VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnana Sangama, Belagavi - 590018

PROJECT ENTITLED

"Simulation of Musical Instruments using Neural Networks"

Submitted in partial fulfilment of the requirements for the award of degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

For the Academic year 2017 – 2018

Submitted by:

Chandra Mahendra Vikram Singh	**1MV14CS027**
Himanshu Kumar	**1MV14CS040**
Jubin George Mathew	**1MV14CS046**
K Vishnudev	**1MV14CS049**

Project carried out at
Sir M. Visvesvaraya Institute of Technology
Bangalore - 562157

Under the guidance of
Mrs. Sheela S Kathavate
Assistant Professor, Dept of CSE
Sir M Visvesvaraya Institute of Technology, Bangalore

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SIR M VISVESVARAYA INSTITUTE OF TECHNOLOGY
HUNASAMARANAHALLI, BANGALORE - 562157

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnana Sangama, Belagavi - 590018

PROJECT ENTITLED

"Simulation of Musical Instruments using Neural Networks"

Submitted in partial fulfilment of the requirements for the award of degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

For the Academic year 2017 – 2018

Submitted by:

Chandra Mahendra Vikram Singh	**1MV14CS027**
Himanshu Kumar	**1MV14CS040**
Jubin George Mathew	**1MV14CS046**
K Vishnudev	**1MV14CS049**

Project carried out at
Sir M. Visvesvaraya Institute of Technology
Bangalore - 562157

Under the guidance of
Mrs. Sheela S Kathavate
Assistant Professor, Dept of CSE
Sir M Visvesvaraya Institute of Technology, Bangalore

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SIR M VISVESVARAYA INSTITUTE OF TECHNOLOGY
HUNASAMARANAHALLI, BANGALORE - 562157

SIR M VISVESVARAYA INSTITUTE OF TECHNOLOGY

Krishnadeveraya Nagar, International Airport Road,
Hunasamaranahalli, Bengaluru – 562157

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

It is certified that the project work entitled "**Simulation of Musical Instruments using Neural Networks**" is carried out **by Chandra Mahendra Vikram Singh (1MV14CS027), Himanshu Kumar (1MV14CS040), Jubin George Mathew (1MV14CS046), K Vishnudev (1MV14CS049)** bonafide student of **Sir M Visvesvaraya Institute of Technology** in partial fulfilment for the award of the Degree of Bachelor of Engineering in Computer Science and Engineering of the **Visvesvaraya Technological University**, Belagavi during the year 2017-2018. It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the course of Bachelor of Engineering.

Mrs. Sheela S Kathavate	**Prof. Dilip K. Sen**	**Dr. V.R. Manjunath**
Internal Guide	HoD	Principal
Assistant Prof., Dept of CSE	Dept of CSE	Sir MVIT
Sir MVIT	Sir MVIT	

Name of the examiners **Signature with date**

1)

2)

DECLARATION

We hereby declare that the entire project work embodied in this dissertation has been carried out by us and no part has been submitted for any degree or diploma of any institution previously.

Place: Bangalore

Date:

Signature of Students:

Chandra Mahendra Vikram Singh
(1MV14CS027)

Himanshu Kumar
(1MV14CS040)

Jubin George Mathew
(1MV14CS046)

K Vishnudev
(1MV14CS049)

ACKNOWLEDGMENT

It gives us immense pleasure to express our sincere gratitude to the management of **Sir M. Visvesvaraya Institute of Technology,** Bangalore for providing the opportunity and the resources to accomplish our project work in their premises.

On the path of learning, the presence of an experienced guide is indispensable and we would like to thank our guide **Mrs Sheela S Kathavate**, Assistant Professor, Dept. of CSE, for his invaluable help and guidance.

We would also like to convey our heartfelt and sincere thanks to **Dr. V.R. Manjunath,** Principal, Sir. MVIT for providing us with the infrastructure and facilities needed to develop our project.

We would also like to thank the staff of Department of Computer Science and Engineering and lab-in-charges for their co-operation and suggestions. Finally, we would like to thank all our friends for their help and suggestions without which completing this project would not have been possible.

Chandra Mahendra Vikram Singh	1MV14CS027
Himanshu Kumar	1MV14CS040
Jubin George Mathew	1MV14CS046
K Vishnudev	1MV14CS049

ABSTRACT

Music is an important entity in everyone's life. Similarly, musical instruments have a paramount place in Music. The art of playing musical instruments have been traditionally carried on from generation to generation, from primitive stage to this modern age. As Music is a performing art, which is being creative and cannot be static, hence the gradual developments and experiments have always given new ideas to the modern generation.

This project highlights experimentation in Instrumental Music which led to digitalised instruments in this sector. Electronic Instruments are instruments which help to electrically produce a symphony of its own may it be an electric guitar in a rock concert or a keyboard in orchestra. This project introduces an end-to-end neural network model for playing the sound of a musical instrument based on a silent video of it being played. At a high level, the model consists of a convolutional neural network (CNN) to extract features from the raw video frames and neural autoregressive models to encode the spatiotemporal features of the video.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1
INTRODUCTION

CHAPTER 1

INTRODUCTION

Nowadays, machine learning resides in every domain of the industry like online advertising, recommendation system, search engine, machine locomotive, pattern recognition etc. Also due to its pronounced effectiveness its applicability keeps growing bigger. Machine learning is an application of artificial intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.

In machine learning, a concept called convolutional neural network (CNN, or ConvNet) is used extensively in image and video recognition. It is a class of deep, feed-forward artificial neural networks that has successfully been applied to analysing visual imagery. CNNs use a variation of multilayer perceptron's designed to require minimal pre-processing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, recommender systems and natural language processing.

This project redefines the way one uses music instruments to generate sound to compose music. A keyboard instrument is a musical instrument played using a keyboard, a row of levers which are pressed by the fingers. The most common of these are the piano, organ, and various electronic keyboards, including synthesizers and digital pianos.

1.1 SCOPE OF WORK

Scope of the project is to develop and implement a working model to detect the movement of hands and play music accordingly. The focus is to develop a cost effective and a portable alternative to existing musical instruments that are expensive and bulky. The setup includes an optical device to capture video for detection of motion. This project is limited only to keyboard instruments and other instruments are out of scope of this project. Accuracy of the model, quality of capturing device and non-deterministic variation in processing time may widely affect the performance of this model. The project is expected to be completed within a period 4 - 6 months. This time span will involve building a system for emulating a piano.

1.2 MOTIVATION

Recent years have marked a sharp increase in the number of ways in which people interact with electronic devices. Keyboards and computer mouse devices are not the only way users interact with their computers. This project aims to investigate and demonstrate a human-computer interaction application in which a user can interact with the layout of a piano on a sheet of paper using computer vision. The image is acquired using a general-purpose camera from a single direction. The work presents detail about methods of piano key detection and labelling, hand movement detection, fingertip detection and tracking, and touch interaction. The detected key press then triggers a sound. The paper covers the details of the development of the virtual piano from concept to realization.

CHAPTER 2

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

There are wide range of instruments for music generation, they are either expensive or colossal to carry and sometimes both. So, this model tries to develop a system to mimic multiple instruments or input devices using the same environment in a more economic manner. Different Input Output devices such as keyboard, mouse or gesture could be used as input for electronic gadgets. They also suffer the same set of limitations as musical instruments. Thus to overcome such limitations, we have an elegant solution which involves building a generic learning model which can be trained for any specific device and the same can be achieved by passing the input received from these devices to a Convolutional neural network.

2.1 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNN, or ConvNet) are a class of deep-forward artificial neural networks that have been used for image analysis. These networks use convolution layers in its core. The images are broken down into smaller units, which are a set of learnable filters called kernels. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

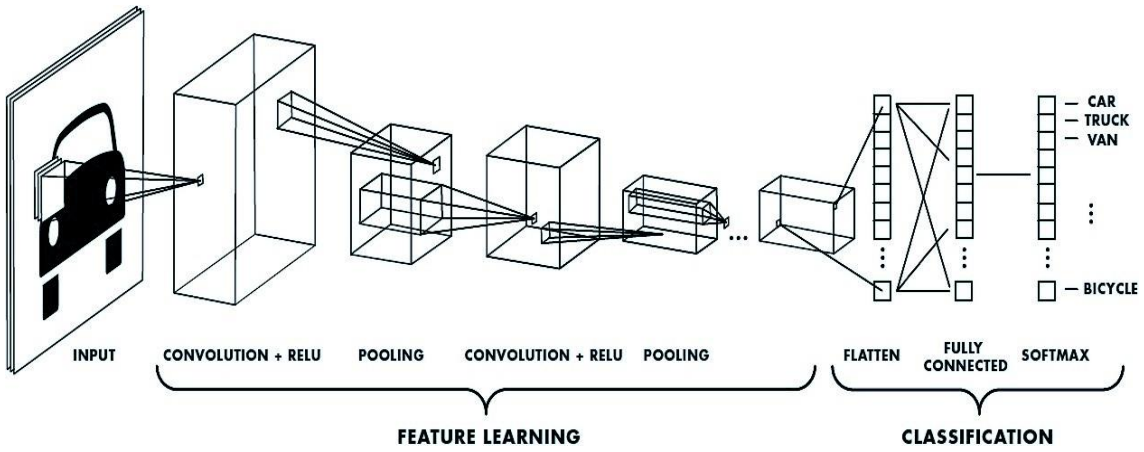

Figure 1: CNN Layers

Recurrent neural networks (RNN) [2] were created in the 1980s but have recently gained popularity with advances to network designs and increased computational power of graphics processing units. They are particularly useful with sequential data because each neuron or unit can use internal memory to maintain information about previous inputs.

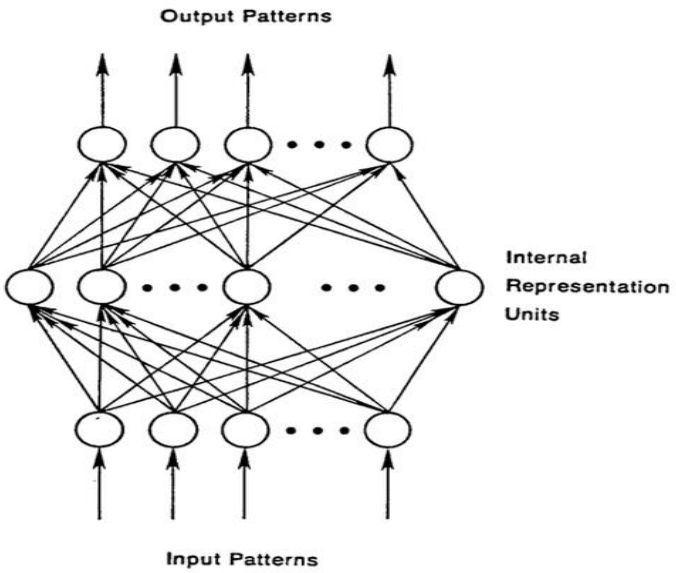

Figure 2: Neural Network

2.2 SUPERVISED LEARNING

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).

A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way (see inductive bias).

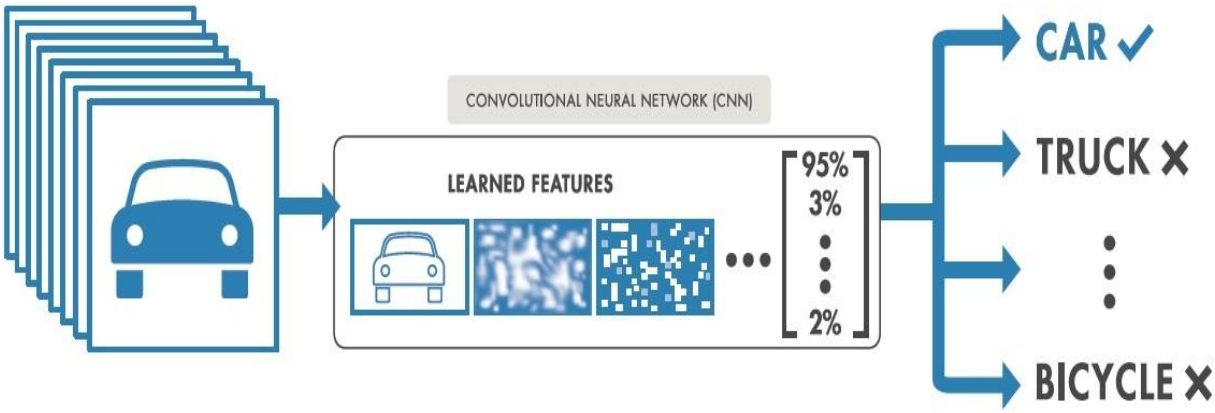

Figure 3: A Sample Convolutional Neural Network

2.3 DATA ACQUISITION AND AUGMENTATION

Deep learning uses Neural Nets with a lot of hidden layers (dozens in today's state of the art), and requires large amounts of training data. These models have been particularly effective in gaining insight and approaching human level accuracy in perceptual tasks like vision, speech, language processing. The theory and mathematical foundations were laid several decades ago.

Primarily two phenomena have contributed to the rise of machine learning:

a) Availability of huge data-sets/training examples in multiple domains

b) Advances in raw compute power and the rise of efficient parallel hardware.

Deep neural networks require lot of data to learn features from images. Many times we don't have lot of data to train such complex models. In such cases we perform Data augmentation to get more data such that model doesn't over-fit the given data. Some basic but powerful data augmentation techniques include the following:

- **Gaussian noise**

  Over-fitting usually happens when your neural network tries to learn high frequency features (patterns that occur a lot) that may not be useful. Gaussian noise, which has zero mean, essentially has data points in all frequencies, effectively distorting the high frequency features. This also means that lower frequency components (usually, your intended data) are also distorted, but your neural network can learn to look past that. Adding just the right amount of noise can enhance the learning capability.

Figure 4: Gaussian noise

- **Lighting conditions**

  Changing lighting conditions using programming technique includes conversion of RGB images to HSV. The HSV model describes colour similar to the colours perceived by human eye. RGB defines colour in terms of combination of primary colours. In situations where colour illustration plays an integral role, the HSV colour model is often preferred over the RGB model.

'Hue' represents the colour, 'Saturation' represents the amount to which that respective colour is mixed with white and 'Value' represents the amount to which that respective colour is mixed with black (Grey level).

In RGB, we cannot separate colour information from luminance. HSV or Hue Saturation Value is used to separate image luminance from colour information. Different lighting conditions can be emulated by tweaking the HSV values.

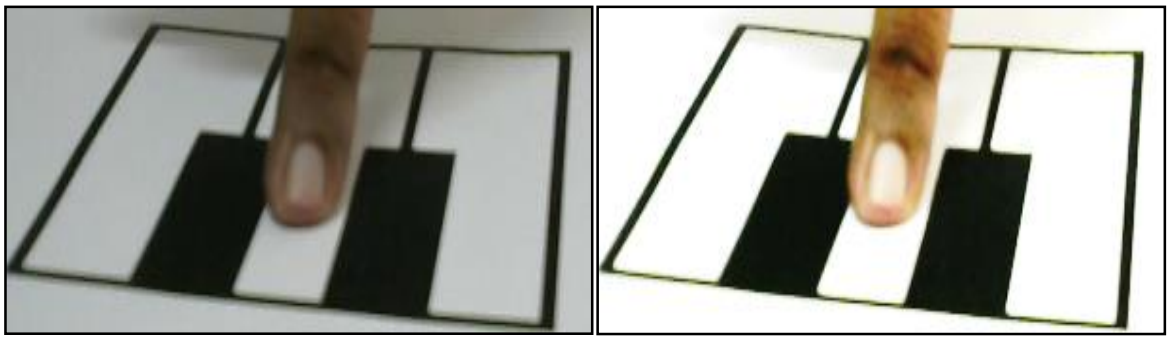

Figure 5: Lighting Property Changes

2.4 KERAS AND TENSORFLOW

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was 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.

Features of Keras are:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).

- Supports both convolutional networks and recurrent networks, as well as combinations of the two.

- Runs seamlessly on CPU and GPU.

Keras is a model-level library, providing high-level building blocks for developing deep learning models. It does not handle itself low-level operations such as tensor products, convolutions and so on. Instead, it relies on a specialized, well-optimized tensor manipulation library to do so, serving as the "backend engine" of Keras. Rather than picking one single tensor library and making the implementation of Keras tied to that library, Keras handles the problem in a modular way, and several different backend engines can be plugged seamlessly into Keras.

TensorFlow is an open source software library for machine learning across a range of tasks, and developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use. It is currently used for both research and production at Google products, often replacing the role of its closed-source predecessor, DistBelief. TensorFlow was originally developed by the Google Brain team for internal Google use before being released under the Apache 2.0 open source license on November 9, 2015.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations which such neural networks perform on multidimensional data arrays. These multidimensional arrays are referred to as "tensors". In June 2016, Google's Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google.

TensorFlow provides a Python API, as well as C++, Haskell, Java, Go, and Rust APIs. In addition, there is a 3rd party package for R.

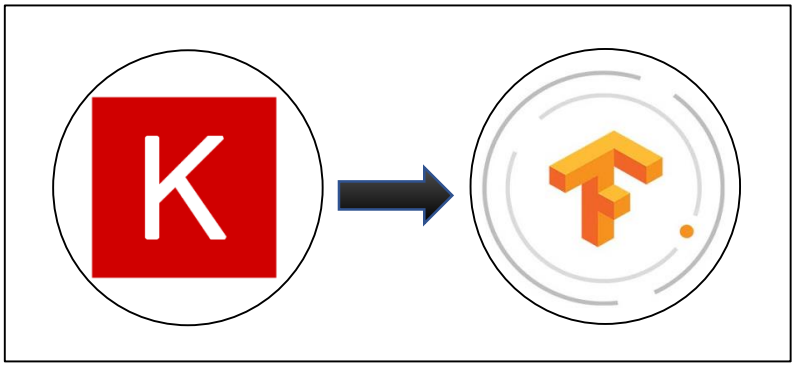

Figure 6: Tensorflow Keras Backend

CHAPTER 3

SYSTEM REQUIREMENTS AND SPECIFICATIONS

CHAPTER 3

SYSTEM REQUIREMENTS SPECIFICATION

The software requirements specification document enlists all necessary requirements that are required for the project development. To derive the requirements, we need to have clear and thorough understanding of the products to be developed. This is prepared after detailed communications with the project team and customer.

3.1 FUNCTIONAL REQUIREMENTS

The Functional Requirements Specification documents the operations and activities that a system must be able to perform.

1) Live video feed as input
2) Placing piano layout inside capture frame
3) Registering keypress events of the piano layout

3.2 NON-FUNCTIONAL REQUIREMENTS

The modeling, automatic implementation and runtime verification of constraints in component-based applications. Constraints have been assuming an ever more relevant role in modeling distributed systems as long as business rules implementation, design-by-contract practice, and fault-tolerance requirements are concerned. Nevertheless, component developers are not sufficiently supported by existing tools to model and implement such features, we propose a methodology and a set of tools that enable developers both to model component constraints and to generate automatically component skeletons that already implement such constraints. The methodology has been extended to support implementation even in case of legacy components.

3.3 OTHER NON-FUNCTIONAL REQUIREMENTS

3.3.1 Safety and Security Requirements

Many software-intensive systems have significant safety and security ramifications and need to have their associated safety- and security-related requirements properly engineered. For example, it has been observed by several consultants, researchers, and authors that inadequate requirements are a major cause of accidents involving software-intensive systems. Yet in practice, there is very little interaction between the requirements, safety and security disciplines and little collaboration between their respective communities. Most requirements engineers know little about safety and security engineering, and most safety and security engineers know little about requirements engineering. Also, safety and security engineering typically concentrates on architectures and designs rather than requirements because hazard and threat analysis typically depend on the identification of vulnerable hardware and software components, the exploitation of which can cause accidents and enable successful attacks.

3.3.2 Software Quality Attributes

Following factors are used to measure software development quality. Each attribute can be used to measure the product performance. These attributes can be used for Quality assurance as well as Quality control. Quality Assurance activities are oriented towards prevention of introduction of defects and Quality control activities are aimed at detecting defects in products and services.

Reliability

Measure if product is reliable enough to sustain in any condition give consistently correct results. Product reliability is measured in terms of working of project under different working environment and different conditions.

Maintainability

Different versions of the product should be easy to maintain. For development it should be easy to add code to existing system, should be easy to upgrade for new features and new technologies time to time. Maintenance should be cost effective and easy. System be easy to maintain and correcting defects or making a change in the software

Usability

This can be measured in terms of ease of use. Application should be user friendly. The system must be Easy to use for input preparation, operation, and interpretation of output, and provide consistent user interface standards or conventions with our other frequently used systems. They should be easy for new or infrequent users to learn to use the system.

Portability

This can be measured in terms of Costing issues related to porting, Technical issues related to porting, Behavioral issues related to porting

Correctness

Application should be correct in terms of its functionality, calculations used internally and the navigation should be correct. This means application should adhere to functional requirements.

Efficiency

To Major system quality attribute. Measured in terms of time required to complete any task given to the system. For example, system should utilize processor capacity, disk space and memory efficiently. If system is using all the available resources, then user will get degraded performance failing the system for efficiency. If system is not efficient then it cannot be used in real time applications.

3.4 DEVELOPMENT REQUIREMENTS

Model Training Requirement

- Google Colaboratory - It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud.

Hardware Requirement

PROCESSOR	:	Intel i3 or higher
RAM	:	4 GB or higher (8 GB Recommended)
MONITOR	:	13” or higher
HARD DISK	:	8 GB or higher
PIANO LAYOUT	:	5 Key Layout
WEBCAM	:	420p webcam or higher

Software Requirement

OPERATING SYSTEM	:	Linux
DRIVERS	:	Cuda Toolkit, cuDNN SDK, NVIDIA GPU Drivers
WEB BROWSER	:	Chrome, Firefox

Tools and Libraries

PROGRAMMING LANGUAGE	:	Python 3 or higher
TEXT EDITOR	:	Any Text Editor
REQUIRED LIBRARIES	:	Numpy, Tensorflow, Keras, Flask

CHAPTER 4
DATASET AND FEATURES

CHAPTER 4

DATASET AND FEATURES

4.1 DATASET

The dataset used in the project was generated from the live video feed using webcam. The fetched video feed was processed to generate the image frames for the respective key classes of the piano. Data set is typically divided into Training, Testing and Cross Validation sets. Training and Testing set has 90% and 10% data split respectively, whereas data is randomly picked from training set for cross validation. Mostly all three data sections are kept mutually exclusive for better generalization of the model. We are also performing data augmentation in order to avoid overfitting of model on training data.

CLASS	TRAINING SET	CROSS VALIDATION SET	TEST SET
Class 0	673	673	73
Class 1	1538	1538	170
Class 2	1598	1598	177
Class 3	2678	2678	296
Class 4	800	800	88
Class 5	1303	1303	144

Table 1 : Data Set for all classes

4.2 FEATURES

The Neural Network identifies all features during the learning phase itself. These features range from few to some crores in number based on its complexity.

Here we use white keys, black keys, layout of the piano as primary hyper features to enable successful learning. Thereafter multiple classes are used accordingly to successfully classify key press events of the paper layout.

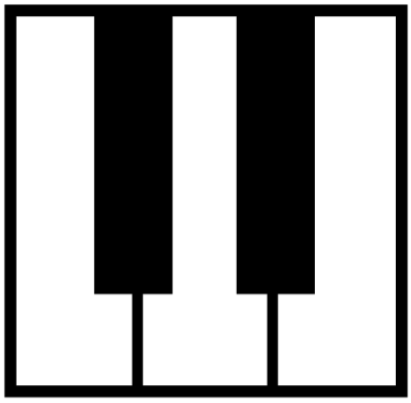

Figure 7: Keys Layout

The system is performing a visual input to scan paper piano layout in order to play the music.

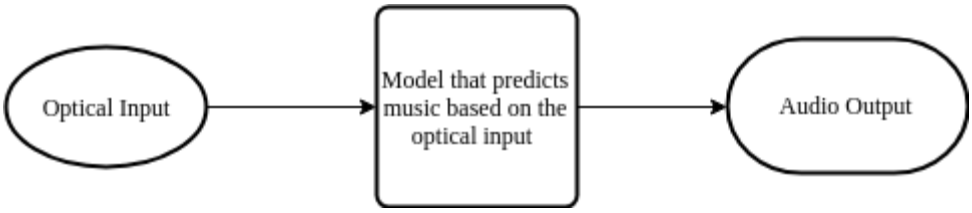

Figure 8: Overview of the System

CHAPTER 5

SYSTEM ANALYSIS & DESIGN

CHAPTER 5

SYSTEM ANALYSIS AND DESIGN

5.1 SYSTEM ANALYSIS

5.1.1 System Development Life Cycle

The SDLC is an application systems approach to development of information system. The tools of SDLC are using diagrams so it will be easier to understand, its stages related to each other. When changes occur in all phases of the system then it does not repeat again, SDLC phase is simpler.

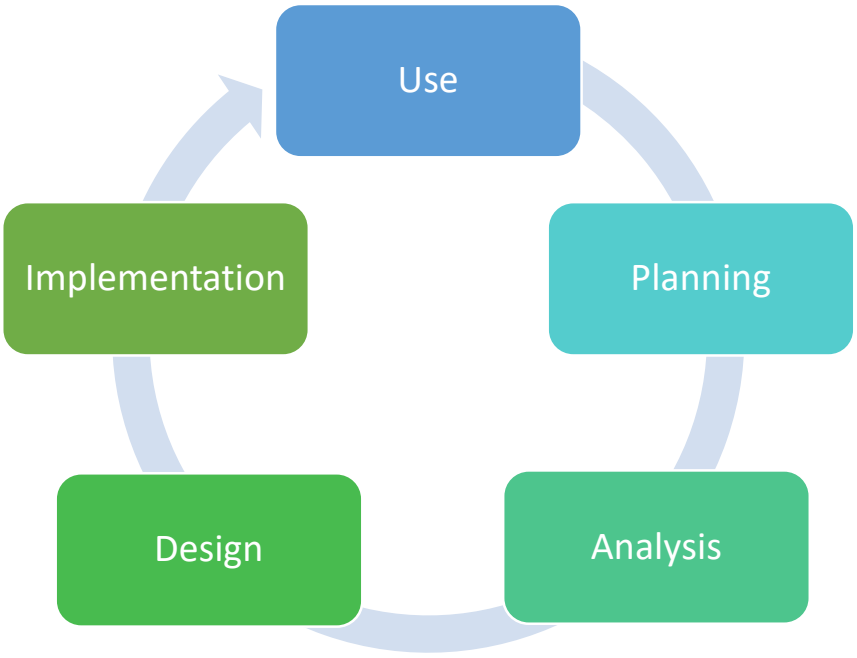

Figure 9: Software Development Life Cycle

5.1.2 The Development Phases in Brief

a.) Planning Phase

The Planning phase began with figuring out a way to develop a framework with organized deep reinforcement learning modules working at an environment agnostic level. Hence, to implement this model we planned to build an environment which will simulate and learn to predict keypress on a piano, and with our observations acquired by training the model we will be able to implement the concept of Supervised Reinforcement Learning.

b.) Analysis Phase

This phase began with the analysis of the requirements needed to build the working model of our paper piano system, by preparing an abstract and carrying out a literature survey and realizing about the different approaches to go about and develop the system. The analysis revealed that the primary difficulty arises due to insufficient computational power, resulting in a model with a limited number of keys. Upon further research we came to a conclusion that our problem lies in supervised learning. Image recognition plays a key role in the system hence a Convolutional Neural Network is used.

c.) Design Phase

We constructed our design by partitioning the design of the whole system into two levels, the high level design and the low level design. In the High level we presented a system represented in its most abstract level which consisted of the methodologies and necessary approaches to go about designing the environment, model and the convolutional neural networks. In the lower layer of the design we focus mainly on the build and the detailed design of the Convolutional Neural Network and how it could perhaps be used to train the model.

d.) Implementation Phase

In order to develop a working model, implementation was divided into following segments, namely, Preprocessor, Classifier and User Interface. In the first segment we apply data augmentation techniques on raw images to generate our dataset. Classifier segment involves training of the model with our dataset and is responsible for accurately predicting keypress events. User interface segment deals with the simulation of a piano with the help of computer graphics and production of musical notes in accordance with the keys pressed.

e.) Use Phase

After the implementation phase we move on to the use phase where we have trained our model to predict keypress events based on user's interaction on the paper piano layout via a live video feed. This use phase was used to test the model in a restricted environment. We can further extend our model to train different environments and use them in different classes of musical instruments. Also we can scale our model to a larger set of keys in a more flexible environmental conditions by training with a diverse and large dataset.

5.2 SYSTEM DESIGN

5.2.1 Low Level Design

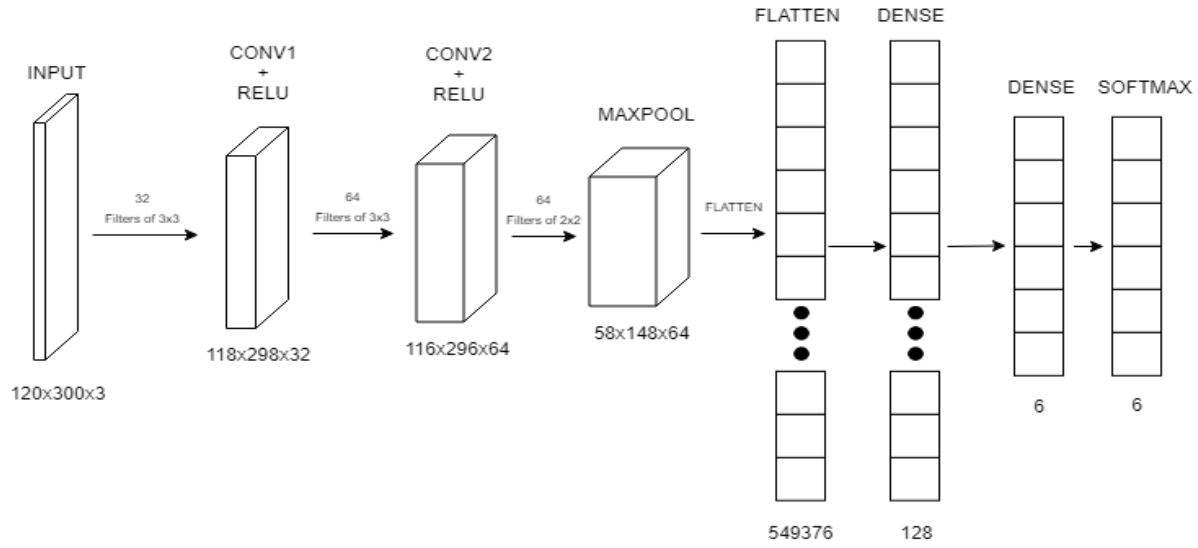

Figure 10: Low Level CNN Design

5.2.2 High Level Design

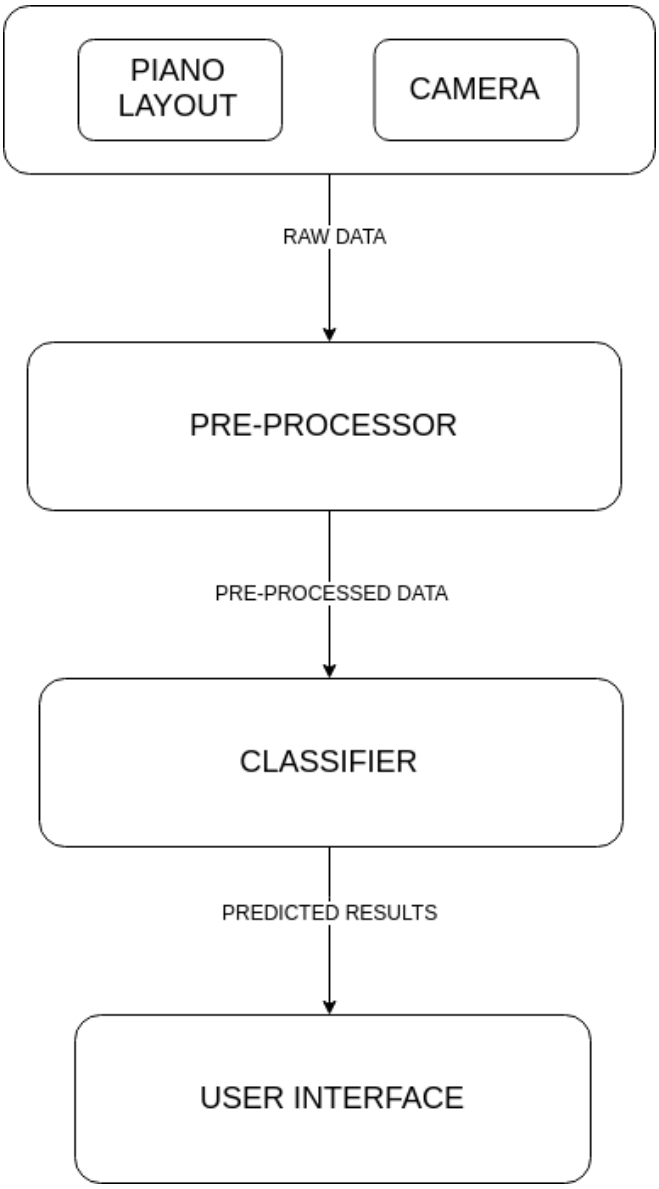

Figure 11: High Level System Architecture

CHAPTER 6

IMPLEMENTATION AND INTERFACE

CHAPTER 6

IMPLEMENTATION

6.1 IMPLEMENTATION

The design carried out in order to develop the model for being able to train and test and imply supervised reinforcement learning was translated into a working model by implementing our design process which was broken down into different segments of programmed source code. Each of these segments were responsible for handling an aspect of the working model. We have described the working of each of the segments along with the code snippets and implementation in the following subsections.

6.1.1 Model

Our neural network model consists of 2 layers of Conv2D layers stacked sequentially followed by MaxPooling2D, Dropout, Flatten and Dense layers. The Conv2D layers uses relu activation function. The Dense layer uses softmax activation function. Before training the model, the learning process is configured using the compile method of the Keras model class, which takes the type of optimizer, type of loss and other metrics as parameters. The code snippet given below shows the model segment of our system.

```
.
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
.
.
```

The JSON format of the model looks like the following:


```
{
   "class_name": "Sequential",
   "config": [
   {
       "class_name": "Conv2D",
       "config":
       {
           "name": "conv2d_5",
           "trainable": true,
           "batch_input_shape": [null, 120, 300, 3],
           "dtype": "float32",
           "filters": 32,
           "kernel_size": [3, 3],
           "strides": [1, 1],
           "padding": "valid",
           "data_format": "channels_last",
           "dilation_rate": [1, 1],
           "activation": "relu",
           "use_bias": true,
           "kernel_initializer":
           {
               "class_name": "VarianceScaling",
               "config":
```

```
                {
                    "scale": 1.0,
                    "mode": "fan_avg",
                    "distribution": "uniform",
                    "seed": null
                }
            },
            "bias_initializer":
            {
                "class_name": "Zeros",
                "config":
                {}
            },
            "kernel_regularizer": null,
            "bias_regularizer": null,
            "activity_regularizer": null,
            "kernel_constraint": null,
            "bias_constraint": null
        }
    },
    {
        "class_name": "Conv2D",
        "config":
        {
            "name": "conv2d_6",
            "trainable": true,
            "filters": 64,
            "kernel_size": [3, 3],
            "strides": [1, 1],
            "padding": "valid",
            "data_format": "channels_last",
            "dilation_rate": [1, 1],
            "activation": "relu",
            "use_bias": true,
            "kernel_initializer":
            {
```

```
                "class_name": "VarianceScaling",
                "config":
                {
                    "scale": 1.0,
                    "mode": "fan_avg",
                    "distribution": "uniform",
                    "seed": null
                }
            },
            "bias_initializer":
            {
                "class_name": "Zeros",
                "config":
                {}
            },
            "kernel_regularizer": null,
            "bias_regularizer": null,
            "activity_regularizer": null,
            "kernel_constraint": null,
            "bias_constraint": null
        }
    },
    {
        "class_name": "MaxPooling2D",
        "config":
        ...
    },
    {
        "class_name": "Dropout",
        "config":
        ...
    },
    {
        "class_name": "Flatten",
        "config":
        ...
```

```
},
{
    "class_name": "Dense",
    "config":
    {
        "name": "dense_5",
        "trainable": true,
        "units": 128,
        "activation": "relu",
        "use_bias": true,
        "kernel_initializer":
        ....
        "kernel_regularizer": null,
        "bias_regularizer": null,
        "activity_regularizer": null,
        "kernel_constraint": null,
        "bias_constraint": null
    }
},
{
    "class_name": "Dropout",
    "config":
    {
        "name": "dropout_6",
        "trainable": true,
        "rate": 0.5,
        "noise_shape": null,
        "seed": null
    }
},
{
    "class_name": "Dense",
    "config":
    {
        "name": "dense_6",
        "trainable": true,
```

```
            "units": 6,
            "activation": "softmax",
            "use_bias": true,
            "kernel_initializer":
            ...
            "kernel_regularizer": null,
            "bias_regularizer": null,
            "activity_regularizer": null,
            "kernel_constraint": null,
            "bias_constraint": null
        }
    }],
    "keras_version": "2.1.6",
    "backend": "tensorflow"
}
```

Model can also be saved into a YAML file.

6.1.2 Real-time Image Capture

The video frame is captured with the help of **VideoCapture()** which reads image from a specific buffer of the memory. Continuous images are fetched using **VideoCapture.read()**. The captured frames of images are displayed using **cv2.imread()**. Keyboard events are captured with the help of the **cv2.waitKey()** which is detecting the ESC key for exiting out of the infinite loop. **VideoCapture.release()** is used for freeing the device which is being used for capturing the images. **cv2.destroyAllWindows()** is used to close all the opened windows which is used to display the captured frames of images.

```
import numpy as np
import cv2

# setting the video device capture to device 1 i. e external
webcam
```

```
cap = cv2.VideoCapture(1)

while(True):
      # read the frames from the device
  ret, frame = cap.read()
  .
  .
  .
  # display the image captured from the device
  cv2.imshow('frame',gray)

  # taking the keyboard events
  k = cv2.waitKey(33)
  if k == 27:
      # ESC key pressed - break the Loop
      break

# releasing the video device which is being used for data
capturing.
cap.release()

# destroy all the windows.
cv2.destroyAllWindows()
```

6.1.3 Training and saving the model

Dataset is divided into batches of 20 images each and is clustered as such via batchGenerator function. After successful batching, data is reshaped and normalized to ease the process of training. Now the normalized data is trained using the fit function of Keras.models class.

Thus after all the epochs are completed for training the model, it is then saved to the local user system with the help of to_json and save_weights functions of

Keras.models class which returns a representation of the model as a JSON string and saves the weights of the model as a HDF5 file respectively.

```
.
.
# number of epochs the model should be training.
for e in range(epochs):

  # fetching batched of images 20 each
  for x_train, y_train in batchGenerator():
      # reshaping of the training data
      x_train = x_train.reshape(x_train.shape[0], img_rows,
img_cols, 3)
      input_shape = (img_rows, img_cols, 3)

  # normalization of the training data
  x_train = x_train.astype('float32')
  x_train /= 255

  # convert class vectors to binary class matrices
  y_train = keras.utils.to_categorical(y_train, num_classes)

  # training of the model
  history = model.fit(x_train, y_train,
            epochs=1,
            verbose=0,
            validation_split=0.05)

# save model to JSON
print("Saving model to disk...")
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)

# save weights to HDF5
```

```
model.save_weights("model.h5")
print("Saved model to disk")

# save history
np.save("history", history.history)
```

6.1.4 Imports and Hyper parameters

The **OS** library is used to fetch images from the generated dataset and to access some low level services of operating system's interfaces of the for the same cause. The high-level Neural Network library Keras is used for creating our CNN model, and it runs on top of TensorFlow backend.

We are using linear stacked layer model called Sequential model. Layers used here include Dense, Dropout, Flatten, Conv2D and MaxPooling2D layers. We are setting the hyper parameters for the network as batch size of 20 images.

```
import os
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 20
num_classes = 6
epochs = 20
img_rows, img_cols = 120, 300
input_shape = (img_rows, img_cols, 3)
.
.
```

6.1.5 Loading the model and prediction

For loading the model, we use **model_from_json** and **load_weights** functions of the Keras.models class for loading back the model configuration from the JSON file and loading the different weights of the network respectively. The model is then configured with the help of the **compile** method of the Keras.models class.

Now the data which is to be predicted is reshaped and normalised before it is send for prediction. Now for prediction we use either **evaluate** or **predict** method of the Keras.models class, which gives us a vector of probabilities a particular sample belonging to a specific class.

```
from keras.models import model_from_json
.
.
.
# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# load weights into new model
loaded_model.load_weights("model.h5")

loaded_model.compile(loss=keras.losses.categorical_crossentrop
y,
            optimizer=keras.optimizers.Adadelta(),
            metrics=['accuracy'])

# reshaping the data
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols,
3)
```

```
# normalization of the new data.
x_test = x_test.astype('float32')
x_test /= 255
y_test = keras.utils.to_categorical(y_test, num_classes)
# prediction of the results.
score = model.evaluate(x_test, y_test, verbose=0)
# or
score = model.predict(x_test)
```

6.1.6 Model Upload and Download

Because of the scarcity of resources for computation the model is trained in the cloud in the Google Colab servers. Colaboratory is a Google research project created to help disseminate machine learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud.

Colaboratory notebooks are stored in Google Drive and can be shared just as you would with Google Docs or Sheets. Colaboratory is free to use.

To save the model from their servers to a local machine or our Google Drive we are using pydrive module provided by google for storage purposes. The object of **drive.CreateFile** is called to create a pydrive object to which the file is specified which is to be uploaded using **SetContentFile** and then uploaded using the **Upload** method of the pydrive object.

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
.
.
.
# Create & upload model.h5 file.
uploaded = drive.CreateFile()
```

```
uploaded.SetContentFile('model.h5')
uploaded.Upload()

# Create & upload model.json file.
uploaded = drive.CreateFile()
uploaded.SetContentFile('model.json')
uploaded.Upload()
```

Similarly loaded model and weights can also be downloaded from drive using PyDrive as:

```
# Download a file based on its file ID.
#
# A file ID looks like: laggVyWshwcyP6kEI-y_W3P8D26sz
# Downloading model.json
file_id = <file ID>
downloaded = drive.CreateFile({'id': file_id})
downloaded.GetContentFile('model.json')
print('Downloaded content "{}"'.format('model.json'))

# Downloading model.h5
file_id = <file ID>
downloaded = drive.CreateFile({'id': file_id})
downloaded.GetContentFile('model.h5')
print('Downloaded content "{}"'.format('model.h5'))
```

Before performing upload and download operation it needs authentication only once for a program. It can be installed and authenticated simultaneously using following code in Jupyter Notebook:

```
# Install the PyDrive wrapper & import libraries.

# This only needs to be done once in a notebook.
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
```

```
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once in a notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials =
GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

We are also downloading and extracting data from Google Drive to Google Colaboratory using above method as:

```
file_id = <file ID>

downloaded = drive.CreateFile({'id': file_id})
downloaded.GetContentFile('data.zip')
print('Downloaded content "{}"'.format('data.zip'))
```

6.1.7 Augmentation using ImageDataGenerator

ImageDataGenerator class in Keras can generate batches of tensor image data with real-time data augmentation. The data is looped over (in batches).

```
# this is the augmentation configuration we will use for
training
train_datagen = ImageDataGenerator(
        rescale=1./255)
# this is the augmentation configuration we will use for
testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1./255)
```

6.1.8 Image data flow from directory

flow_from_directory method takes the path to a directory, and generates batches of augmented/normalized data.

```
# this is a generator that will read pictures found in
# subfolers of 'data/train', and indefinitely generate
# batches of augmented image data
train_generator = train_datagen.flow_from_directory(
        'data/train',  # this is the target directory
        target_size=(120, 300),  # all images will be resized
to 150x150
        batch_size=batch_size,
        class_mode='categorical')

# this is a similar generator, for validation data
validation_generator = test_datagen.flow_from_directory(
        'data/test',
        target_size=(120, 300),
        batch_size=948,
        class_mode='categorical')
```

6.1.9 Fit Generator

Keras spawns multiple threads when using fit_generator, each calling your generator trying to fetch examples in advance. This helps parallelizing data fetching on the CPU.

```
history = model.fit_generator(
              train_generator,
              steps_per_epoch = 8590 // batch_size,
              epochs = epochs,
              validation_data = validation_generator,
              validation_steps = 948 // batch_size)
```

6.2 INTERFACE IMPLEMENTATION

6.2.1 BACKEND – Flask and Web Sockets

Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. The **Web Server Gateway Interface (WSGI)** is a simple calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language. In this project the server runs on flask with WSGI as Interface for productionisation of server environment.

WebSockets are an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.

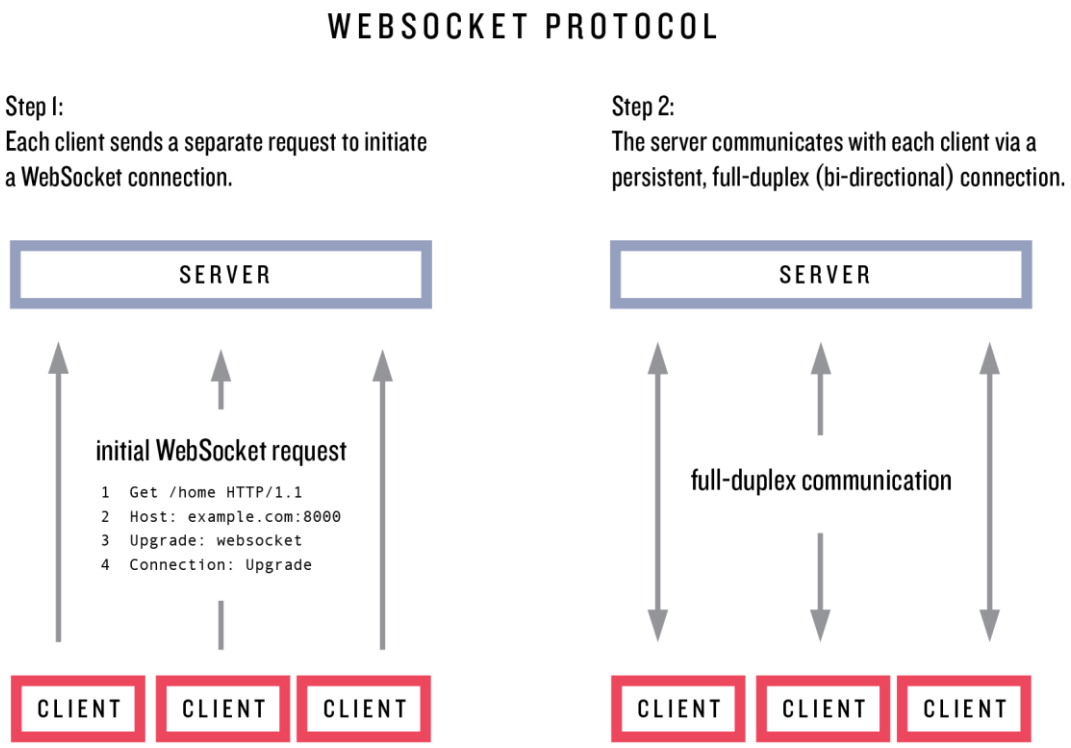

Figure 12 : Web-Socket Protocol

Imports

```
import socketio
import eventlet
from flask import Flask, render_template
```

Usage

```
sio = socketio.Server()
app = Flask(__name__, static_folder='static',
template_folder='template', static_url_path='')

@app.route('/')
def index():
   """Serve the client-side application."""
   return render_template('index.html')

# wrap Flask application with socketio's middleware
app = socketio.Middleware(sio, app)

# deploy as an eventlet WSGI server
eventlet.wsgi.server(eventlet.listen(('', 8000)), app)
```

6.2.2 FRONTEND

User Interface is built using Three.js and MIDI player.

Three.js is a cross-browser JavaScript library and Application Programming Interface (API) used to create and display animated 3D computer graphics in a web browser. Three.js uses **WebGL**.

WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser without the use of plugins. WebGL is integrated completely into all the web standards of the browser, allowing GPU-accelerated usage of physics and image processing and effects as part of the web page canvas.

MIDI (Musical Instrument Digital Interface) is a protocol designed for recording and playing back music on digital synthesizers that is supported by many makes of personal computer sound cards. Originally intended to control one keyboard from another, it was quickly adopted for the personal computer. Rather than representing musical sound directly, it transmits information about how music is produced. The command set includes note-ons, note-offs, key velocity, pitch bend and other methods of controlling a synthesizer. The sound waves produced are those already stored in a wavetable in the receiving instrument or sound card.

dat.GUI is a lightweight graphical user interface for changing variables in JavaScript.

Scripts included are :

```
<script src="/js/three.js"></script>
<!-- COLLADA 3D MODEL LOADER -->
<script src="/js/ColladaLoader.js"></script>
<!-- MIDI Player Library -->
<script src="/js/MIDI/AudioDetect.js"></script>
<script src="/js/MIDI/LoadPlugin.js"></script>
<script src="/js/MIDI/Plugin.js"></script>
<script src="/js/MIDI/Player.js"></script>
<script src="/js/MIDI/Loader.js"></script>
<!-- Simple GUI from Google -->
<script type="text/javascript" src="/js/dat.gui.js"></script>
```

Implementation

```
// Begin MIDI loader widget
MIDI.loader = new widgets.Loader({
      message: "Loading: Soundfont..."
});
// Initialize three.js scene and camera
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(7,
window.innerWidth/window.innerHeight, 2.0, 5000);
```

```
// Load 3D model into the three.js scene
var loader = new THREE.ColladaLoader();
loader.load( '/piano-small.dae', prepare_scene );

// Render the scene
function render( delta )
{
      renderer.render(scene, camera);
};
// Initialize piano keys and draw it in scene
function initialize_keys( obj)
{
      keys_obj.push(obj);
      obj.rotation.y = 0;
      obj.rotation.z = 0;
      obj.position.x += 0.1;
      obj.position.y -= 0.165;
      obj.keyState = keyState.unpressed;
      obj.clock = new THREE.Clock(false);
      obj.castShadow = true;
      obj.receiveShadow = true;

      // only add meshes in the material redefinition (to make
      keys change their color when pressed)

      if (obj instanceof THREE.Mesh)
      {
            old_material = obj.material;
            obj.material = new THREE.MeshPhongMaterial( {
            color:old_material.color} );
            obj.material.shininess = 35.0;
            obj.material.specular = new
            THREE.Color().setRGB(0.25, 0.25, 0.25);;
            obj.material.note_off = obj.material.color.clone();
      }
```

```
}

// Show key press by rotation of keys

function update_key( obj, delta )
{
	if (obj.keyState == keyState.note_on)
	{
		obj.rotation.x = mix(-Math.PI/2, -
		controls.key_max_rotation, smoothstep(0.0, 1.0,
		controls.key_attack_time*obj.clock.getElapsedTime())
		);
		if (obj.rotation.x >= -controls.key_max_rotation)
		{
			obj.keyState = keyState.pressed;
			obj.clock.elapsedTime = 0;
		}
		obj.material.color = noteOnColor;
	}
	else if (obj.keyState == keyState.note_off)
	{
		obj.rotation.x = mix(-controls.key_max_rotation, -
		Math.PI/2, smoothstep(0.0, 1.0,
		controls.key_attack_time*obj.clock.getElapsedTime())
		);
		if (obj.rotation.x <= -Math.PI/2)
		{
			obj.keyState = keyState.unpressed;
			obj.clock.elapsedTime = 0;
		}
		obj.material.color = obj.material.note_off;
	}
}

function keyPress(ev)
```

```
{
        console.log(ev);
        if (keys_down[ev.keyCode] != true)
        {
                var obj = keyCode_to_note(ev.keyCode);
                var disp_note = obj.disp, note = obj.note;
                if (note != -1)
                {
                        console.log("note", disp_note);
                        key_status(disp_note, keyState.note_on);
                        keys_down[ev.keyCode]=true;



                        var delay = 0;
                        // play one note every quarter second
                        // var note = parseInt(note.substr(1))+21;
                        // the MIDI note
                        var velocity = 127; // how hard the note hits
                        console.log(ev, MIDI);
                        MIDI.setVolume(0, 127);
                        MIDI.noteOn(0, note+21, velocity,
                        delay);
                }
        }
        setInterval(function() {
                for (let index = 0; index < 5; index++) {
                        keyRelease({"keyCode":65+index});
                }
        }, 2000);
}
```

CHAPTER 7

EXECUTION AND TEST CASES

CHAPTER 7

EXECUTION AND TEST CASES

7.1 EXECUTION

7.1.1 Visualization

Here we have given a sample input image:

Figure 13: Raw Input Image

Above image after passing through first conv_2d layer of size 118 x 298 x 32. Here this image is filtered with the help of 32 filters.

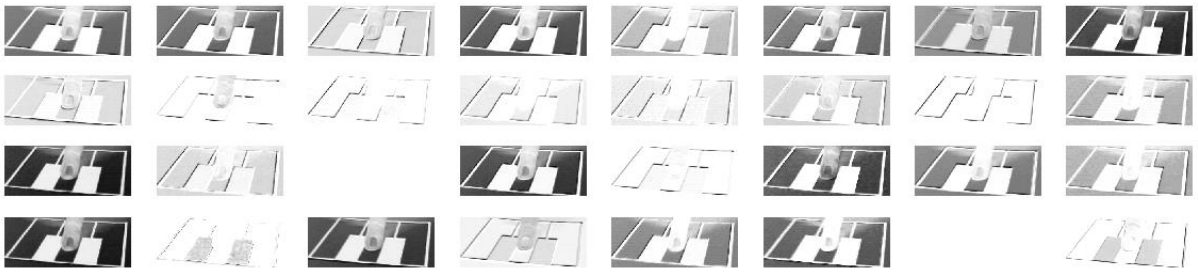

Figure 14: Conv_2D Layer 1 output

Above filtered images are then passed through second conv_2d layer of size 116 x 296 x 64. Here images are filtered with the help of 64 filters. This layer provides a more detailed approach in identifying features.

7.1.2 User Interface

The User Interface which is displayed for the user to interact with the working, completely trained models running in the system. We have captured the screenshots of the test runs performed on the system as shown below.

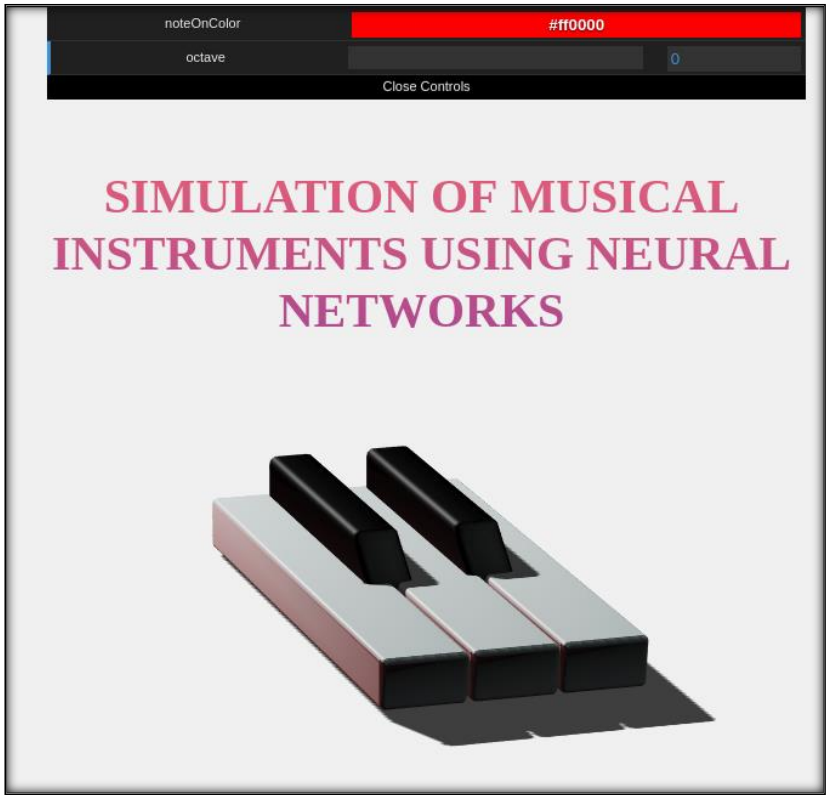

Figure 15: 3D Piano Interface

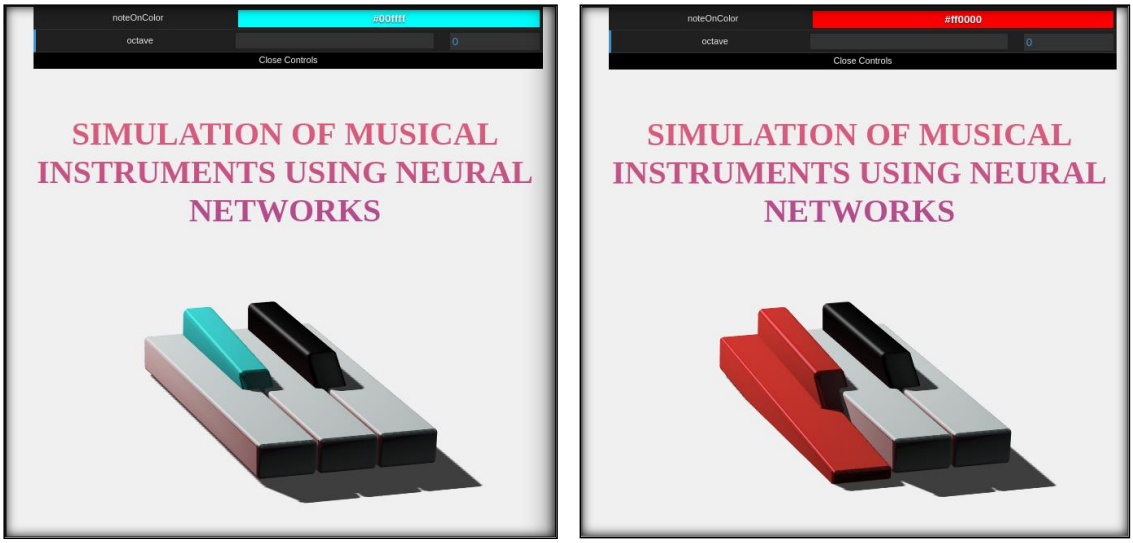

Figure 16: Key Press Events [Single and Multiple]

7.1.3 Observations and Graphs

LAYER (TYPE)	OUTPUT	SHAPE	PARAM #
conv2D_1	Conv2D	(None, 118, 298, 32)	896
conv2d_2	Conv2D	(None, 116, 296, 64)	18496
max_pooling2d_1	MaxPooling2	(None, 58, 148, 64)	0
dropout_1	Dropout	(None, 58, 148, 64)	0
flatten_1	Flatten	(None, 549376)	0
Dense_1	Dense	(None, 128)	70320256
dropout_2	Dropout	(None, 128)	0
Dense_2	Dense	(None, 6)	774
TOTAL		PARAMS	70320422
TRAINABLE		PARAMS	70320422
NON – TRAINABLE		PARAMS	0

7.1.4 Model Training History

We can create plots from the collected history data from Keras.

Model Accuracy Curve

From the plot of accuracy its evident that the model is sufficiently trained to satisfy required need. We can also conclude that the model has not yet over-learned the training dataset, showing comparable accuracy on both datasets.

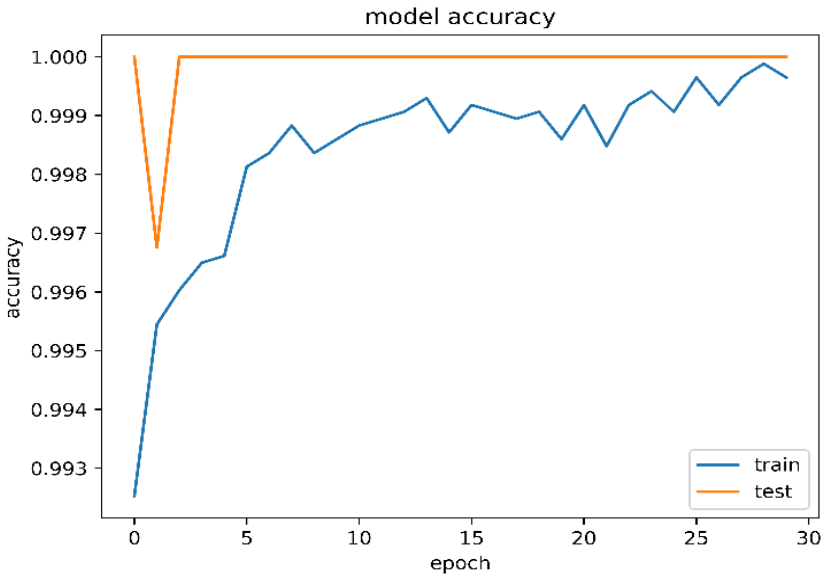

Figure 17: Accuracy on the training and validation datasets over training epochs

Model Loss Curve

From the plot of loss, we can conclude that the model has comparable performance on both train and validation datasets. If these parallel plots start parting away consistently, it might be a sign to stop training at an earlier epoch.

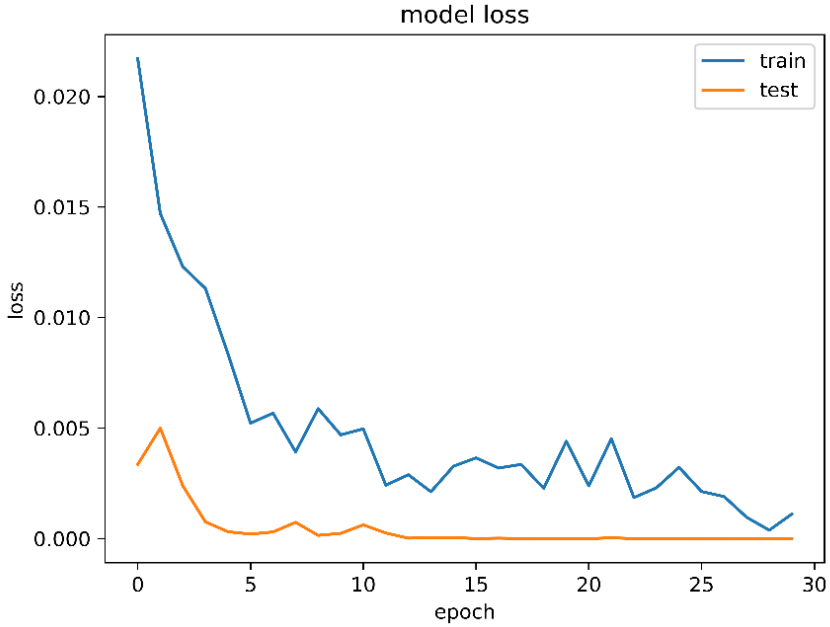

Figure 18: Loss on the training and validation datasets over training epochs

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

CHAPTER 8

CONCLUSION AND FUTURE ENCHANCEMENTS

8.1. CONCLUSION

This project uses Convolutional Neural Networks model for detecting finger interaction of the end user with the piano layout from the video feed. This model parses each video frame and extracts respective feature set to feed the model. The model requires explicit manual labelling, which results in some overhead and thus increasing latency of the system. In future, we will be working on improving its latency in order to make it more realistic and expeditious. This would lead to an increase in its real time applications and gradually an alternative for bulky and costly instruments.

This is a difficult problem due to following aspects.

- As the system involves model training with image recognition a lot of computation power is required to train and run the system.
- There is no such dataset available that can be used to train model for such problem which are based on different musical instruments. Hence, manual collection of data is a must and this leads to hindrance in scalability of such models.
- A camera module is used to input a live video feed to the model, hence the quality of the camera plays an important role in the system. The dataset generated from the camera is exclusive to devices with same configuration of the camera used for generating dataset. If a lower quality or a far superior quality camera is used to run the system, it can result in lower accurate predictions.

8.2 FUTURE WORK

- This model accurately predicts the keypress events of the paper piano. The number of keys of the layout have been kept to minimal due to limited computational power to train the model. In future we want to work on an optimized version of the model which can accurately predict the keypress events for a larger set of keys. With more computational prowess we can make a much more robust model for the same.

- There is a certain amount of latency in the system due to the limited processing power of the system hence optimization of the model can be done to have an overall smooth interaction with the system.

- This approach can be scaled to multiple keyboard based musical instruments.

REFERENCES

REFERENCES

[1] E. Rumelhart, David & E. Hinton, Geoffrey & J. Williams, Ronald. (1986). Learning Representations by Back Propagating Errors. Nature. 323. 533-536. 10.1038/323533a0.

[2] LeCun, Y., Kavukcuoglu, K., & Farabet, C. (2010, May). Convolutional networks and applications in vision. In Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on (pp. 253-256). IEEE.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: Advances in neural information processing systems. 2012, pp. 1097– 1105.

[4] D. Cire¸san, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. Arxiv preprint arXiv:1202.2745, 2012.

[5] LeCun, Yann, Bengio, Yoshua & Hinton, Geoffrey (2015). Deep learning. nature, 521, 436.