize the data I'm going to train with in some way... let's try taking the mean of all the numerical values in our PCA components and map them to their respective genre.

```
y_df = pd.DataFrame(y)
y_df = y_df.replace(np.nan, 'Other', regex=True)
y_map = dict(zip(*np.unique(y_df, return_counts=True)))
plt.bar(y_map.keys(), y_map.values())
plt.xticks(rotation='vertical')
plt.title("Genre Distro")

print(y_map)
```

```
{'Blues': 66, 'Classical': 265, 'Electronic': 2170, 'Experimental': 17, 'Folk': 874, 'Hi
```



Time to Train!

Saved successfully!

```
mport train_test_split
principalDf = principalDf.replace(np.nan, 0, regex=True)

data = principalDf[:300]
labels = y_df[:300]

data_train, data_test, label_train, label_test = train_test_split(data, labels, test_si
```

```
scaler = StandardScaler()

# Fit on training set only.
scaler.fit(data_train)

# Apply transform to both the training set and the test set.
data_train = scaler.transform(data_train)
```

```
data_train = scaler.transform(data_train)
data_test = scaler.transform(data_test)
```

```
pca = PCA(.93) # this is about 200 components as we saw earlier
pca.fit(data_train)
```

```
PCA(copy=True, iterated_power='auto', n_components=0.93, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

Bad

↳ *4 cells hidden*

So there's a problem here: I'm trying to use string values as labels for my dataset, which is not allowed. I decided to try this route (without thinking it all the way through) because the numerical genre values, genre_ids, were stored as a string of a list of a list, and I wanted to try to avoid dealing with that mess. Looks like it's unavoidable so let's...

Deal with the genre Label Mess

```
with open("genre_labels.pkl", "rb") as handle:
  genre_labels = pickle.load(handle)

genrelabels = pd.DataFrame.from_dict(genre_labels)
genrelabels.head()
```

	genre_id	genre_title
0	1	Avant-Garde
1	2	International
2	3	Blues
3	4	Jazz

Saved successfully!

```
label_train2 = pd.merge(label_train, genrelabels, how="left", left_on=0, right_on="genre
```

```
label_train2.head()
```

	0	genre_id	genre_title
0	Other	NaN	NaN
1	Other	NaN	NaN
2	Rock	12.0	Rock
3	Rock	12.0	Rock
4	Other	NaN	NaN

```
abel_test2 = pd.merge(label_test, genrelabels, how="left", left_on=0, right_on="genre_ti
abel_test2.head()
```

	0	genre_id	genre_title
0	Other	NaN	NaN
1	Other	NaN	NaN
2	Rock	12.0	Rock
3	Other	NaN	NaN
4	Other	NaN	NaN

```
label_test = label_test2["genre_id"]
label_train = label_train2["genre_id"]
label_test = label_test.replace(np.nan, 0, regex=True)
label_train = label_train.replace(np.nan, 0, regex=True)

label_test, label_train
```

```
(0        0.0
 1        0.0
 2       12.0
 3        0.0
 4        0.0
```

Saved successfully!

```
 4330    12.0
 4331    12.0
 4332     4.0
 Name: genre_id, Length: 4333, dtype: float64, 0       12.0
 1       17.0
 2       12.0
 3       17.0
 4       17.0
         ...
 196     12.0
 197     12.0
 198     12.0
 199     21.0
 200     12.0
 Name: genre_id, Length: 201, dtype: float64)
```

OK! let's try to train again..

```
one_hot_train_labels = to_categorical(label_train.values)
one_hot_test_labels = to_categorical(label_test.values)

one_hot_train_labels.shape, one_hot_train_labels.shape
```

```
((201, 22), (201, 22))
```

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(200, )))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='tanh'))

model.add(Dense(161, activation='softmax'))

output = 164
model.add(Dense(output, activation='sigmoid')) # all genres

model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy', metrics=['ac
model.summary()
```

```
Model: "sequential_53"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_253 (Dense)            (None, 32)                6432
_________________________________________________________________
dense_254 (Dense)            (None, 64)                2112
_________________________________________________________________
dense_255 (Dense)            (None, 64)                4160
_________________________________________________________________
dense_256 (Dense)            (None, 32)                2080
_________________________________________________________________
                             (None, 161)               5313
_________________________________________________________________
dense_258 (Dense)            (None, 164)               26568
=================================================================
Total params: 46,665
Trainable params: 46,665
Non-trainable params: 0
_________________________________________________________________
```

Saved successfully!

```
model.fit(data_train, label_train, epochs=35, batch_size=512, validation_split=0.1)
```

```
Epoch 7/35
180/180 [==============================] - 0s 51us/step - loss: 4.5861 - acc: 0.7611 - v
Epoch 8/35
180/180 [==============================] - 0s 47us/step - loss: 4.5842 - acc: 0.7611 - v
Epoch 9/35
180/180 [==============================] - 0s 47us/step - loss: 4.5823 - acc: 0.7611 - v
Epoch 10/35
180/180 [==============================] - 0s 63us/step - loss: 4.5804 - acc: 0.7611 - v
Epoch 11/35
180/180 [==============================] - 0s 59us/step - loss: 4.5786 - acc: 0.7611 - v
Epoch 12/35
180/180 [==============================] - 0s 55us/step - loss: 4.5767 - acc: 0.7611 - v
Epoch 13/35
180/180 [==============================] - 0s 54us/step - loss: 4.5748 - acc: 0.7611 - v
Epoch 14/35
180/180 [==============================] - 0s 54us/step - loss: 4.5730 - acc: 0.7611 - v
Epoch 15/35
180/180 [==============================] - 0s 55us/step - loss: 4.5711 - acc: 0.7611 - v
Epoch 16/35
180/180 [==============================] - 0s 58us/step - loss: 4.5692 - acc: 0.7611 - v
Epoch 17/35
180/180 [==============================] - 0s 54us/step - loss: 4.5674 - acc: 0.7611 - v
Epoch 18/35
180/180 [==============================] - 0s 72us/step - loss: 4.5655 - acc: 0.7611 - v
Epoch 19/35
180/180 [==============================] - 0s 112us/step - loss: 4.5636 - acc: 0.7611 -
Epoch 20/35
180/180 [==============================] - 0s 103us/step - loss: 4.5617 - acc: 0.7611 -
Epoch 21/35
180/180 [==============================] - 0s 53us/step - loss: 4.5598 - acc: 0.7611 - v
Epoch 22/35
180/180 [==============================] - 0s 66us/step - loss: 4.5579 - acc: 0.7611 - v
Epoch 23/35
180/180 [==============================] - 0s 67us/step - loss: 4.5561 - acc: 0.7611 - v
Epoch 24/35
180/180 [==============================] - 0s 64us/step - loss: 4.5542 - acc: 0.7611 - v
Epoch 25/35
180/180 [==============================] - 0s 65us/step - loss: 4.5524 - acc: 0.7611 - v
Epoch 26/35
180/180 [==============================] - 0s 64us/step - loss: 4.5505 - acc: 0.7611 - v
Epoch 27/35
180/180 [==============================] - 0s 62us/step - loss: 4.5486 - acc: 0.7611 - v
Epoch 28/35
============] - 0s 60us/step - loss: 4.5468 - acc: 0.7611 - v
Saved successfully!
============] - 0s 70us/step - loss: 4.5449 - acc: 0.7611 - v
Epoch 30/35
180/180 [==============================] - 0s 70us/step - loss: 4.5431 - acc: 0.7611 - v
Epoch 31/35
180/180 [==============================] - 0s 65us/step - loss: 4.5412 - acc: 0.7611 - v
Epoch 32/35
180/180 [==============================] - 0s 61us/step - loss: 4.5394 - acc: 0.7611 - v
Epoch 33/35
180/180 [==============================] - 0s 60us/step - loss: 4.5375 - acc: 0.7611 - v
Epoch 34/35
180/180 [==============================] - 0s 73us/step - loss: 4.5356 - acc: 0.7611 - v
Epoch 35/35
180/180 [==============================] - 0s 68us/step - loss: 4.5338 - acc: 0.7611 - v
```

```
# serialize model to JSON
model_json = model.to_json()
with open("model_1.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_1.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

Not bad, we got to 76% (model_1) which is much better than the 21% we were getting before!

Let's play with a different regularizer to see if we can do better..

```
rom keras.models import Sequential
rom keras.layers import Dense, Activation
rom keras import optimizers

odel = Sequential()
odel.add(Dense(32, activation='relu', input_shape=(200, )))
odel.add(Dense(64, activation='relu'))
odel.add(Dense(128, activation='relu'))
 model.add(Dropout(0.1))
 model.add(Dense(128, activation='relu'))
odel.add(Dense(64, activation='relu'))
odel.add(Dense(32, activation='relu'))
odel.add(Dense(161, activation='relu'))
```

Saved successfully!

```
ion='sigmoid')) # all genres

gd = optimizers.SGD(lr=1e-08, clipvalue=1)
odel.compile(optimizer=sgd, loss='sparse_categorical_crossentropy', metrics=['accuracy']
odel.summary()
```

```
Model: "sequential_41"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_168 (Dense)            (None, 32)                6432
_________________________________________________________________
dense_169 (Dense)            (None, 64)                2112
_________________________________________________________________
dense_170 (Dense)            (None, 128)               8320
_________________________________________________________________
dense_171 (Dense)            (None, 64)                8256
_________________________________________________________________
dense_172 (Dense)            (None, 32)                2080
```

```
model.fit(data_train, label_train, epochs=20, batch_size=512, validation_split=0.33)
```

```
Train on 134 samples, validate on 67 samples
Epoch 1/20
134/134 [==============================] - 2s 15ms/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 2/20
134/134 [==============================] - 0s 82us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 3/20
134/134 [==============================] - 0s 69us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 4/20
134/134 [==============================] - 0s 61us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 5/20
134/134 [==============================] - 0s 71us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 6/20
134/134 [==============================] - 0s 82us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 7/20
134/134 [==============================] - 0s 81us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 8/20
134/134 [==============================] - 0s 73us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 9/20
134/134 [==============================] - 0s 78us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 10/20
134/134 [==============================] - 0s 78us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 11/20
134/134 [==============================] - 0s 78us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 12/20
134/134 [==============================] - 0s 83us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 13/20
134/134 [==============================] - 0s 77us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 14/20
            ============] - 0s 83us/step - loss: 5.1049 - acc: 0.0075 - v
Saved successfully!
134/134 [==============================] - 0s 68us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 16/20
134/134 [==============================] - 0s 72us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 17/20
134/134 [==============================] - 0s 70us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 18/20
134/134 [==============================] - 0s 90us/step - loss: 5.1049 - acc: 0.0075 - v
Epoch 19/20
134/134 [==============================] - 0s 107us/step - loss: 5.1049 - acc: 0.0075 -
Epoch 20/20
134/134 [==============================] - 0s 73us/step - loss: 5.1049 - acc: 0.0075 - v
```

Nope. Let's just stick with the weights from before.

1

Saved successfully!