

```

1 import os
2 import pandas as pd
3 import numpy as np
4 import librosa
5 import random
6 import time
7 import pickle
8
9 from librosa.display import specshow, waveplot
10
11 from sklearn.preprocessing import MinMaxScaler
12 from sklearn.model_selection import train_test_split
13 from sklearn.preprocessing import LabelBinarizer
14
15 import IPython.display as ipd
16
17 import matplotlib
18 import matplotlib.pyplot as plt
19
20 np.random.seed(42)
21
22 %matplotlib inline

1 from keras.models import Sequential
2 from keras.layers import Dense, MaxPooling2D, Conv2D, Flatten, Dropout, Input, BatchNormali
3 from keras.models import Model, load_model
4 from keras.callbacks import Callback, EarlyStopping
5 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](#).

```

1 from google.colab import drive
2 drive.mount('/gdrive')

```

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 simplifying the output of our neural net to include only the top 5.

Previously, I attempted outputting 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
			Experimental	38
			Electronic	15
12	32923	0	Rock	12
1235	14938	0	Instrumental	1235
10	13845	0	Pop	10

▼ 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*.

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

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

↗

echonest							
	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

5 rows × 249 columns

```
1 for col in echonest:
2     if col[0] == "metadata":
3         echonest.drop(col, axis=1, inplace=True)
4     elif col[0] == "ranks":
5         echonest.drop(col, axis=1, inplace=True)
6     elif col[0] == "social features":
7         echonest.drop(col, axis=1, inplace=True)

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

```
1 echonest_sub = echonest[['acousticness', 'danceability', 'energy', 'instrumentalness',
2 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.

```
1 tracks = pd.read_csv("tracks.csv", header=[0, 1], skipinitialspace=True, index_col=0)
2 tracks.columns = tracks.columns.droplevel(0)
3 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	<p></p>	
3	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1	<p></p>	
-	-	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1	<p></p>	
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	<p>"spiritual songs" from Nicky Cook</p>	

Saved successfully!

✕

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



	listens	listens	name	duration	genre_top	genres	title	title
track_id								
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()
```



	listens_album	listens_track	name	duration	genre_top	genres	title_album	title_track
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	

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()
```



	listens_album	listens_track	name	duration	genre_top	genres	title_album	id
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	

```
1 tracks_echo_genres = pd.merge(tracks_echo, genres, how="left", left_on="genre_top", right_on="genre_top")
1 tracks_echo_genres.head()
```



	listens_album	listens_track	name	duration	genre_top	genres	title_album	id
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	

Saved successfully! ✕

```
1 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.

```
1 features = pd.read_csv("features.csv", header=[0, 1, 2], skipinitialspace=True, index_col=0)
2 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	0.86
3	1.888963	0.760539	0.345297	2.295201	1.654031	0.067592	1.366848	1.054094	0.10
5	0.527563	-0.077654	-0.279610	0.685883	1.937570	0.880839	-0.923192	-0.927232	0.66
10	3.702245	-0.291193	2.196742	-0.234449	1.367364	0.998411	1.770694	1.604566	0.52
20	-0.193837	-0.198527	0.201546	0.258556	0.775204	0.084794	-0.289294	-0.816410	0.04

5 rows × 518 columns

```
1 # MERGING!!!
2 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)

```
1 monster.head()
```

	listens_album	listens_track	name	duration	genre_top	genres	title_album	1
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	
		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	

5 rows × 539 columns

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

PCA Shenanigans for Dimensionality Reduction

```
1 from sklearn.preprocessing import StandardScaler
2 feats = monster.columns
3
4 # Separating out the features
5 numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
6 x = monster.select_dtypes(include=numerics).values
7
8 # x = monster.loc[:, feats].values
9 # Separating out the target
10 y = monster.loc[:, ['genre_top']].values
11
12 # Standardizing the features
13 X = StandardScaler().fit_transform(x)
```

```
1 X.shape
```

```
↳ (13129, 532)
```

```
1 # no nan vals allowed!!!
2 from sklearn.impute import SimpleImputer
3
4 imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
5 imputer = imputer.fit(X[:,1:532])
6 X[:,1:532] = imputer.transform(X[:,1:532])
```

```
1 from sklearn.decomposition import PCA
2
3 pca = PCA(n_components=9) # for number of big attributes?
4 principalComponents = pca.fit_transform(X)
5 principalDf = pd.DataFrame(data = principalComponents
6                             , columns = ['principal component 1', 'principal component 2', 'principal component 3', 'principal component 4', 'principal component 5', 'principal component 6', 'principal component 7', 'principal component 8', 'principal component 9'])
```

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

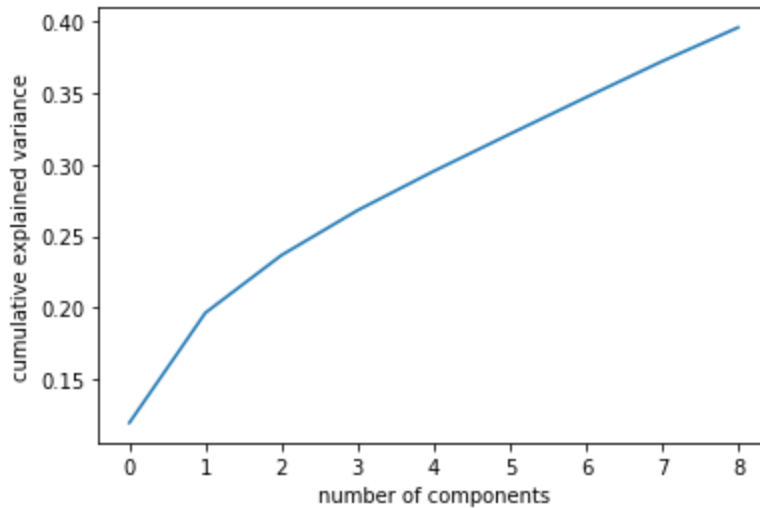
```
4 plt.plot(np.cumsum(pca.explained_variance_ratio_))
5 plt.xlabel('number of components')
6 plt.ylabel('cumulative explained variance')
7 plt.show()
```

```
↳
```

Saved successfully!



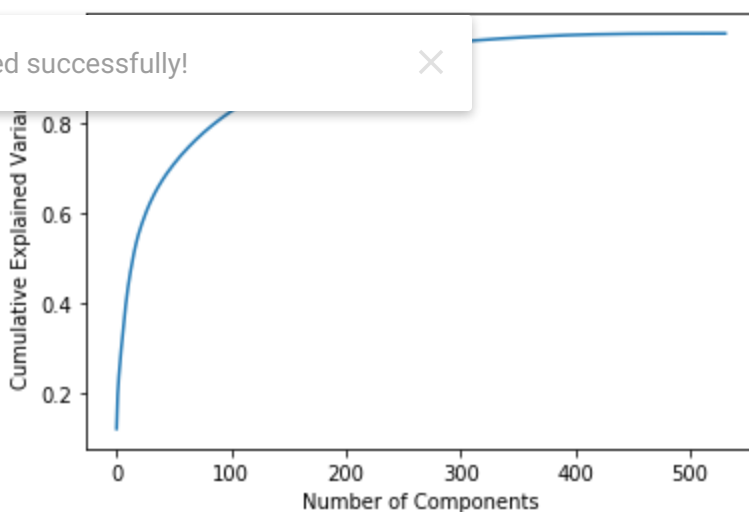

```
[0.11959576 0.07689131 0.04026169 0.03136196 0.02734066 0.0259367
 0.02560827 0.02520922 0.02367956]
[0.11959576 0.19648707 0.23674876 0.26811072 0.29545139 0.32138809
 0.34699636 0.37220558 0.39588514]
```



▼ 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:

```
1 # SCREE PLOT
2
3 #Explained variance
4 pca = PCA().fit(X)
5 plt.plot(np.cumsum(pca.explained_variance_ratio_))
6 plt.xlabel('Number of Components')
7 plt.ylabel('Cumulative Explained Variance')
8
9 plt.plot(250, np.cumsum(pca.explained_variance_ratio_)[250], "ob")
10 plt.plot(200, np.cumsum(pca.explained_variance_ratio_)[200], "or")
11 plt.plot(150, np.cumsum(pca.explained_variance_ratio_)[150], "og")
12
13 plt.show()
```



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

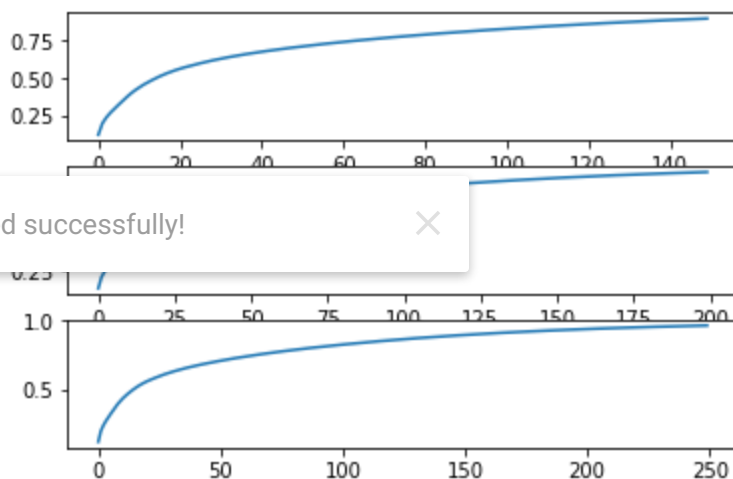
```

1 #Explained variance
2
3 fig, axs = plt.subplots(3)
4 fig.suptitle('Comparing Num of PCA Components')
5 # fig.xlabel('Number of Components')
6 # fig.ylabel('Cumulative Explained Variance')
7
8 # PLOT 1: 150 COMPONENTS
9 pca = PCA(n_components=150)
10 principalComponents = pca.fit_transform(X)
11 col_names = [("col_" + str(i)) for i in range(150)]
12 principalDf = pd.DataFrame(data = principalComponents
13                             , columns = col_names)
14 axs[0].plot(np.cumsum(pca.explained_variance_ratio_))
15
16 # PLOT 2: 200 COMPONENTS
17 pca = PCA(n_components=200)
18 principalComponents = pca.fit_transform(X)
19 col_names = [("col_" + str(i)) for i in range(200)]
20 principalDf = pd.DataFrame(data = principalComponents
21                             , columns = col_names)
22 axs[1].plot(np.cumsum(pca.explained_variance_ratio_))
23
24 # PLOT 3: 250 COMPONENTS
25 pca = PCA(n_components=250)
26 principalComponents = pca.fit_transform(X)
27 col_names = [("col_" + str(i)) for i in range(250)]
28 principalDf = pd.DataFrame(data = principalComponents
29                             , columns = col_names)
30 axs[2].plot(np.cumsum(pca.explained_variance_ratio_))

```

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

Comparing Num of PCA Components



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

```

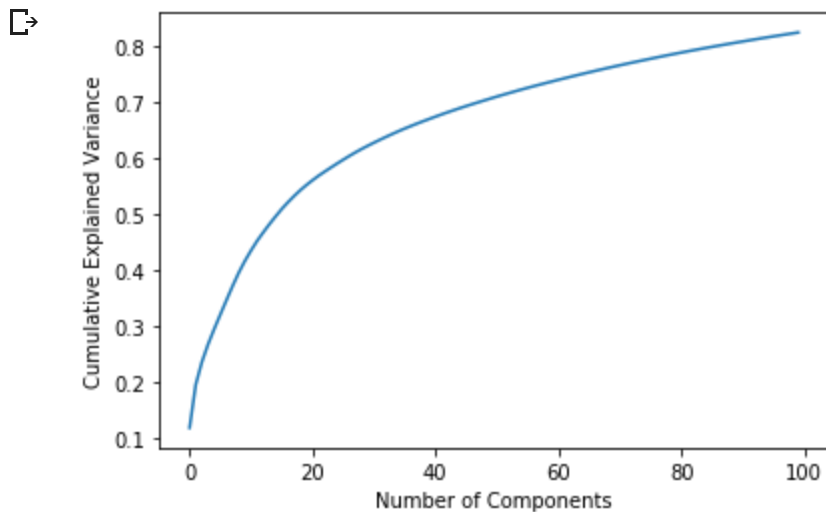
1 #Explained variance
2 pca = PCA(n_components=100) # for number of big attributes?
3 principalComponents = pca.fit transform(X)

```

```

4 col_names = [("col_" + str(i)) for i in range(100)]
5 principalDf = pd.DataFrame(data = principalComponents
6                             , columns = col_names)
7
8 plt.plot(np.cumsum(pca.explained_variance_ratio_))
9 plt.xlabel('Number of Components')
10 plt.ylabel('Cumulative Explained Variance')
11 plt.show()
12
13 print(100, np.cumsum(pca.explained_variance_ratio_)[99])

```



```
100 0.8235167577314023
```

```

1 def compare_n_comp(n):
2     pca = PCA(n_components=n)
3     principalComponents = pca.fit_transform(X)
4     print(n, str((np.cumsum(pca.explained_variance_ratio_)[n-1])*100), str('%'))
5
6 compare_n_comp(150)
7 compare_n_comp(175)
8 compare_n_comp(200)

```

```

150 89.51656816113069 %
175 91.96887994360743 %
200 93.89739765751429 %

```

Saved successfully!

ts. That'll be a 62% reduction!

```

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

```

```

1 print(principalDf.shape)
2 print(y.shape)

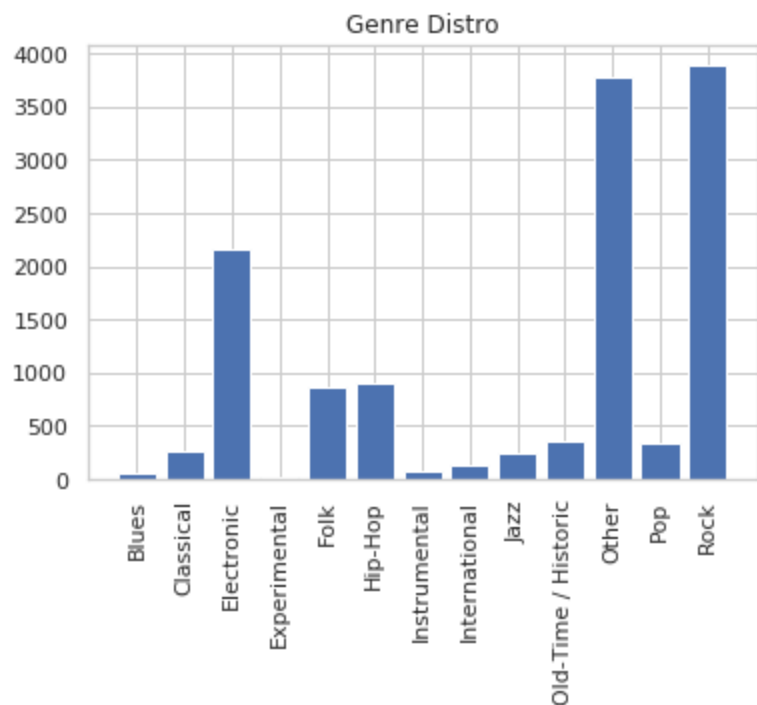
```



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.

```
1 y_df = pd.DataFrame(y)
2 y_df = y_df.replace(np.nan, 'Other', regex=True)
3 y_map = dict(zip(*np.unique(y_df, return_counts=True)))
4 plt.bar(y_map.keys(), y_map.values())
5 plt.xticks(rotation='vertical')
6 plt.title("Genre Distro")
7
8 print(y_map)
```

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



▼ Time to Train!

Saved successfully! ✕

```
import train_test_split
2 principalDf = principalDf.replace(np.nan, 0, regex=True)
3
4 data = principalDf[:300]
5 labels = y_df[:300]
6
7 data_train, data_test, label_train, label_test = train_test_split(data, labels, test_si:
```

```
1 scaler = StandardScaler()
2
3 # Fit on training set only.
4 scaler.fit(data_train)
5
6 # Apply transform to both the training set and the test set.
7 data_train = scaler.transform(data_train)
```

```

7 data_train = scaler.transform(data_train)
8 data_test = scaler.transform(data_test)

1 pca = PCA(.93) # this is about 200 components as we saw earlier
2 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

```

1 with open("genre_labels.pkl", "rb") as handle:
2     genre_labels = pickle.load(handle)
3
4 genrelabels = pd.DataFrame.from_dict(genre_labels)
5 genrelabels.head()

```

```

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

```

Saved successfully!



```

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

1 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

```
1 label_test2 = pd.merge(label_test, genrelabels, how="left", left_on=0, right_on="genre_title")
2 label_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

```
1 label_test = label_test2["genre_id"]
2 label_train = label_train2["genre_id"]
3 label_test = label_test.replace(np.nan, 0, regex=True)
4 label_train = label_train.replace(np.nan, 0, regex=True)
5
6 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..

```
1 one_hot_train_labels = to_categorical(label_train.values)
2 one_hot_test_labels = to_categorical(label_test.values)
3
4 one_hot_train_labels.shape, one_hot_train_labels.shape
```

↳ ((201, 22), (201, 22))

```
1 from keras.models import Sequential
2 from keras.layers import Dense, Activation
3
4 model = Sequential()
5 model.add(Dense(32, activation='relu', input_shape=(200, )))
6 model.add(Dense(64, activation='relu'))
7 model.add(Dense(128, activation='relu'))
8 model.add(Dense(64, activation='relu'))
9 model.add(Dense(32, activation='tanh'))
10
11 model.add(Dense(161, activation='softmax'))
12
13 output = 164
14 model.add(Dense(output, activation='sigmoid')) # all genres
15
16 model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy', metrics=['ac
17 model.summary()
```

↳ Model: "sequential_59"

Layer (type)	Output Shape	Param #
dense_289 (Dense)	(None, 32)	6432
dense_290 (Dense)	(None, 64)	2112
dense_291 (Dense)	(None, 128)	8320
dense_292 (Dense)	(None, 64)	8256
dense_293 (Dense)	(None, 32)	2080
dense_294 (Dense)	(None, 161)	5313
dense_295 (Dense)	(None, 164)	26568
Total params: 59,081		
Trainable params: 59,081		
Non-trainable params: 0		

Saved successfully!

```
1 model.fit(data_train, label_train, epochs=50, batch_size=512, validation_split=0.1)
```

↳

Epoch 22/50
180/180 [=====] - 0s 68us/step - loss: 4.9481 - acc: 0.6556 - v

Epoch 23/50
180/180 [=====] - 0s 63us/step - loss: 4.9457 - acc: 0.6556 - v

Epoch 24/50
180/180 [=====] - 0s 64us/step - loss: 4.9432 - acc: 0.6556 - v

Epoch 25/50
180/180 [=====] - 0s 59us/step - loss: 4.9408 - acc: 0.6556 - v

Epoch 26/50
180/180 [=====] - 0s 58us/step - loss: 4.9383 - acc: 0.6556 - v

Epoch 27/50
180/180 [=====] - 0s 61us/step - loss: 4.9359 - acc: 0.6556 - v

Epoch 28/50
180/180 [=====] - 0s 71us/step - loss: 4.9335 - acc: 0.6556 - v

Epoch 29/50
180/180 [=====] - 0s 68us/step - loss: 4.9311 - acc: 0.6556 - v

Epoch 30/50
180/180 [=====] - 0s 69us/step - loss: 4.9287 - acc: 0.6556 - v

Epoch 31/50
180/180 [=====] - 0s 76us/step - loss: 4.9264 - acc: 0.2611 - v

Epoch 32/50
180/180 [=====] - 0s 75us/step - loss: 4.9240 - acc: 0.2611 - v

Epoch 33/50
180/180 [=====] - 0s 71us/step - loss: 4.9216 - acc: 0.2611 - v

Epoch 34/50
180/180 [=====] - 0s 68us/step - loss: 4.9192 - acc: 0.3000 - v

Epoch 35/50
180/180 [=====] - 0s 66us/step - loss: 4.9168 - acc: 0.2611 - v

Epoch 36/50
180/180 [=====] - 0s 70us/step - loss: 4.9144 - acc: 0.2667 - v

Epoch 37/50
180/180 [=====] - 0s 74us/step - loss: 4.9121 - acc: 0.3667 - v

Epoch 38/50
180/180 [=====] - 0s 51us/step - loss: 4.9097 - acc: 0.2667 - v

Epoch 39/50
180/180 [=====] - 0s 84us/step - loss: 4.9072 - acc: 0.3667 - v

Epoch 40/50
180/180 [=====] - 0s 96us/step - loss: 4.9048 - acc: 0.3667 - v

Epoch 41/50
180/180 [=====] - 0s 84us/step - loss: 4.9023 - acc: 0.3667 - v

Epoch 42/50
180/180 [=====] - 0s 88us/step - loss: 4.8999 - acc: 0.3667 - v

Epoch 43/50
180/180 [=====] - 0s 81us/step - loss: 4.8974 - acc: 0.3667 - v

Epoch 44/50
180/180 [=====] - 0s 79us/step - loss: 4.8949 - acc: 0.3667 - v

Epoch 45/50
180/180 [=====] - 0s 82us/step - loss: 4.8922 - acc: 0.3667 - v

Epoch 46/50
180/180 [=====] - 0s 81us/step - loss: 4.8896 - acc: 0.3667 - v

Epoch 47/50
180/180 [=====] - 0s 77us/step - loss: 4.8870 - acc: 0.3667 - v

Epoch 48/50
180/180 [=====] - 0s 79us/step - loss: 4.8845 - acc: 0.3667 - v

Epoch 49/50
180/180 [=====] - 0s 57us/step - loss: 4.8819 - acc: 0.3667 - v

Epoch 50/50
180/180 [=====] - 0s 70us/step - loss: 4.8794 - acc: 0.3667 - v

Saved successfully!




```
1 # serialize model to JSON
```

```
Saved successfully!  as json_file:
```

```
model_json = model.to_json()
```

```
5 # serialize weights to HDF5
```


```
6 model.save_weights("model_1.h5")
```

```
7 print("Saved model to disk")
```

```
 Saved model to disk
```

```
1 results = model.evaluate(data_train, label_train)
```

```
2 results
```

```
 201/201 [=====] - 0s 147us/step  
[4.878632616640917, 0.33333333333704009]
```

▶ Another training attempt w another optimizer (worse)

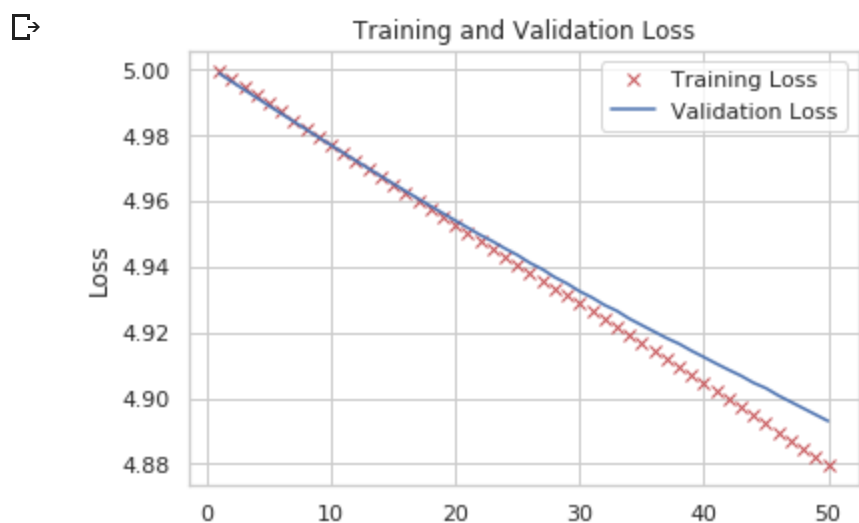
Not bad, we got to 76% (model_1) which is much better than the 21% we were getting before!

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

↳ 4 cells hidden

▶ Plotting Training and Validation Loss + Acc

```
1 loss = history.history['loss']
2 val_loss = history.history['val_loss']
3 epochs = range(1, len(loss) + 1)
4
5 plt.plot(epochs, loss, 'rx', label="Training Loss")
6 plt.plot(epochs, val_loss, 'b', label="Validation Loss")
7 plt.title("Training and Validation Loss")
8 plt.xlabel("Epochs")
9 plt.ylabel("Loss")
10 plt.legend()
11
12 plt.show()
```

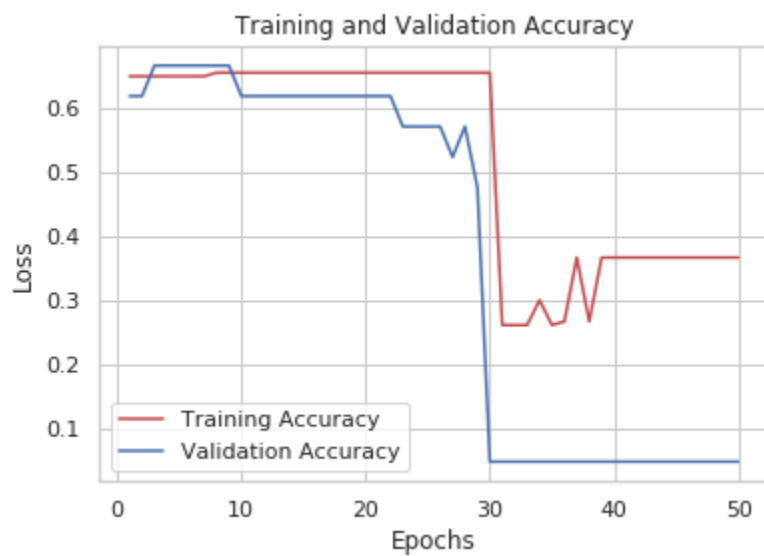


Saved successfully!

```

1 plt.clf()
2
3 acc = history.history['acc']
4 val_acc = history.history['val_acc']
5 # epochs = range(1, len(loss) + 1)
6
7 plt.plot(epochs, acc, 'r', label="Training Accuracy")
8 plt.plot(epochs, val_acc, 'b', label="Validation Accuracy")
9 plt.title("Training and Validation Accuracy")
10 plt.xlabel("Epochs")
11 plt.ylabel("Loss")
12 plt.legend()
13
14 plt.show()

```



1

Saved successfully!

