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

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

```python
from keras.models import Sequential
from keras.layers import Dense, MaxPooling2D, Conv2D, Flatten, Dropout, Input, BatchNormali
from keras.models import Model, load_model
```

```
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_ver` `1.x` magic: more info.

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989

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 neu to include only the top 5.

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

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

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

genre_id	#tracks	parent	title	top_level
38	38154	0	Experimental	38
15	34413	0	Electronic	15
12	32923	0	Rock	12
1235	14938	0	Instrumental	1235
10	13845	0	Pop	10

```
top_titles = ["Experimental", "Electronic", "Rock", "Instrumental", "Pop"]
top_titles
```

```
['Experimental', 'Electronic', 'Rock', 'Instrumental', 'Pop']
```

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-- tl
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=0)
echonest.head()
```

	echonest						
	acousticness	**danceability**	**energy**	**instrumentalness**	**liveness**	**speechiness**	**tem**
track_id							
2	0.416675	0.675894	0.634476	0.010628	0.177647	0.159310	165
3	0.374408	0.528643	0.817461	0.001851	0.105880	0.461818	126
5	0.043567	0.745566	0.701470	0.000697	0.373143	0.124595	100
10	0.951670	0.658179	0.924525	0.965427	0.115474	0.032985	111
134	0.452217	0.513238	0.560410	0.019443	0.096567	0.525519	114

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":
        echonest.drop(col, axis=1, inplace=True)
```

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

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

track_id	acousticness	danceability	energy	instrumentalness	liveness	speechiness	te
2	0.416675	0.675894	0.634476	0.010628	0.177647	0.159310	165
3	0.374408	0.528643	0.817461	0.001851	0.105880	0.461818	126
5	0.043567	0.745566	0.701470	0.000697	0.373143	0.124595	100
10	0.951670	0.658179	0.924525	0.965427	0.115474	0.032985	111
134	0.452217	0.513238	0.560410	0.019443	0.096567	0.525519	114

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()
```

track_id	comments	date_created	date_released	engineer	favorites	id	information	list
2	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1		6
3	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1		6
5	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1		6
10	0	2008-11-26 01:45:08	2008-02-06 00:00:00	NaN	4	6	NaN	47
20	0	2008-11-26 01:45:05	2009-01-06 00:00:00	NaN	2	4	"spiritual songs" from Nicky Cook	2

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

	listens	listens	name	duration	genre_top	genres	title
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
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

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

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

	listens_album	listens_track	name	duration	genre_top	genres	title_album
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
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

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	ti
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	S

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

```
tracks_echo_genres.head()
```

track_id	listens_album	listens_track	name	duration	genre_top	genres	title_album	ti
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	S

```
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 some dimensionality reduction here. I will be using PCA post-merge.

```
features = pd.read_csv("features.csv", header=[0, 1, 2], skipinitialspace=True, index_col=0
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	0.8668
3	1.888963	0.760539	0.345297	2.295201	1.654031	0.067592	1.366848	1.054094	0.1081
5	0.527563	-0.077654	-0.279610	0.685883	1.937570	0.880839	-0.923192	-0.927232	0.6666
10	3.702245	-0.291193	2.196742	-0.234449	1.367364	0.998411	1.770694	1.604566	0.5212
20	-0.193837	-0.198527	0.201546	0.258556	0.775204	0.084794	-0.289294	-0.816410	0.0438

5 rows × 518 columns

```python
# 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: merg
  warnings.warn(msg, UserWarning)
```

```python
monster.head()
```

	listens_album	listens_track	name	duration	genre_top	genres	title_album	ti
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	
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	S

5 rows × 539 columns

```python
monster = monster[monster.genre_top.notnull()]
monster.head()
```

	listens_album	listens_track	name	duration	genre_top	genres	title_album	ti
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	
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	S

5 rows × 539 columns

```
monster = monster[monster.genre_top.isin(top_titles)]
monster.head()
```

	listens_album	listens_track	name	duration	genre_top	genres	title_album
track_id							
10	47632	50135	Kurt Vile	161	Pop	[10]	Constant Hitmaker
153	628	424	Arc and Sender	405	Rock	[26]	Arc and Sender
154	628	205	Arc and Sender	319	Rock	[26]	Arc and Sender
155	197	197	Arc and Sender	756	Rock	[26]	unreleased demo
169	716	270	Argumentix	144	Rock	[25]	Boss of Goth

5 rows × 539 columns

```
monster.to_pickle("./monster_top.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
```

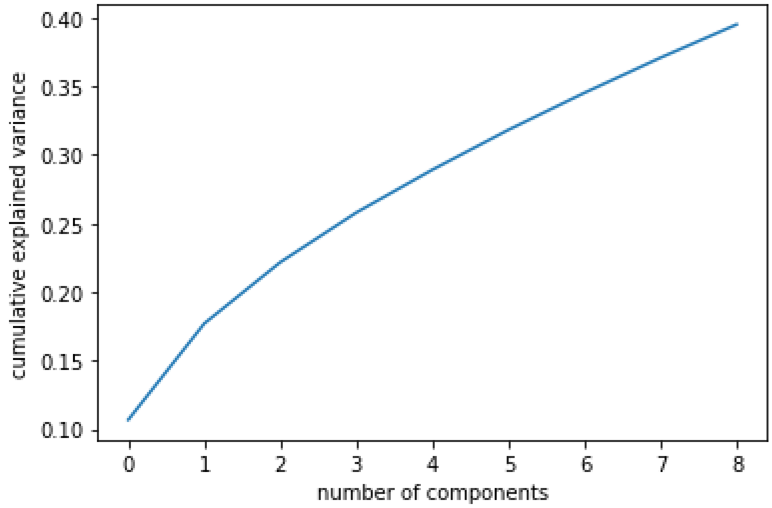
```
(6509, 532)
```

```
# 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 compo
```

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

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

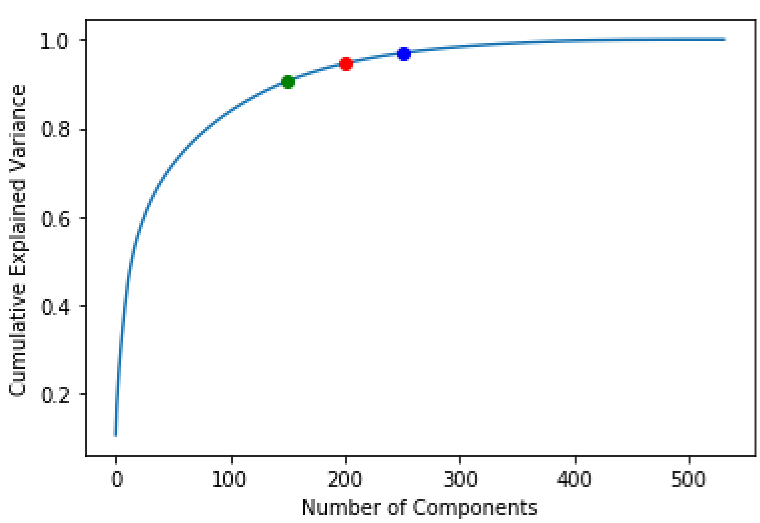
```
[0.10689284 0.07021918 0.0446891  0.03621707 0.03121396 0.02907616
 0.02699325 0.02559262 0.02420273]
[0.10689284 0.17711202 0.22180113 0.25801819 0.28923215 0.31830831
 0.34530156 0.37089418 0.39509691]
```



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:

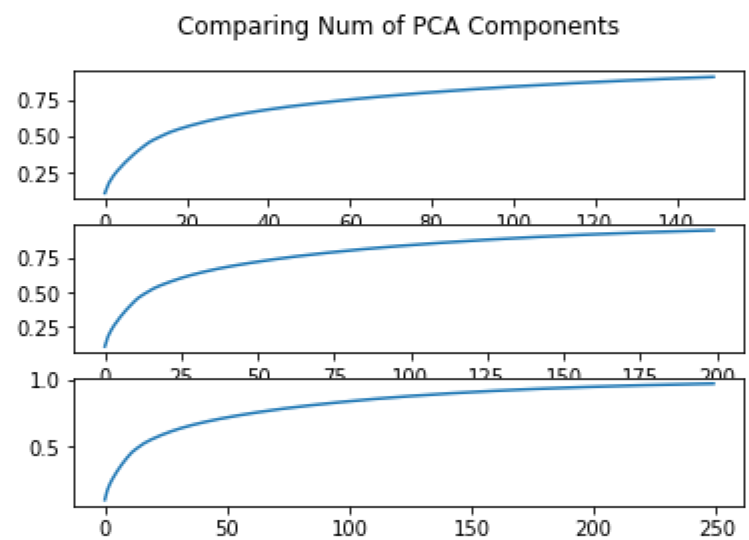
```
# SCREE PLOT

#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_))
```

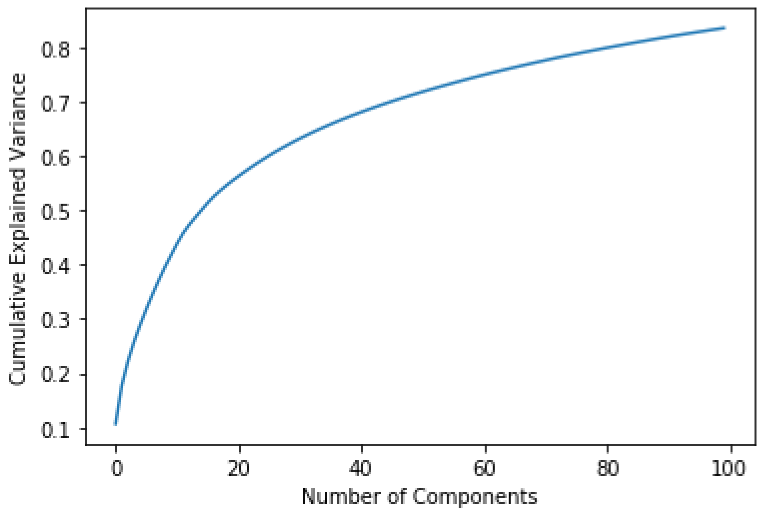
```
[<matplotlib.lines.Line2D at 0x7f8437761f60>]
```



So.... they are looking basically the same! Let's see if we can go smaller than 150 components so that we can be
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 0.8350745469250968
```

```
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 90.430565705896 %
175 92.73442261031477 %
200 94.49492280474016 %
```

Alright let's just go with 200 components. That'll be a 62% reduction!

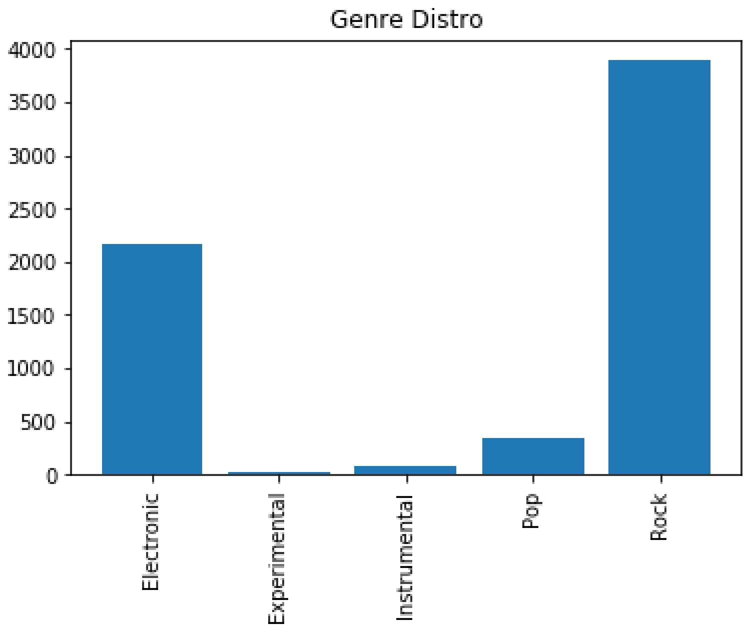
```
#Explained variance
pca = PCA(n_components=200)
principalComponents = pca.fit_transform(X)
col_names = [("col_" + str(i)) for i in range(200)]
principalDf = pd.DataFrame(data = principalComponents
             , columns = col_names)
```

```
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 valu
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)
```

```
{'Electronic': 2170, 'Experimental': 17, 'Instrumental': 84, 'Pop': 346, 'Rock': 3892}
```



Time to Train!

```
from sklearn.model_selection import train_test_split
principalDf = principalDf.replace(np.nan, 0, regex=True)

data = principalDf
labels = y_df

data_train, data_test, label_train, label_test = train_test_split(data, labels, test_size=0
```

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

```
pca = PCA(.94) # this is about 200 components as we saw earlier
pca.fit(data_train)
```

```
PCA(copy=True, iterated_power='auto', n_components=0.94, 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 this route (without thinking it all the way through) because the numerical genre values, genre_ids, were stored as 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
4	5	Classical

```
el_train2 = pd.merge(label_train, genrelabels, how="left", left_on=0, right_on="genre_title"
```

```
label_train2.head()
```

	0	genre_id	genre_title
0	Rock	12	Rock
1	Rock	12	Rock
2	Rock	12	Rock
3	Rock	12	Rock
4	Rock	12	Rock

```
label_test2 = pd.merge(label_test, genrelabels, how="left", left_on=0, right_on="genre_titl
label_test2.head()
```

	0	genre_id	genre_title
0	Rock	12	Rock
1	Electronic	15	Electronic
2	Rock	12	Rock
3	Rock	12	Rock
4	Electronic	15	Electronic

```
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       12
 1       15
 2       12
 3       12
 4       15
         ..
 2143    15
 2144    12
 2145    15
 2146    15
 2147    12
 Name: genre_id, Length: 2148, dtype: int64, 0       12
 1       12
 2       12
 3       12
 4       12
         ..
 4356    12
 4357    15
 4358    12
 4359    12
 4360    12
 Name: genre_id, Length: 4361, dtype: int64)
```

OK! let's try to train again..

```
from keras.utils import to_categorical

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
```

```
((4361, 1236), (4361, 1236))
```

```
om keras.models import Sequential
```

```
:om keras.models import Sequential
:om keras.layers import Dense, Activation

odel = Sequential()
odel.add(Dense(32, activation='relu', input_shape=(200, )))
 model.add(Dense(64, activation='relu'))
 model.add(Dense(128, activation='relu'))
odel.add(Dense(64, activation='relu'))
odel.add(Dense(32, activation='tanh'))

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

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

odel.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy', metrics=['accuracy
odel.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 32)                6432
_________________________________________________________________
dense_26 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_27 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_28 (Dense)             (None, 161)               5313
_________________________________________________________________
dense_29 (Dense)             (None, 164)               26568
=================================================================
Total params: 42,505
Trainable params: 42,505
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(data_train, label_train, epochs=50, batch_size=512, validation_split=0.:
```

```
Train on 2921 samples, validate on 1440 samples
Epoch 1/50
2921/2921 [==============================] - 0s 140us/step - loss: nan - acc: 0.0000e+00 -
Epoch 2/50
2921/2921 [==============================] - 0s 15us/step - loss: nan - acc: 0.0000e+00 -
Epoch 3/50
2921/2921 [==============================] - 0s 14us/step - loss: nan - acc: 0.0000e+00 -
Epoch 4/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 5/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 6/50
2921/2921 [==============================] - 0s 15us/step - loss: nan - acc: 0.0000e+00 -
Epoch 7/50
2921/2921 [==============================] - 0s 15us/step - loss: nan - acc: 0.0000e+00 -
Epoch 8/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 9/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 10/50
2921/2921 [==============================] - 0s 14us/step - loss: nan - acc: 0.0000e+00 -
Epoch 11/50
2921/2921 [==============================] - 0s 13us/step - loss: nan - acc: 0.0000e+00 -
Epoch 12/50
2921/2921 [==============================] - 0s 14us/step - loss: nan - acc: 0.0000e+00 -
Epoch 13/50
2921/2921 [==============================] - 0s 14us/step - loss: nan - acc: 0.0000e+00 -
Epoch 14/50
2921/2921 [==============================] - 0s 13us/step - loss: nan - acc: 0.0000e+00 -
Epoch 15/50
2921/2921 [==============================] - 0s 14us/step - loss: nan - acc: 0.0000e+00 -
Epoch 16/50
2921/2921 [==============================] - 0s 12us/step - loss: nan - acc: 0.0000e+00 -
Epoch 17/50
2921/2921 [==============================] - 0s 14us/step - loss: nan - acc: 0.0000e+00 -
Epoch 18/50
2921/2921 [==============================] - 0s 13us/step - loss: nan - acc: 0.0000e+00 -
Epoch 19/50
2921/2921 [==============================] - 0s 15us/step - loss: nan - acc: 0.0000e+00 -
Epoch 20/50
2921/2921 [==============================] - 0s 13us/step - loss: nan - acc: 0.0000e+00 -
Epoch 21/50
2921/2921 [==============================] - 0s 13us/step - loss: nan - acc: 0.0000e+00 -
Epoch 22/50
2921/2921 [==============================] - 0s 13us/step - loss: nan - acc: 0.0000e+00 -
Epoch 23/50
2921/2921 [==============================] - 0s 13us/step - loss: nan - acc: 0.0000e+00 -
Epoch 24/50
2921/2921 [==============================] - 0s 14us/step - loss: nan - acc: 0.0000e+00 -
Epoch 25/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 26/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 27/50
2921/2921 [==============================] - 0s 15us/step - loss: nan - acc: 0.0000e+00 -
Epoch 28/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 29/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 30/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 31/50
```

```
Epoch 31/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 32/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 33/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 34/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 35/50
2921/2921 [==============================] - 0s 18us/step - loss: nan - acc: 0.0000e+00 -
Epoch 36/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 37/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 38/50
2921/2921 [==============================] - 0s 15us/step - loss: nan - acc: 0.0000e+00 -
Epoch 39/50
2921/2921 [==============================] - 0s 15us/step - loss: nan - acc: 0.0000e+00 -
Epoch 40/50
2921/2921 [==============================] - 0s 18us/step - loss: nan - acc: 0.0000e+00 -
Epoch 41/50
2921/2921 [==============================] - 0s 14us/step - loss: nan - acc: 0.0000e+00 -
Epoch 42/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 43/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 44/50
2921/2921 [==============================] - 0s 18us/step - loss: nan - acc: 0.0000e+00 -
Epoch 45/50
2921/2921 [==============================] - 0s 17us/step - loss: nan - acc: 0.0000e+00 -
Epoch 46/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 47/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 48/50
2921/2921 [==============================] - 0s 15us/step - loss: nan - acc: 0.0000e+00 -
Epoch 49/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
Epoch 50/50
2921/2921 [==============================] - 0s 16us/step - loss: nan - acc: 0.0000e+00 -
```

```
# serialize model to JSON
model_json = model.to_json()
with open("model_2.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_2.h5")
print("Saved model to disk")
```

```
results_test = model.evaluate(data_test, label_test[:99])
print("results_test:", results_test)

results_train = model.evaluate(data_train, label_train)
print("results_train:", results_train)
```

Another training attempt w another optimizer (worse)

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..

↳ *4 cells hidden*

Plotting Training and Validation Loss + Acc

```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'rx', label="Training Loss")
plt.plot(epochs, val_loss, 'b', label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```

```
plt.clf()

acc = history.history['acc']
val_acc = history.history['val_acc']
# epochs = range(1, len(loss) + 1)

plt.plot(epochs, acc, 'r', label="Training Accuracy")
plt.plot(epochs, val_acc, 'b', label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
```

```
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```

Comparing to a Random Baseline

```
import copy
np.random.seed(4242)

test_labels_copy = copy.copy(label_test)
np.random.shuffle(test_labels_copy)
hits_array = np.array(label_test) == np.array(test_labels_copy)
float(np.sum(hits_array)) / len(label_test)
```

Predictions on New Data

```
predictions = model.predict(data_test)
predictions[2].shape
```

That's good! This is what's expected-- our 164 possible genres.

```
np.sum(predictions[0])
predictions[0]
#hmm...
```

```
np.argmax(predictions[2])
```

```
genre_labels["genre_title"][17]
```

```
genre_labels["genre_title"][int(label_test[2])]
```

```
for i in range(0, 50):
  p_idx = np.argmax(predictions[i])
  p_genre = genre_labels["genre_title"][p_idx]
  a_genre = genre_labels["genre_title"][int(label_test[i])]
  print(i, '\t', p_genre, '\n\t', a_genre, '\n')
```

1