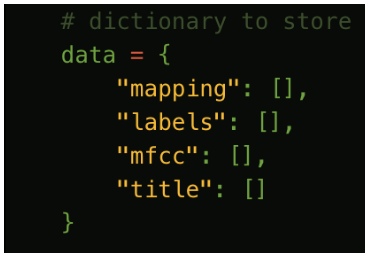
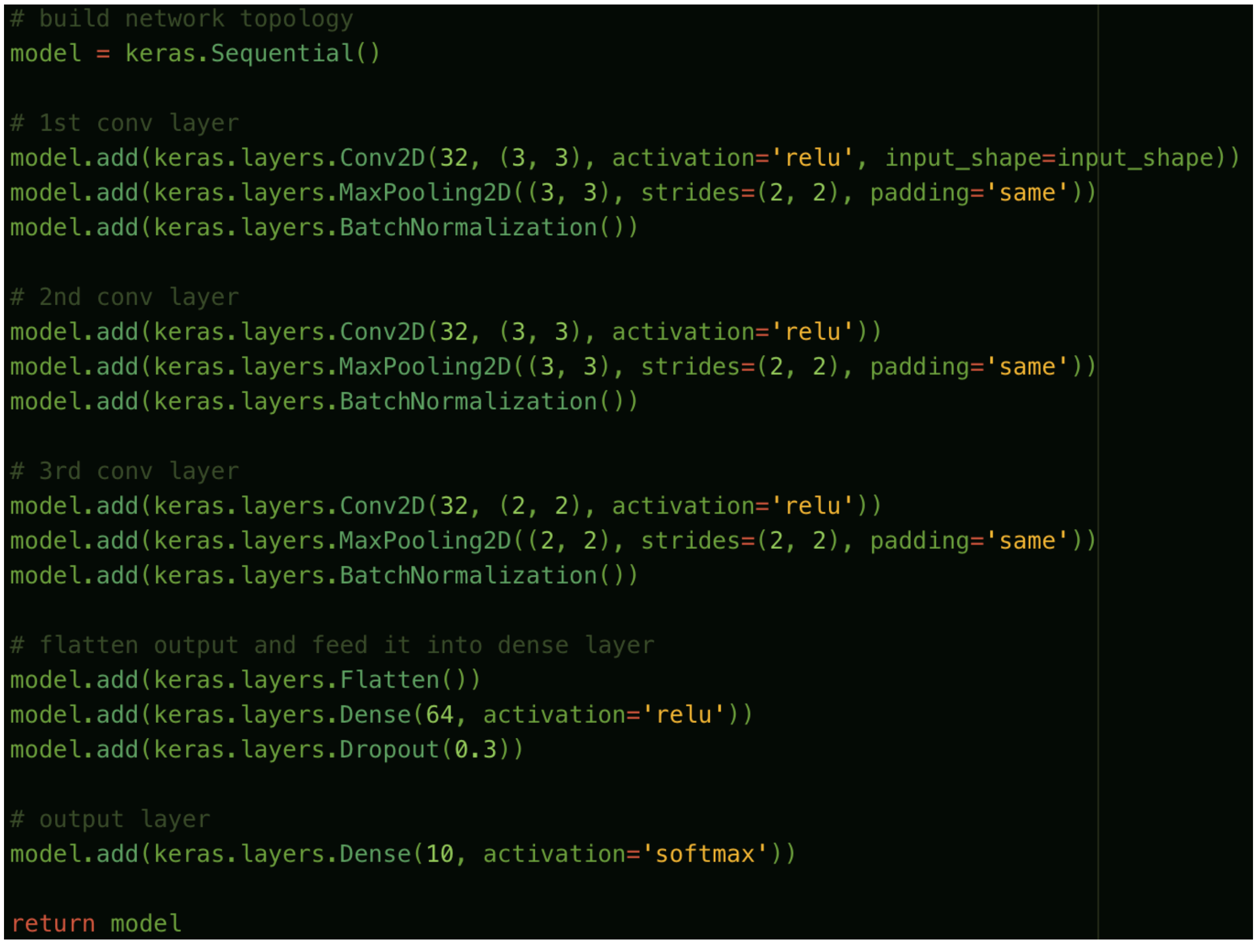
For my project, I followed a tutorial on using TensorFlow for music genre classification. extractData.py works by gathering data from a music database from the website Marsyas.com such as genre and mfcc data and puts it in a dictionary in the format below.



```
    # dictionary to store
    data = {
        "mapping": [],
        "labels": [],
        "mfcc": [],
        "title": []
    }
```

This data is dumped to a JSON file that will be access by the program that handles the TensorFlow model. The “mapping” key stores all the genre options and the “label” key is the index of “mapping” key.

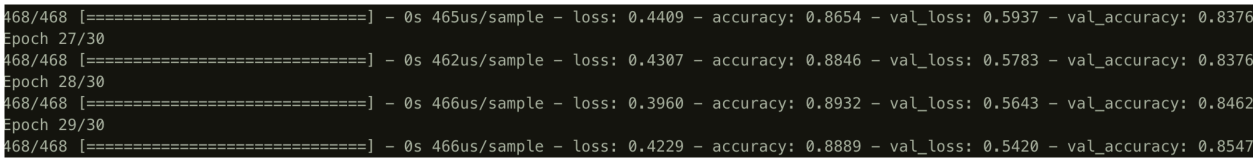
The newCNN.py file creates a TensorFlow model that uses a Convolution Neural Network.



```
# build network topology
model = keras.Sequential()

# 1st conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 2nd conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 3rd conv layer
model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# flatten output and feed it into dense layer
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.3))

# output layer
model.add(keras.layers.Dense(10, activation='softmax'))

return model
```

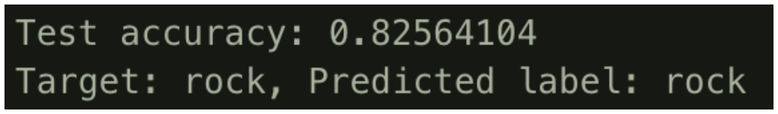
This code is responsible for making the layers of the CNN. The CNN is fed the data from the JSON file after being split into what percentages will go to training and to testing. As you can see, this uses ‘ReLU’ or Rectified Linear Unit activation function.

The program displays lots of interesting information including the stats as the machine learns, including test vs. train accuracy, and overall loss.



```
468/468 [==============================] - 0s 465us/sample - loss: 0.4409 - accuracy: 0.8654 - val_loss: 0.5937 - val_accuracy: 0.8376
Epoch 27/30
468/468 [==============================] - 0s 462us/sample - loss: 0.4307 - accuracy: 0.8846 - val_loss: 0.5783 - val_accuracy: 0.8376
Epoch 28/30
468/468 [==============================] - 0s 466us/sample - loss: 0.3960 - accuracy: 0.8932 - val_loss: 0.5643 - val_accuracy: 0.8462
Epoch 29/30
468/468 [==============================] - 0s 466us/sample - loss: 0.4229 - accuracy: 0.8889 - val_loss: 0.5420 - val_accuracy: 0.8547
```

This program also ends with making the model make a prediction based on what it has learned. I modified this code to be a little easier on the eyes as far as all the complicated music information goes, so it just shows what the target is and what the model predicts the target is.



```
Test accuracy: 0.82564104
Target: rock, Predicted label: rock
```

I'm really happy with this tutorial it is a video series of about 20 on YouTube and it has covered a lot such as ideas of overfitting and many other concepts that I learned some of in Fourier Analysis. This program will also produce a graph that shows the difference in the train and the test accuracy.

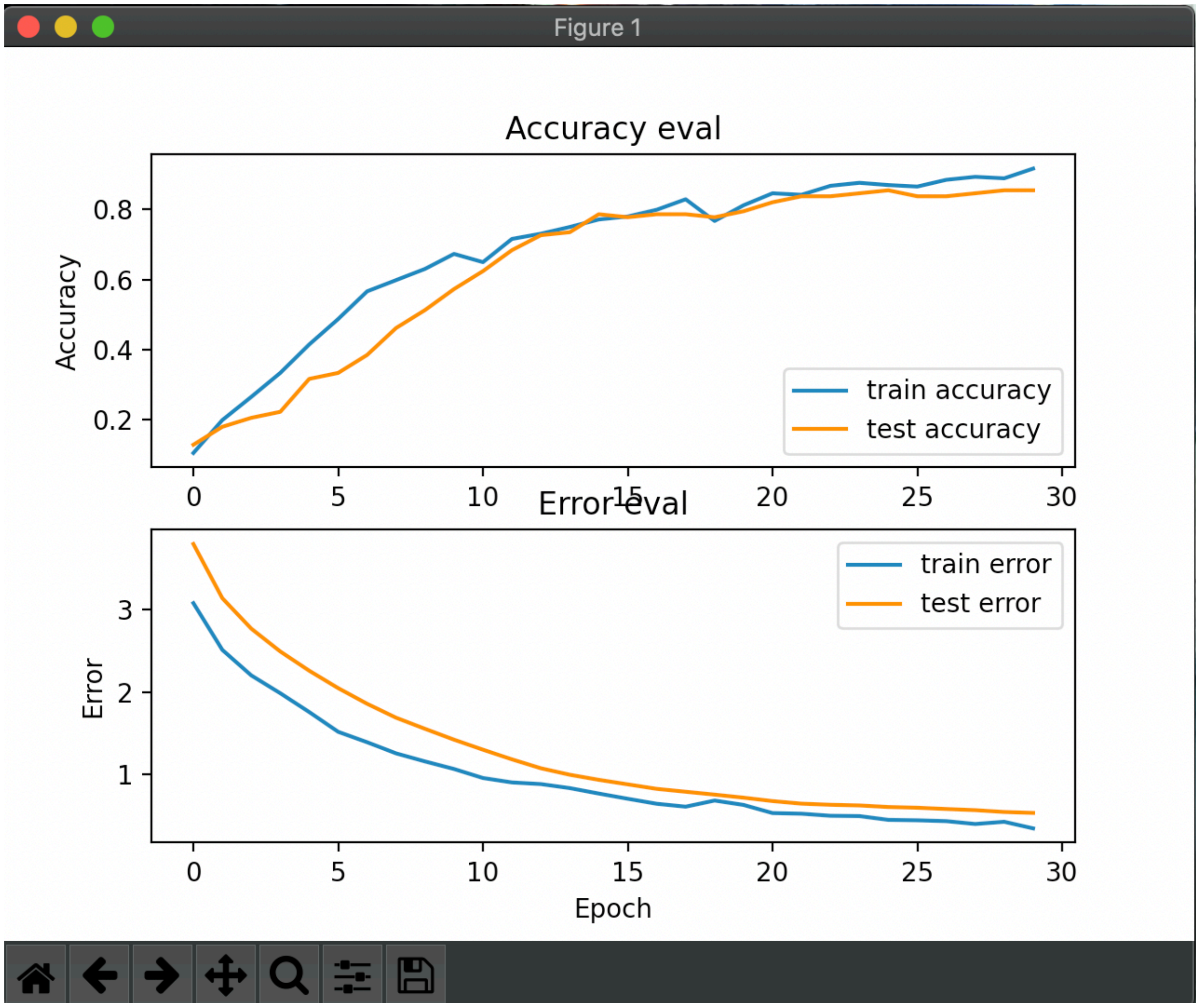


Figure 1
Accuracy eval
Accuracy
train accuracy
test accuracy
Error eval
Error
train error
test error
Epoch