Music classification Neural Network

Presented by:
Bhavesh Baviskar
Naveen Tiwari
Abdul Raqeeb

In this project we are trying to identify various audio

We have used the concept of neural networks to classify different music genre

Neural networks are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.
Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity.

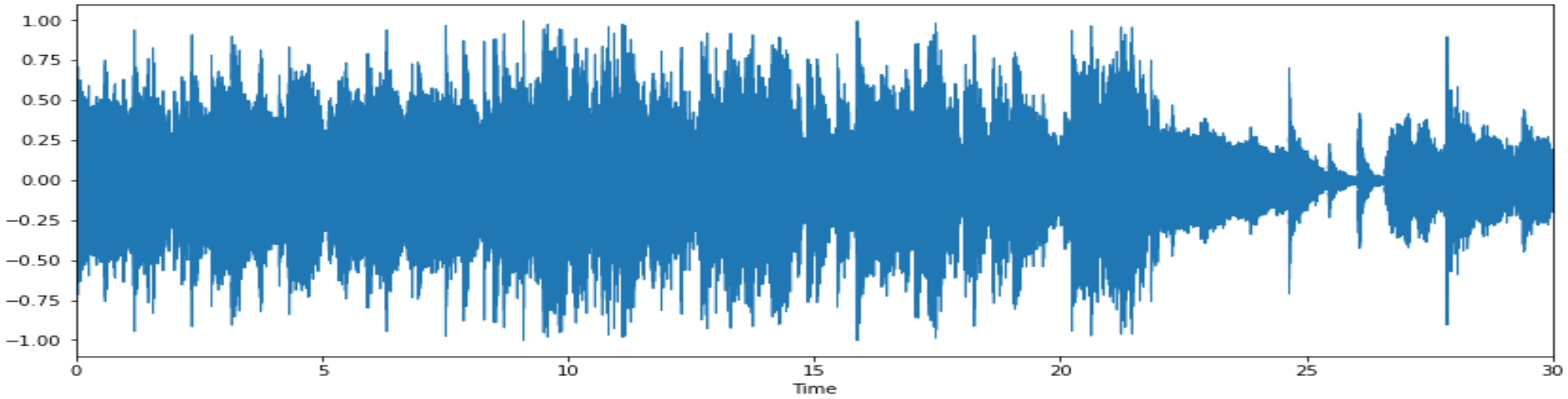
Information related to the data:

- **CSV files** - it consists of two csv files. There is one file for every song, we have calculated the mean and variance using multiple features that can be extracted from an audio file . Different file include the same structure, and we have splitted the songs before three seconds audio files, just to increase the data for classification.

- **Genres original** - A collection of various genres (reggae, blues, rock, classical, metal, jazz, hiphop, disco, country, pop) with 100 audio files each. including the GTZAN dataset, the MNIST of sounds

```
rock_path = 'Data/genres_original/rock/rock.00001.wav'
rock_audio = plot_sound(rock_path)

ipd.Audio(rock_path)
```



Each datasets has a length of 30 seconds.

For these neural network project we have imported several different types of libraries.

```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import math
import os, librosa
```

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
Json for
In scikit-learn a random split into training and test sets can be quickly computed with the train_test_split helper function.
Split arrays or matrices into random train and test subsetsQuick utility that wraps input validation and next(ShuffleSplit().split(X, y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Keras has the following key features:Allows the same code to run on CPU or on GPU, seamlessly.User-friendly API which makes it easy to quickly prototype deep learning models.Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.

Import math functions in python help us to do various mathematical calculations.

The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The os and *os

Librosa is a Python package for music and audio analysis. Librosa is basically used when we work with audio data like in music generation(using LSTM's), Automatic Speech Recognition. It provides the building blocks necessary to create the music information retrieval systems

```
def save_mfcc(dataset_path, json_path, num_mfcc=13, n_fft=2048, hop_length=512, num_segments=5):
    data = {
        "mapping": [],
        "labels": [],
        "mfcc": []
    }

    samples_per_segment = int(SAMPLES_PER_TRACK / num_segments)
    num_mfcc_vectors_per_segment = math.ceil(samples_per_segment / hop_length)

    for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):

        if dirpath is not dataset_path:

            semantic_label = dirpath.split("/")[-1]
            data["mapping"].append(semantic_label)
            print("\nProcessing: {}".format(semantic_label))

            for f in filenames:

                file_path = os.path.join(dirpath, f)

                if file_path != 'python exapmle/genres_original/jazz/jazz.00054.wav':

                    signal, sample_rate = librosa.load(file_path, sr=SAMPLE_RATE)

                    for d in range(num_segments):
                        start = samples_per_segment * d
                        finish = start + samples_per_segment
                        mfcc = librosa.feature.mfcc(signal[start:finish], sample_rate, n_mfcc=num_mfcc, n_fft=n_fft,
                                                    hop_length=hop_length)
                        mfcc = mfcc.T
                        if len(mfcc) == num_mfcc_vectors_per_segment:
                            data["mfcc"].append(mfcc.tolist())
                            data["labels"].append(i - 1)
                            print("{}, segment:{}".format(file_path, d + 1))

    with open(json_path, "w") as fp:
        json.dump(data, fp, indent=4)


save_mfcc(DATASET_PATH, JSON_PATH, num_segments=6)

DATA_PATH = "./data_10.json"
```

Here we have converted the file from music format to the Json format because it eases in visualizing the data. Also while working with Json we can work with more variables ,which help us in getting more numerical data which help us in synchronizing it with the other data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.07)
```

In scikit-learn a random split into training and test sets can be quickly computed with the train_test_split helper function.
We can easily split our data in different section such as test data and train data. Through which we can build the neural network
We have split the test data into 7 %

```
model = keras.Sequential([

    # input layer
    keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])),

    # 1st dense layer
    keras.layers.Dense(512, activation='relu'),

    # 2nd dense layer
    keras.layers.Dense(256, activation='relu'),

    # 3rd dense layer
    keras.layers.Dense(64, activation='relu'),

    # output layer
    keras.layers.Dense(10, activation='softmax')
])
```

RELU activation function

In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.

The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.In this tutorial, you will discover the rectified linear activation function for deep learning neural networks.After completing this tutorial, you will know:The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem.

Applies the rectified linear unit activation function. With default values, this returns the standard ReLU activation: max(x, 0) , the element-wise maximum of 0 and the input.

Use the keyword argument input_shape (tuple of integers, does not include the batch axis) when using this layer as the first layer in a model.tensor.

- Output shape
- Same shape as the input.
- Arguments
- max_value: Float >= 0. Maximum activation value. Default to None, which means unlimited.
- negative_slope: Float >= 0. Negative slope coefficient. Default to 0.
- threshold: Float >= 0. Threshold value for thresholded activation. Default to 0.

Softmax activation Function

Softmax is often used as the activation for the last layer of a classification network because the result could be interpreted as a probability distribution. The softmax of each vector x is computed as exp(x) / tf. reduce_sum(exp(x)) . The input values in are the log-odds of the resulting probability.

Compile and fit the data using the keras functions

```
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

# train model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
batch_size=32, epochs=50)
```

Optimizer ADAM

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled “Adam: A Method for Stochastic Optimization“. I will quote liberally from their paper in this post, unless stated otherwise.

The algorithm is che name Adam is derived from adaptive moment estimation.When introducing the algorithm, the authors list the attractive benefits of using Adam on non-convex optimization problems, as follows:alled Adam. It is not an acronym and is not written as “ADAM”.

Straightforward to implement.

Computationally efficient

Little memory requirements.

Invariant to diagonal rescale of the gradients.

Well suited for problems that are large in terms of data and/or parameters.

Appropriate for non-stationary objectives.

Appropriate for problems with very noisy/or sparse gradients.

Hyper-parameters have intuitive interpretation and typically require little tuning.

Loss Sparse categorical crossentropy

Computes the cross-entropy loss between true labels and predicted labels.Use this cross-entropy loss for binary (0 or 1) classification applications. The loss function requires the following inputs:

y_true (true label): This is either 0 or 1.

y_pred (predicted value): This is the model's prediction, i.e, a single floating-point value which either represents a logit, (i.e, value in [-inf, inf] when from_logits=True) or a probability (i.e, value in [0., 1.] when from_logits=False

Metrics accuracy

The metrics that you choose to evaluate your machine learning algorithms are very important.

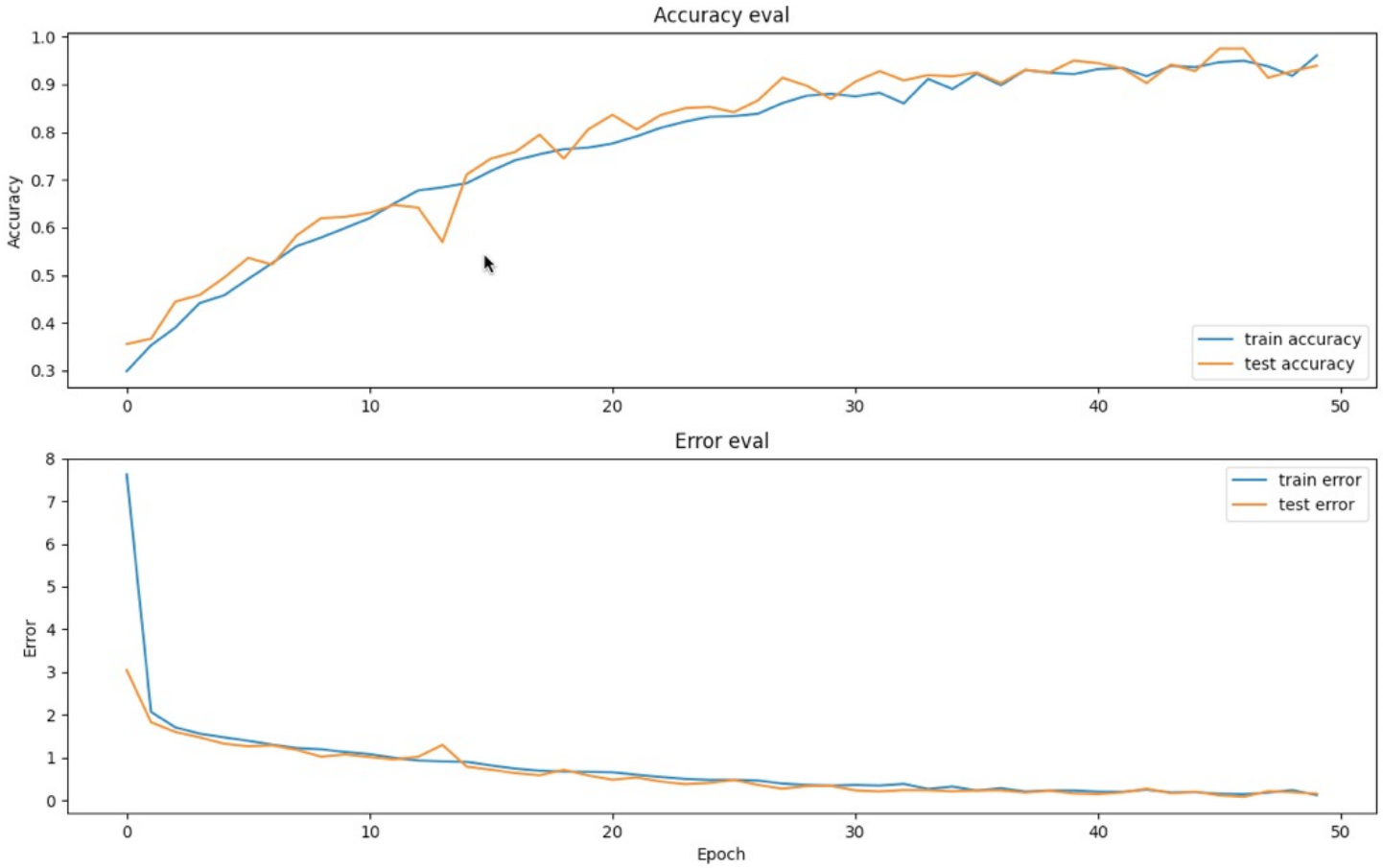
Choice of metrics influences how the performance of machine learning algorithms is measured and compared. They influence how you weight the importance of different characteristics in the results and your ultimate choice of which algorithm to choose.

In this post, you will discover how to select and use different machine learning performance metrics in Python with scikit-learn.

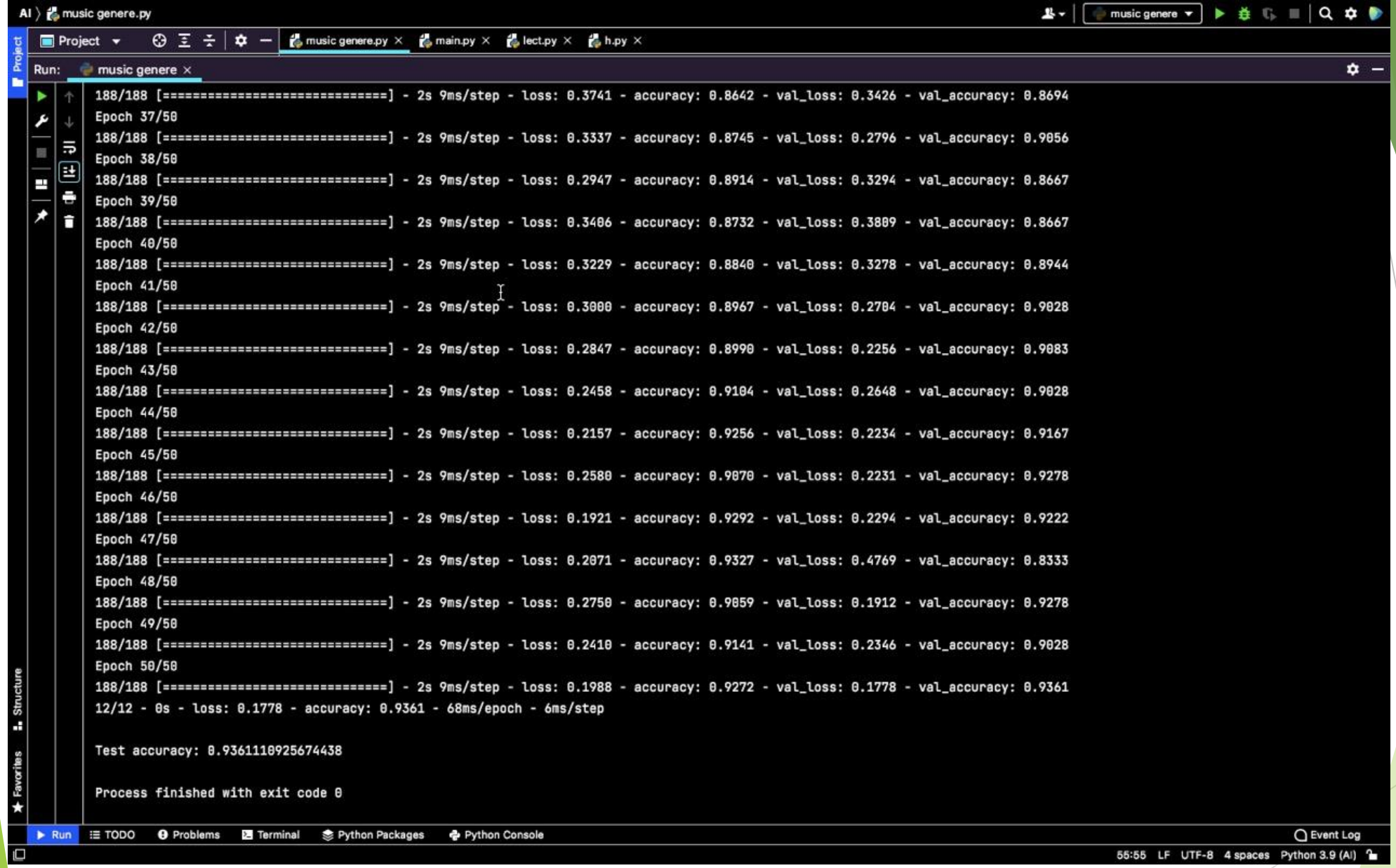
Plotting the precision of the model

for this we have used the matplotlib library.

```
def plot_history(history):    fig, axs = plt.subplots(2)
axs[0].plot(history.history["accuracy"], label="train accuracy")
axs[0].plot(history.history["val_accuracy"], label="test accuracy")
axs[0].set_ylabel("Accuracy")    axs[0].legend(loc="lower right")
axs[0].set_title("Accuracy eval")        axs[1].plot(history.history["loss"],
label="train error")    axs[1].plot(history.history["val_loss"], label="test
error")    axs[1].set_ylabel("Error")    axs[1].set_xlabel("Epoch")
axs[1].legend(loc="upper right")    axs[1].set_title("Error eval")
plt.show()
```


Accuracy eval
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50
train accuracy
test accuracy
Error eval
Error
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
Epoch
train error
test error

x=14.74 y=0.538



As shown in the above image the accuracy of the model is 93.61%

The data taken in these case is 93 to 7 percentage for train and test respectively.

The End