UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Music Recognition Using Convolutional Neural Networks

propusă de

Andrei Vavilov

Sesiunea: Iulie, 2021

Coordonator științific

Conf. Dr. Vitcu Anca

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Music Recognition Using Convolutional Neural Networks

Andrei Vavilov

Sesiunea: Iulie, 2021

Coordonator științific

Conf. Dr. Vitcu Anca

Avizat,

Îndrumător lucrare de licență,

Conf. Dr. Vitcu Anca.

Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Vavilov Andrei** domiciliat în **România, jud. Iași, orș. Târgu Frumos, Strada Tudor Vladimirescu, nr. 59**, născut la data de **05 iulie 1999**, identificat prin CNP **1990705225638**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2021, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Music Recognition Using Convolutional Neural Networks** elaborată sub îndrumarea doamnei **Conf. Dr. Vitcu Anca**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Music Recognition Using Convolutional Neural Networks**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Andrei Vavilov**

Data: ..... Semnătura: .....

Contents

Motivation

In the age of *big data* and immense computational power, artificial intelligence has come to be the new standard in the computer science field. Various types of data can be understood, learnt, predicted and even produced by a well-tuned neuronal network, making the principles of machine learning a must for a scientist nowadays.

Applications of neuronal networks can be found in any discipline: from medicine to physics, social sciences and languages. The purpose of this thesis is to depict how artificial intelligence can find its place and purpose in a previously profoundly human field: art.

Introduction

The current thesis is an attempt to materialize the intersection of artificial intelligence and arts, especially, music.

The application consists of two major parts:

- First, the machine learning component, represented by a neuronal network framework, implemented from scratch (i.e. without the explicit support of existent frameworks or libraries). The module presents the necessary functionalities for constructing a neuronal network: layers, activation functions, metrics, optimizers, models and support for data generation and preprocessing.

- Second, the visual support, compound of two illustrative animations, created in Autodesk Maya 2019, and a Graphical User Interface, in which a hypothetical user can interact with the application by feeding it a YouTube link of a song. As a result, the application will decide whether the input song is played on a piano or not (prediction Piano/Other). Depending on the decision of the neuronal network, the corresponding 3D animation will be played.

0.1 Context

The first analogy between the way computers can process information and the way the human brain works (as we know, at least, to this day), has been made by Warren McCullough and Walter Pitts in 1944, who later became the founding members of what is sometimes called the first cognitive science department [7]. The primary idea

is elegant in its simplicity: the neuron, the basis of the human cognitive apparatus, can be modelled in machine learning as an unit in a network.

Since then, numerous studies and breakthroughs have been made, as well as various frameworks and tools which make implementing a neuronal network accessible without possessing the full mathematical background needed prior.

0.2 Purpose and modus operandi

One of the purposes of this thesis is to implement the functionalities and the logic behind a neuronal network, as well as creating and fine-tuning a model. In order to complete this task, we consulted multiple sources(e.g. [9],[17],[13]) which presented the theoretical and mathematical aspects of constructing the aforementioned classifier. The implementation was constructed with the support of various tools from the Python programming language (e.g. Numpy, Librosa, Tenserflow Keras, Kapre), which will be extensively discussed in the following chapters.

As discussed before, the second component of the thesis regards the visual part of the application. For obtaining an interactive use of the project, we used Autodesk Maya 2019 (for creating the animations) and Python tkinkter for creating a simplistic GUI.

The logical flow of the pipeline is as follows: the user feeds a YouTube link of a song to the GUI. The model (previously trained and saved) computes a prediction regarding the category under which the input falls (Piano/Other). Given this result, the corresponding animation is played.

0.3 Personal contributions

Numerous attempts of creating a medium between artificial intelligence and other disciplines have been made since the rise of this field. Arts, especially music, is no exception.

The particularity of the current thesis is the approach we had in completing the task: implementing from scratch the neural network framework, and, implicitly, understanding the mathematical and theoretical subtleties of it, as well as creating the visual aid which aims to touch on (although briefly) 3D animations.

0.4 Structure

The structure of the present thesis follows the major constituent parts described before and is as follows:

1. Chapter One
    - Similar applications : in which other akin projects are mentioned;
2. Chapter Two
    - Technologies used : in which technologies needed for creating the application and their purpose and functionalities are discussed;
3. Chapter Three
    - Architecture and implementation : in which the actual implementation is discussed explicitly. This section contains technical and theoretical aspects of the thesis.
4. Chapter Four
    - Use cases: in which a part of the functionalities of the project are presented;

0.5 Acknowledgements

I would like to express my special thanks of gratitude to the academical staff of the Faculty of Computer Science Iași for the opportunity to do this project.

Obviously, the present thesis would not be possible without the help and guidance of the professor that directed it, PhD. Anca Vitcu.

Chapter 1

Similar applications

Neural Network Models as well as Machine Learning algorithms provide various versatile and adaptive methods for non-linear problem solving. Because of these reasons, there are numerous applications which perform the task of musical/audio classification using the aforementioned techniques. We will discuss in the following sections about three of those applications.

1.1 Spektrum

Spektrum is a multi-platform music genre classificator and music recommandation system developed in the context of the course "Application Challenges for Machine Learning on the example of IBM Power AI", by the team consisting of Marte Vinje, Moritz Klimmek, Thomas Salzer, Aaron Hümmecke, Lukas Vorwerk [16] . The application consists of two parts:

- Music Genre Classification: Performed by a Convolutional Neural Network Model which analyzes the MEL spectrogram of a given input song.
- Music Recommendation: Generating suggestions by making use of a combination collaborative filtering and content based filtering.

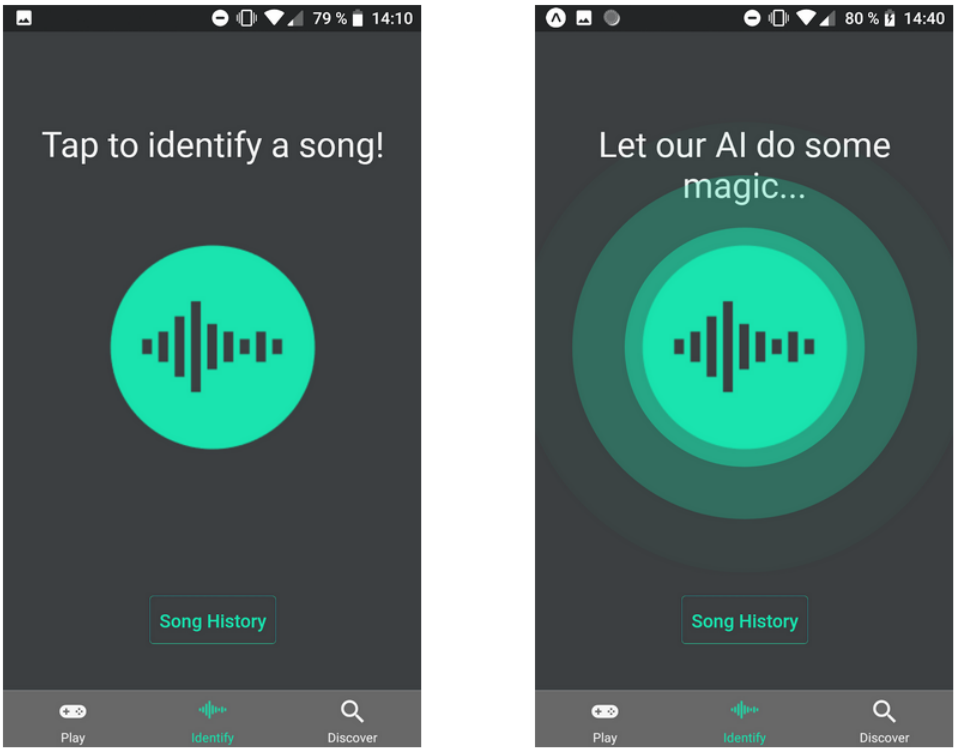

Figure 1.1: The interface of the Spektrum application (Photo source: [16])

1.2 MuseNet

MuseNet is a Deep Neural Network Framework developed by OpenAI that can generate 4-minute snippets of original musical composition with up to 10 different instruments, combining various styles ranging from Mozart to The Beatles. MuseNet creates music by determining the patterns over a given style as input, generating the respective sequence of notes and chords. It also computes a relational graph between its various current styles, in order to incorporate as many related musical features as possible.

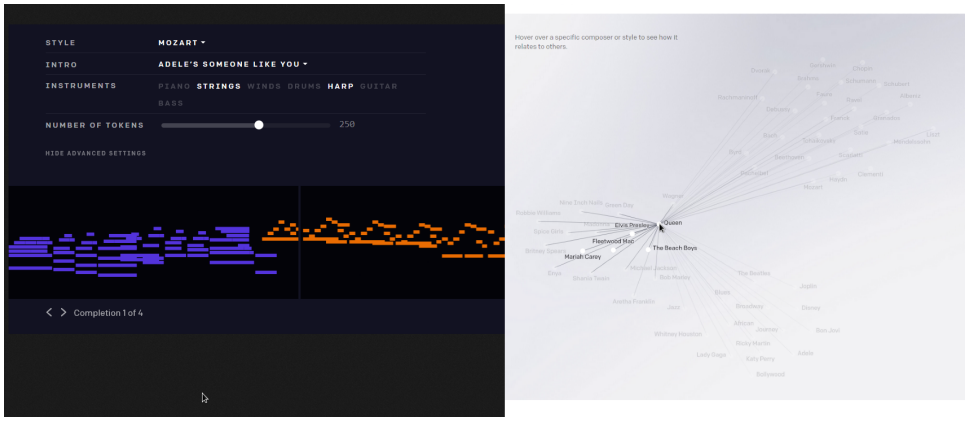

Figure 1.2: An illustration of the MuseNet's functionalities (Photo sources : [15])

1.3 Magenta

Magenta is an open source project (started by a group of engineers from the Google Brain team), based on deep learning and reinforcement learning used for generating and creating art. By using the Google Tensorflow as well as Magenta.js, the application produces original songs, images, drawings and material textures.

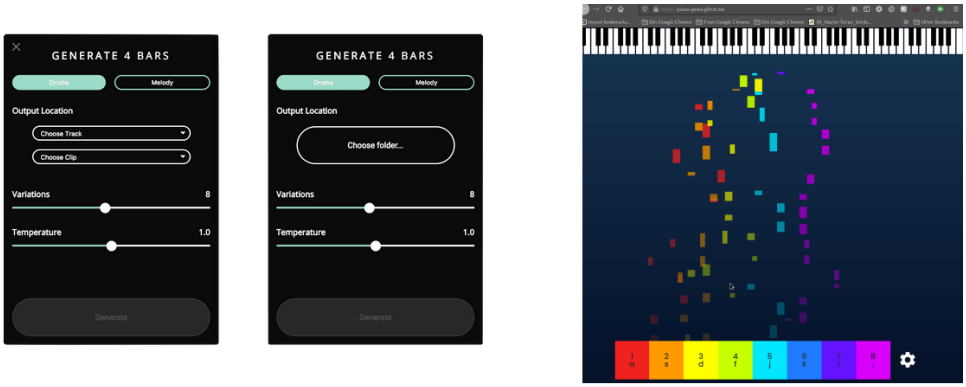

Figure 1.3: Applications based on Magenta (Photo sources: [14])

Chapter 2

Technologies used

Python provides many modules for numeric calculus, data processing and manipulation as well as tools for user interaction. In the following chapter we will briefly touch on the modules and libraries used for achieving the goal of the application, instrument classification.

2.1 Scipy

SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering. The SciPy library contains a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics, and much more.[4] The aforementioned library's submodule *Input and output* provides a variety of ways of reading and writing data in numerous file formats. For the data processing part of the application, we had chosen the *scipy.io.wavfile.read* function because it facilitates the data manipulation process by being compatible with the numpy module, mentioned below (the output of the *scipy.io.wavfile.read* function is a tuple consisting of the sample rate(int) and the audio data(as a *ndarray*).

2.2 NumPy

Numpy is a Python library (part of the SciPy ecosystem) that provides the needed tools in order to perform the mathematical operations behind the Neural Network Framework. The core of the module is the *ndarray* object, which encapsulates an optimized n-dimensional array. Thus, we use *ndarray* objects for computing and storing

the various operations performed throughout the application (e.g. the dot product, the data reshaping). The *ndarray* array is represented as a matrix, whose dimensions are referred to as *shape*. Some of the advantages brought by the numpy module are:

- Forward and backward operation compatibility for the various objects contained within the application
- Fast execution time caused by the optimization of the library [3] (by using pre-compiled C code)
- Support for large number/big data processing.

2.3 TensorFlow

TensorFlow is an open source platform for machine learning applications. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.[5] From the vast amount of methods and submodules contained within TensorFlow, we used Keras, an utilitary library built on top of the TensorFlow API. In order for the Neural Network Framework to be able to handle large amounts of data, we inherited in our custom *DataGenerator class* the *tensorflow.keras.utils.Sequence* class. The *Keras DataGenerator class* along with the custom Neural Network Framework *DataGenerator* provide among other functionalities the *getitem* method, which yields a batch of data at a given index, thus overcoming the handling of large data amounts.

2.4 Kapre

Kapre (Keras Audio Preprocessors) is an audio data preprocessing library built on top of *Keras*, specialized on audio signal handling.[1] Kapre facilitates sound transformation by efficiently performing various operations (e.g. *Short Time Fourier Transformation*, *Mel scaling*) in a GPU optimized consistent manner . For the input data modeling we had chosen the *get mel spectrogram layer* function from the module *kapre.composed*. It applies sequentially *STFT, Magnitude and mel filterbank* transformations over the given input data, returning a mel spectrogram with an user specified sample rate and output shape.

2.5 Librosa

Librosa is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems [11]. Out of the Librosa API we have chosen the following tools :

- *resample*: A function that resamples an audio file from its original rate to an user given sample rate.
- *to mono*: A function which converts a multi channel signal to mono signaling by averaging the sample values.

2.6 youtube-dl

Youtube-dl is an open source tool for downloading videos and songs from the YouTube platform. In the context of the final application, the youtube-dl module provides the end user with the possibility of classifying a song just by inserting a link to a YouTube video.

2.7 Tkinter

Tkinter is the standard Python GUI Toolkit, a simple yet versatile module was used for the minimalistic Graphic User Interface of the final application.

2.8 Autodesk Maya

Maya is a software program developed by Autodesk which provides a complex environment for 3D modeling and animation. In the context of the final application we have chosen Maya for projecting and animating the 3D visualisation of the classification.

Chapter 3

Theoretical Aspects

Regarding the theoretical aspects of the current paper, we present the following concepts:

3.1 Neuronal Network

The preliminary principle of a *neuron* consists in a set of weights and a set of *biases*. *Weights* are a set of numerical values (initially arbitrary) which throughout the passing of the input through the Neural Network will be transformed and later, at the end of the training process will be saved (basically representing the model). *Biases*, similar to the intercept of a linear equation, represent additional parameters to the network whose purpose is to adjust the output of the weighted sum of the input.

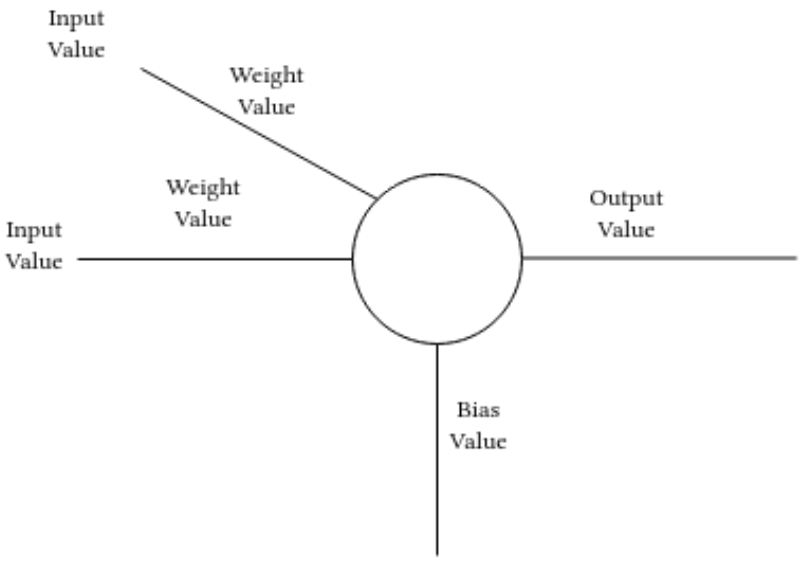

Figure 3.1: Illustration of a generic neuron

A Neuronal Network is a set of *neurons* (as previously defined), grouped in *layers*, during the traversal of which the process of learning takes place. Depending on the type of layers as well as their configuration multiple various operations regarding the input feeded take place.

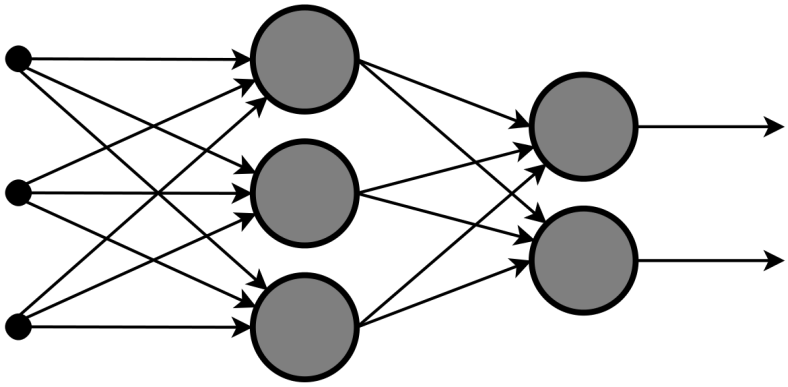

Figure 3.2: Illustration of a generic neural network (Image source [6])

3.2 Layers

As seen before, a layer is a set of neurons, which is described by an input (having a specific shape) and an output with a shape resulting from the specific calculations.

Throughout a Neural Network architecture we may find multiple types of layers such as: Fully connected layer(*Dense* layers), *Flattening* layers, *2D Convolutional* layers, *Batch Normalization* layers etc. Of these aforementioned layers, we will go into the specifics of *Batch Normalization*.

Batch Normalization

A *batch* of data is a subset of the training data with a fixed size. Normalization is a technique which involves mapping a given set of values to a new one, preserving its value ratios.

BatchNormalization is a technique for standardizing the inputs receieved from the previous layer, with the effect of stabilizing the learning process and reducing the number of training epochs required. The BatchNormalization layer works by applying a linear scale and then shift the result of the batch, following the formula:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 3.3: The Batch Normalization formula(Image source [10])

The Batch Normalization layer formula is composed of three components:

- gamma: The layer scaling factor.
- x̂ : The output of the normalization process, computed by subtracting a batch element by the mean of the whole batch and then dividing the output by the standard variation of the batch.
- beta: The layer shifting factor.

3.3 Activation Functions

The activation function is a function added to a layer which enables the Neural Network to fit nonlinear problems. The activation functions fall into 2 categories:

- Linear(e.g. ReLU) which emulates the *firing* of a neuron by checking whether or not the function output is below a critical value.
- Non-Linear(e.g. Sigmoid) used most of the time as an activation function for the last layer, because it normalizes the values given between 0 and 1, thus producing an informative output in regards to the confidence of the model prediction.

We will now discuss the *Rectified Linear Unit(ReLU)* function:

Rectified Linear Unit(ReLU)

ReLU is an activation function which clips negative input values using the formula illustrated below. Among the advantages of the ReLU function we mention the simplicity of the computation and the fact that ReLU prevents the *vanishing gradient problem*(the invalidation of the neuron values during backpropagation due to the derivation process).

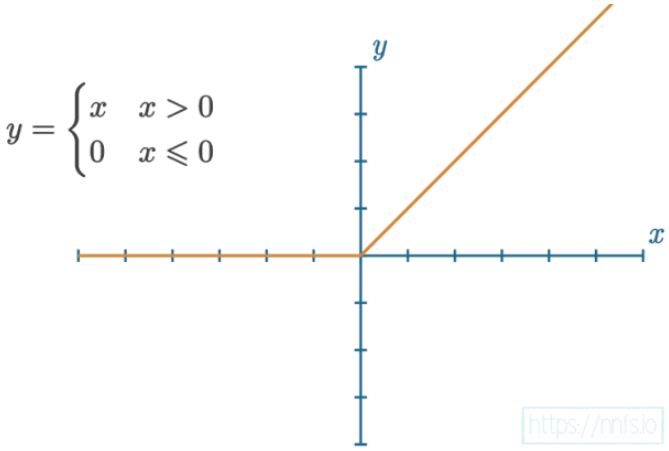

Figure 3.4: The ReLU formula and graphic illustration(Image source [9])

3.4 Optimizers

Optimizers are algorithms used to improve the parameters of a given neural network (such as weights and biases) in order to reduce the *model loss value*. Depending on the type of the problem to solve we can choose from multiple optimizers such as SGD (Stochastic Gradient Descent), RMSProp (Root Mean Squared Error Propagation), Adagrad(Adaptive Gradient Algorithm) etc.

Stochastic Gradient Descent(SGD)

The SGD optimizers works by slicing a batch of data into individual inputs then updating the weights and biases of a layer by subtracting the product of the *learning rate*(a parameter that controls the rate at which a model changes) and the backpropagation output.

3.5 Metrics

The metrics function provides the means for analyzing and interpreting the performance of a Neural Network. The two most utilised metrics in Neural Network analysis are:

- *Loss*: the cost function of the network with the main goal of computing the incorrectness of the model prediction . Common loss functions include *Cross-Entropy Loss*, *Categorical Cross-Entropy*, *Mean Square Error*.

- *Accuracy*: is a metric for evaluation of the model classification, computed by dividing the number of correct predictions by the total number of predictions. The accuracy is the most "human readable" metric, illustrating the performance of the model in terms of successful predictions. Amongst the accuracy functions we recall the *Categorical Accuracy* and *Regression Accuracy*.

In order to accuracy as well as other related metrics (*sensitivity,recall,precision*) we defined the following notations:

		Actual Values: positives	Actual Values: negatives
Predicted Values	positives	TP True Positive	FP False Positive
Predicted Values	negatives	FN False Negative	TN True Negative

Figure 3.5: Confusion matrix.

Thus we compute the accuracy using the following formula:

$$accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

Chapter 4

Architecture and implementation

The application is divided into three parts:

- The Neuronal Network Engine
- The User Input Handling and Processing
- The Application-User interaction via the GUI

We will continue this chapter by entering into the details of each of the aforementioned components.

4.1 The Neuronal Network Engine

The Neural Network Engine was developed using the Neural Network Framework which we implemented from scratch. The Neural Network Framework contains various tools for Neural Network problem solving such as layers, optimizers, metrics, etc. The Neural Network Framework operates a *model* file, a binary serialized file which supports the following operations:

- Training: Initializing/improving the performance by performing predictions over a given input data in order to adjust the *weights and biases* (the model *neurons*).
- Inference: Performing predictions over unseen data.
- Loading: Loading a previously trained model in order to perform inference/ continue the training process
- Saving: Creating/writing the results of the previous operations.

The main application problem can be reduced to a binary classification problem (the engine has to predict if a given input falls into the category Piano or Other), thus we chosen the following architecture:

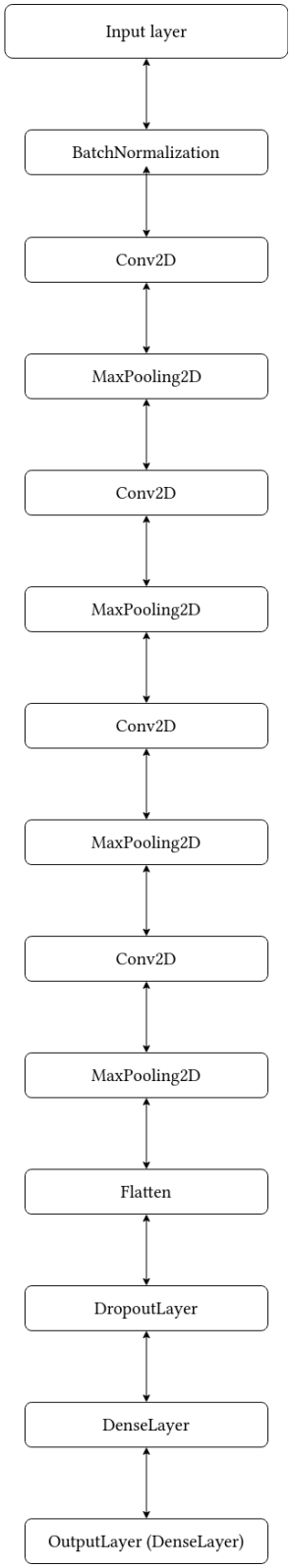

Figure 4.1: A summary of the model architecture

The model is composed of layers, which fall into two categories: trainable (Conv2D, MaxPooling2D, DenseLayer), and non-trainable (InputLayer, FlattenLayer, DropoutLayer, BatchNormalization). The difference between the two categories is the type of the operations performed, and whether or not they facilitate the learning process.

For the trainable layers we chosen the *Adam*(Adaptive Moment estimation) optimizer, which works by combining the *Adagrad* and *RMSProp* optimizers by calculating an exponential moving average of the gradient and the squared gradient, as well as controlling the decay rates of these moving averages.[9]

The activation function we had chosen for the trainable layers is *ReLU*(Rectified Linear Unit), while for the activation layer we preferred the *SoftmaxActivation* because, unlike linear activation functions, the aforementioned activation function returns the probability values normalized between 0 and 1, handling very small negative or exploding values by penalizing them.

The metrics measured throughout the learning process are computed using *CategoricalAccuracy* and *CategoricalCrossentropy*.

The Neural Network Engine consists of a trained model, containing the weights and biases of the trainable layers from the figure 4.1.

The model values and its performance are the result of the training process, which consisted of five epochs over 33602 input values (one second wav audio snippets), presented to the model in the form of *batches* provided by the custom *Data generator*, with each batch containing 32 samples. The batch size as well as other configuration parameters (e.g. *filter shape, pool size, dropout rate*) are arbitrary values chosen after performing the *fine tuning* phase of the training, which consists of repeatedly training models with different configurations (also known as *hyperparameter adjusting*), in order to obtain an optimal configuration.

The data was gathered by web scraping various YouTube playlists using the *youtube-dl* utilitary, obtaining a diverse and feature rich dataset. The ”Piano” category consists of isolated tracks which only contain piano snippets(any other combination of instruments will not be classified as expected by the model), while the ”Other” category contains every other category of sounds.

The distribution of the data is presented in the below figure:

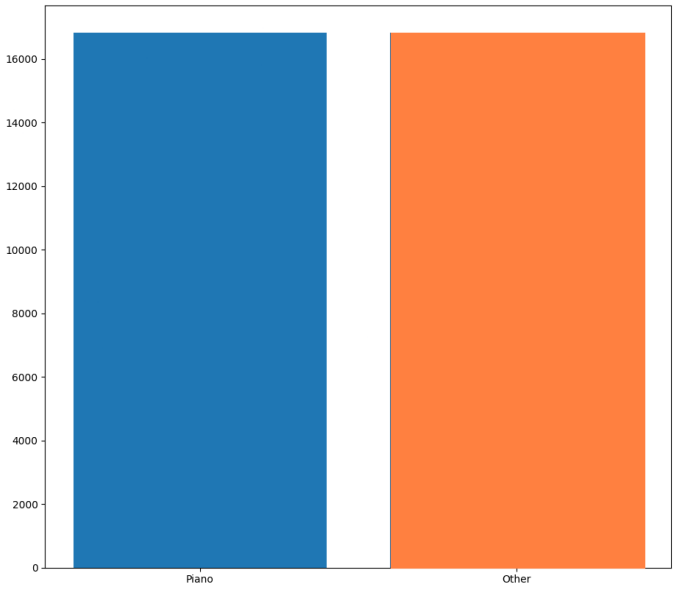

Figure 4.2: Data distribution by number of samples.

Initially we approached the problem without using Convolutional Operations, but after testing numerous configurations the results were below satisfying (with the loss stagnating at a value of about 0.5) and the accuracy fluctuating between 40-60%, as illustrated in the figures below.

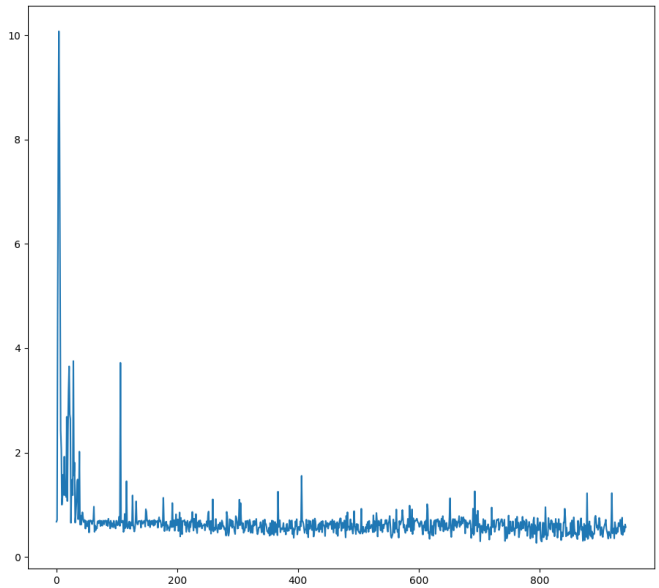

Figure 4.3: Model loss before adding Convolutional Operations to the Model

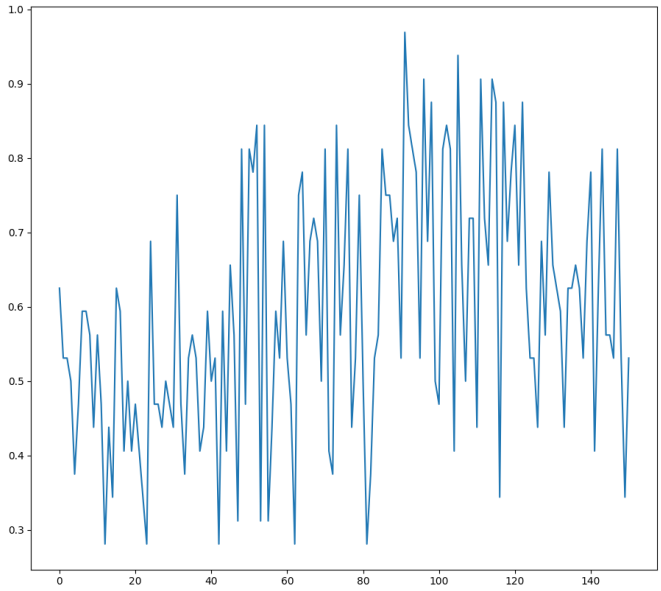

Figure 4.4: Model accuracy before adding Convolutional Operations to the Model

By incorporating the Convolutional Operations and Layers into the Neural Network Framework, the model accuracy as well as the loss improved at a visible rate, as shown below.

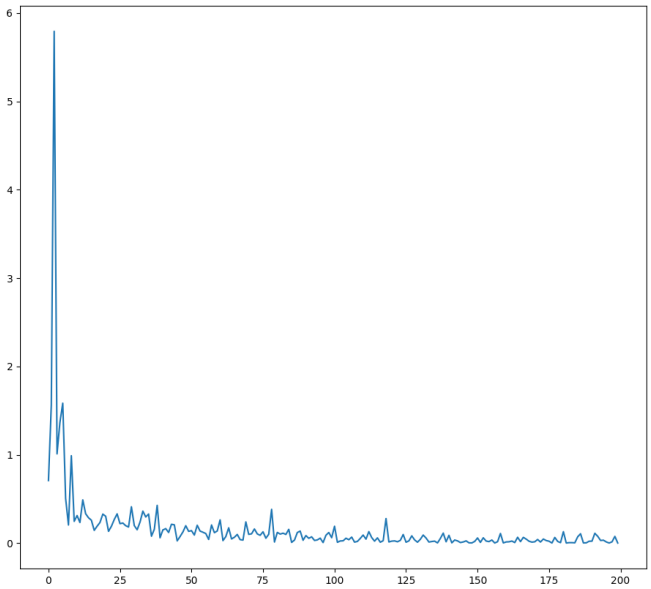

Figure 4.5: The model loss after incorporating Convolutional Layers

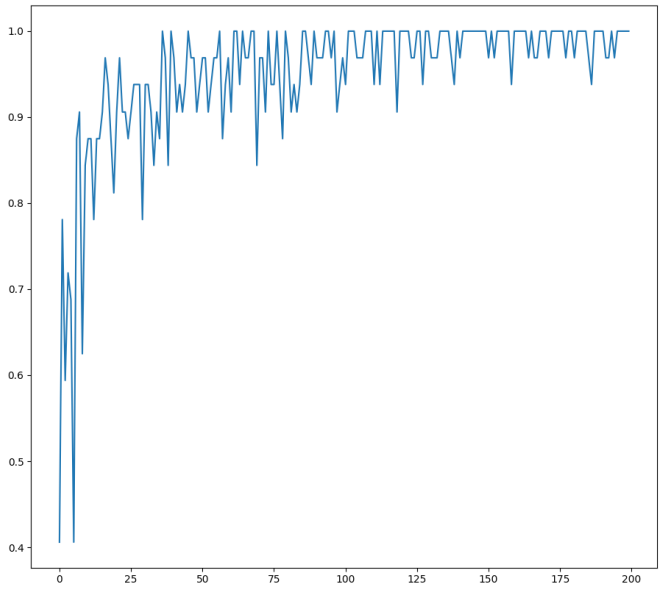

Figure 4.6: The model accuracy after incorporating Convolutional Layers

The general flow of the data in the Neural Network Framework is based on the forward-backward propagation principle, specific to the *Convolutional Neural Network* architecture. Each of the trainable model's layers have the following methods:

1. :function:forward(inputs): A method which takes the previous layer output as the parameter(inputs) and performs the specific operations, thus performing the forward data processing.

2. :function:backward(derived values): A method which takes the previous layer output, which consists of the gradient of the forward propagation end result(derived values), performing the inverse computations in regards to the forward method, and then propagating the end result to the next layer.

3. :parameter:output: A *ndarray* where the *forward* function computations are stored.

4. :parameter:derived input: A *ndarray* where the *backward* function computations are stored.

The following diagram illustrates the forward-backward propagation of the Neural Network Model.

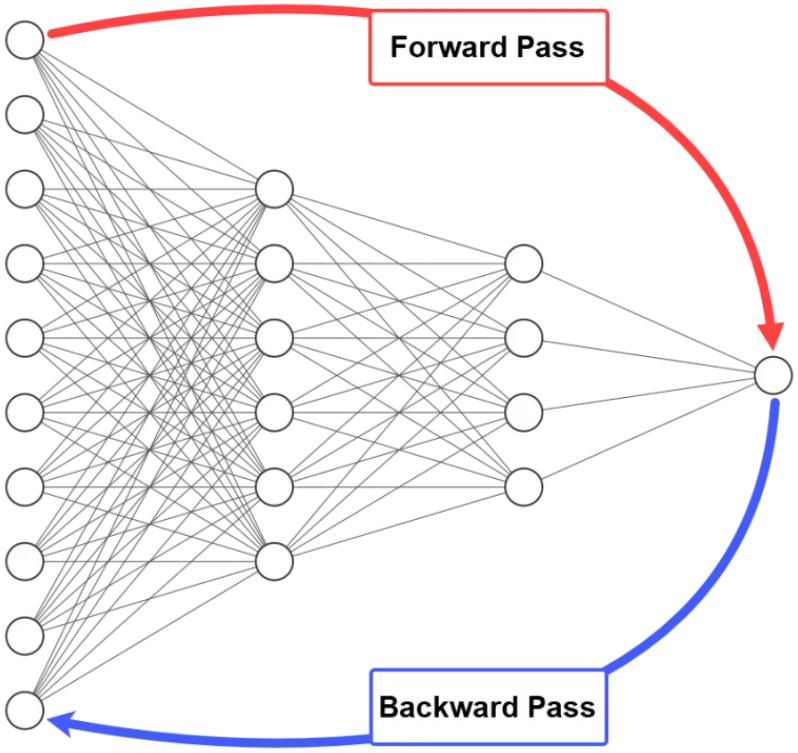

Figure 4.7: Data flow throughout the model (image source [12])

Because the model of the Neural Network Engine is effectively a Convolutional Neural Network, the training process follows the following schema:

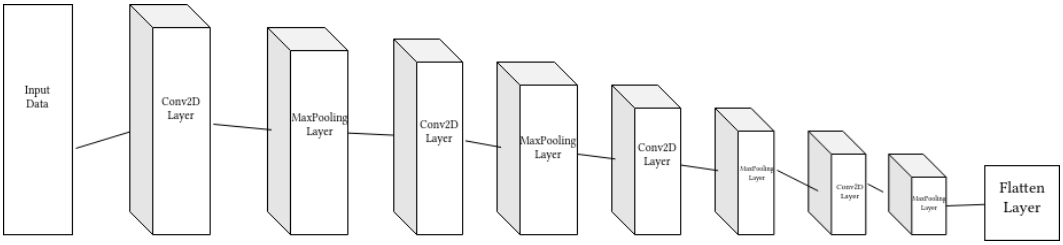

Figure 4.8: Convolutional Neural Network architecture illustration.

At the end of each iteration the model has to classify unseen data (the *validation data*), and based on the results of the classification, the model *metrics* are computed, in the form of loss (computed with *Categorical CrossEntropy*) and accuracy (computed with *CategoricalAccuracy*).

4.2 User Input Handling and Processing

The modus operandi of the input handling and processing pipeline is presented in the following diagram.

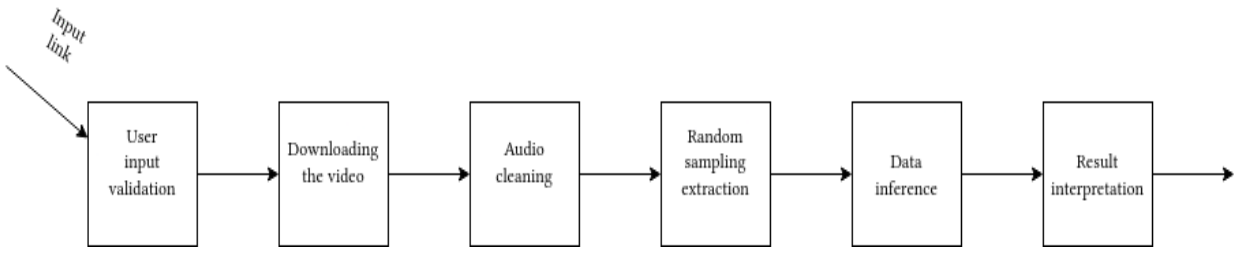

Figure 4.9: User input handling pipeline

The steps performed in order process the user input are as follows:

1. Validating the user input : Checking if the input link inserted by the user is a valid YouTube link to either a video or a playlist.

2. Downloading the video : Saving and converting the video at the specified link in a temporary folder. The *youtube-dl* utilitary facilitates this step by downloading the file using a list of specified configuration parameters such as audio quality and the output format. After this step, we will have obtained an audio file of format wav in a temporary folder, which will be deleted after the finish of the prediction operation.

3. Audio file cleaning: Given the reason that the model was trained on 1 second snippets of audio files of sample rate of 16000, the following operations are necessary to be performed to the audio file in order for it to be compatible with the Neural Network Engine:

    - Downsampling the wav file to mono: Reducing the number of channels of the file to mono with the function *to mono* from *librosa*.
    - Cleaning the wav file by sound envelope.

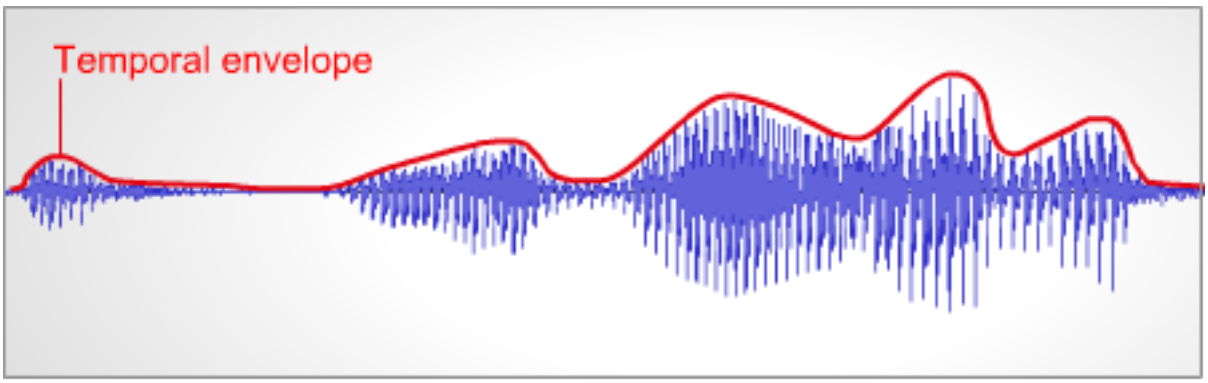

Figure 4.10: Sound envelope example (Source [2])

The sound envelope (presented in the figure 3), represents the changes in audio frequency over a period of time. The *split wav by sound envelope* function computes a *filtered mask* of the audio file by comparing each value of the sound envelope to a given threshold, thus eliminating the audio noise(such as crowd cheering and silence), keeping only the relevant audio information.

- Splitting the masked audio file: After the first two steps of the audio processing, the audio file has to be splitted into 1 second samples and converted to a sample rate of 16000.

4. Extracting a number of random sound samples: In order to offer an accurate prediction and reduce the time cost required by the Neural Network Engine, we randomly extract a number of samples from the splitted audio file.

5. Loading the random sound samples into the model: The Neural Network Engine also presents the *inference* functionality. The inference/prediction is performed by transforming the input data (transformation performed by the input layer), then performing the forward-backward data propagation once. The transformations applied by the input layer are:

   - Short Time Fourier Transformation

     Short Time Fourier Transformation is a mathematical transformation technique used for decomposing a function relative to its spatial and temporal dependencies. Since the audio signal is composed of sound wave which only illustrate the amplitude of a recording, the Fourier transform allows for the creation of an illustration of a frequency over time,called spectrum. [8]

   - MEL Filterbank transformation

     The *MEL* scale is a representation of the sounds that can be heard and distinguished by the human ear. The *MEL spectrogram* is a graphical representation of the sound variations and intensities of the a given audio sample. With the *MEL Filterbank transformation* we effectively represent the relevant information of the input data, thus transforming the problem from an audio classification into an image classification one.

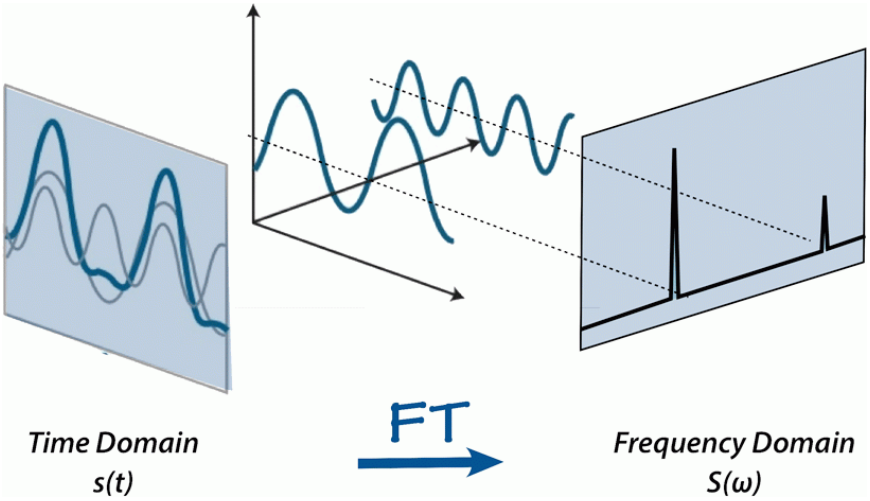

Figure 4.11: Fourier transformation (Source [8])

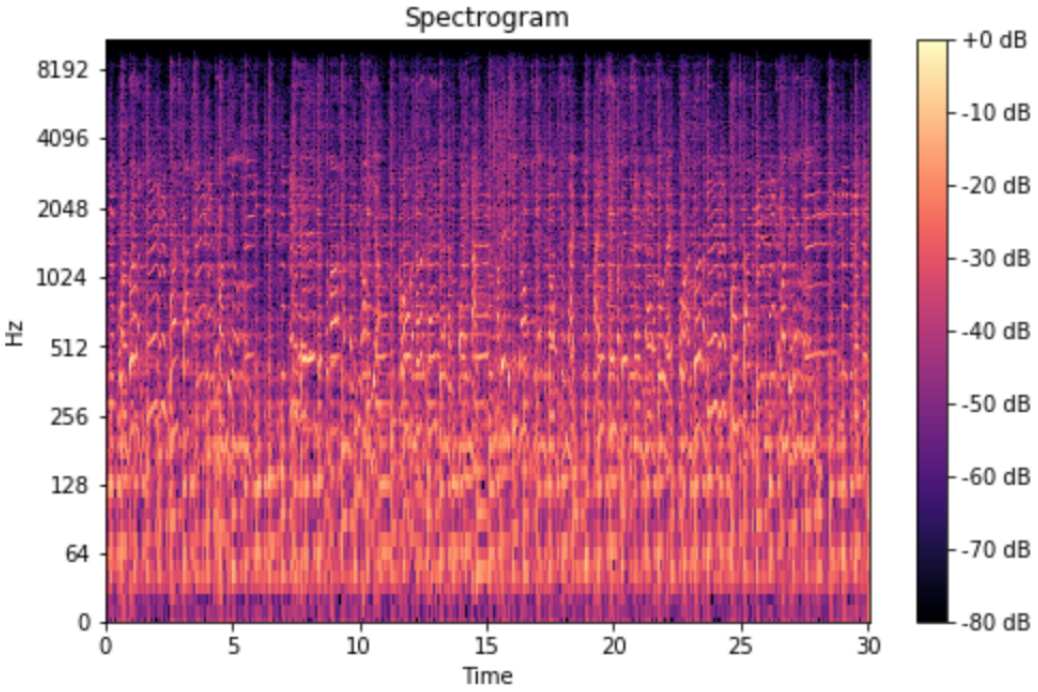

Figure 4.12: MEL Spectrogram example

The output of the transformation is a *ndarray* containing the MEL spectrogram of the input sample which represents the neural network model's input.
The inference output is an array containing floating point numbers which represent the confidence value of the model prediction.
The predicted label is computed by extracting the *argmax*(the index of the element with the biggest value), then mapping it to the specific class name.

6. Interpreting the result and returning the final prediction: The output label is

stored for each prediction extracted from the random samples batch. The final label is computed by extracting the prediction which was suggested most often.

4.3 The Application-User Interaction

The application provides a minimalistic Graphic User Interface which offers the end user the means for classifying an YouTube song, as well as visualise the specific 3D Animation depending on the model prediction.
In order to enrich the visual experience, we have chosen two 3D animations, projected using *Autodesk Maya*.
The main elements of the GUI are the textbox where the user inputs the link to the song, the submit button which grabs the link inserted by the user and begins the inference process and the video stream box, which appears after the model has performed the classification. The application runs over a *main loop*, meaning that the user can input as many links as desired in a session.

The following figure illustrates some of the application states.

Figure 4.13: The Graphic User Interface at the start of the application.

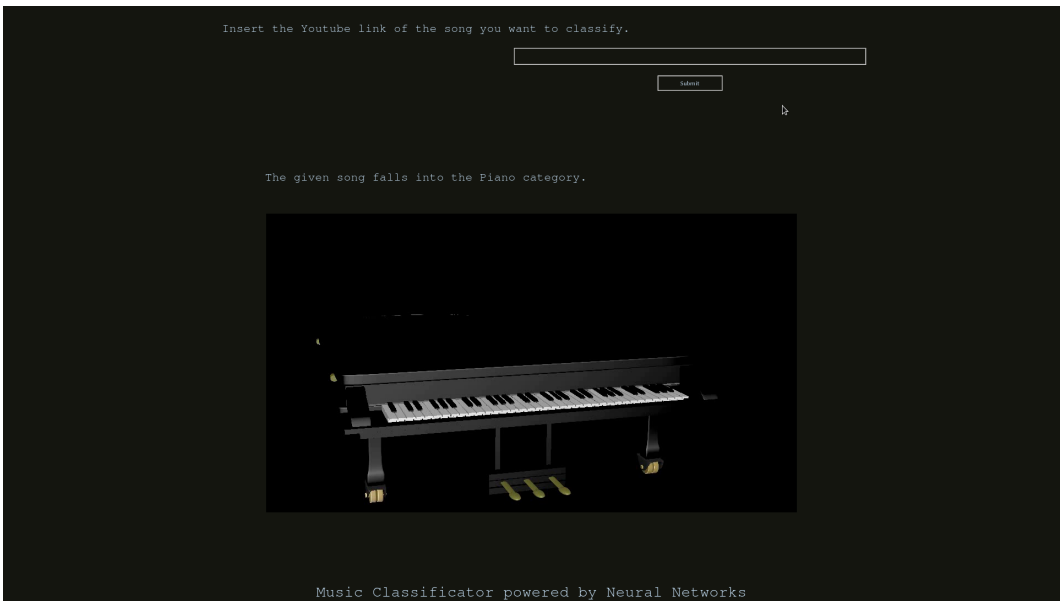

Figure 4.14: The Graphic User Interface after predicting a song as a piano song.

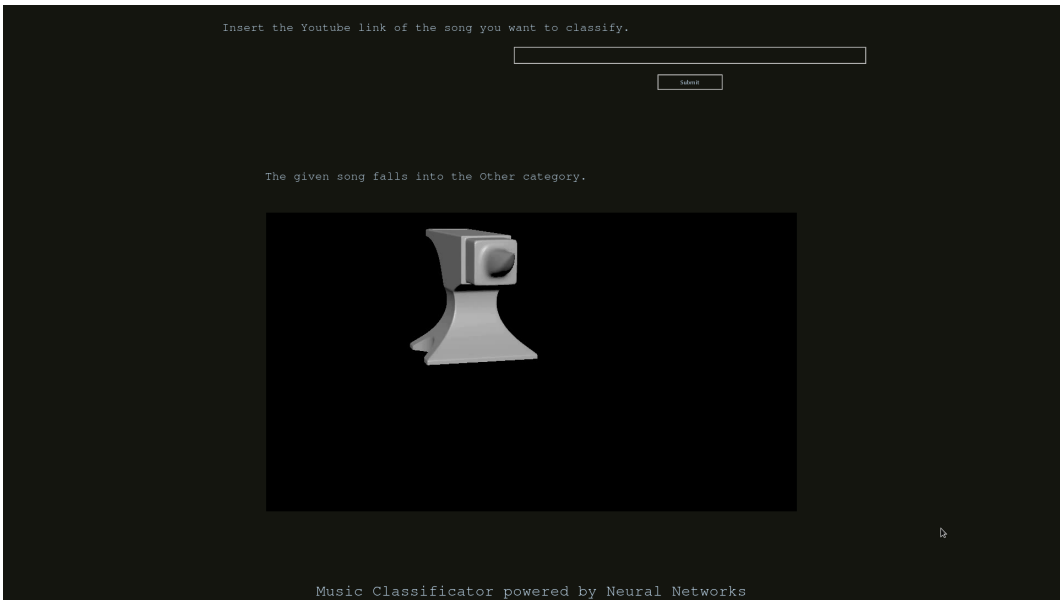

Figure 4.15: The Graphic User Interface after classifying a song as other.

Chapter 5

Use cases

For illustrating the functionality of the application, we propose the following three use cases:

5.1 First use case

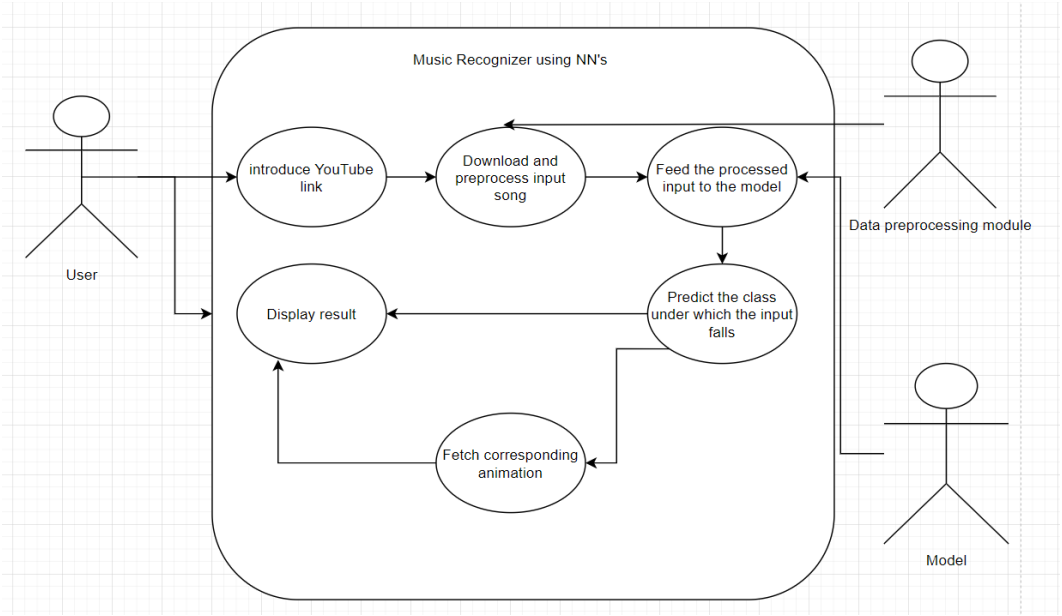

Figure 4.1: Illustration of the behaviour when the user inputs a correct link.

After inserting a valid link, it is inserted into the main application pipeline and the behaviour is the expected one.

5.2 Second use case

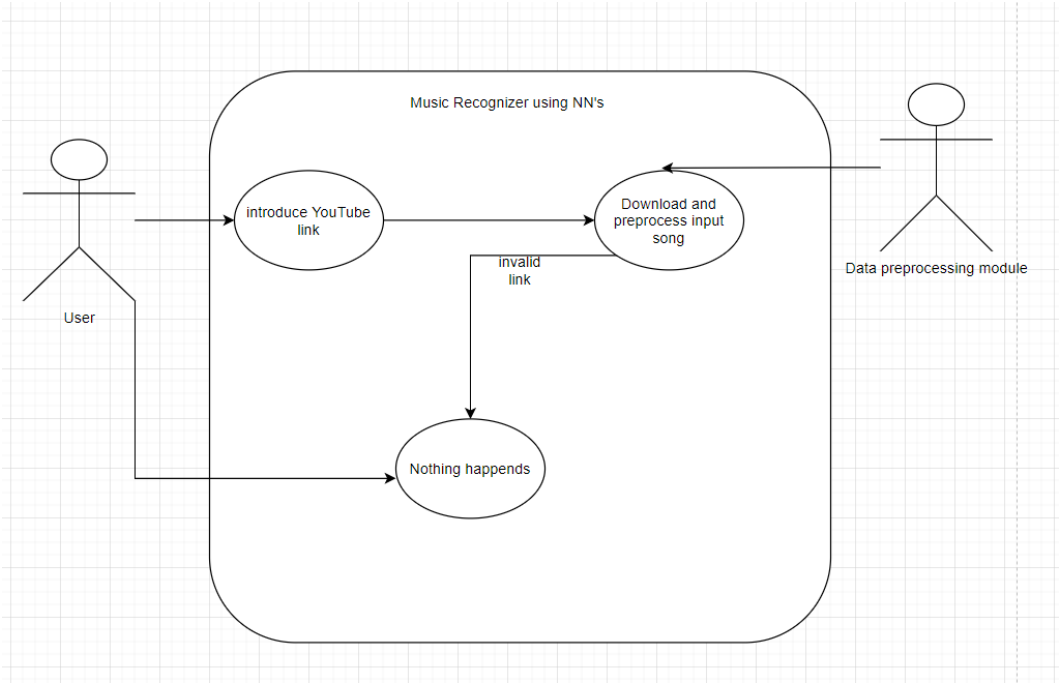

Figure 4.2: Illustration of the behaviour when the user inputs an invalid link.

The link introduced by the user is invalid. The youtube-dl library will throw an exception and the execution will ceases, given that the input cannot be fetched.

5.3 Third use case

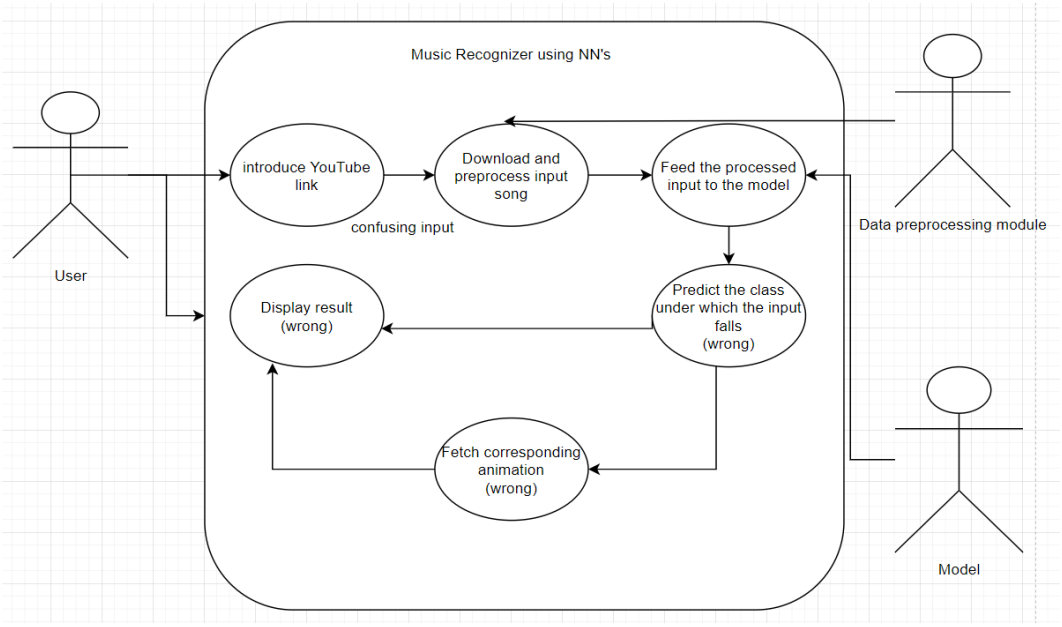

Figure 4.3: Illustration of the behaviour when the user inputs a confusing song to the application.

The input is confusing (e.g. a symphonic concert containing multiple other instruments beside a piano). Given that the model has been trained on piano-only samples, the prediction for such an input will be 'Other'.

Conclusions and possible improvements

Possible Improvements

Given that the problem the application solves is a binary one, a possible improvement is to increase the number of classes in order to represent multiple instruments. One possible scenario would be distinguishing between the instruments of an orchestra. Obviously that implies creating the corresponding 3D animations.
On the same note another improvement would refer to implementing GPU numeric operations support in order to improve the speed of the training process which is known to be lengthy in convolutional learning.

Final Conclusions

The three constituent parts of the application (as presented in Chapter 4 of the current paper) form a project that serves its purpose: to both create a middle way between arts and computer science, making the applications of machine learning interesting, yet engaging and to get a deeper understanding, by implementing the neuronal network framework from scratch, of the mathematical and theoretical subtleties of artificial intelligence.

The application succeeds in creating the liaison between an intuitive and friendly user experience and the complex background of the custom machine learning framework. Therefore, we believe that the current thesis corresponds with the initial ambition of the project.

Bibliography

[1] Keunwoo Choi, Deokjin Joo, and Juho Kim. "Kapre: On-GPU Audio Preprocessing Layers for a Quick Implementation of Deep Neural Network Models with Keras". In: *Machine Learning for Music Discovery Workshop at 34th International Conference on Machine Learning*. ICML. 2017.

[2] cochlea.eu. "Sound Representation Models". In: ().

[3] The SciPy community. "Numpy Official Documentation". In: (2008-2021).

[4] The SciPy community. "SciPy Official Documentation". In: (2008-2021).

[5] The Tensorflow community. "Tensorflow Official Documentation". In: ().

[6] The Wikipedia Community. "Generative adversarial network". In.

[7] Larry Hardesty. "Explained: Neural networks". In: (2017).

[8] AAVOS International. "Fourier Transform". In: ().

[9] Harrison Kinsley and Daniel Kukieła. *Neural Networks from Scratch in Python*. 2020.

[10] Frederik Kratzert. "Understanding the backward pass through Batch Normalization Layer". In: (2016).

[11] Brian McFee et al. "librosa: Audio and music signal analysis in python". In: *Proceedings of the 14th python in science conference*. Vol. 8. 2015.

[12] Quick overview of Neural Network architecture. "NeptuneAI team". In: ().

[13] Narayan Srinivasan. "Building an Audio Classifier using Deep Neural Networks". In: (2017).

[14] The GoogleAI/Magenta Team. "Magenta Demos". In.

[15] The OpenAI Team. "MuseNet Projects". In.

[16] Marte Vinje et al. "Music Genre Detection – A Complete System". In: (2020).

[17] Mingqing Yun and Jing Bi. “DEEP LEARNING FOR MUSICAL INSTRUMENT RECOGNITION”. In: (2017).