**Music Generation Using Machine Learning**

**Guide:**

Dr. Arpita Gupta

**Presenters:**

Jyothin Movva (2010030071)

SatyaVarsan Kattuva (2010030151)

Devaraj Acharya (2010030040)

**Introduction:**

* The project is about music generation. Music generation is very important now. It can be used in many applications.
* Automatic Music Generation is a process of composing a short piece of music with minimum human intervention.
* Music Generation a viable tool that can and is being used by producers to help in the creative process.
* This is achieved by recombination of musical phrases extracted from existing music, either live or pre-recorded.

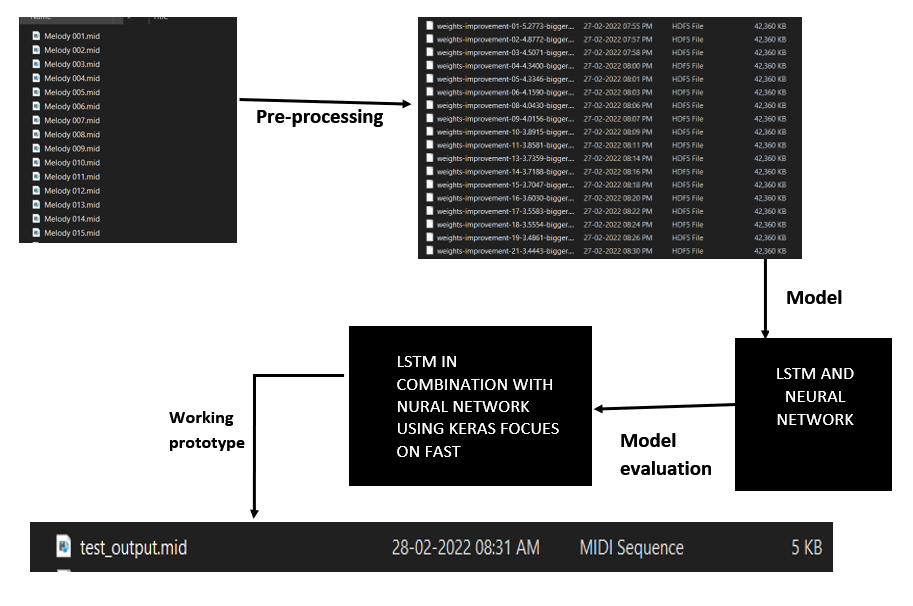
**Objective:**

* Train a model to generate new music by giving a sample tracks.
* To accomplish music generation using LTSM.

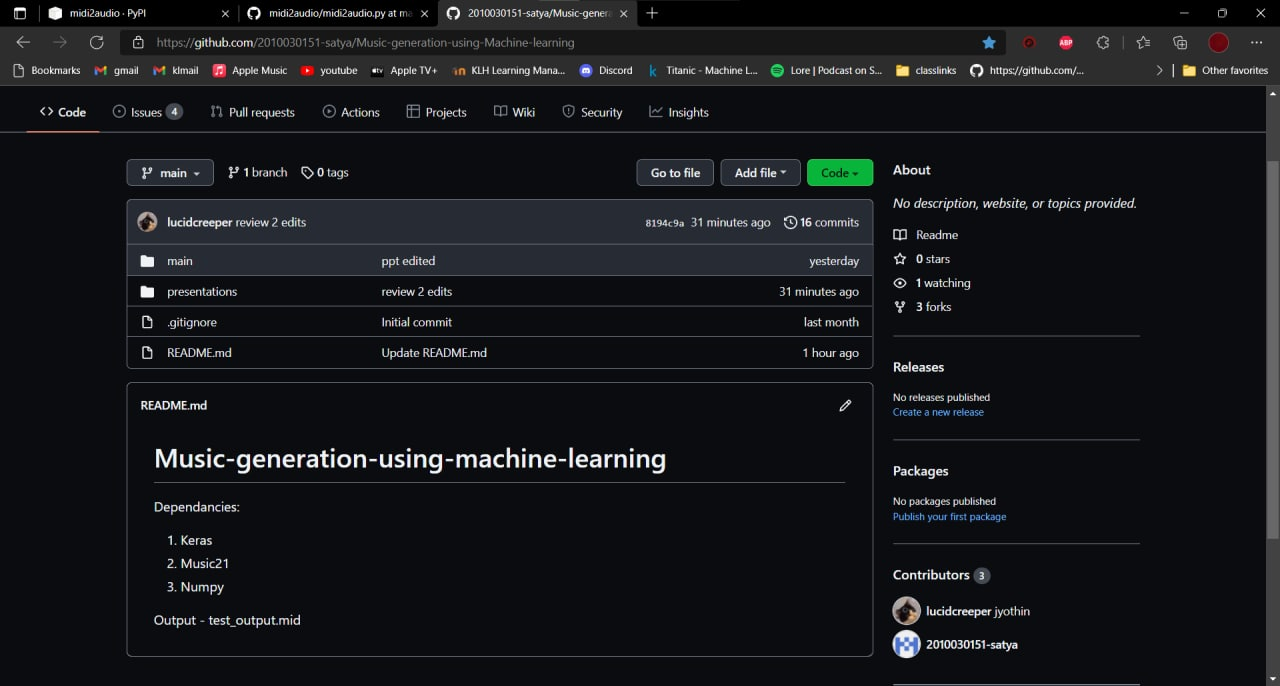
**Dataset and Techniques:**

|  |  |  |
| --- | --- | --- |
| Dataset | Characteristics | Models and Technique |
| Midi melody files for theme song | 60 midi files | LTSM |
| Midi piano files | 92 midi files | LTSM |
| Lakh piano-roll dataset | 174,154 multi track piano rolls | 1)JamBot (LSTM)  2)Convolutional Generative Adversarial Networks with Binary Neurons for Polyphonic Music Generation (GAN)  3)MuseGAN (GAN) |
| Lo-Fi Hip Hop MIDIs | 93 Midi files | Lo-Fi Hip Hop Generation (LTSM) |
| Multi-modal MIREX Emotion Dataset | 193 Midi files |  |

**Flowchart:**



**GitHub Repository:**



**Selected Model:**

LTSM: Long Short Term Memory

Webhosting: Django

**Implementation:**

**from** django.shortcuts **import** redirect, render

**from** .models **import** Document

**from** .forms **import** DocumentForm

**import** pickle

**import** numpy

**from** music21 **import** instrument, note, stream, chord

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense

**from** keras.layers **import** Dropout

**from** keras.layers **import** LSTM

**from** keras.layers **import** BatchNormalization **as** BatchNorm

**from** keras.layers **import** Activation

*def* generate():

    """ Generate a piano midi file """

    #load the notes used to train the model

**with** open('data/notes', 'rb') **as** filepath:

        notes **=** pickle.load(filepath)

    # Get all pitch names

    pitchnames **=** sorted(*set*(item **for** item **in** notes))

    # Get all pitch names

    n\_vocab **=** len(*set*(notes))

    network\_input, normalized\_input **=** prepare\_sequences(notes, pitchnames, n\_vocab)

    model **=** create\_network(normalized\_input, n\_vocab)

    prediction\_output **=** generate\_notes(model, network\_input, pitchnames, n\_vocab)

    create\_midi(prediction\_output)

*def* prepare\_sequences(*notes*, *pitchnames*, *n\_vocab*):

    """ Prepare the sequences used by the Neural Network """

    # map between notes and integers and back

    note\_to\_int **=** *dict*((note, number) **for** number, note **in** enumerate(pitchnames))

    sequence\_length **=** 100

    network\_input **=** []

    output **=** []

**for** i **in** range(0, len(notes) **-** sequence\_length, 1):

        sequence\_in **=** notes[i:i **+** sequence\_length]

        sequence\_out **=** notes[i **+** sequence\_length]

        network\_input.append([note\_to\_int[char] **for** char **in** sequence\_in])

        output.append(note\_to\_int[sequence\_out])

    n\_patterns **=** len(network\_input)

    # reshape the input into a format compatible with LSTM layers

    normalized\_input **=** numpy.reshape(network\_input, (n\_patterns, sequence\_length, 1))

    # normalize input

    normalized\_input **=** normalized\_input **/** *float*(n\_vocab)

**return** (network\_input, normalized\_input)

*def* create\_network(*network\_input*, *n\_vocab*):

    """ create the structure of the neural network """

    model **=** Sequential()

    model.add(LSTM(

        512,

*input\_shape***=**(network\_input.shape[1], network\_input.shape[2]),

*recurrent\_dropout***=**0.3,

*return\_sequences***=**True

    ))

    model.add(LSTM(512, *return\_sequences***=**True, *recurrent\_dropout***=**0.3,))

    model.add(LSTM(512))

    model.add(BatchNorm())

    model.add(Dropout(0.3))

    model.add(Dense(256))

    model.add(Activation('relu'))

    model.add(BatchNorm())

    model.add(Dropout(0.3))

    model.add(Dense(n\_vocab))

    model.add(Activation('softmax'))

    model.compile(*loss***=**'categorical\_crossentropy', *optimizer***=**'rmsprop')

    model.load\_weights('media/weights.hdf5')

**return** model

*def* generate\_notes(*model*, *network\_input*, *pitchnames*, *n\_vocab*):

    start **=** numpy.random.randint(0, len(network\_input)**-**1)

    int\_to\_note **=** *dict*((number, note) **for** number, note **in** enumerate(pitchnames))

    pattern **=** network\_input[start]

    prediction\_output **=** []

**for** note\_index **in** range(500):

        prediction\_input **=** numpy.reshape(pattern, (1, len(pattern), 1))

        prediction\_input **=** prediction\_input **/** *float*(n\_vocab)

        prediction **=** model.predict(prediction\_input, *verbose***=**0)

        index **=** numpy.argmax(prediction)

        result **=** int\_to\_note[index]

        prediction\_output.append(result)

        pattern.append(index)

        pattern **=** pattern[1:len(pattern)]

**return** prediction\_output

*def* create\_midi(*prediction\_output*):

    offset **=** 0

    output\_notes **=** []

**for** pattern **in** prediction\_output:

**if** ('.' **in** pattern) **or** pattern.isdigit():

            notes\_in\_chord **=** pattern.split('.')

            notes **=** []

**for** current\_note **in** notes\_in\_chord:

                new\_note **=** note.Note(*int*(current\_note))

                new\_note.storedInstrument **=** instrument.Piano()

                notes.append(new\_note)

            new\_chord **=** chord.Chord(notes)

            new\_chord.offset **=** offset

            output\_notes.append(new\_chord)

**else**:

            new\_note **=** note.Note(pattern)

            new\_note.offset **=** offset

            new\_note.storedInstrument **=** instrument.Piano()

            output\_notes.append(new\_note)

        offset **+=** 0.5

    midi\_stream **=** stream.Stream(output\_notes)

    midi\_stream.write('midi', *fp***=**'test\_output.mid')

*def* my\_view(*request*):

    message **=** ''

    # Handle file upload

**if** request.method **==** 'POST':

        generate()

        form **=** DocumentForm(request.POST, request.FILES)

**if** form.is\_valid():

            newdoc **=** Document(*docfile***=**request.FILES['docfile'])

            newdoc.save()

            # Redirect to the document list after POST

**return** redirect('my-view')

**else**:

            message **=** 'The form is not valid. Fix the following error:'

**else**:

        form **=** DocumentForm()  # An empty, unbound form

    # Load documents for the list page

    documents **=** Document.objects.all()

    # Render list page with the documents and the form

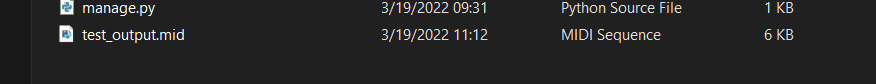
    context **=** {'documents': documents, 'form': form, 'message': message}

**return** render(request, 'list.html', context)

**Results:**

Graphical user interface, text, application

Description automatically generated



A screenshot of a computer

Description automatically generated with medium confidence