

**Music Instrument Localization in Virtual Reality  
Environments using audio-visual cues**

**Siddhartha Bhattacharyya B.Tech**

**A Dissertation**

Presented to the University of Dublin, Trinity College  
in partial fulfilment of the requirements for the degree of

**Master of Science in Computer Science (Data Science)**

Supervisor: Aljosa Smolic and Cagri Ozcinar

September 2020

# Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

---

Siddhartha Bhattacharyya

September 6, 2020

## Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

---

Siddhartha Bhattacharyya

September 6, 2020

# Acknowledgments

I would like to thank my supervisors Professors Aljosa Smolic and Cagri Ozcinar for their dedicated support, understanding and leadership. My heartfelt gratitude goes out to my peers and the batch of 2019-2020 for their support and friendship. I would also like to thank the Trinity VR community for their help. I would like to extend my sincerest gratitude to the admins at SCSS labs who gave endless support in ensuring the availability of remote servers. During this time of crisis and remote work, this dissertation would not have been possible without their support. Last but not the least, I would like to thank my family for their trust and belief in me.

SIDDHARTHA BHATTACHARYYA

*University of Dublin, Trinity College  
September 2020*

# Music Instrument Localization in Virtual Reality Environments using audio-visual cues

Siddhartha Bhattacharyya, Master of Science in Computer Science  
University of Dublin, Trinity College, 2020

Supervisor: Aljosa Smolic and Cagri Ozcinar

This research work aims to develop and assess the capabilities of convolution neural networks to identify and localize musical instruments in 360 videos. Using audio and visual cues from 360 video frames along with object detection technologies, sound source separation technologies and sound classification technologies, the research aims to provide a single demonstrable unit that highlights the location of the musical instruments in a video segment, along with their corresponding annotations in the form of bounding boxes. The research is extended to 360 video frames as they are an essential format of a typical virtual reality experience. An input 360 video is split into its constituent audio-visual components, the former being .wav files used as inputs to the sound classifier model, and the latter being the visual image frames being used as input to the object detection framework that outputs the annotation and localization information. An all-inclusive demonstrable unit showcases both the models' functionality and can be used on two-dimensional video frames as well.

# Summary

The research introduces the motivation behind the experiment and the objectives it aims to achieve, followed by a technical insight into the technologies and state-of-the-art tools used to develop the models. Some related and similar work along with state-of-the-art models are discussed to establish a benchmark for our research at hand. After which the proposed method is discussed in detail including the implementation strategy that highlights how the state-of-the-art tools and technologies have been modified and extended to achieve the objectives of our research. The data set collection process, sources and statistics are also described, following which the details of the experiments conducted are discussed. The results of the experiments conducted with various parameters are described and the various demonstrable units along with their implementations are discussed. Finally, the research concludes by acknowledging the scopes for future work and improvements that can be made in the proposed architecture.

# Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>Summary</b>	<b>v</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>ix</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Objective . . . . .	2
1.3 Structure . . . . .	3
<b>Chapter 2 Fundamentals</b>	<b>4</b>
2.1 Virtual vs Augmented Reality . . . . .	4
2.1.1 360 Videos . . . . .	5
2.1.2 Capturing 360 Videos . . . . .	5
2.2 Convolution Neural Networks . . . . .	6
2.3 Object Detection . . . . .	11
2.3.1 Methodologies . . . . .	12
2.3.2 Evaluation Metrics . . . . .	12
2.3.3 Types of Object Detectors . . . . .	18
2.3.4 Feature Extractors . . . . .	21
2.3.5 Applications . . . . .	23

2.4	Tensorflow Object Detection API . . . . .	24
2.5	Sound Source Separation . . . . .	26
2.6	Audio Classification . . . . .	28
2.6.1	Evaluation Metrics . . . . .	29
2.6.2	Applications . . . . .	31
<b>Chapter 3 Related Work and State-of-the-art</b>		<b>32</b>
<b>Chapter 4 Design and Implementation</b>		<b>35</b>
4.1	Proposed Method . . . . .	35
4.2	Data Acquisition . . . . .	36
4.2.1	Images Data Set . . . . .	36
4.2.2	Audio Data Set . . . . .	38
4.2.3	Ground Truth Set Up . . . . .	38
4.3	Experiments . . . . .	42
4.3.1	Object Detection Model . . . . .	43
4.3.2	Audio Source Separation . . . . .	45
4.3.3	Audio Classification Model . . . . .	46
<b>Chapter 5 Experimental Results</b>		<b>48</b>
<b>Chapter 6 Applications and Demonstrable Units</b>		<b>53</b>
<b>Chapter 7 Conclusion</b>		<b>55</b>
7.1	Limitations . . . . .	56
7.2	Future Work . . . . .	57
<b>Bibliography</b>		<b>59</b>
<b>Appendices</b>		<b>68</b>



# List of Tables

2.1	Activation Functions . . . . .	9
2.2	Precision and Recall formulae . . . . .	13
5.1	Results on base object detection model . . . . .	48
5.2	Results on fine tuned object detection model . . . . .	50
5.3	Results on audio classification . . . . .	52

# List of Figures

2.1	McCulloch-Pitts single Perceptron for boolean operations . . . . .	7
2.2	Biological neuron perceiving an image. . . . .	7
2.3	Convolutions . . . . .	8
2.4	Pooling with 2x2 filter . . . . .	10
2.5	A simplistic representation of a convolution neural network. . . . .	11
2.6	Mean Average Precision Calculation . . . . .	16
2.7	The Tensorflow Object Detection API. . . . .	26
2.8	Spectrogram representations of musical instrument sounds. . . . .	29
2.9	Categorical accuracy calculations. . . . .	30
4.1	High-level proposed method architecture. . . . .	36
4.2	Images data set information . . . . .	37
4.3	Images pre-processing . . . . .	39
4.4	Ground truth statistics . . . . .	41
4.5	High-level proposed method architecture. . . . .	43
4.6	Deep Audio Prior Network . . . . .	45
4.7	Keras MobileNet architecture. . . . .	46
5.1	Object Detection accuracy and loss plots against iterations . . . . .	50
5.2	Audio classification loss and accuracy plots . . . . .	52
6.1	Google Colab Notebook UI/UX application . . . . .	54
1	Visual Object Tagging Tool . . . . .	69
2	360 image object detection with 1 instrument . . . . .	70
3	360 image object detection with 2 instruments . . . . .	70

# Chapter 1

## Introduction

This section introduces the motivation behind the research at hand and sets up the grounds on which the research was conducted.

### 1.1 Motivation

The advent of *Virtual Reality*, abbreviated as *VR* has opened up a myriad of possibilities with respect to computer vision problems. It is essentially an extension of the two-dimensional environment, however, in addition to the traditional features of two dimensional entertainment, it takes into consideration the viewers inputs and actions. The most prominent of such VR experiences are the ones which are had through the use of *Head Mounted Devices*, abbreviated as *HMDs* such as *Samsung Gear VR* [1] or the *Google Cardboard* [2]. These devices enable viewers to experience and interact with VR environments with ease. VR has found its applications in a number of fields such as those of *360* or *omnidirectional videos* [3] which we use in our research, fully immersive gaming consoles such the *Playstation VR* [4], 3D animations etc. The background and technical details of omnidirectional videos that we use in our research is explained in Chapter 2.

Our research is also motivated from the aspect of *Music Information Retrieval* abbreviated as *MIR* [5, 6] which can be defined as extracting characteristic features from pieces or segments of music that can be in video format or audio format or both. The characteristic features of any musical piece can include the information regarding

its sound tracks, the notes being played, the class of instruments being used. Such information can be retrieved and manipulated for various other applications. MIR finds its application across a number of fields such as signal processing, instrument identification and categorization [7] where each piece of music instrument is identified based on its sound, music notes recognition [8] which includes the identification of musical notes being played for educational purposes or transcription, source or track separation [9, 10] which involves the segregation of instruments and their respective tracks, and even music generation [11]. We have developed our work on instrument identification and localization using cues from the video frames which form the visual component, as well as using the audio streams which forms the audio component.

A combination of the aforementioned concepts forms the basis of this dissertation. We have used music videos in VR environments as our target data to identify and localize the instruments. It can be considered analogous to the X-Ray feature of *Amazon Prime Video* [12], which displays the details of the actors on screen, their related work, what roles they play in the current show etc. It is an application to extend user knowledge, and is being performed in real time. The research at hand is somewhat similar, wherein we aim to identify musical instruments active in a particular video frame of a 360 enabled video file.

## 1.2 Objective

The aim of this research is to experiment on and assess the capabilities of convolution neural network models capable of identifying and localizing musical instruments using video frame images as well as audio segments from a 360 video. A part of this objective also includes curating applications and demonstrable units that can showcase the achieved results. The research aims to delve into the various approaches through which this can be achieved and also conduct a thorough investigation of the accuracies achieved by the studied approaches. Any music video definitely contains visual and audio features. The research aims to study these two features individually as separate inputs to 2 distinct neural network architectures. The first architecture works on video frames while the latter on audio segments.

To detect any activity in an image a common approach used is object detection, wherein a neural network is trained to detect and learn regions of interests. The neural

network learns to identify only those objects which are predicated as important during the training process. This process of object detection is utilized to help our developed model detect a collection of selected musical instruments in image frames and is then extended to identify musical instruments in video frames stitched together. The object detection framework delivers the instrument annotations and positional data in the form of bounding boxes. We also make use of the audio embedded within the video file. The video may contain multiple instruments playing at the same time. Source separation techniques are studied in an attempt to segregate the various instrument sounds. Once the sounds have been separated the individual audio segments are used as an input to the second neural network which classifies the instrument.

Combining object detection, source separation and audio classification techniques, we are able to develop a single demonstrable unit that generates a new video file with annotated and localized musical instruments from the input video file. The research also demonstrates the capabilities of the object detection model to be exported as an android application to be used for live object detection.

## 1.3 Structure

The dissertation is divided further into 7 chapters described as follows:

- Chapter 2 introduces the fundamental concepts and tools used in the dissertation.
- Chapter 3 explores related work and describes state-of-the-art methodologies implemented in similar fields of research.
- Chapter 4 delves into the implementation of our proposed methods, the network architectures and various models trained.
- Chapter 5 discusses the results of the conducted experiments.
- Chapter 6 gives an overview of some of the demonstrable units curated to test our developed models.
- Chapter 7 concludes the research and provides details about the research limitations and scope for future work .

# Chapter 2

## Fundamentals

This section delves into the basic technical fundamentals behind the research conducted. It explains background information on the technologies and tools used, accuracy metrics used and their calculation.

### 2.1 Virtual vs Augmented Reality

Since our research includes omni-directional videos which are a form of VR, it is essential to know what VR actually means. VR can be defined as an extension of the traditional two-dimensional audio visual experience [13]. In the traditional sense the viewer has no way to interact with the audio-visual interface except to play, pause, control the volume levels and other basic functions. However, VR enables the user to be a part of the simulated environment. This gives the user a fully immersive experience wherein they can directly interact with the simulated environment and see what they choose to see. In order to achieve this fully immersive experience a dedicated device that enables the user to interact with the simulated world is required. This is achieved using *Head mounted devices* abbreviated as *HMDs* which are essentially audio-visual headsets that serve as the gateway to the VR world. Some of the major HMDs available today are *Samsung Gear VR* [1], *Google Cardboard* [2], the *Oculus Rift Series* [14] and gaming VR consoles such as the *Playstation VR* [4]. Therefore, VR's main diversion from traditional two dimensional formats is in the way the environment is viewed. The user is not restricted to view a flat screen but can now

be a part of the simulated environment and actively interact with it.

Augmented reality is slightly different from VR wherein the environment as we see it is manipulated or augmented in a certain way with the help of computer technology. It involves rendering of three-dimensional objects on our own reality plane as if it were actually present. For example, the ***IKEA Place*** [15] app uses augmented reality to help viewers simulate how particular furniture would look in their apartment.

An essential format of a VR experience are ***360*** or ***omni-directional videos***, which we have used for our research. The following section describes them in further detail.

### 2.1.1 360 Videos

***360 videos*** or ***omni-directional videos*** are the most common format of a VR experience. The audio visual format constitutes capturing an entire surrounding environment as opposed to the traditional format of two-dimensional videos wherein the frames captured only include a particular perception range. In 360 videos the viewer cannot view his entire surroundings at once which is the case in two-dimensional videos. In this way 360 videos mimic the real environment and brings it to the viewer in the form of VR. It is also not compulsory to use an HMD to view 360 videos. Any video player with 360 video compatibility can aid in viewing 360 videos and have controls on the player that help them navigate the entire surrounding using an input device. ***YouTube*** [16] also supports this functionality. However, this negates the most characteristic feature of 360 videos, which is the fully immersive VR experience that can only be fully experienced using an HMD or some other VR simulator. For this very reason of being able to view 360 videos in traditional 2-dimensional format, we have chosen 360 videos as our test subject so as to not be totally dependent on HMDs and VR simulators for our experiments.

### 2.1.2 Capturing 360 Videos

Capturing a 360 video requires that the recording devices capture all of its surroundings instead of just what is in front of it or in a rectangular perception range, which is the case with traditional cameras. Specialized 360 cameras help capture the entire surrounding environment and format them into a fully spherical view. Placement of the

camera defines its surroundings and the content that is captured. Most such cameras nowadays contain dual lenses and there is no additional post-record processing such as stitching required to get the final spherical omni-directional videos. Such cameras for example are the **Rylo** [17] and **GoPro** [18] Max. However, certain cameras like the **Yi 360** [19] require post processing before the fully spherical form of the 360 video can be obtained.

## 2.2 Convolution Neural Networks

The concept of biological learning based on the capabilities of the human brain traces its way back to the 1940s when two researchers **Warren McCulloch** and **Walter Pitts** put forward the idea of extending the phenomenon of activation of neurons inside a human brain to the world of artificial supervised and unsupervised learning. Their works included the implementation of basic logical operations such as *and*, *not* and *or* in the form of artificial neural connections, which led them to believe that if such basic operations can be fulfilled with artificial neural networks, then any other operation conceivable to mankind can be replicated with combinations and activation of these artificial neurons [20]. The single layer **Perceptron** with its operations are shown graphically in Figure 2.1

Neural networks are biologically inspired artificial learning networks that imitate a mammal's visual capabilities of perceiving its environment and processing that information as a potential pattern recognition system using a layered architecture of neurons in the brain. In biological terms a neuron accepts information in the form of electrical signals from its input nodes called *synapses*. These *synapses* have the capability of amplifying or reducing the strength of the input signal. Once all inputs have reached the neuron, the neuron cell decides the output based on a threshold and sends the output to its neighbouring neuron using an *axon* which is a connection between the two neurons. This architecture repeats itself in the form of a mesh to create an intricate perceptory network. This very basic function of information processing and transfer to retrieve a final output, as depicted in Figure 2.2 is replicated in artificial neural networks also called a **Perceptron**

A **Convolution Neural Network** is essentially an artificial learning network that performs an operation called *convolution*. Abbreviated as **CNNs** they usually



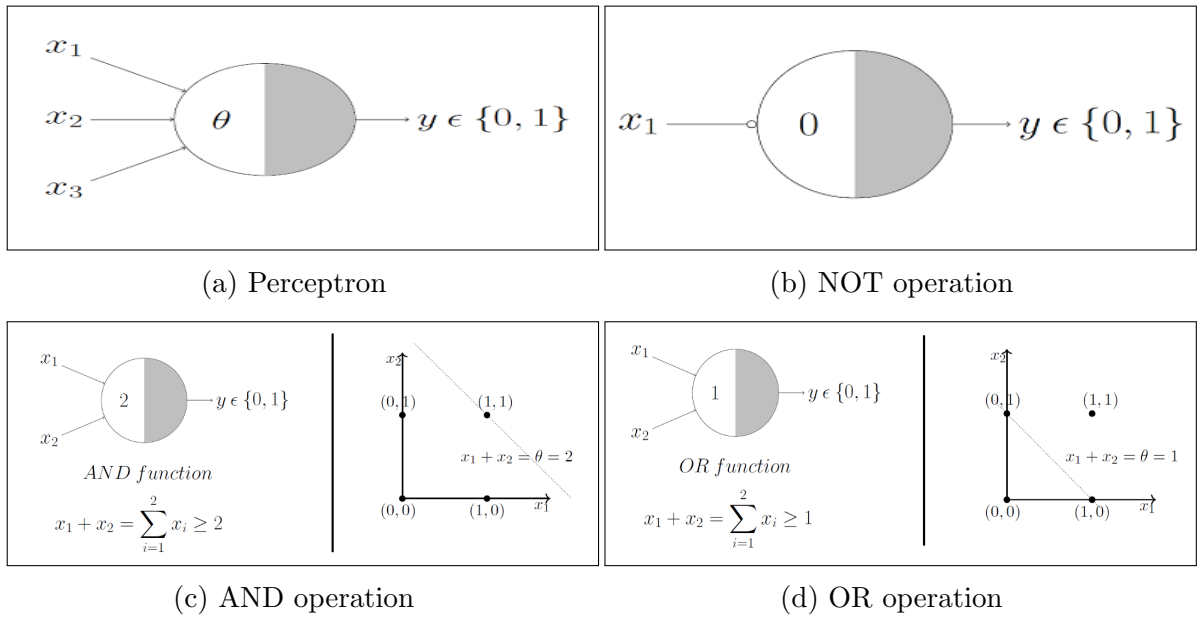


Figure 2.1: McCulloch-Pitts single Perceptron for boolean operations

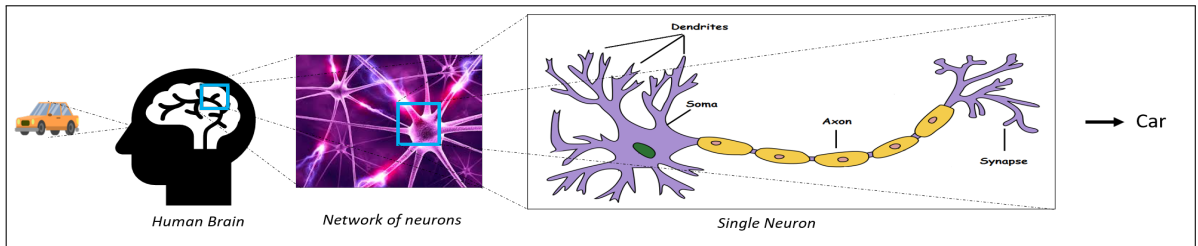
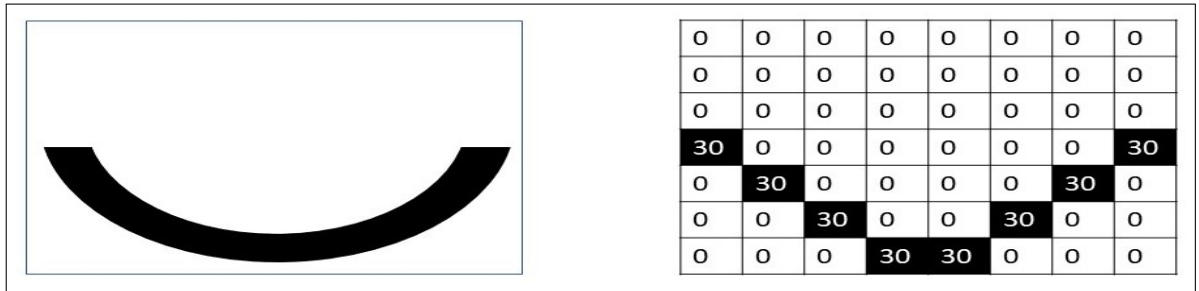


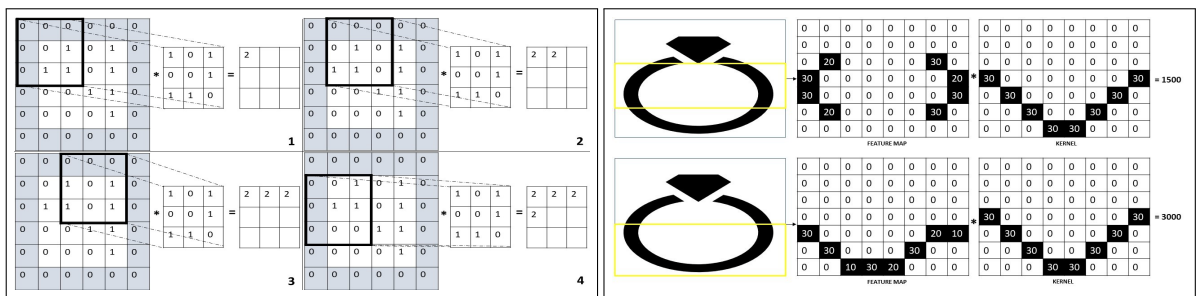
Figure 2.2: Biological neuron perceiving an image.

contain multiple layers, wherein the output of one layer is propagated to the input of its neighbouring layer after processing the information into a more complex format, until a final generalized form of the information is retrieved. A CNN is comprised of the following building blocks:

- **The Convolution Block:** This block serves the purpose of *feature extraction* or identifying the salient feature that we need our model to detect. It has a *convolution layer* and the *pooling layer*. The convolution layer uses a *kernel* or *filter* to extract or identify important features within the input information. The *filter* or *kernel* is the feature that the layer tries to detect in the input information through the operation of convolution. For example Figure 2.3a shows a kernel



(a) Sample Kernel and its matrix representation



(b) Convolution Operation

(c) Matching features

Figure 2.3: Convolutions

in image format and also in the form of its corresponding 2D matrix as seen by the layer. The convolution operation is shown in Figure 2.3b. The first matrix is the input feature with a single row and column of *padding*. Padding cells are added to the actual image information in order to give equal importance to the boundary cells of the actual image. The padding cells are always valued at 0 so they do not effect the actual convolution weightage. The *kernel* or *filter* is moved along the input matrix with a particular *stride*. The stride value can be increased to 2 or 3 to make the filter shift across the input at 2 or 3 columns and rows at a time. This helps reduce the size of the output and also the number of operations thus reducing computational complexity. All CNNs have this stride input as a hyperparamter which decides the accuracy of the model. As seen from Figure 2.3b the kernel is shifted across the input matrix with a stride of 1 in this case to get the dot products of the input matrix and the kernel which gives the final *Convolved Feature*.

From Figure 2.3c we try to extract the feature represented in the kernel from

a larger image represented as a matrix. We see that the convolution results for the first feature in the yellow box yields a lower score, which shows that the representation is not present in that frame. The second feature we get a higher convolution score which suggests a larger probability of finding the feature in that particular frame. Some of the activation functions used in convolution layers are defined in Table 2.1. **ReLU** or **Rectified Linear Unit** is the most commonly used activation function. The function ignores all negative activations while the positive ones are left as is. While ReLU is a linear function and the **Sigmoid** function also known as the logistic function is a non-linear function and restricts the output between 0 and 1. This function is generally used when networks need to predict the probability of a class. The **Softmax** function is similar to the Sigmoid function, however it converts prediction scores into probabilities that sum to one. This function is also widely used for multi-class classification tasks as it outputs probabilities.

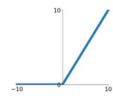
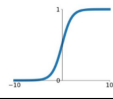
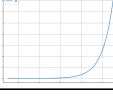
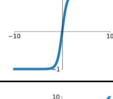
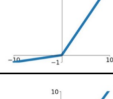
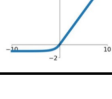
Function	Equation	Graph
ReLU	$\max(0, x)$	
Sigmoid	$\frac{1}{1 + \exp(-x)}$	
Softmax	$S(y_i) = \frac{\exp y_i}{\sum_j \exp y_j}$	
tanh	$\tanh(x)$	
Leaky ReLU	$\max(0.1x, x)$	
ELU	$f(x) = \begin{cases} x & \text{for } x \geq 0 \\ \alpha(\exp(x) - 1) & \text{for } x < 0 \end{cases}$	

Table 2.1: Activation Functions

The **Pooling Layer** serves the task of reducing the model size, number of parameters so as to reduce the computational overhead of the network by only

extracting the relevant information from the outputs of the convolution layers, and ignoring other unwanted predictions. In other words, a convolution layer outputs a set of convolved values as seen above, while the pooling layer extracts either an average of them, known as **Average Pooling** as seen in Figure 2.4b or takes the maximum value from them, known as **Maximum Pooling** as seen in Figure 2.4a. The images show a 2x2 filter pooling from a 4x4 feature map. The task performed by the Pooling Layer is also known as **Downsampling**. Every convolution layer is followed by a pooling layer to extract only the important or averaged information from the output of a pooling layer.



Figure 2.4: Pooling with 2x2 filter

- **The Fully Connected Block:** The final blocks of a CNN is the fully connected layer which is usually used for simple classification tasks, and categorizing the output based on a collection of classes provided to the network. The fully connected block comprises the classical neural network structure wherein each neuron or node is connected with every other node, which is different from the convolution layer as discussed above. The convolution layer is not fully connected, wherein it only works on features that would be essential for the prediction task at hand. The activity of a fully connected layer in a CNN is also called **Flattening**. The features that prioritize the target label most accurately are decided by the neurons in the fully connected block.

The main reason to use a convolution neural network is to optimize large scale problems such as image classification and object detection, where the input is in the form of pixel information. A single coloured megapixel image that is sized as  $1000 * 1000$  pixels essentially has  $3 * 1000 * 1000 = 3000000$  input features to be handled, as the 3 colors RGB also forms a part of the input feature. Even with a neural network with 1 hidden layer (1000 neurons) and 1 output layer with 10 neurons there would be

$(3000000 * 1000) + (1000 * 10)$  there would be approximately 3 billion weights and parameters to be learned, which would considerably increase computational overhead. To manage such large volume and scale of input, CNNs and Deep Learning strategies are established where all neurons are not fully connected in the convolution layer and incorporate convolution and down-sampling operations as demonstrated above. A simplified version of a convolution neural network is shown in Figure 2.5.

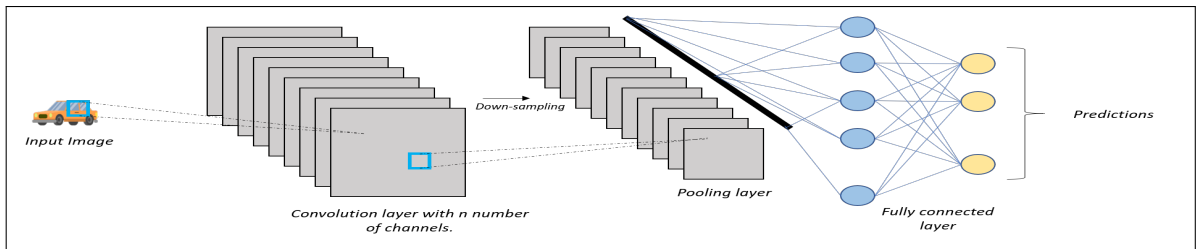


Figure 2.5: A simplistic representation of a convolution neural network.

## 2.3 Object Detection

Object detection is a computer vision technology and can be defined as the process of identifying, localizing or annotating objects of importance within a particular region of interest. Object detection is usually performed with a known set of classes, that is, the object detector should know prior to the training or detection what it is trying to detect. Object detection can be performed on static images, video files and even live stream data. There are numerous object detection applications which range from fields of research such as scene analysis, image segmentation to real world applications such as facial recognition, autonomous driving, surveillance and security. The learning based object detectors use CNN backbone architectures to extract salient features from the data set and also to generate the resulting classification and localization information [21]. Object detection methodologies including feature extractors and accuracy matrices are discussed as follows.

### 2.3.1 Methodologies

Object detection methodologies are typically based on pattern-matching or based on learning methods. Early studies in object detection involve pattern matching that go back as early as the 1970s, such as the works of Fishchler and Elschlager [22] which was based on defining a set of patterns or primitives to detect whether a certain input stream of data (any visual object) was actually a photograph or not. However, recent works on template and pattern matching have also shown improvement in this particular methodology. For example the works of Nguyen, Li et al. [23], which involve weights and distance transformation information to detect a collective class of objects in images.

Learning based approaches include techniques like CNNs, Deep Learning, Adaboost, SVM etc. These learning based approaches create a trainable network that learns on a pre-defined ground truth data set until a particular loss function is minimized [24]. Learning based classification paved the way for similar research as it proved to be more robust and efficient. Some of the modern problems that have been solved using object detection mechanisms usually involve human-object interaction. Object detection provides a way to segregate the two and helps to identify the object or the actions that are deemed important for a particular application [25, 26].

Most object detectors use the '*sliding window*' technique to identify important objects in a visual frame. The dimensions and depth of the sliding window are set as hyper-parameters and affect the accuracy of the detection. This window slides over the entire image frame based on the stride factors and matches areas of interest. The sliding window scheme is common application and forms the basis of all deep learning strategies [27].

### 2.3.2 Evaluation Metrics

Information retrieval and object detection projects have a specific mechanism for measuring their respective model accuracy. One such metric is the ***mean average precision*** abbreviated as ***mAP***. The mAP is considered as a culmination of the accumulated precision and recall scores of the model developed at several stages of the training. To understand mAP we define certain key terms that would be crucial in its calculation. With reference to Table 2.2 they are as follows:

- **Ground Truth:** For any machine learning model the trainable data set is considered as the ground truth, based on which the model learns the various characteristic features. This is a fixed and immutable collection of data.
- **True Positives:** When the actual class and the predicted class is positive the prediction is considered True Positive.
- **False Positives:** When the actual class is negative however it is predicted as positive by the model, the prediction is considered as a False Positive.
- **False Negative:** When the actual class is positive however it is predicted as negative by the model, the prediction is considered as a False Negative.
- **True Negative:** When the actual class and the predicted class is negative, the prediction is considered to be True Negative.
- **Precision:** Precision is defined as the percentage of predictions that are correct. Mathematically it is defined as the ratio of the True Positives and the total predicted positives.
- **Recall:** Recall is defined as the measure of how good the positives found are. It is also known as *True Positive Rate* or *Sensitivity*. Mathematically it is defined as the ratio of the True Positives and the total ground truth positives.

		Actual		Formulae	
		Positive	Negative	Precision	Recall
Predicted	Positive	TP	FP	$\frac{TP}{TP + FN}$	$\frac{TP}{TP + FP}$
	Negative	FN	TN		

Table 2.2: Precision and Recall formulae

From the definition of precision and recall we can say that precision is essentially the number of selected items that are relevant, while the recall is how many of the relevant items are selected. Therefore, for object detection models, there is always a trade-off between precision and recall. We would always want the number of True Positives to be high to get a good precision score and decrease the number of False

Positives, this would in turn decrease the recall score. Alternatively, if we reduce the number of False Negatives, the recall score would increase but would in turn decrease the precision score.

A generalized calculation of mAP can be defined considering the following variables:

- A defined ground truth labelled data set  $D$  with  $N$  items such that  $D = (d_1, d_2, \dots, d_N)$
- A test item  $T$  that is to be queried against the ground truth data set  $D$
- A confidence scoring function  $CF(T_i, T_j)$  that compares the similarity between the 2 items  $T_i$  and  $T_j$  and returns a confidence score.
- An ordered data set  $D_{sorted}$  that contains the data items of  $D$  in order of their confidence scores as retrieved from the above function.
- $P$  the number of ground truth positives in the data set  $D$  for the test image  $T$ .

After comparing the test item with each of the data items in  $D$  and retrieving their confidence scores, the ordered data set  $D_{sorted}$  is created and the average precision for each of the items in  $D_{sorted}$  is calculated. Let the average precision at  $k^{th}$  item in  $D_{sorted}$  be defined as  $AP_k$ . Thus, we can define the Average Precision for an item  $k$  in  $D_{sorted}$  as:

$$AP_k = \frac{\text{True Positives Seen Till } k}{k} \quad (2.1)$$

The overall mean average precision can be defined as:

$$mAP = \frac{1}{P} \sum_{i=1}^N AP_i \quad (2.2)$$

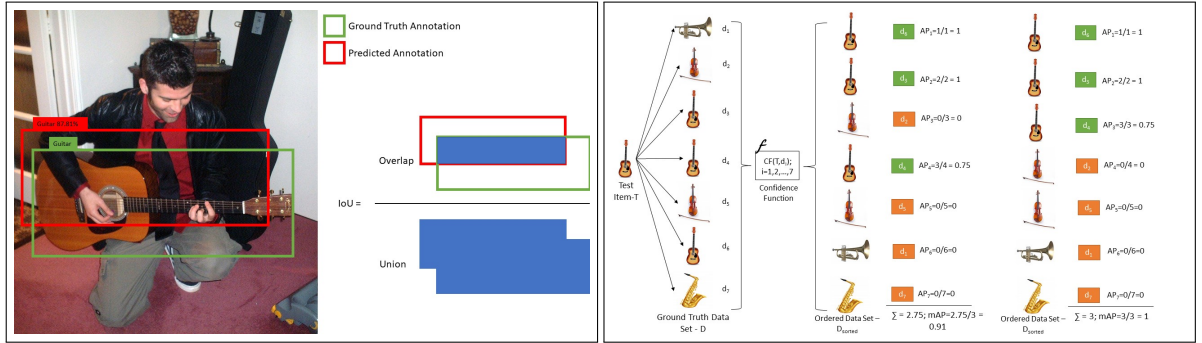
An example can be seen from Figure 2.6b we see that the test image  $T$  of a guitar is compared with 7 images from the ground truth data set  $D$  containing 3 Ground Truth Positive items, specifically  $d_3, d_4$  and  $d_6$ . The model's confidence function returns the ordered data set  $D_{sorted}$  where the 4 items,  $d_2, d_5, d_1$  and  $d_7$  have been found to be False Positives and 3 items,  $d_6, d_3$  and  $d_4$  have been found as true positives ordered according to their decreasing confidence levels. Using the formulae above we see that



the mAP for the first ordered data set is  $0.91$ . For a perfect mAP of 1 all the true positives should be in the beginning of the data set as seen from the last ordered data set which gets a mAP of 1.0.

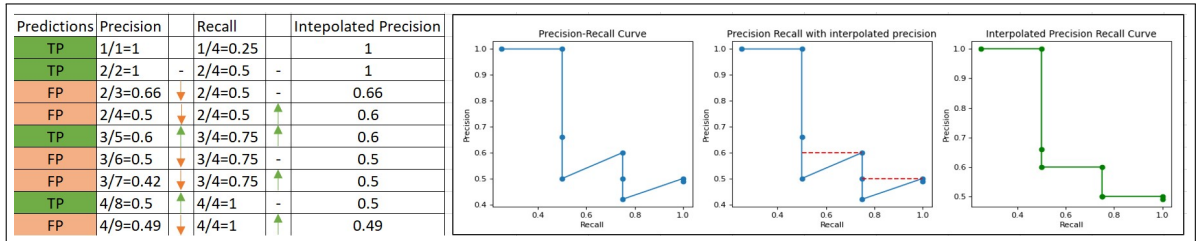
The calculation of mAP for object detection utilizes a hyperparameter called ***Intersection of Union*** abbreviated as IoU whose value determines whether a predicted annotation should be considered a True Positive or a False Positive. It is defined as the ratio of area of overlap and the area of union between the bounding boxes of the ground truth data and the predicted data. For example, from the Figure 2.6a the ground truth annotation is shown as the green bounding box while the predicted annotation is shown as a red bounding box. As is clear from both the bounding boxes the two boxes are not fully aligned with each other, that is, there is some error in the prediction annotation. IoU helps decide if this error is large enough to consider the annotation as a false positive. Ideally the IoU is set as 0.5 and/or 0.75, that is if IoU is greater than 0.5 or 0.75 then the image is annotated correctly and is therefore a True Positive, else it is a False Positive. IoU can be considered as the localization error in object detection models. The prediction would be considered a False Negative, if the annotation is misclassified or there is no detection at all.

Since the true positive, false positive and false negative cases have been defined for object detection scenarios, we can now define the precision and recall scores for the predictions across the test data set by ranking the predictions based on the confidence level produced by the model. Analogous to the use case presented earlier with the 7 image test data set, the predictions are sorted by the confidence levels of the bounding boxes. However, the overall mAP is calculated slightly differently than the previous example. Object detection use cases make use of ***Interpolated Precision*** to derive the final mAP score of the model. With reference to Figure 2.6c if we sketch the precision recall curve for any object detection model it will be clear that the recall gradually increases as we go down the sorted predictions. This is because, from the definition, recall is the measure of how many positives have been predicted from among all the ground truth positives. Therefore, since we know the total number of ground truth positives, as we go down the sorted predictions, the number of true positives gradually increase and so does the recall ratio value. However, the precision score follows a squiggly zig-zag pattern, that is, increasing at times and decreasing at times. This is because, as seen from the previous use case, the first 2 predictions are true positives



(a) Intersection over Union

(b) Calculation of mAP



(c) Interpolated precision

Figure 2.6: Mean Average Precision Calculation

which increases the precision value, however, the third is a false positive which lowers the precision value, thus giving the curve a zig-zag pattern. An example of a precision recall curve with sample data is shown in Figure 2.6c. 4 out of 9 images are the correct images while the remaining are false. Which brings our total Ground Truth Positives = 4. The table shows the predictions sorted in descending order of the confidence score as retrieved by the model. We see that the recall scores stay the same with each prediction or increase, this is because the total ground truth positives is known. However, with every false positive found the precision decreases while with every true positive found it increases, thus leading to the aforementioned zig-zag pattern as demonstrated in the graph alongside. To diminish the impact of these zig-zag patterns which are caused due to imperfect ranking of the predictions and maintain the monotonicity of the precision, we use interpolated precision. The interpolated precision is the altered precision value at each recall point, which is calculated by taking the next highest precision value at that recall point. Mathematically it can be defined as:

$$\text{Interpolated Precision } P_{interp}(r) = \max\{P_i : R_i \geq r\} \quad (2.3)$$

Graphically, it can be defined as at each recall score  $r$ , we find the max precision at a recall score to the right of the  $r$ . The precision recall curve from Figure 2.6c can be seen to have the zig-zag patterns because the 2 True Positive predictions are at a lower confidence level and thus at the lower end of the sorted data set. In order to remove these anomalies the precision value at fourth recall level i.e. at points (0.5,0.5) is updated to the maximum precision value to the right which is (0.5,0.6). Similarly the precision value at seventh recall level i.e at points (0.75,0.42) is updated to (0.75,0.5). Finally, the interpolated precision recall curve is shown in green.

Now that we have our precision values interpolated at the various recall levels, we can calculate the average precision. This is done by calculating the area under the precision-recall curve. The area under the precision recall curve is calculated by taking the average of the precision value at equal intervals of the total recall scale. The recall values can only lie between 0 and 1. Therefore, the precision values at 11 intervals are averaged, which return us the average precision. The recall intervals are : 0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1. Mathematically:

$$AP = \frac{1}{11} \sum_{r \in \{0,0.1,0.2\dots 1\}} P_{interp}(r) = \frac{1}{11}(1+1+1+1+1+0.6+0.6+0.6+0.5+0.5+0.49) = 0.75 \quad (2.4)$$

For object detection cases the mAP is calculated as the precision for all the predicted classes of the object detection model. Our object detection model uses the **COCO** detection metrics [28] which calculates the average precision over an IoU of 0.5 which is the standardised method of AP calculation and also over an IoU of 0.75. This is done so as to achieve a comparative study of the model at various settings of the IoU hyperparameter as certain objects detected can be comparatively smaller in size than the others. The COCO detection metrics scheme calculates the average precision over a range of IoUs and scales and also calculates the average recall over a range of detections per image and scales. In addition to calculating the Average precision at  $IoU=0.5$  and  $IoU=0.75$ , the precision is also calculated by taking the average of precision values from 0.5 to 0.95 with a step size of 0.05. This is called the primary challenge metric and is denoted as  $AP@IoU[0.5 : 0.05 : 0.95]$ . Scales indicate the size of detection which are categorized into small detections (area less than  $32^2$ ), medium

detections (area between  $32^2$  and  $96^2$  ) and large detections (area greater than  $96^2$  ). The recall is averaged over 1,10 and 100 detections per image and also similarly over the small, medium and large scales.

### 2.3.3 Types of Object Detectors

The task of object detection is broadly categorized into two sub-tasks which are feature extraction or classification task that takes in an input image frame and provides the feature maps of the classification. This task is performed by several backbone supporting networks discussed further. The second and main task is of generating Regions of Interest (RoIs) or pooling these regions into feature vectors. This is the task of the object detection module [21]. Object detectors are categorized by the number of stages they have in their framework. There are two such categories described as follows.

#### Two Stage Object Detectors

The two stage object detector is named so because it essentially contains two distinct stages in its object detection framework. The first being the *Region Proposal* stage, wherein candidate objects are proposed as potential detectable entities with bounding boxes. This is called the pooling layer. The second stage includes extraction of features from these proposed regions of interest. Two stage detectors have higher accuracy, localization and classification scores than one stage detectors, however, they can be computationally slower and have low inference speeds. Some critical two stage detectors are the *Region based CNN* abbreviated as *R-CNN*, proposed by Girshick et al. [29], which was one of the first works to show the capability of a convolution neural network performing an object detection task. The R-CNN object detector is comprised of four modules, the first being the region proposal layer, the second layer extracts fixed length feature vectors from the proposed regions, the third layer is an SVM (Support Vector Machine) based classifier that classifies the defined classes while the last module is a regressor which predicts the bounding boxes. The R-CNN model suffers a drawback of forward passing each region proposed to the feature extractor which slows down the SVM classifier. An improved version of the R-CNN model named *Fast R-CNN*, was developed by Girshick and Ross [30] wherein the features are extracted at once from all the regions proposed and sent to the classification layer

for categorization. This improved the performance of the performance of the R-CNN model by 0.9% on the *PASCAL VOC* data set [31]. Further improvements to this model lead to the development of *Faster R-CNN* [32] which uses a dedicated neural network architecture for the region proposal network, thus further improving the inference speed. Furthermore, improvements to the R-CNN model were achieved by using masking layers for image segmentation purposes as in *Mask-RCNN* [33]. To improve processing and inference speeds further, *pyramid networks* [34] of feature extractions are used and the traditional RoI pooling layer is replaced with RoI alignment layer that extracts only a small portion of the candidate regions to be sent to the classification layer.

### One Stage Object Detectors

The One stage object detector does away with the region proposal stage and predicts the output bounding boxes directly from input frames. Thus it contains only a single stage and is faster as compared to its two stage predecessor. One stage detectors are often used in live object detection scenarios such as monitoring traffic flow or pedestrians and other surveillance tasks, wherein the input feed is a continuous stream of data. One of the major developments in one stage detectors is the *YOLO (You Only Look Once)* object detector developed by Redmon, Joseph et al. [35]. The YOLO architecture divides the input image into smaller equally sized grid cells. Each of these grid cells are responsible for making a single class prediction based on the fact if the center of the object being detected lies within that grid cell. A single grid cell can predict multiple bounding boxes, however, only one will be considered. The proposed or predicted bounding boxes are called anchors. Each grid cell's predictions have a box confidence score and a conditional class probability which corresponds to the likeness or probability of the correct class predicted. The overall confidence score of the prediction can be defined as: Confidence score = Confidence Score\*Conditional class probability. YOLO also does away with errors such as duplicate bounding boxes of the same object by using *Non-Maximal Suppression*. Non-maximal suppression orders the multiple bounding box predictions in descending order by their confidence scores and removes those predictions that have the same class as their previous predictions and an IoU score of greater than 0.5. Single stage detectors are faster however give lesser

accuracy in predictions. Testing on the PASCAL VOC data set YOLO gives an mAP of 63.4% at a speed of 45 fps, while Fast R-CNN achieves an mAP of 70% at 0.5 fps.

Further improvements to the YOLO pipeline came in the form of **YOLOv2** [36] which introduces batch normalization layers that helps improve the mAP by almost 2%. Batch normalization involves computing of the mean and variance of a input batches of pre-configured size and then normalize the resultant activations of these batches to have a mean of 0 and a variance of 1. YOLOv2 also uses a high resolution classifier, wherein the classifier is trained with image frames of 224 x 224 size, while the object detection task is performed with images sizes of 448 x 448. This improves mAP by almost 4%.

Improvements were made furthermore by the introduction of YOLOv3 [37] which replaced the single label classification using **Softmax activation** to multi label classification with logistic regression. It also makes three predictions per location and allocates an '*objectness*' score to them which is similar to the Feature Pyramid Network feature extractor [34]. YOLOv3 boasts 3 times faster inference speeds and is efficient in detecting smaller objects, however, it has a higher localization error and an AP@IoU=0.75 is significantly lesser.

More recent advancements in the current year include **YOLOv4** [38] which shows astounding improvements from their predecessors. YOLOv4 incorporates a **Bag of Specials (BoS)** features such as non maximal suppression, FPNs, skip connections, attention modules such as **Spatial Attention Modules (SAM)** and feature integration such as **Path Aggregation Networks (PAN)** which improves the detection and localization accuracy, and also incorporates a **Bag of Freebies (BoF)** features that do not advance the accuracy in any major way. Such features are data augmentation, label smoothing and regularization. An unofficial release of **YOLOv5** [39] also shows huge improvements in real time object detection accuracy with COCO Average Precision at IoU=0.5 score as 67.4%.

Another important contribution in the field of Single Stage Object Detectors is the **Single Shot Detector** abbreviated as **SSD** developed by Liu, Anguelov et al. [40] The single shot detector comprises of a single deep neural network architecture that predicts the classification scores and bounding box offsets for a pre-defined set of bounding boxes or anchors. The predictions are made on multiple feature maps extracted by the supporting backbone network of varying aspect ratios and scales. The

methodology requires a defined ground truth setup comprising of annotated images in the form of bounding boxes. During training the aforementioned proposed bounding boxes are matched with the ground truth boxes and the ones that are matched are considered positive. For the final detection SSD incorporates non-maximal suppression. The SSD pipeline adds several feature layers or fully connected layers beyond the feature extractor to make predictions of the bounding box offsets. The SSD512 version of the Single Shot Detector essentially takes in 300 x 300 sized input image frames and outperforms the YOLO pipeline in both accuracy (by 11%) and inference speed (by 14 fps).

This research also uses the Single shot object with a variety of base model feature extractors and compares the mAP accuracy of each architecture by fine tuning certain hyper-parameters. Some of the feature extractors are discussed in the following sections. The SSD incidentally gives best results compared to other object detectors for our research with input frames on 360 videos frames.

### 2.3.4 Feature Extractors

Feature extractors are used in all deep learning problems. The main aim of feature extraction is to extract only the necessary characteristic properties for the network architecture to work on, instead of the entire image frame. This considerably reduces the dimensionality of the network as it now has to work on a reduced set of input parameters rather than the entire input frame. These extracted salient features are often representative of the classes to be predicted. Using these extracted features the final predictions of the problem are made. For object detection tasks most feature extractors are generally pre-trained image classification models that serve as the backbone network for the classification task in object detection. The image classifiers that paved the way for modern feature extractors are the *AlexNet* [41], *SqueezeNet* [42], *Quantized Convolution Neural Networks* [43] among many others. Some of the backbone networks used in this research are discussed as follows.

The *MobileNet* classifier [44] developed by researchers at Google is one of the multi-task image processing neural networks that is generally used for image classification, facial recognition, landmark recognition and object detection tasks for small scale mobile-device-based applications. The MobileNet architecture introduces fac-

torization of the convolution layers with *depthwise convolution* and *point wise convolution*. The depthwise convolution separates each input channel and applies the convolution filter separately on each. The output from the depthwise convolution filters is then passed on to a 1 x 1 pointwise convolution that combines the results into a new output. Therefore, the filtering and combination tasks which are traditionally performed by a single convolution are now separated tasks in MobileNet. This separation considerably reduces the computation overhead and reduces the model size. A 3 x 3 depthwise convolution is used by MobileNet which reduces the computational overhead by at least 8-9 times. The MobileNet network architecture consists of a total of 28 layers, a full convolution layer followed by 3 x 3 depthwise separable convolution layers. Each layer is followed by a batch normalization layer and a ReLU activation layer, except the final layer which is connected to a softmax classifier. The MobileNet architecture also introduces two new parameters for scalability. The first parameter is the **Width Multiplier**  $\alpha \in (0, 1]$ . This parameter reduces the width of each network layer equally thus reducing the computational overhead and number of parameters. If the number of input channels are  $M$  and the number of output channels are  $N$  using the width multiplier they become  $\alpha M$  and  $\alpha N$  respectively. The second parameter is called the **Resolution Multiplier**  $\rho \in (0, 1]$  which is applied to each input image. These two parameters combined are known as the **Model Shrinking Hyperparameters** and help to generate a small sized model with acceptable accuracy and reduced latency. Object Detection results on the COCO data set with MobileNet feature extraction and SSD gives an AP@IoU=[0.5:0.05:0.95] of 19%, while with Faster-RCNN gives an AP@IoU=[0.5:0.05:0.95] of 16.4%.

Improved versions of MobileNet include **MobileNetv2** [45] that removes the ReLU activation after each layer as in its predecessor. MobileNetv2 architecture is comprised of a residual block and a downsizing block. Each block has 3 layers individually, a 1 x 1 layer with ReLU activation, a depthwise convolution layer and non-linear 1 x 1 convolution final layer. Experiments show that MobileNetv2 with SSD object detector has an AP@IoU=[0.5:0.05:0.95] of 22.1%. MobileNetv3 [46] further improved on this architecture by including Neural Architecture Search [47] that aims at creating a recurrent neural network that outputs a collection of models. The Neural architecture search then searches all possible combinations of these models to choose a collection that gives the best accuracy.



Another experiment on image classifier is the *InceptionNet* or GoogleNet [48] finds the sweet spot between very deep networks which are computationally expensive and prone to overfitting and overly wide networks to create a pipeline that uses multiple sized convolution filter making the network essentially a bit wider rather than deeper. It uses a 1x1 convolution layer before a stacked 3x3 and 5x5 convolution layer followed by a 3x3 max pooling layer to reduce the dimensions of the pipeline. This is a single module. The entire architecture of GoogleNet or InceptionNetv1 is made of 9 such modules, giving a total of 27 layers. *InceptionNetv2* [49] made use of factorization methods to divide the 5x5 convolution layer into two 3x3 convolution layers. This reduced the computational complexity as a 5x5 convolution is 2.78 time more expensive than a 3x3 convolution layer. In addition to the adjustments made in Inceptionv2, *Inceptionv3* [49] included *RMSProp* or *Root Mean Square Propagation Optimization*, factorized 7x7 convolution layers, batch normalization and label smoothing methods. These methods all together made the model less prone to overfitting and improved accuracy measures.

### 2.3.5 Applications

As stated earlier the applications of object detection methodologies ranges from research-based fields to real world scenarios. Deep learning methods has made these application easily achievable. Some of the applications developed are discussed as follows.

Sai and Sasikala [50] have developed an application that is able to identify potentially harmful objects such as knives, guns etc. using the *Tensorflow Object Detection API* [51]. In addition to detection and localizing these harmful objects, they have also counted the number of specific potentially harmful items in a particular image. Another similar application using the tensorflow object detection API is the work done by Hsieh, Lin et al. [52], where they have used the tool to detect vehicles and the distances between two vehicles in an attempt to warn the drivers if they were heading towards an accident. The result of the procedure would give an indicative warning if the driver was safe or in danger.

Facial recognition software is one of the key applications of object detection. It also extends to the field of security and surveillance. Some works done in this field are those of Sosorbaram, Batchimeg et al. [53], wherein they utilize unmanned aerial

drones as real time object detectors, in this case, identifying human faces. Facial recognition technologies have deviated from the traditional object detection methodologies towards biometric technologies. Some of the formidable front-runners in the race of facial recognition are Google and Facebook with applications like *DeepFace* (by Facebook) [54] which determines whether 2 images containing human faces are of the same person or not, *FaceNet* (by Google) [55] which uses deep learning methods to identify faces and link it to individuals with an accuracy of 99.63%. Amazon’s cloud-based facial recognition software named *Rekognition* [56] boasts the capability of identifying multiple faces in a single image frame and matches them against public data sets to identify individuals.

Apart from these object detection also finds its uses in applications like active speaker detection as in the works of Roth, Chaudhuri et al. [57], food detection, object tracking and facial recognition as in the works of Qiu, Lo et al. [58], optical music recognition application such as musical notes identification and transcription as seen in the works of Pacha, Hajič et al. [59]

## 2.4 Tensorflow Object Detection API

Researchers at Google, Huang, Rathod et al. [51], have developed a one stop shop for all things related to image classification and object detection. The *Tensorflow Object Detection API* makes for a ready to use, user friendly functional API which allows researchers to pick and choose the key elements that are required for any object detection task. This API provides a single point of entry into the process of object detection in the form of a configuration file defined with *.config* extension. This configuration file contains all the relevant information pertaining to the training and evaluation process that a particular object detection task entails, this includes, the training data set information, the validation data set information, the class labels files containing the list of classes the object detection is supposed to predict, the feature extractor details and lastly the object detector details. While the meta data features such as the training and validation files are fed into the configuration file as their respective absolute paths, the feature extractor and object detection details including their hyperparameters are fine tuned using *Googles’ Protocol Buffers* [60].

Protocol Buffers developed by Google are an XML-like message representation sys-

tem that is simple, platform and language independent and is made for '*serializing structured data*'. Protocol buffer serves as a service bus between the object detection model and its corresponding feature extractor and object detector counterparts, enabling the setting and un-setting of hyperparameters. Protocol buffers are stored with the *.proto* extension. The Tensorflow object detection package comes with a collection of *.proto* files for all available feature extractors. These *.proto* files are compiled into corresponding *.protoc* files using the google protocol executable binary. The command is given in Appendix B. This compilation is required so that the feature extractor details can be exchanged with the object detection model. The message structure along with their corresponding values are defined in these compiled *proto* files. Each *proto* file has its own message structure defined with its corresponding attributes. For example from Figure 2.7 the **faster\_rcnn.proto** file consists of three optional parameters **number\_of\_stages**, **num\_class** and **image\_resizer**. If one wants to update the **image\_resizer** parameters in the configuration file, the protocol buffer needs to look up the corresponding **image\_resizer.proto** file and then find the corresponding dependencies which can be seen as **min\_dimension** and **max\_dimension**. This channel of communication is maintained during the training or validation process by the Tensorflow Object detection API which acts as a buffer and simultaneously performs read and write operations to and from the compiled *proto* files and the configuration files to fine tune the object detection model.

Other relevant files required while using the Tensorflow object detection API are the **class labels file** saved with a *.pbtxt* extension which contains the relevant class labels to be detected along with their corresponding id numbers as seen in Figure 2.7. General information regarding the training procedure such as fine tune checkpoints, transfer learning models, epochs, validation epochs etc. are placed within the master configuration file along with other feature extractor and object detector details and hyper-parameters. The API trains and saves the trained model as checkpoint files which are converted to frozen inference graphs, again using the configuration file used for training. It is important to note that the object detection package is a part of a larger python directory structure. The repository of the object detection framework is contained within the *models/research* directory. To use the object detection framework, the current python executable path should include the packages within the object detection folder. This export command is shown in Appendix B. To begin training,

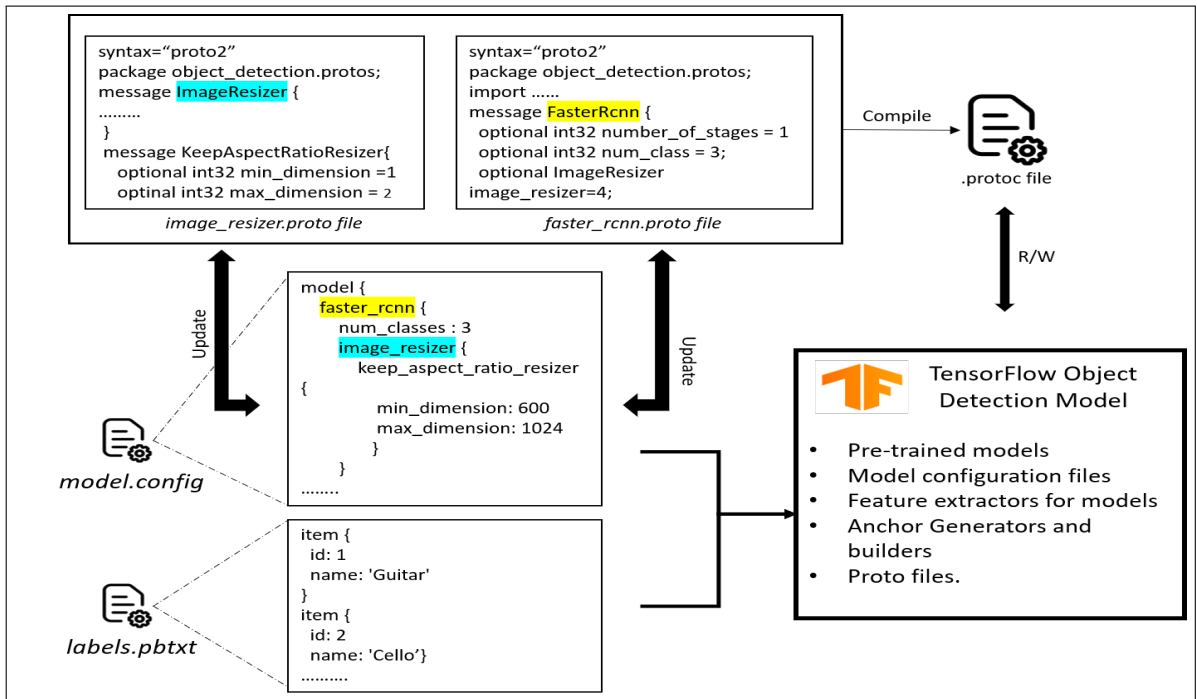


Figure 2.7: The Tensorflow Object Detection API.

firstly the model configuration file should be edited to include all required paths to training and validation records, other relevant hyper-parameters such as number of epochs, model checkpoint locations etc. Using the `model_main.py` script the training can be launched. A sample is shown in Appendix B. The `pipeline_config_path` parameter holds the path to the model configuration file.

## 2.5 Sound Source Separation

Sound source separation is a task that has been highly experimented upon, and its origins traces back to classic problems such as the *Cocktail Party Effect* coined by *Collin Cherry* in 1953. Cherry studied the capability of the human brain to isolate and focus on targeted sound sources in an environment with a variety of sounds originating from a range of sources, similar to a *cocktail party* environment, hence the name [61]. This research paved the way for attention models and filter models [62, 63] in an attempt to actively locate sound sources and separate them. The main task of audio source separation is to replicate the human brain's capacity to detect individual sound

sources in a multi-sound-source environment to an automated artificial architecture capable of imitating this natural ability.

An audio file constitutes of a mixture signal made up of several sounds originating from different sources. These sources may include human sources such as speech sounds [64], musical instrument sources including musical notes and singing voices [65] or sounds from other inanimate objects such as automobiles, animals, speakers etc. categorized as natural sounds [66]. Each classified sound source has its own unique characteristics which differentiates it from the other categories. The task is to cleanly and efficiently extract the different sound sources and their sounds in order for further processing and applications such as music transcription, classification etc. One of the most commonly used techniques to achieve source separation is called **Blind Audio Source Separation** abbreviated as **BASS** [67]. BASS involves processing of signals from microphone arrays [68] which capture the sound signals and masking them. BASS method involves localizing different sound sources spatially and apply masking methods in order to enhance the targeted sound source while dissolving the other sound sources. This method is repeated for all other individual sound sources. BASS methods are used to separate synthetic sound mixtures where the individual sound sources originate from point sources. BASS can be represented mathematically as defined by Vincent, Jafari et al. [67].

$$x_i(t) = \sum_{j=1}^J \sum_{\tau=-\infty}^{+\infty} a_{ij}(t - \tau, \tau) s_j(t - \tau) \quad (2.5)$$

From (2.5),  $x_i(t)$  are the channels for the mixture signal while  $s_j(t)$  are the single channel source signals,  $a_{ij}(t)$  are the mixing functions which may include information about spatial positions and transfers. The problem statement of BASS is to identify or estimate the single sound sources  $s_j(t)$  given the full mixture signal channel  $x_{ij}(t)$ . The two major sub problems that BASS needs to solve is estimating the number of sound sources or localizing them, and the second is to filter these sound sources to obtain their individual signals. **Beamforming** is one of the most common forms of filtering [69, 70]. It is also considered as a sound localization technique. Beamforming uses fixed stationary filters on the mixture signals and sums up the weighted results to estimate the sound sources. To separate the sound sources in this research we have used self-supervised blind audio source separation technique as developed by Tian, xu

and Li [71] which is discussed in Chapter 4.

Musical instrument sound source separation has been experimented upon quite extensively. Li and Wang [72] have proposed Time-Frequency masking techniques and use the fact that similar instruments have similar spectral features to their advantage in order to isolate sound sources in the time-frequency domain. Kim and Choi [73] attempt music source separation from polyphonic audio by incorporating non-negativity and shift-invariance. Other novel approaches to localizing and separating musical instrument sound sources are those of Zhao, Gan et al [74], where they have attempted to identify the audio waveform from individual pixels of a video frame. They incorporate the joint usage of audio and visual features to localize and separate sound sources without any manual intervention.

## 2.6 Audio Classification

Sound classification can be defined as the task of automatically identifying and labelling a sound source based on only the audio information. The human brain has the capability to connect an audio source to a semantic entity, for example, the sound being produced by an animal can be mapped to which animal is making that sound. Similarly, for our research, we are interested in identifying which particular instrument is producing a particular sound. Sound classification is similar to image classification, however, rather than depending on visual features, the model has to work on audio features, or the spectral representation of these features. Sound classification is different from automatic speech recognition or ASR, as the number of identifiable classes are larger. Specific benchmark data sets are also available for sound classification tasks such as for environmental sounds there are *UrbanSound8K* [75] and *ECS* (Environmental Sound Classification) [76] data sets. Additionally there are also musical instrument data sets which we have used in our thesis for instrument classification such as the *Philharmonia* data set [77] and the *IRMAS* or *Instrument Recognition in Music Audio Signals* data set [78].

Most classification tasks are achieved by CNNs and deep learning approaches. Sound classification tasks are no different. Several state of the art back-bone networks used for feature extraction and image classification have been used to classify sounds on the ECS data sets [79]. For example Boddapati, Petef et al. [80] have used

the AlexNet [41] and GoogleNet [44] CNNs to classify the environmental sounds data set and achieved an accuracy of 86% and 92% on the ESC and UrbanSound8K data sets respectively. One of the major approaches to classify sounds is to convert the audio signals to their image counterparts. These images are usually *spectrogram* images. Spectrogram represent the sound waveform in 2 dimensional graphs and provide the loudness or signal strength of the waveform at different time points of the audio. Therefore, a spectrogram representation of an audio waveform depicts the characteristic features or intensities of the sound at different points in time. A third dimension which is color is also added to spectrograms to depict this intensity. Spectrogram representations of musical instrument are shown in Figure 2.8

Therefore, essentially audio classification task can also be converted to the classical image classification task. Since, there are several backbone networks trained on a large array of image data sets, the problem scope is shortened. In our research, the separated sound sources are converted to spectrogram images of a specific dimension and fed into an image classifier network to learn certain characteristic features for different musical instrument sounds. This is discussed further in detail in Chapter 4.

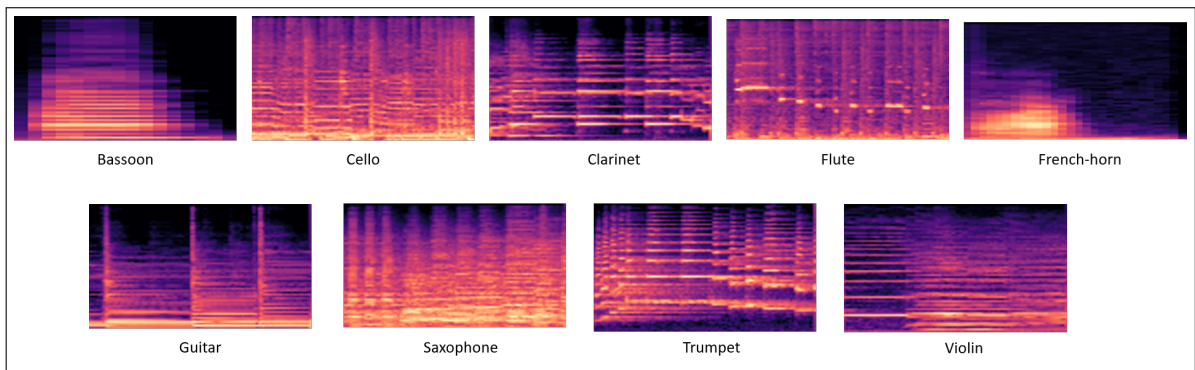


Figure 2.8: Spectrogram representations of musical instrument sounds.

### 2.6.1 Evaluation Metrics

A classification problem requires its accuracy measured in the form of the how correct the probabilities of each class predicted by the network are. The accuracy of a model is defined as the number of predicted values that match with the actual ground truth values. A number of accuracy measures have been defined for its classification prob-

lems. As a list of classes are defined for the classification model to predict from, the model makes its predictions in the form of probability or confidence scores for each of the classes. We have used the *Keras MobileNet* [81] application for our classification model which contains several pre-defined accuracy metrics functions. One such accuracy metric that measures probabilities of predictable classes is the *sparse categorical accuracy* metric. Sparse categorical accuracy takes the index of the maximum probabilities of each individual class and matches it with the ground truth index. The percentage of match is considered the sparse categorical accuracy. As an example from Figure 2.9 the we compare the column-wise indices of the max values from the ground truth data as well as the predicted data. We see that two indices match while the rest do not. Therefore, the sparse categorical accuracy is 50%.

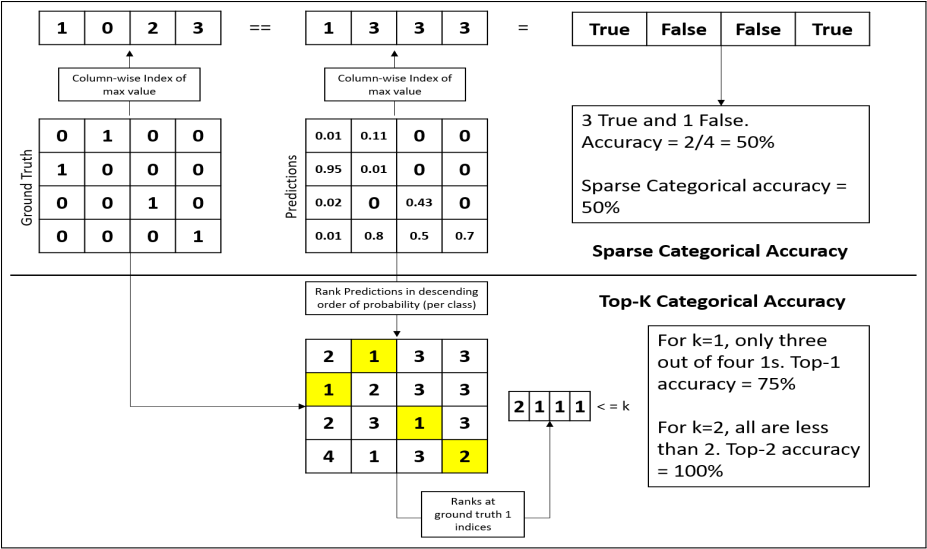


Figure 2.9: Categorical accuracy calculations.

Another metric used is the *sparse top-K categorical accuracy*, where  $k$  is a hyperparameter. It only considers the top  $k$  predictions based on their probability scores. The predictions made are ranked on their indices according to their probability scores in descending order and compared with the indices of the ground truth. The number of predictions in the correct positions less than equal to  $k$  are considered correct predictions. As an example from Figure 2.9 the predictions made from each class are given a rank, in this case from 1 to 4. Then, those ranks are taken from the true value positions in the ground truth data, highlighted as yellow. This gives us the ranks 2,1,1



and 1. Based on the value of  $k$  we get our overall top- $k$  categorical accuracy. If  $k=1$ , all predictions less than rank 1 are only 3 out of 4, therefore the top- $k$  categorical accuracy is 75%. If  $k=2$  then all predicted ranks are less than 2, which makes the top- $k$  categorical accuracy is 100%.

## 2.6.2 Applications

Audio classification models finds its applications in several fields of research. One of the most common application is *Automatic Speech Recognition*, abbreviated as *ASR* [82]. ASR is the technology where a machine is able to understand human speech and represent them coherently either in textual or audio format. ASR includes sound classification as its basis, as the network model needs to differentiate and identify various phonetic syllables and phrases and represent them correctly. A variant of ASR is *Natural Language Processing*, abbreviated as *NLP*. NLP is an artificial intelligence technology wherein the machine imitates a human's capability to process phrases and words, and also provide an appropriate response to achieve a desired function. Common examples of such AIs are virtual assistant technologies such as Apple's *Siri* [83], or Amazon's *Alexa* [84], which have the capability to understand human voice commands and carry out necessary actions based on those commands.

Several other interesting applications of sound classification in the field of music include musical note recognition [8]. This application can be further extended to automated music transcription applications [11]. And subsequently, identifying musical instruments itself using their characteristic sound features which we have implemented in our research.

## Chapter 3

# Related Work and State-of-the-art

This section delves into some of the related work that have employed the technologies and tools used by this research to achieve similar objectives. Roth, Chaudhuri et al. [57] have proposed a data set as well as an audio-visual trained model for *active speaker detection* in video frames. The research has led to the creation of a fully annotated data set wherein humans are labeled as either speaking or not and also whether their speech is audible or not. The lack of a fully annotated data set to measure state-of-the-art accuracy of neural network architectures for problems in similar fields such as speech recognition, speech transcription, video re-targeting etc was met by the creation of this data set. Videos from this data set [85] comprising of scenes from different movies of varying lengths were used to create the annotated data set highlighting active speakers and audio only. The end to end multi-modal model learns mappings from a sequence of faces and their corresponding audio signal to the probability of that face actually speaking and achieves a mean F1 score of 92.8%.

Morgado, Vasconcelos et al. [86] have proposed an architecture wherein they localize sound sources in a 360 spherical environment using heat maps. Most 360 video cameras do not incorporate spatial audio within them and restrict themselves to recording the audio in the traditional two-dimensional or mono format. *Spatial audio* is an essential part of virtual reality experience. The model proposed by this research localizes the sound sources by first separating them and then generating the exact or close to exact spatial sound for the entire video frame. The data set used contains of publicly available 360 videos which include general activities as well as people playing

musical instruments. The mono tracks form the input to the network and the output retrieved is spatial and localized audio within the 360 spherical view.

Owens and Efros [87] have worked on a multisensory separation application with several applications using self-supervised learning mechanisms. Using a multi-modal fused audio-visual convolution network and residual connections between the layers they have developed multiple models that support on-screen and off-screen audio separation, blind source separation from audio-visual cues and signals, separation of audio sources from a single video frame and also localization of audio source in the form of heat maps. The model developed can be fine tuned with custom data sets and parameters to fit other requirements.

Delforouzi, Tabatabaei et al. [88] in their work aim to track objects such as humans, vehicles, animals etc. in polar representation of 360 images which are essentially the outputs of a 360 degree camera. The deviation from object tracking in rectangular frames to object tracking in polar image frames is the challenge that they have targeted in their work. The distorted nature of polar images from 360 image frames poses a challenge for traditional object detection methods. However, they have proposed a novel RoI mapping technique in which the proposed regions are essentially denoted by small sectors of two concentric circles, thus establishing a polar or circular RoI detection mechanism. These RoIs are then forwarded to the object detection module for tracking and prediction.

Tian, Shi et al. [89] have proposed an architecture for cross-modality event localization in videos. They have developed a fusion network wherein the audio events are learnt from visual cues and vice versa. This is known as *cross-modality localization*. The models are then fused with the help of residual networks into one single model that can localize events based on actions in the audio mode as well as the video mode. The ground truth is extracted from the *Audio-Visual Event* data set or *AVE* data set [89]. The cross modality localization is performed using an audio-visual distance learning network that can measure the deviation in a particular event between the audio and visual cues.

The work done by Tepljakov, Astapov et al. [90] is involved in the acoustic localization of sound sources by representing sound in the form of visual attributes such as colors and shapes. The sizes of the shapes for instance are representative of the loudness of the sound. Therefore, creating a synesthetic experience of visualizing the

audio mode in visual format. The experiments are performed using *Distance of Arrival* or *DoA* techniques and *Mel-frequency cepstral coefficient* calculations, and finally experienced in a virtual reality environment using head mounted devices. The amplitude and spectral forms of audio waves are represented as spherical shapes of corresponding sizes.

Shreevathsa, Harshith et al. [91] have worked on identifying musical instrument sounds using CNNs as essentially an image classification task. Several classes of instruments are targeted to be classified by a simple neural network that learns the audio wave representation in the form of Mel-frequency spectrogram images.

Aytar, Vondrick et al. [92] propose a large scale convolution neural network trained on a considerably large unlabelled video data set to recognizing sound entities by making use of the synchronization of audio and video cues in video files. The large unlabeled video data set comprises of publicly available videos from *YouTube* [16] and *Flickr* [93]. The unlabeled video is dissected into its corresponding visual frames that are fed into image recognition pre-trained CNNs such as *ImageNet*. In addition to this the raw waveform of the unlabeled video's audio content is an 8-layer neural network named *SoundNet* that synchronizes with the outputs from the visual frame network. Experiments have also been conducted with a 5-layer version of this architecture. The experimental setup was tested on acoustic scene classification challenge data sets such as *DCASE (Detection and classification of Acoustic Scenes and Events)* [94] and *ESC50* [79]

Liu, Yang et al. [95] have worked on a weakly supervised multi modal separate musical instrument detector considering the visual as well as audio aspects of musical instrument detection. The research takes into consideration the connection between '*action*' and position of the '*object*'. It leverages the fact that the location of any action would be close to the location of the object being acted upon. Additionally, the research also considers the relation between an action and the corresponding sound the action produces. Keeping these in mind they have developed an 8 layer neural network with a global pooling layer as the final fully connected layer that localizes action and object pairs. The manually annotated ground truth for the experiment also consists of image frames with action and object location points.

# Chapter 4

## Design and Implementation

This section describes the low-level details of our proposed method with which we aim to answer our research question along with details of the data sets acquired and used to train the models. It also includes a low-level design implementation of dual model architecture including some of its applications and possible scope for improvement.

### 4.1 Proposed Method

In order to identify and annotate musical instruments, this research proposes a dual model architecture. The models are built respectively on visual cues and audio cues. The input for our detection architecture are 360 video files. Therefore, the two essential components of any video file are the image frames which forms the input to our first image based object detection model, and secondly the audio waveform, which forms the input to our sound source separation framework, whose output is subsequently fed into our sound classifier network. The image based object detection model outputs annotated and labeled image frames highlighting bounding boxes around localized instruments. The audio leg of the network outputs the classification scores of the instrument classes. Although the localization and annotation task is accomplished using only the visual cues, the audio classification task is added further to make use of both the visual and audio components of the input 360 video. A list of identifiable instruments or class labels is also maintained for both the networks, so that the models know which instruments it needs to identify. A high-level design of the method is shown in

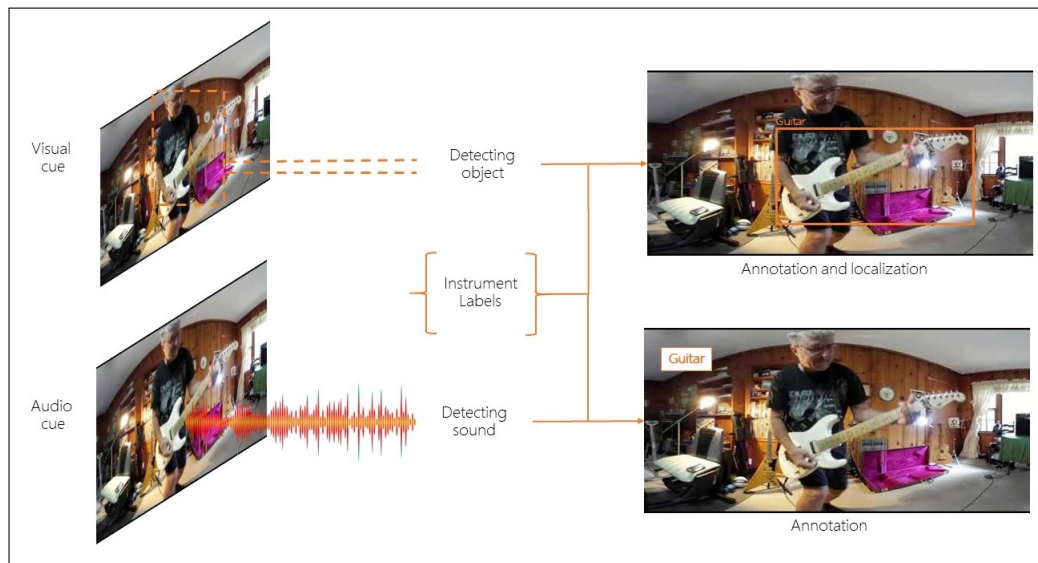


Figure 4.1: High-level proposed method architecture.

Figure 4.1

## 4.2 Data Acquisition

Our dual model design brings up the need to use two separate data sets based on the model’s input requirements. Our image based object detection model uses images to learn instrument specifications and features so that it can detect them in video frames. While our audio classification model works on audio segments sampled at a pre-defined wavelength level and channel. Both the data sets, their annotations and collection along with the ground truth setup for our models has been described as follows.

### 4.2.1 Images Data Set

The image collection used is a pre-collected data set gathered by Yao et al.[96]. This data set is collected on the premise of human interaction with different inanimate objects. Their research delves into automatically distinguishing whether people are actually interacting with the objects or are just holding them. For example, they have considered musical instruments in their research. A person can interact with a musical



Figure 4.2: Images data set information

instrument in two separate ways. One is to actually play them, while the other is to stand by them or hold them without playing them. In this regard they have accumulated several two dimensional images of humans interacting with different musical instruments in a variety of environments. The different musical instruments collected are *Bassoon*, *Cello*, *Clarinet*, *Erhu*, *Flute*, *French Horn*, *Guitar*, *Harp*, *Recorder*, *Saxophone*, *Trumpet* and *Violin*. These 12 visually distinguishable images are divided into two separate folder structures based on the depicted action with humans. The two separate folders are labelled as '*Playing Instruments*' and '*With Instruments*'. The images are distributed amongst varying background environments such as orchestras, auditoriums, domestic environments, parks, streets etc. Also the images have varying background colors and resolutions, so that the neural network can differentiate the same instrument effectively. Although our detection model does not consider the action of humans with the instruments, we have leveraged all the images collected irrespective of their actions with humans. In addition to this we have excluded '*Erhu*' instrument from consideration, because of its lesser known status. Since our research delves mainly in the virtual reality domain and there are not many videos with the *Erhu*, we have chosen to not consider it. Therefore, we have essentially collected unlabelled images distributed among eleven different classes. A total of 4209 images have been used. All images were then arranged respectively in their class folders for manual annotation and tagging. This collection of instruments will be referred to as *PPMI* data set or *People Playing Music Instruments* and constitutes our larger two dimensional data set for object detection. Sample images can be seen in Figure 4.2a.

Since our objective is to detect instruments in 360 videos, we would like to fine tune our model to learn instruments on a 360 image frame, which is quite different from a 2 dimensional video frame. A 360 image frame is more spherical or polar in nature, given it captures more of the environment than its traditional two dimensional counterpart. In order to suffice this need, we have collected several 360 videos containing people playing musical instruments. The videos have been collected from publicly available sources on YouTube. A total of **276** videos were collected. 10 image frames were extracted from each of the videos at a separation of five seconds, to create a 360 image data set comprising of **2760** images. This forms our smaller 360 image data set that would be used for fine tuning our object detection model. Sample 360 image frames can be seen in Figure 4.2b

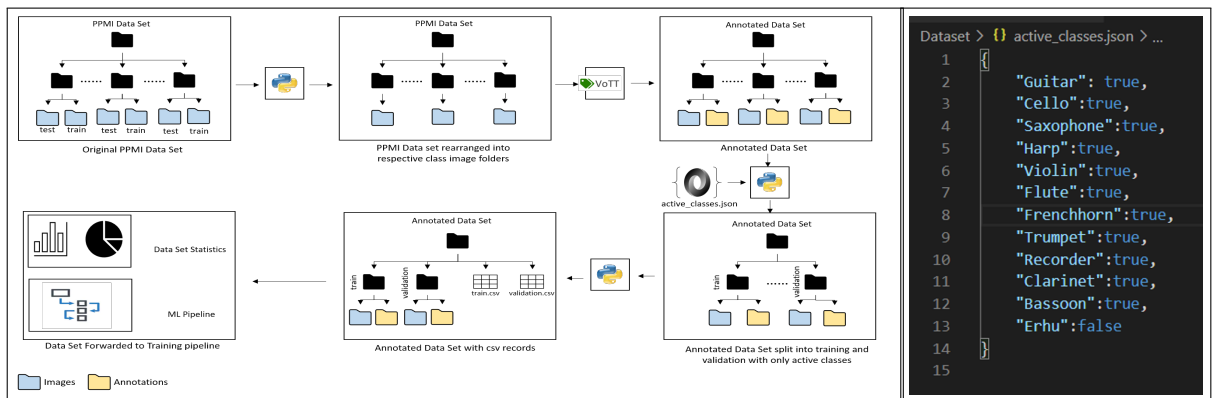
### 4.2.2 Audio Data Set

The audio data set is collected from publicly available pre-compiled data sets. Musical instrument audio excerpts from the *Instrument Recognition in Musical Audio Signals* or *IRMAS* [78] data set, and the *Philharmonia* [77] data set have been used to compile the audio data set of musical instruments. To maintain consistency only those instruments are selected from the audio data sets that are available in the image data sets. From the IRMAS data set *Cello, Clarinet, Flute, Guitar, Saxophone, Trumpet* and *Violin* were selected, while from the *Philharmonia* data set sounds of *Basoon* and the *French horn* were selected. A total **8224** instruments distributed among these **9** classes were used to create our audio data set. The IRMAS data set comprised of acoustic guitar and electric guitar sounds as separate classes. In our research both these sounds are considered to be generated from a single class 'guitar', because of which the wav files from these classes have been merged into a single class. The distribution of audio excerpts can be seen in Figure 4.4

### 4.2.3 Ground Truth Set Up

Both the images and the audio data set were pre-processed in order to fit as inputs to the object detection and classification models respectively. The PPMI images as downloaded from its source is distributed into two separate folders, that is, '*Playing Instruments*' and '*With Instruments*'. However, this is not how we want our





(a) Pre-processing of PPMI images

(b) Active classes file

1	filename	width	height	class	xmin	ymin	xmax	ymax
2065	vjm4_QuByQY.mp4_frame2.jpg	640	360	Guitar	141.783822	197.179292	248.05593	255.074013
2066	vjm4_QuByQY.mp4_frame2.jpg	640	360	Saxophon	321.183724	203.495082	348.014797	246.126661
2067	vjm4_QuByQY.mp4_frame2.jpg	640	360	Cello	483.748459	141.389803	541.35635	272.442434
2068	vjm4_QuByQY.mp4_frame3.jpg	640	360	Guitar	141.783822	197.179292	248.05593	255.074013
2069	vjm4_QuByQY.mp4_frame3.jpg	640	360	Saxophon	321.183724	203.495082	348.014797	246.126661
2070	vjm4_QuByQY.mp4_frame4.jpg	640	360	Cello	483.748459	141.389803	541.35635	272.442434
2071	vjm4_QuByQY.mp4_frame4.jpg	640	360	Guitar	141.783822	197.179292	248.05593	255.074013
2072	vjm4_QuByQY.mp4_frame4.jpg	640	360	Saxophon	321.183724	203.495082	348.014797	246.126661
2073	vjm4_QuByQY.mp4_frame4.jpg	640	360	Cello	483.748459	141.389803	541.35635	272.442434
2074	vjm4_QuByQY.mp4_frame5.jpg	640	360	Guitar	141.783822	197.179292	248.05593	255.074013
2075	vjm4_QuByQY.mp4_frame5.jpg	640	360	Saxophon	321.183724	203.495082	348.014797	246.126661
2076	vjm4_QuByQY.mp4_frame5.jpg	640	360	Cello	483.748459	141.389803	541.35635	272.442434
2077	vjm4_QuByQY.mp4_frame6.jpg	640	360	Guitar	141.783822	197.179292	248.05593	255.074013
2078	vjm4_QuByQY.mp4_frame6.jpg	640	360	Saxophon	321.183724	203.495082	348.014797	246.126661
2079	vjm4_QuByQY.mp4_frame6.jpg	640	360	Cello	483.748459	141.389803	541.35635	272.442434
2080	vjm4_QuByQY.mp4_frame7.jpg	640	360	Guitar	141.783822	197.179292	248.05593	255.074013
2081	vjm4_QuByQY.mp4_frame7.jpg	640	360	Saxophon	321.183724	203.495082	348.014797	246.126661
2082	vjm4_QuByQY.mp4_frame7.jpg	640	360	Cello	483.748459	141.389803	541.35635	272.442434
2083	vjm4_QuByQY.mp4_frame8.jpg	640	360	Guitar	141.783822	197.179292	238.323058	240.789474
2084	vjm4_QuByQY.mp4_frame8.jpg	640	360	Saxophon	321.183724	203.495082	348.014797	246.126661
2085	vjm4_QuByQY.mp4_frame8.jpg	640	360	Cello	483.748459	141.389803	541.35635	272.442434
2086	vjm4_QuByQY.mp4_frame9.jpg	640	360	Guitar	141.783822	197.179292	243.847102	246.315789
2087	vjm4_QuByQY.mp4_frame9.jpg	640	360	Saxophon	321.183724	203.495082	348.014797	246.126661
2088	vjm4_QuByQY.mp4_frame9.jpg	640	360	Cello	483.748459	141.389803	541.35635	272.442434

(c) Pascal VOC xml

(d) Comma separated values for localization information

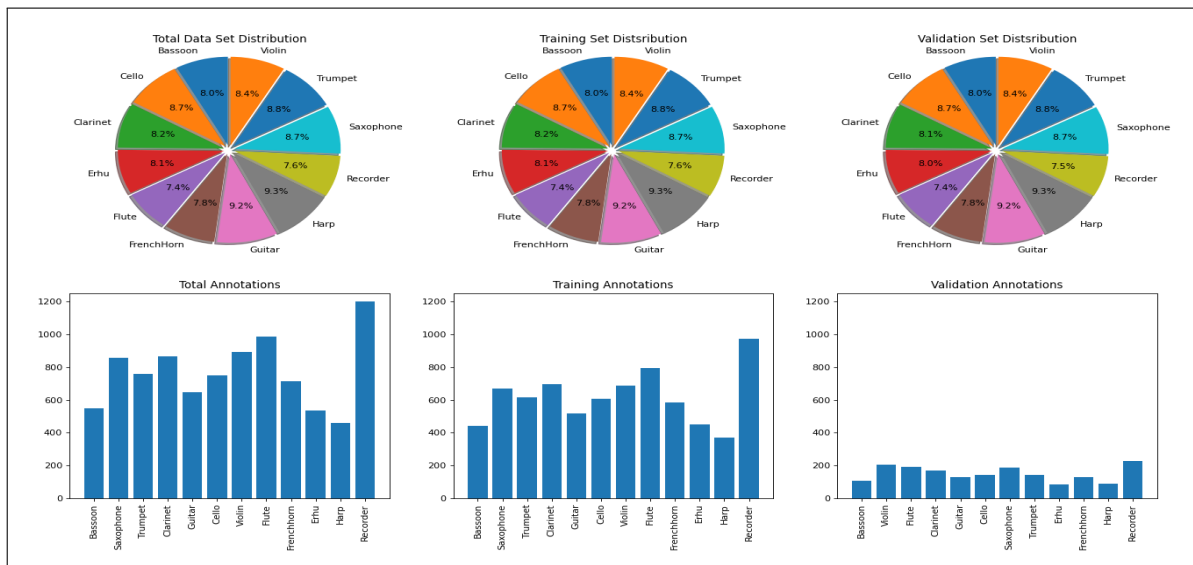
Figure 4.3: Images pre-processing

data set to be structured for the object detection task. The instrument images should be segregated based on instrument labels. To accomplish this task we have created several scripts available in our project repository (link available in Appendix C). An overview of this pre-processing stage is shown in Figure 4.3a.

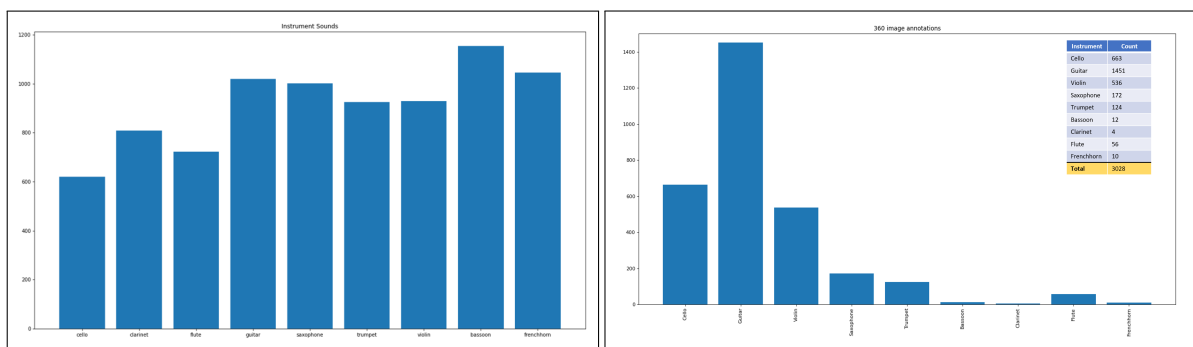
Firstly, the original folder structure is formatted to separate instrument folders. Thus, now we have separate instrument folders that contain image folders with all images of that particular instrument. The next step is to annotate these images together and save the annotation information in a separate *'annotations'* folder under each instrument folder. The raw images were required to be labelled and annotated to show instrument locations. Both the images data set were manually annotated using *Visual Object Tagging Tool (VoTT)* [97]. The labelling and annotation software

allows users to load all images for a project at once and manually annotate regions of interest and label them with corresponding class labels (Screen : Appendix A). Once the export was complete, the saved project is exported as ***Pascal Visual Object Classes*** files which are saved in xml format. Each image has its corresponding annotation information stored in a corresponding Pascal VOC file. The annotation information is stored as bounding box dimensions and location. This format is chosen because if there is a correction to be made on any bounding box on a particular image, the image’s corresponding xml file can be easily located and the data can be updated as required. Additionally PASCAL VOC is the standard format for object detection data representation, that can be used by Python modules for training. An example exported xml file can be seen in Figure 4.3c.

Once all images have been annotated, they are split into training and validation data sets. We have chosen an ***80-20*** split. Separate folders are created to contain the images as well as their annotation xmls. Using the ***generate\_dataset.py***. The script takes in an active classes configuration file which allows the user to list a set of instruments to be considered for the combined data set. This option is kept so that we can include or exclude certain instruments in our combined data set for the purposes of training. A sample configuration file is shown in Figure 4.3b. *True* means the instrument should be considered while *False* means otherwise. The script also takes in a ***'split'*** parameter which defines the training and validation splits of the combined data set. Running this script gives us our final combined data set split into training and validation data sets along with the raw images and localization information. A sample command is shown in Appendix B. After the split, the training and validation data are converted to single comma separated value files using the same script. These files hold the image file name, the classes annotated in that file and their corresponding bounding box locations as seen in Figure 4.3d. Using the raw image files along with the bounding box information in the csv files, ***Tensorflow records*** are created with the help of ***Tensorflow*** python package [98]. Using the ***generateTFRecords.py*** the csv files along with the raw image information can be converted to Tensorflow Records Appendix B. The script is available within the repository created for this research Appendix B. In this way we obtain our training and validation records for our large two dimensional image object detection data set. Furthermore, the ***generate\_data\_stats.py*** script generates visualizations of the data set generated as seen in Figure 4.4. A sample command is



(a) 2d images annotation stats



(b) Audio data set statistics

(c) 360 images stats

Figure 4.4: Ground truth statistics

shown in Appendix B.

On comparing the number of images and annotations it was found that although the number of images under each class was almost same as can be seen from the pie charts in Figure 4.4, the distribution of instrument annotations were not as equitable. We can see from the bar graphs that depict instrument annotations, that the **Recorder** has a high number of annotations. This can be attributed to the fact that the recorder appeared on multiple images, especially in orchestral setups. These distributions can be seen in Figure 4.4a. A total of **9180** annotations were obtained from the PPMI image data set. Also from Figure 4.4c, it can be seen that the annotations in 360

images are highly skewed to guitar. This is because most available 360 videos largely contained the guitar than any other instrument. A total of **1451** annotations were made among the 2760 images in the 360 images data set. A total of **3028** annotations were made in the 360 image data set. Also, it is important to note that all **11** eleven instruments as previously considered in the PPMI image data set are not present in the 360 image data set, rather only **9** instruments could be found. This reduces the scope of the 360 image object detector model, which is one of the reasons for its improved accuracy as is discussed in Chapter 5.

The audio wav files in the audio data set are pre-processed as well. The **16 bit** audio samples from the IRMAS data set are all **2 to 3 seconds** in length with **stereo channel** and sampled at **44.1 kHz**. The audio samples extracted from the Philharmonia data set are also sampled with a similar configuration to maintain consistency. This sampling and trimming of audio files is done using *ffmpeg* [99] library installed on a linux powered machined. The command for this is mentioned in Appendix B. All wav files were kept in their separate class folders and individual converted to **128 x 64** sized spectrogram images using the *libROSA* python library [100]. Spectrogram images are condensed cepstral representations of audio signals also known as **Mel-Frequency Cepstrum**. The MFC of an audio signal is retrieved by taking the Fourier Transform followed by a collection of mathematical functions to get a condensed excerpt of the audio signal. MFC of an audio signal comprises of the **mel-frequency cepstrum coefficients** which are the essential features of the audio signal. These are the basic features extracted from audio signals and have been used to benchmark the environment sound classification data sets [79] and also automatic speech recognition tasks [82]. These spectrogram images are used for classification training going forward.

## 4.3 Experiments

The experimentation is divided into two separate legs. The first being object detection training on the annotated two dimensional images data set followed by fine-tuning on the annotated 360 images. The second being classification training on the audio data set. An overall low-level design diagram is shown in Figure 4.5. The object detection training on two dimensional PPMI images gives us our base object detection model which is further fine tuned on the smaller 360 image set, while the audio classifier

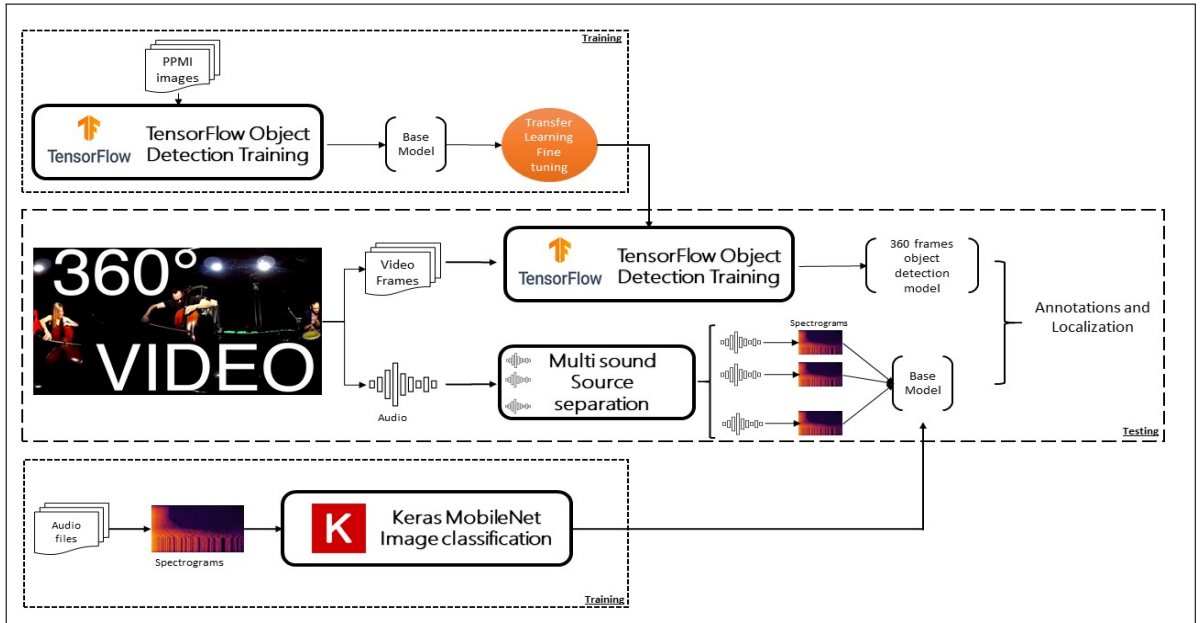


Figure 4.5: High-level proposed method architecture.

model is trained on the spectrogram images compiled during data pre-processing of the audio data set. In this way we obtain both our audio and visual modal models. The next step is to test the models. For testing we use 360 image frames from input 360 videos, run our object detection model on the frames and stitch them together to obtain our annotated and localized video. The audio from 360 video is extracted and passed onto the source separation which segregates the separate instrument sound sources. Each of these separated audio wav files are converted to spectrogram having the same dimensions as the spectrogram images during training and passed onto the classification model to obtain class scores. Thus, the experimentation covers both the audio and visual cues from any 360 video.

### 4.3.1 Object Detection Model

An effective way to speed up convergence of object detection model is to pre-train on a large data set and then fine tune that model on a smaller target-oriented data set [21]. The same concept is applied to our object detection model. The PPMI image data set is a larger data set comprising of 4209 annotated images which is used to train our base object detection model using the Tensorflow Object Detection API

[51]. As discussed earlier, the Tensorflow Object Detection API supports a collection of backbone networks such as MobileNet, InceptionNet and ResNet, in addition to the final object detection stage which includes two stage and single stage detectors. For our experiments the single shot detector SSD [40] was used along with pre-trained feature extraction models such as MobileNet, ResNet and InceptionNet. The Tensorflow training and validation records and all other relevant hyperparameters are set in the model configuration file used for training the object detection model. The experiments were conducted on varying batch sizes including **4,8,16** and **24**, keeping in mind GPU memory restrictions. All object detection networks were trained for two hundred thousand iterations on an **8GB, CUDA enabled NVIDIA GeForce RTX 208 Ti GPU**. The backbone feature extractor models and their configuration files used for training the music instrument object detector were pre-trained on the COCO data set [28]. The hyperparameters of the configuration were kept unchanged except for the number of classes as we have 11 classes for our object detection, learning-rate decay parameters, epochs, batch sizes, training and validation record paths. The batch size defines the number of images taken together by the model for a complete forward and backward pass through the network. Increasing the batch size means more images will be processed in a single run, which also means the training time would be reduced, however, more memory would be used. Therefore, there is a trade-off between training time, accuracy and memory management. Experiments have shown that higher batch sizes provide better results [101]. Therefore, keeping in mind our GPU restriction the batch sizes were increased till 24. Learning rate decay parameters were also used in certain model configurations. This is because experiments have shown that starting with a relatively large learning rate helps increase the training speed and helps avoid unnecessary features, and then decaying the learning rate at certain intervals or epochs helps stabilize the model, improves convergence speed and also lets the network learn intricate features from the data set [102]. Different model configurations have different learning rate decay parameters. The input images in the Tensorflow records were also resized to the classic **300 x 300** dimensions before training. The *model\_main.py* script available under the *object\_detection* package is used to launch the training. The command is available in Appendix B.

The object detection model with the best mAP was used to fine tune on the 360 image data set with the same configurations to retrieve our final visual object

detection model. The results are discussed in Chapter 5. To monitor the progress of the training *Tensorboard* was used. Object detection training stores the checkpoint weights after regular epoch intervals in the log directory mentioned during training. The *Tensorboard* application can be pointed to this directory for regular monitoring of accuracy and loss. The command for launching *Tensorboard* is shown in Appendix B. The *Tensorboard* application launches the monitoring UI on the localhost server on an available port which can be viewed from any web browser. The *logdir* parameter stores the path to the checkpoint directory. The evaluation of our model has been discussed in Chapter 6.

### 4.3.2 Audio Source Separation

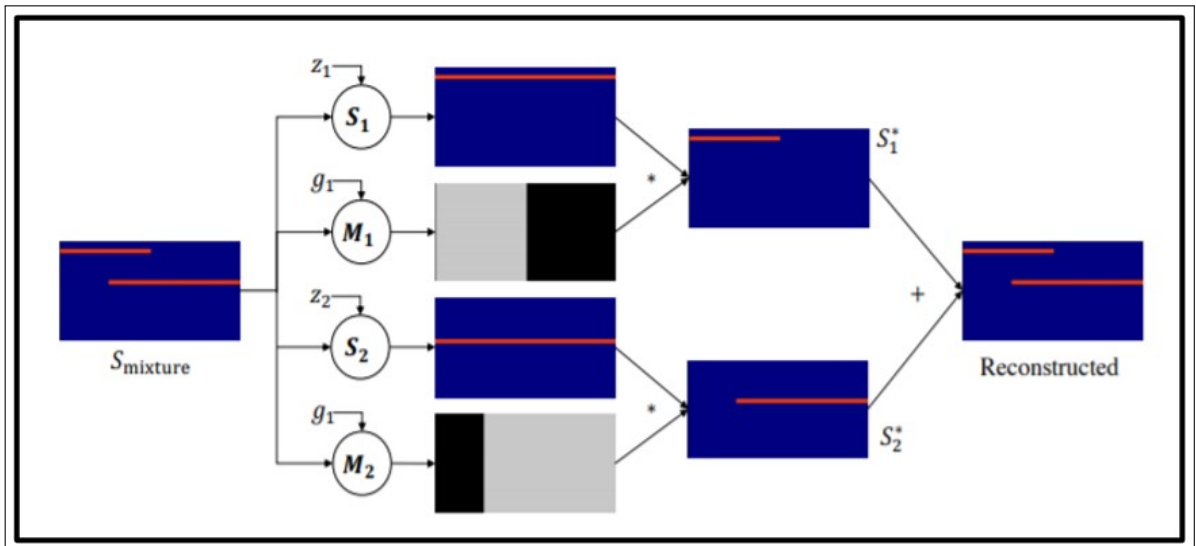


Figure 4.6: Deep Audio Prior Network

The audio extracted from the 360 video frame may contain multiple instruments. Therefore in order to categorize them individually it is necessary to separate the sources and create individual audio files for each instrument. For audio source separation we have used the unsupervised source separation framework named *Deep Audio Prior* abbreviated as *DAP* developed by Tian, Xu et al. [71]. This network is a blind source separator that is independent of any external training data and uses the current audio mixture to train itself into segregating the sound sources. The DAP architecture is

shown in Figure 4.6. It uses two sound generator networks  $S_1$  and  $S_2$  and  $M_1$  and  $M_2$  are the masking networks which filter out the sound sources by suppressing one source and maximizing the other. The DAP network assumes that the sound mixture contains only 2 distinguishable sound sources and not more. This is a critical bottleneck of the network which does not allow us to test it on sound mixture having more than 2 instruments. All networks follows the *Wave-U-Net* architecture for end-to-end source separation [103]. For source separation from the audio extracted from our test 360 videos. The unformatted wav file is fed to the DAP network and DAP blind source separation script is used to train the network for **5000** iterations. The repository for Deep Audio Prior contains the scripts required for unsupervised training. We have used the *dap\_sep.py* script to segregate sound sources, which is the blind source separator script. The command is available in Appendix B. The *input\_mix* parameter stores the input wav mixture signal that is to be separated, while the *output* parameter is the directory that stores the separated wav files at different checkpoints. The 2 separated tracks are saved as wav files. The separate wav files are then sent to the audio classification network for classification.

### 4.3.3 Audio Classification Model

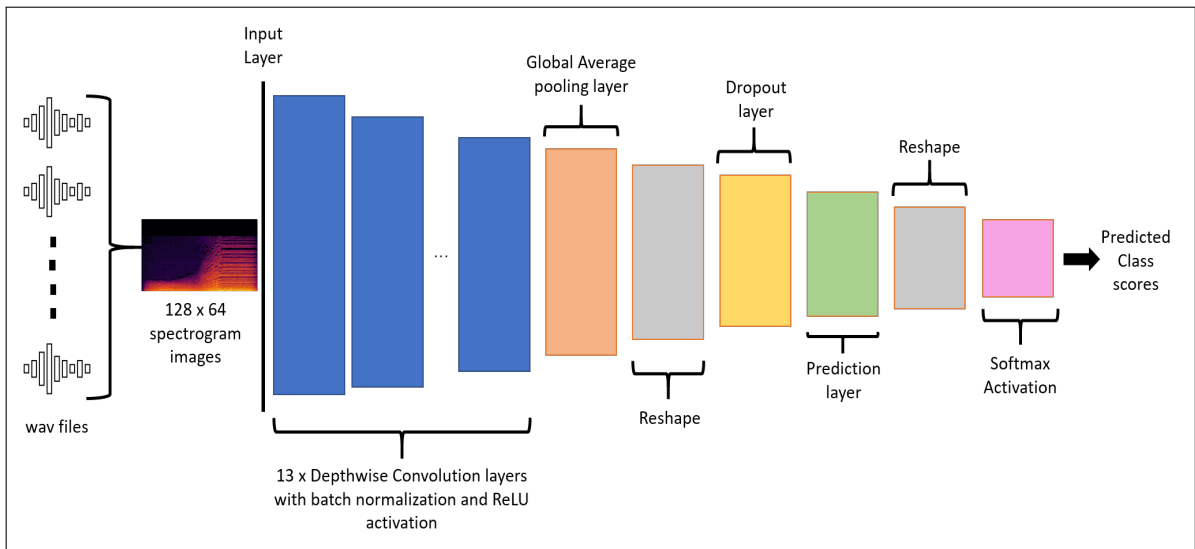


Figure 4.7: Keras MobileNet architecture.



The audio classification model used was the *MobileNet* image classifier from the *Tensorflow Keras* applications package [81]. This model was fine tuned on the spectrogram images exported from the audio data set . The Keras MobileNet application comprises of **13** depthwise convolution layers with *batch normalization* and *ReLU activation* functions. It uses *global average pooling* and *softmax activation* as the final layer to predict the class scores. The model image can be seen in Figure 4.7. The **9** class **128 x 64** spectrogram image data sets are used to fine tune this pre-trained model with a batch size of **4,8** and **16** keeping in mind GPU restrictions. The model was trained for a total of **1000** iterations. 2 experiments were conducted with early stop monitoring on the training loss with a patience level of **300** epochs. This means that if the training loss did not improve even after 300 iterations the training would be stopped early. *Adam optimizer* was used to minimize the loss functions and the images were re-scaled to keep their RGB coefficients between 0 and 1 in order to maintain scalability. The model checkpoints were saved for testing on unlabeled data. The loss functions used are sparse categorical accuracy and top 3 categorical accuracy discussed in Section 2.6.1. The training is launched using the *train\_sc.py* script within the *SoundClassifier* directory of our repository. This script contains functions to building the MobileNet model, along with the accuracy metric functions. Both of which are available under the *applications* and *metrics* package of the Keras python module. The command for launching the training is shown in Appendix B. The *width* and *height* parameters store the spectrogram dimensions which correspondingly create the model dimensions. The *batch-size* parameter stores the batch size for training. The *epochs* parameter stores the number of training iterations. The *dataset* parameter stores the path to the spectrogram images data. With the *output-model-param* parameter we can save out trained weights to a desired directory, and finally the *plot-save-path* stores the training and validation loss and accuracy plots for the particular training session. The results and evaluation are discussed in Chapter 5.

# Chapter 5

## Experimental Results

The results of the experiments conducted are divided into three broad categories. The first being the results of object detection on the two dimensional PPMI images data set in Table 5.1, the second being the results of fine tuning the base model on the 360 image data set in Table 5.2 and finally the audio classification results in Table 5.3. Additionally the accuracy plots, classification loss plot and localization loss plots against each epoch for object detection training on 360 images data set can also be seen in Figure 5.1. The audio classification loss and accuracy plots can be seen in Figure 5.2

Num of classes	Configuration	Parameters	Batch Size	Iterations	Learning Rate Decay	map @0.5IoU	map @0.75IoU
11	MobileNetv1 with SSD	4.2M	4	200K	False	54%	22.8%
11	MobileNetv1 with SSD	4.2M	8	200K	False	55%	30%
11	MobileNetv1 with SSD	4.2M	16	200K	False	55%	28.7%
11	MobileNetv1 with SSD	4.2M	24	200K	False	64.5%	37.3%
11	Inceptionv2 with SSD	24M	4	200K	False	64.5%	35.3%
11	MobileNetv1 with SSD	4.2M	4	200K	True	58%	29.15%
11	Resnet50v1 with SSD	26M	4	92K	False	32%	14.16%
11	MobileNetv1 with SSD	4.2M	16	86K	True	55%	30.3%
11	Inceptionv2 with SSD	24M	8	200K	False	68%	37%
11	MobileNetv3 large with SSD	5.4M	16	200K	False	34%	11.3%
11	Inceptionv2 with SSD	24M	16	200K	False	69%	40%
11	MobileNetv3 large with SSD	5.4M	16	200K	False	49%	17.7%
11	Inceptionv2 with SSD	24M	24	200K	False	66%	38.21%
11	MobileNetv3 large with SSD	5.4M	24	200K	False	53.16%	22.6%

Table 5.1: Results on base object detection model

The accuracy and loss results are obtained from monitoring the results of the final iteration from the *Tensorboard* application for each model. From Table 5.1 it can be seen that the most promising results were received from *InceptionNet version 2* feature extractor with an *mAP@0.5IoU* of **69%** and *mAP@0.75IoU* of **37%** with an experimental batch size of **16**. All *InceptionNet* feature extractors with *Single Shot Object Detector* outperformed the other feature extractors such as *MobileNet* and *Resnet*. With the *InceptionNet version 2* feature extractor the mean average precision increases on increasing the batch size till a batch size of **16** to **69%** and drops down to **66%** with a batch size of **24**. At batch size **16** the object detector reached its optimum generalized minima. Further increasing the batch size would only result in reduction of the accuracy as the CNN would not be able to generalize well. Additionally due to GPU memory restrictions, the batch size was restricted to **24**. Furthermore, the success of *InceptionNet* when compared to the other feature extractors can be attributed to the fact that it includes batch normalization which reduces the input dimensions for subsequent layers by a large factor. It is also important to point out that some of the images in the PPMI data set have lower resolution or a lower receptive field. *InceptionNet version 2* adopts a unique procedure to handle such low resolution images. By reducing the stride factor of first two layers in the network in case of lower resolution images, it ensures that each image is examined with more care. This reduced stride feature is not implemented for high resolution images. Furthermore, it factorizes large convolution block into multiple smaller convolution blocks. Because of this, the entire training of two hundred thousand epochs was completed for all batch sizes, which was not the case with the *ResNet50* version feature extractor. Because of *ResNet50's* large size (parameter count of 26 million) only 92000 iterations could be completed at a batch size of **4** on the GPU available.

As for *MobileNet version 1* feature extractor, the accuracy increased on increasing the batch size. We would have gotten better results by increasing the batch size beyond 24 because of its smaller parameter count and size, however, due to GPU restrictions it could not be done. With *MobileNet version 1* an accuracy of **64.5%** is attained on a batch size of **24**. Introducing learning rate decay did not help improve the accuracy. The *MobileNet* architecture was constructed for mobile and embedded applications where the image sizes are considerably smaller. Also the model size is

reduced and has fewer parameters. Although the training time was less, it did not achieve the desired results because the PPMI data set contained images of different sizes and resolutions. A similar case was observed with *MobileNet version 3*. The reduced number of parameters helped the model to train faster on larger batch sizes, however, it compromised on accuracy. Perhaps, training the *MobileNet* model further with larger batch sizes and changing the learning rates would give better results. This remains an area for future work.

The fine tuned object detector model achieved an *mAP@0.5IoU* of **94%** as can be seen from Table 5.2.

Num of classes	Configuration	Parameters	Batch Size	Iterations	Learning Rate Decay	map @0.5IoU	map @0.75IoU
11	Inceptionv2 with SSD	24M	16	200K	False	94%	67%

Table 5.2: Results on fine tuned object detection model

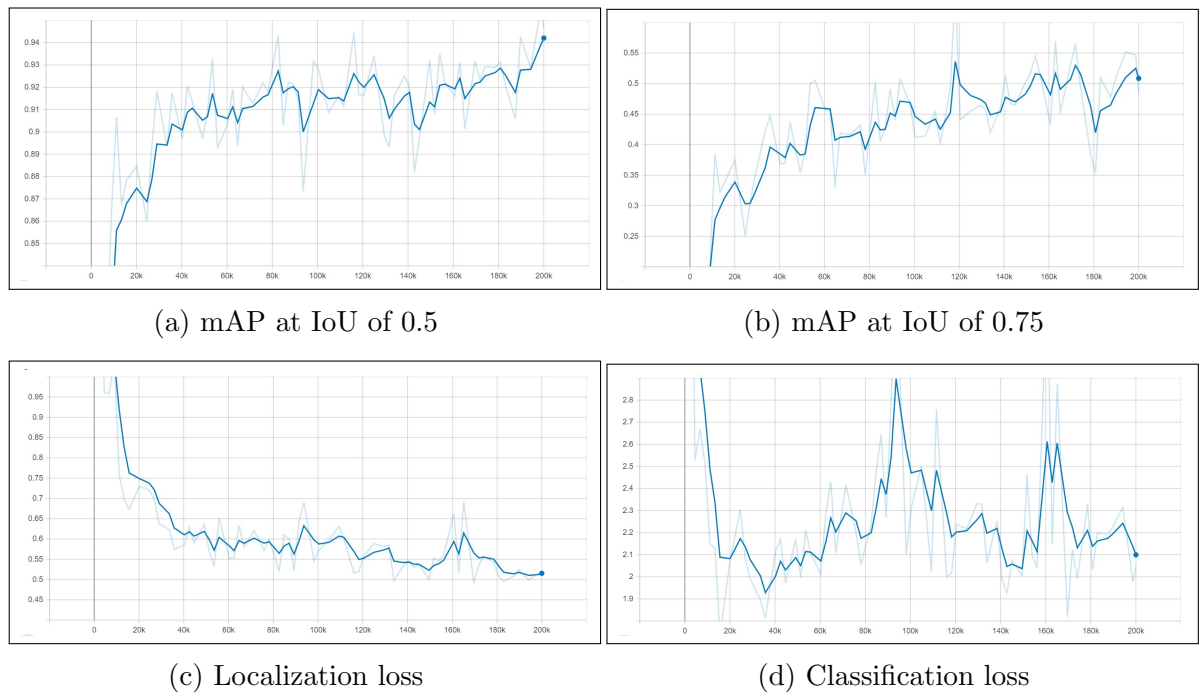


Figure 5.1: Object Detection accuracy and loss plots against iterations

The same training configurations were kept as the highest accuracy model from our base object detector, that is, the *InceptionNet version 2* with *SSD* backed

with a batch size of **16** and training iterations of **200 thousand epochs**. The fine-tuned object detection model is essentially manipulated to make predictions on 360 image frames, having learnt instrument patterns from normal two dimensional frames. Therefore, the model can be said to have begun its learning with a pre-conceived notion of the characteristic features of each individual instrument, which gave it an edge, in identifying instruments on 360 image frames. However, the reduced size of the 360 images data set also helped it gain this accuracy. Perhaps increasing the size of the 360 image data set and training it again would be the actual test of accuracy measure of this model. Additionally the sharp increase in accuracy from the base object detection model (**69%**) to the 360 image object detection model (**94%**) can be attributed to the fact that the 360 object detector only worked on 9 different instrument labels as the 360 image data set contained only the most common instruments as can be seen from Figure 4.4c. Therefore, the 360 model had less class labels to differentiate from. Moreover it can be seen from Figure 4.4c that the 360 image data set had a larger number of Guitar annotations as it is the most commonly used instrument. This also helped the 360 model to accurately detect instruments as most of them were Guitars. Unavailability of 360 videos with a wide variety of instruments is the main reason for such a skewed data set. Perhaps, adding some more instruments in 360 format would help in getting a more generalized 360 object detector model. From Figure 5.1a and Figure 5.1b it can be seen that there is an overall steady increase in the mAP score for both IoU levels. From Figure 5.1c it can be seen that the models localization loss has gradually decreased with each iteration, no spikes are observed. This indicates that our model is able to localize instruments with more confidence. Finally from Figure 5.1d it can be seen the classification loss decreases gradually with the each epoch, however there are certain spikes specifically around the hundred thousandth iteration and then again towards the end. This is attributed to the fact that the inception net model uses *mini-batch gradient descent* to minimize its loss function. Subsequently using small batches on the overall batch size of **16**, there can be some images which get classified wrongly which causes these spikes.

The audio classification results can be seen in Table 5.3. The evaluation was done using the *evaluate.py* script from the *SoundClassifier* directory of our repository. A sample command is shown in Appendix B. The *evaluate.py* script returns the *sparse\_categorical\_accuracy* score and *top\_k\_categorical\_accuracy* with  $k = 3$

Num of classes	Spectrogram dimensions	Parameters	Batch Size	Iterations	Early Stopping	Sparse categorical accuracy	Top 3 accuracy
9	128 x 64	3.4M	4	1000	Train Loss,patience=300	65.3%	85.6%
9	128 x 64	3.4M	8	1000	Train Loss,patience=300	62.9%	83.7%
9	128 x 64	3.4M	4	1000	No	63.7%	84.8%
9	128 x 64	3.4M	8	1000	No	63.7%	84.5%
9	128 x 64	3.4M	16	1000	No	59.1%	80.8%

Table 5.3: Results on audio classification

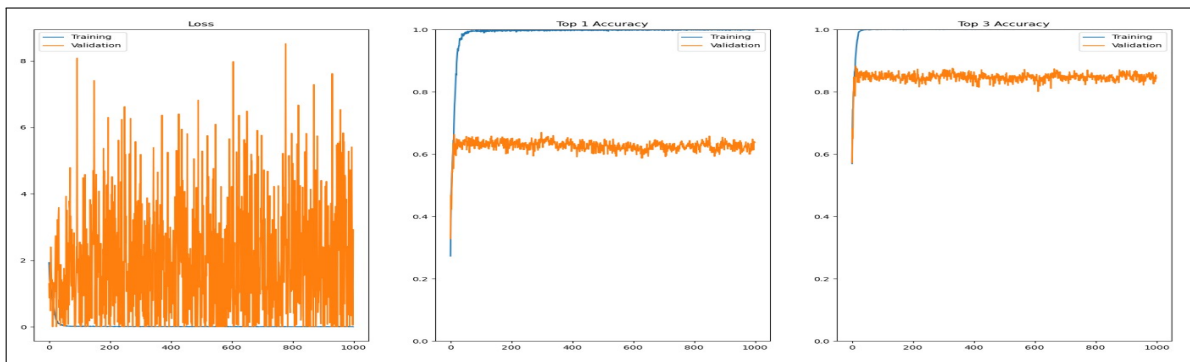


Figure 5.2: Audio classification loss and accuracy plots

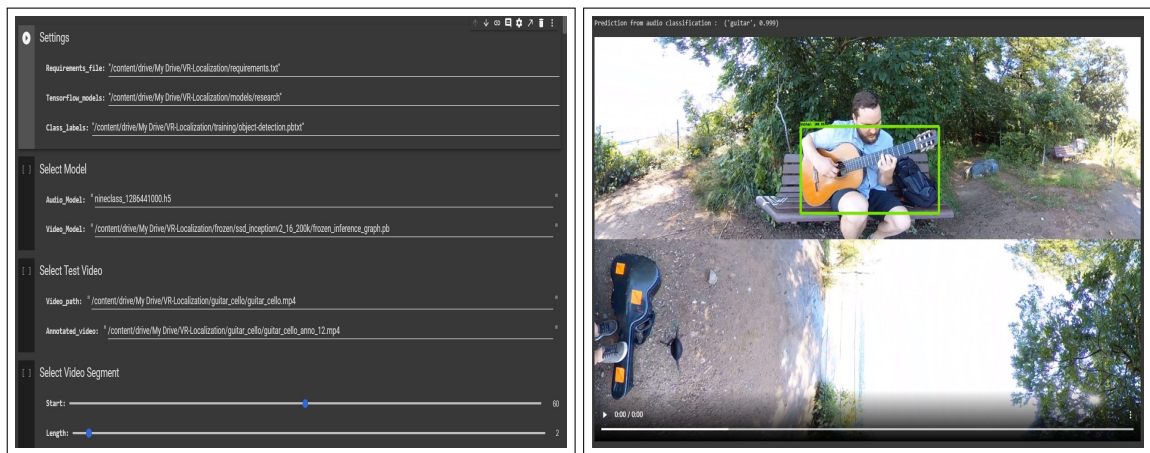
after evaluating the model mentioned in the *model* parameter on the validation data set stored in the *validation\_dir*. The best results were retrieved with a batch size of **4** with an early stopping monitor on training loss having patience value of 300 epochs, although, the training did not stop early and continued to the 1000<sup>th</sup> epoch. An overall categorical accuracy of **65.4%** and top-3 accuracy of **85.6%** was achieved. The **9** class classifier model proved effective on small batch sizes. With a batch size of **16** the accuracy dropped. The learning rate was kept at **0.0004**. As significantly good results were achieved with these hyper-parameter settings, further modification was not required. The model tested well on unlabeled instrument sounds. From Figure 5.2 it can be seen that the training loss flattens out around 0, while the validation loss oscillates greatly. The validation loss oscillation can be attributed to the fact that the Adam optimizer uses *mini-batch gradient descent* on the 20% validation data set which is already pretty small in size. Further experimentation on the learning rate of the optimizer may lead to improved results. The top-1 validation accuracy swiftly increases before flattening out at **62%** while the top-3 validation accuracy swiftly increases before flattening out at **85%**.

## Chapter 6

# Applications and Demonstrable Units

The object detection and sound classification models can be useful in a variety of applications that extend to real-world scenarios. The motivation to create these demonstrable units is to put our models to test and verify their effectiveness on unlabeled 360 videos. Firstly, the checkpoint files saved during the object detection training need to be exported to testable formats. The weights from the checkpoint files are saved as *frozen inference graphs*. These graphs can be used in mobile applications for live object detection. To convert the checkpoint weights to a frozen inference graph the *export\_inference\_graph.py* script from the object detection model can be used. A sample command is given in Appendix B. The *pipeline\_config\_path* parameter stores the model configuration file used during training, the *trained\_checkpoint\_prefix* contains the model checkpoint file to be converted and the *output\_directory* contains the path where the inference graph would be saved. For mobile applications, a *Tensorflow Lite* version of the inference graph is required. This can be retrieved using the *export\_tflite\_ssd\_graph.py* script in the *object\_detection module*. A sample command is shown in Appendix B.

As a part of this dissertation we have curated 2 separate applications that can be used for instrument recognition and localization. The first application is a *Google Colaboratory* notebook UI/UX with the support of Google Colaboratory forms. Since Google Colab offers free GPU support, it is an effective platform to test our models



(a) Forms for settings and inputs

(b) Modelled results

Figure 6.1: Google Colab Notebook UI/UX application

effectiveness. A snapshot of the form and its working can be seen in Figure 6.1. Figure 6.1a shows the form and its inputs. It contains input fields for the python packages requirements files, trained model inputs that is the inference graph exported as discussed above, label file, test video file path and output file path. Additionally the test can also be performed on a particular segment of the video file rather than the entire file. Running the consecutive cells of the notebook will result in the annotated video and result from the sound classification model as shown in Figure 6.1b

Another application curated is the lite version of our object detection model implemented using *Tensorflow Lite*. The base object detection model's Tensorflow Lite inference graph is converted to a Tensorflow lite mobile application model using the Tensorflow lite package, *tf lite\_convert*, in python which makes it compatible with android devices. A sample command is given in Appendix B. The model is loaded onto the Tensorflow lite app via Android Studio. A fully functioning android app which uses the device's camera feed to detect objects based on the model is developed. This application makes it suitable for live object tracking and can be extended to real-world environments. A demo video of the application in action is provided as a link in Appendix C.



# Chapter 7

## Conclusion

As a part of this research we have developed a dual model architecture that incorporates object detection and audio classification methodologies based on deep learning to identify and localize musical instruments in virtual reality environments, specifically in 360 videos. In doing so, a large annotated data set of people interacting with musical instruments was created. The data set images were collected in a separate research. However, this image collection was leveraged to create an annotated image data set of people playing musical instruments in a variety of background environments covering 11 different musical instruments. The music instruments used in our research are *Bassoon*, *Cello*, *Clarinet*, *Erhu*, *Flute*, *French Horn*, *Guitar*, *Harp*, *Recorder*, *Saxophone*, *Trumpet* and *Violin*. Although, the *Erhu* was excluded given that no 360 videos contain this instrument. A total of **4209** images were annotated. Additionally, a collection of 360 videos of people playing musical instruments was also collected. These videos are publicly available on YouTube. From these videos a 360 degree image data set of people playing musical instrument was created by extracting 10 frames from each of these 360 videos. A total of **2760** 360 images are annotated. For the audio classification training, the Philharmonia data set and the Instrument Recognition in Musical Audio Signals data set covering 9 different musical instruments. The audio files in these data sets were normalized to similar wavelengths, channels and lengths and converted to spectrogram images for audio classification training. A total of **8224** audio files and subsequently spectrogram images are generated. Several backbone networks were used along with single shot object detector to train our base

object detection model. The best performing model from these was fine-tuned on the smaller 360 image frame data set to get our final 360 video music instrument object detector. A mean average precision of **94%** was achieved from this model. The audio classification model was trained on the spectrogram images and a **top-3 accuracy** of **85%** was achieved. Furthermore, an audio source separation architecture named **Deep Audio Prior** [71] was used that separates a sound source mixture into two distinct sound wav files using self-supervision.

For testing our proposed methods, visual cues from 360 image frames were fed to our fine-tuned model and the resultant annotated image frames were stitched together to construct an annotated video in Google Colaboratory. Furthermore, audio was extracted from the 360 video file and fed to the source separation network for sound source separation. The separated wav files from this network were converted to spectrograms and tested on our audio classification network.

We can conclude by stating that the research objective was achieved, however, there remains scope for improvement. The main aim to detect musical instruments in 360 videos was achieved by using visual cues and object detection methodologies. This research took a step further to use audio cues as well to detect and classify musical instruments using audio features. Therefore, a dual-modal architecture was curated. In conclusion, with the help of object detection, sound classification and source separation frameworks it is possible to detect and annotate musical instruments in a virtual reality environment. Some limitations and future work for our proposed method are listed as follows.

## 7.1 Limitations

Although the research objective was achieved, there are several limitations of our proposed methods. The discrepancy between the number of classes in the visual and audio framework is one such drawback. While the visual framework works on **11** classes, the audio framework works on **9** only. Although the classes in the audio framework are contained in the visual framework, 2 instruments that is the **harp** and the **recorder** are absent in the audio framework model. This is because the IRMAS and philharmonia data set did not contain audio samples from these instruments. Therefore, although the object detector would be able to locate these 2 instruments, the audio model would

not be able to classify them.

The audio source separation architecture suffers the limitation of considering all sound signal mixtures of having only 2 separate sources. Therefore, if a sound mixture containing more than two instruments are fed to this network, it would still produce 2 separate *wav* files, which is not accurate. Feeding these separate *wav* files to the audio classifier might increase chances of misclassification. Furthermore, each time the audio classifier needs to be run, the source separation needs to be trained on the input signal, which is a time consuming process. This in turn subsequently increases the detection time of the overall model.

The audio model developed only serves the purpose of classification. Localization information cannot be retrieved from the audio model. This is because the audio is extracted out from the input video and considered as a separate entity independent of the video file. This entity is used completely on a different leg of the proposed method that does not depend on the visual object detector model.

## 7.2 Future Work

There is a lot of room for improvement in the research conducted as is the case with any research. Further research can be done in expanding the batch size of the training network and the various hyper-parameters can be fine tuned and experimented with. Also, 2-stage object detectors such as ***RCNN***, ***Fast-RCNN*** and ***Faster-RCNN*** can also be implemented to test their effectiveness on 360 image frames, as they are considered to give better accuracy on benchmark data sets. Using a larger 360 image data set would also help in generalizing the model further. The smaller size of the data set guarantees fine-tuning to a certain extent, however, it still relies on the training performed on the larger PPMI data set. Therefore, increasing the 360 image data set size would help in generalizing the model well over only 360 videos. The 360 image data set is also highly skewed to a particular instrument that is the ***Guitar***, which makes it easier for the model to distinguish between other instruments. The unavailability of a varied collection of instruments in 360 videos makes it difficult to expand the number of classes in 360 image data set. Therefore, it is necessary to collect more 360 videos with a varied range of music instruments. Additionally, more recent object detection frameworks such as ***YOLO (You Only Look Once)***, ***EfficientDet*** etc. can also

be looked into.

The audio classification data set can be expanded to include the 2 missing instruments, which would normalize the 2 modal data sets to have the same class labels. Further experiments by changing the spectrogram dimensions from 128 x 64 can also be performed. The source separation technique suffers the most severe drawback of dividing the mixture signal to only 2 separate sound sources. Other experiments done on source separation and localization such as the Wave U-Net architecture can be looked into. Additionally, since our target data is 360 videos, more research can be conducted on generation of *ambisonics* and localizing sound sources rather than classifying them. Spatial audio plays a vital role in virtual reality environments and using such ambient audio features would help us locate instruments using only audio cues. Thus relieving us of using object detection methodologies.

Furthermore, our proposed method curates a dual-model structure, one using visual cues and the other using audio cues. Residual networks that incorporate both audio and visual features into a single model can be looked upon so as to scale the model further. A single fused model that caters to both the audio as well as the visual cues of a 360 video frame would largely reduce the framework size of the method, and also, make it easier to test.

# Bibliography

- [1] “Samsung gear vr,” [Online] Available: [https://en.wikipedia.org/wiki/Samsung\\_Gear\\_VR](https://en.wikipedia.org/wiki/Samsung_Gear_VR) [Accessed 06 September 2020].
- [2] “Google cardboard,” [Online]. Available: <https://arvr.google.com/cardboard/> [Accessed 06 September 2020].
- [3] “360 videos,” [Online] Available: [https://en.wikipedia.org/wiki/360-degree\\_video](https://en.wikipedia.org/wiki/360-degree_video) [Accessed 06 September 2020].
- [4] “Playstation vr,” [Online] Available: <https://www.playstation.com/en-us/explore/playstation-vr/> [Accessed 06 September 2020].
- [5] A. Tiwari, “Music information retrieval: An overview,” 08 2017.
- [6] J. Burgoyne, I. Fujinaga, and J. Downie, “Music information retrieval,” pp. 213–228, 11 2015.
- [7] M. Ihara, S.-i. Maeda, and S. Ishii, “Instrument identification in monophonic music using spectral information,” pp. 595 – 599, 01 2008.
- [8] Y. Sazaki, R. Ayuni, and S. Kom, “Musical note recognition using minimum spanning tree algorithm,” pp. 1–5, 10 2014.
- [9] A. Defossez, N. Usunier, L. Bottou, and F. Bach, “Demucs: Deep extractor for music sources with extra unlabeled data remixed,” 09 2019.
- [10] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam, “Spleeter: a fast and efficient music source separation tool with pre-trained models,” *Journal of Open Source Software*, vol. 5, p. 2154, 06 2020.

- [11] E. Benetos, S. Dixon, D. Giannoulis, H. Kirchhoff, and A. Klapuri, “Automatic music transcription: Challenges and future directions,” *Journal of Intelligent Information Systems*, vol. 41, 12 2013.
- [12] “X-ray feature. amazon prime video.,” [Online] Available: <https://www.amazon.com/primeinsider/video/pv-xray-tips.html> [Accessed 06 September 2020].
- [13] K. Mathivanan, S. T, A. B, and S. R, “A study of virtual reality,” *International Journal of Trend in Research and Development*, vol. 4, pp. 2394–9333, 06 2017.
- [14] “Oculus rift.,” [Online] Available: <https://www.oculus.com/> [Accessed 06 September 2020].
- [15] “Ikea place app.,” [Online] Available: <https://apps.apple.com/us/app/ikea-place/id1279244498> [Accessed 06 September 2020].
- [16] “Youtube.,” [Online] Available: <https://www.youtube.com/> [Accessed 06 September 2020].
- [17] “Rylo 360 camera.,” [Online] Available: <https://www.rylo.com/> [Accessed 06 September 2020].
- [18] “Gopro 360 camera.,” [Online] Available: <https://gopro.com/en/us/shop/cameras/max/CHDHZ-201-master.html> [Accessed 06 September 2020].
- [19] “Yi 360 camera.,” [Online] Available: <https://uk.pcmag.com/digital-camcorders/93317/yi-360-vr-camera> [Accessed 06 September 2020].
- [20] G. Piccinini, “The first computational theory of mind and brain: A close look at mcculloch and pitts’s “logical calculus of ideas immanent in nervous activity”,” *Synthese*, vol. 141, 08 2004.
- [21] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, “A survey of deep learning-based object detection,” *IEEE Access*, vol. PP, pp. 1–1, 09 2019.

- [22] M. A. Fischler and R. A. Elschlager, “The representation and matching of pictorial structures,” *IEEE Transactions on Computers*, vol. C-22, no. 1, pp. 67–92, 1973.
- [23] D. T. Nguyen, W. Li, and P. Ogunbona, “An improved template matching method for object detection,” vol. 5996, pp. 193–202, 09 2009.
- [24] H. Schneiderman and T. Kanade, “A statistical method for 3d object detection applied to faces and cars,” *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 746–751 vol.1, 02 2000.
- [25] C. Papageorgiou and T. Poggio, “A trainable system for object detection,” *International Journal of Computer Vision*, vol. 38, pp. 15–33, 06 2000.
- [26] S. Paisitkriangkrai, C. Shen, and A. Hengel, “Pedestrian detection with spatially pooled features and structured ensemble learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, 09 2014.
- [27] J. Lee, J. Bang, and S.-I. Yang, “Object detection with sliding window in images including multiple similar objects,” pp. 803–806, 10 2017.
- [28] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick, “Microsoft coco: Common objects in context,” 05 2014.
- [29] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11 2013.
- [30] R. Girshick, “Fast r-cnn,” 04 2015.
- [31] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International Journal of Computer Vision*, vol. 88, pp. 303–338, 06 2010.

- [32] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, 06 2015.
- [33] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” 03 2017.
- [34] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” 12 2016.
- [35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” pp. 779–788, 06 2016.
- [36] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” 12 2016.
- [37] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” 04 2018.
- [38] A. Bochkovskiy, C.-Y. Wang, and H.-y. Liao, “Yolov4: Optimal speed and accuracy of object detection,” 04 2020.
- [39] G. Jocher, “Yolov5,” 6 2020.
- [40] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. Berg, “Ssd: Single shot multibox detector,” vol. 9905, pp. 21–37, 10 2016.
- [41] A. Krizhevsky, I. Sutskever, and G. Hinton, “Imagenet classification with deep convolutional neural networks,” *Neural Information Processing Systems*, vol. 25, 01 2012.
- [42] F. Iandola, M. Moskewicz, K. Ashraf, S. Han, W. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and textless1mb model size,” 02 2016.
- [43] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, “Quantized convolutional neural networks for mobile devices,” 12 2015.
- [44] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 04 2017.



- [45] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” pp. 4510–4520, 06 2018.
- [46] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. Le, and H. Adam, “Searching for mobilenetv3,” 05 2019.
- [47] B. Zoph and Q. Le, “Neural architecture search with reinforcement learning,” 11 2016.
- [48] Z. Lu, J. Yang, and Q. Liu, “Face image retrieval based on shape and texture feature fusion,” *Computational Visual Media*, vol. 3, pp. 1–10, 08 2017.
- [49] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” 06 2016.
- [50] B. Sai and T. Sasikala, “Object detection and count of objects in image using tensor flow object detection api,” pp. 542–546, 11 2019.
- [51] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, “Speed/accuracy trade-offs for modern convolutional object detectors,” pp. 3296–3297, 07 2017.
- [52] C.-H. Hsieh, D.-C. Lin, C.-J. Wang, Z.-T. Chen, and J.-J. Liaw, “Real-time car detection and driving safety alarm system with google tensorflow object detection api,” 07 2019.
- [53] B. Sosorbaram and S. Bold, “Implementation of autonomous unmanned aerial vehicle with moving-object detection and face recognition,” 01 2016.
- [54] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “Deepface: Closing the gap to human-level performance in face verification,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 09 2014.
- [55] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” pp. 815–823, 06 2015.
- [56] A. Mishra, “Amazon rekognition,” pp. 421–444, 08 2019.

- [57] J. Roth, S. Chaudhuri, O. Klejch, R. Marvin, A. Gallagher, L. Kaver, S. Ramaswamy, A. Stopczynski, C. Schmid, Z. Xi, and C. Pantofaru, “Ava active speaker: An audio-visual dataset for active speaker detection,” pp. 4492–4496, 05 2020.
- [58] J. Qiu, P. W. Lo, and B. Lo, “Assessing individual dietary intake in food sharing scenarios with a 360 camera and deep learning,” pp. 1–4, 05 2019.
- [59] A. Pacha, J. Hajič, jr, and J. Calvo-Zaragoza, “A baseline for general music object detection with deep learning,” *Applied Sciences*, vol. 8, p. 1488, 08 2018.
- [60] “Google protocol buffers.” [Online] Available: <https://developers.google.com/protocol-buffers> [Accessed 06 September 2020].
- [61] A. Bronkhorst, “The cocktail-party problem revisited: early processing and selection of multi-talker speech,” *Attention, perception psychophysics*, vol. 77, 04 2015.
- [62] W. N. Cowan N, “Constraints on awareness, attention, processing, and memory: Some recent investigations with ignored speech,” *Conscious Cogn.*, pp. 6(2/3):182–203, 1997.
- [63] P. FB., “The cognitive determinants of behavioral distraction by deviant auditory stimuli: a review,” *Psychol Res.*, pp. 78(3):321–338, 2014.
- [64] X. He and I. Deng, “Speech-centric information processing: An optimization-oriented approach,” *Proceedings of the IEEE*, vol. 101, pp. 1116–1135, 05 2013.
- [65] C. Brooks and D. Hall, “Musical acoustics, 3rd edition,” 2001.
- [66] B. Gygi, G. Kidd, and C. Watson, “Spectral-temporal factors in the identification of environmental sounds,” *The Journal of the Acoustical Society of America*, vol. 115, pp. 1252–65, 03 2004.
- [67] E. Vincent, M. Jafari, S. Abdallah, M. Plumbley, and M. Davies, “Blind audio source separation,” 11 2005.

- [68] S. Nordholm, T. Abhayapala, D. Simon, S. Gannot, N. Patrick, and I. Tashev, “Microphone array speech processing,” *EURASIP Journal on Advances in Signal Processing*, 09 2010.
- [69] N. Mitianoudis and M. Davies, “Using beamforming in the audio source separation problem,” pp. 89 – 92 vol.2, 08 2003.
- [70] J.-M. Valin, F. Michaud, and J. Rouat, “Robust localization and tracking of simultaneous moving sound sources using beamforming and particle filtering,” *Robotics and Autonomous Systems*, vol. 55, pp. 216–228, 03 2007.
- [71] Y. Tian, C. Xu, and D. Li, “Deep audio prior,” 12 2019.
- [72] Y. Li and D. Wang, “Musical sound separation based on binary time-frequency masking,” *EURASIP J. Audio, Speech and Music Processing*, vol. 2009, 01 2009.
- [73] M. Kim and S. Choi, “Monaural music source separation: Nonnegativity, sparseness, and shift-invariance,” pp. 617–624, 03 2006.
- [74] H. Zhao, C. Gan, A. Rouditchenko, C. Vondrick, J. McDermott, and A. Torralba, “The sound of pixels,” 04 2018.
- [75] J. Salamon, C. Jacoby, and J. P. Bello, “A dataset and taxonomy for urban sound research,” pp. 1041–1044, Nov. 2014.
- [76] K. J. Piczak, “ESC: Dataset for Environmental Sound Classification,” in *Proceedings of the 23rd Annual ACM Conference on Multimedia*, pp. 1015–1018, ACM Press.
- [77] “Philharmonia sound samples.” [Online] Available: <https://philharmonia.co.uk/resources/sound-samples/> [Accessed 06 September 2020].
- [78] J. J. F. F. . H. P. Bosch, J. J., “A comparison of sound segregation techniques for predominant instrument recognition in musical audio signals,” pp. 559–564, 2012.
- [79] K. Piczak, “Esc: Dataset for environmental sound classification,” pp. 1015–1018, 10 2015.

- [80] V. Boddapati, A. Petef, J. Rasmusson, and L. Lundberg, “Classifying environmental sounds using image recognition networks,” *Procedia Computer Science*, vol. 112, pp. 2048–2056, 12 2017.
- [81] “Keras mobilenet,” [Online] Available: <https://keras.io/api/applications/mobilenet/> [Accessed 06 September 2020].
- [82] M. Sahidullah and G. Saha, “Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition,” *Speech Communication*, vol. 54, pp. 543–565, 05 2012.
- [83] M. Galeso, *Apple Siri for Mac: An Easy Guide to the Best Features*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2017.
- [84] “Alexa,” [Online] Available: <https://www.alexa.com> [Accessed 06 September 2020].
- [85] A. Li, M. Thotakuri, D. Ross, J. Carreira, A. Vostrikov, and A. Zisserman, “The ava-kinetics localized human actions video dataset,” 05 2020.
- [86] P. Morgado, N. Vasconcelos, T. Langlois, and O. Wang, “Self-supervised generation of spatial audio for 360 video,” 09 2018.
- [87] A. Owens and A. Efros, “Audio-visual scene analysis with self-supervised multi-sensory features,” 04 2018.
- [88] A. Delforouzi, S. Tabatabaei, K. Shirahama, and M. Grzegorzec, “Polar object tracking in 360-degree camera images,” pp. 347–352, 12 2016.
- [89] Y. Tian, J. Shi, B. Li, Z. Duan, and C. Xu, “Audio-visual event localization in unconstrained videos: 15th european conference, munich, germany, september 8-14, 2018, proceedings, part ii,” pp. 252–268, 09 2018.
- [90] A. Tepljakov, S. Astapov, E. Petlenkov, K. Vassiljeva, and D. Draheim, “Sound localization and processing for inducing synesthetic experiences in virtual reality,” pp. 159–162, 10 2016.

- [91] P. Shreevathsa, M. Harshith, A. M, and Ashwini, “Music instrument recognition using machine learning algorithms,” pp. 161–166, 01 2020.
- [92] Y. Aytar, C. Vondrick, and A. Torralba, “Soundnet: Learning sound representations from unlabeled video,” 10 2016.
- [93] “Flickr.,” [Online] Available: <https://www.flickr.com/> [Accessed 06 September 2020].
- [94] D. Giannoulis, E. Benetos, D. Stowell, M. Rossignol, M. Lagrange, and M. Plumbley, “Detection and classification of acoustic scenes and events: An ieeeaasp challenge,” *Proceedings of the 2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA 2013)*, pp. 1–4, 10 2013.
- [95] J.-Y. Liu, y.-h. Yang, and S.-K. Jeng, “Weakly-supervised visual instrument-playing action detection in videos,” 05 2018.
- [96] B. Yao and F. F. Li, “Grouplet: A structured image representation for recognizing human and object interactions,” pp. 9–16, 10 2010.
- [97] “Visual object tagging tools,” [Online] Available: <https://github.com/microsoft/VoTT> [Accessed 06 September 2020].
- [98] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from [tensorflow.org](http://tensorflow.org).
- [99] “Ffmpeg developers. (2016). ffmpeg tool (version be1d324) [software].,” [Online] Available: <http://ffmpeg.org/> [Accessed 06 September 2020].
- [100] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, “librosa: Audio and music signal analysis in python,” vol. 8, 2015.

- [101] I. Kandel and M. Castelli, “The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset,” *ICT Express*, 05 2020.
- [102] K. You, M. Long, J. Wang, and M. I. Jordan, “How does learning rate decay help modern neural networks?,” 2020.
- [103] D. Stoller, S. Ewert, and S. Dixon, “Wave-u-net: A multi-scale neural network for end-to-end audio source separation,” 06 2018.

# Appendix A

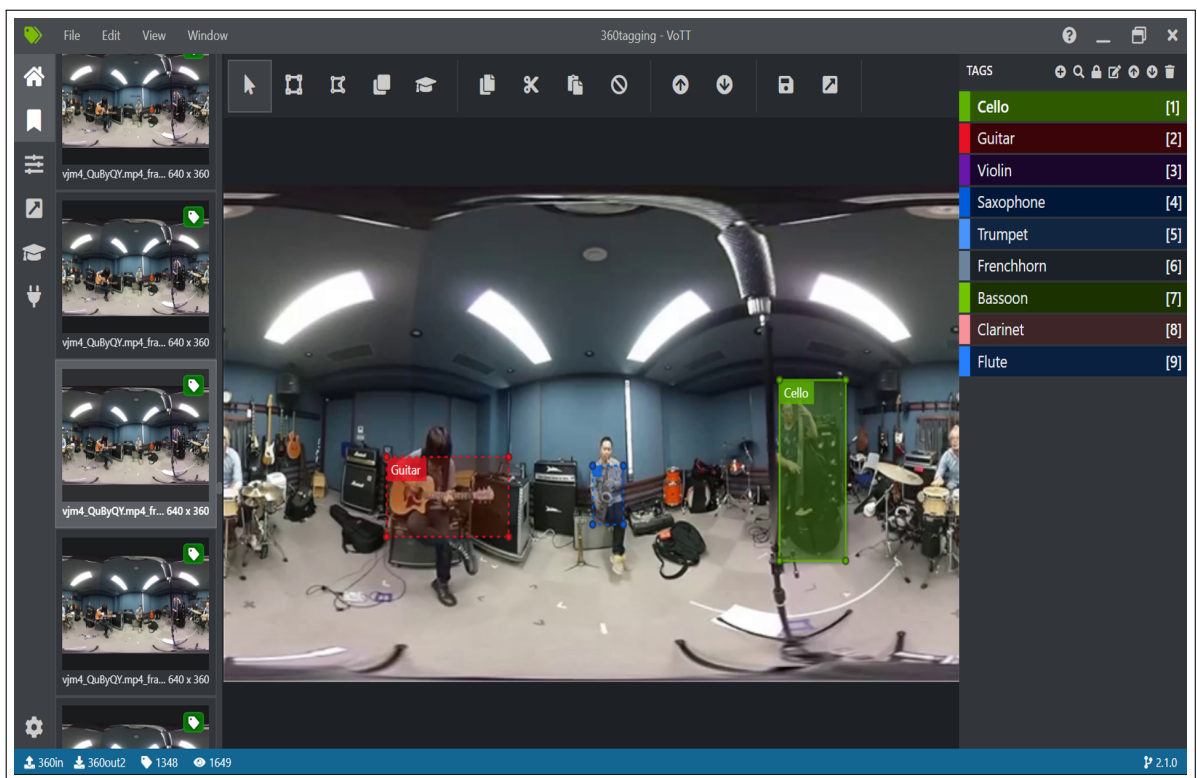


Figure 1: Visual Object Tagging Tool



Figure 2: 360 image object detection with 1 instrument

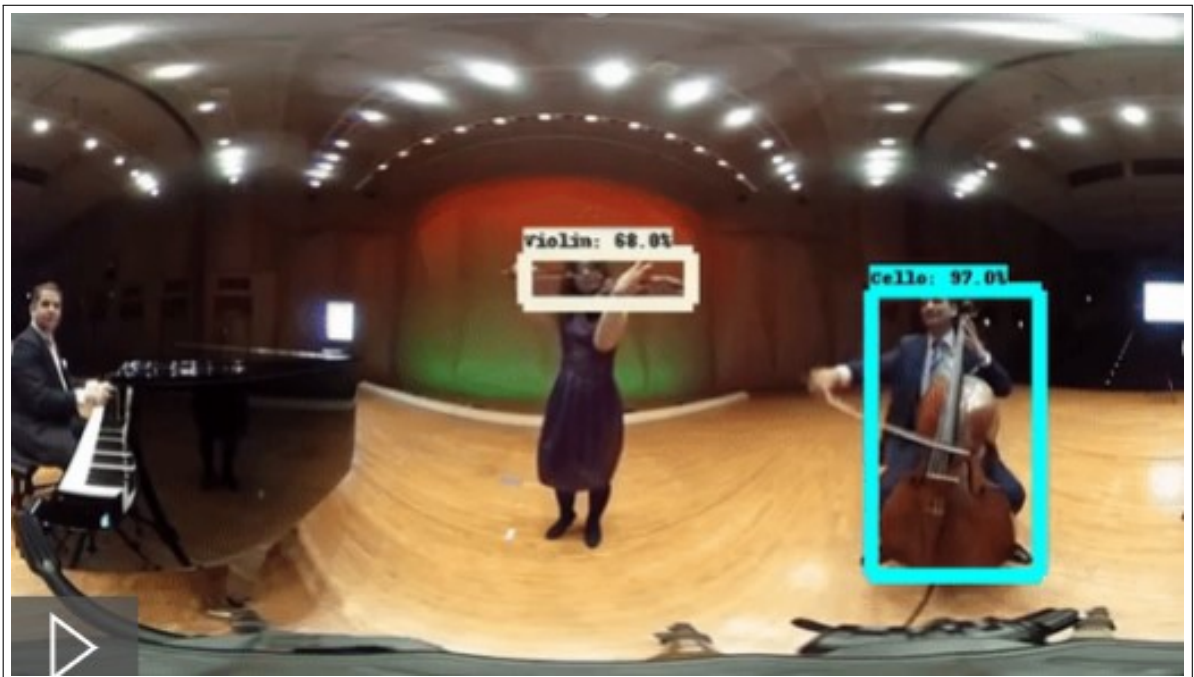


Figure 3: 360 image object detection with 2 instruments



# Appendix B

## Important Linux Commands

- Generating agglomerated training and validation data sets:

```
$ python3 .Scripts/generate_dataset.py \  
  --instrument_images_path <Base image directory> \  
  --target_directory <Combined split data set> \  
  --active_classes_config <path for config file> \  

```

- Generating data set statistics:

```
$ python3 .Scripts/generate_data_stats.py \  
  --data_dir <Combined Data set directory>
```

- Compiling Protocol Buffer files :

```
$ protoc object_detection/protos/*.proto --python_out=.
```

- Exporting object detection framework to python path :

```
$ cd /models/research  
$ export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

- Training object detection model :

```
$ python3 \  
/models/research/object_detection/model_main.py \  
--logtostderr \  
--train_dir=/training \  

```

```
--model_dir=/PreTrainedModels/360_inceptionv2_model1 \  
--pipeline_config_path=/configurations/model.config
```

- Generating Tensorflow Records :

```
$ python3 \  
  generateTFRecords.py \  
  --image_dir=training/360/train/images \  
  --csv_input=training/360/train/train.csv \  
  --output_path=training/360/train/train.record
```

- Formatting and sampling audio files :

```
$ ffmpeg -i 'original_audio.wav' -acodec pcm_s16le \  
-ac 2 -ar 44100 -ss 60 -t 5 -vn 'sampled_audio.wav'
```

- Deep audio prior blind source separation command :

```
$ python3 Deep-Audio-Prior/code/dap_sep.py \  
  --input_mix Deep-Audio-Prior/guitar_cello.wav \  
  --output guitarcellosep
```

- Launching sound classifier training :

```
$ python3 SoundClassifier/train_sc.py --width 128 --height 64 \  
  --batch-size 16 --epochs 1000 \  
  --dataset SoundClassifier/Dataset \  
  --output-model-name SoundClassifier/models/nineclass_12864161000 \  
  --plot-save-path SoundClassifier/plots/nineclass_12864161000.png
```

- Launching Tensorboard :

```
$ tensorboard --logdir='trained_model_path_checkpoints'
```

- Evaluate sound classifier :

```
$ python3 SoundClassifier/evaluate.py \  
  --width 128 \  
  --height 64 \  
  --batch-size 8 \  
  --plot-save-path SoundClassifier/plots/nineclass_12864161000.png
```

```
--validation_dir SoundClassifier/Dataset/validation \  
--model SoundClassifier/models/nineclass_1286481000.h5
```

- Export to inference graph :

```
$ python3 object_detection/export_inference_graph \  
--input_type=image_tensor \  
--pipeline_config_path=training/model.config \  
--trained_checkpoint_prefix=360_inceptionv2_model1/model.ckpt \  
--output_directory=frozen/360_ssd_inceptionv2_16_200k
```

- Export Tensorflow Lite inference graph :

```
$ python3 object_detection/export_tflite_ssd_graph.py \  
--input_type=image_tensor \  
--pipeline_config_path=training/model.config \  
--trained_checkpoint_prefix=360_inceptionv2_model1/model.ckpt \  
--output_directory=frozen/360_ssd_inceptionv2_16_200k
```

- Export Tensorflow Lite mobile application model :

```
$ tflite_convert --input_shapes=1,300,300,3 \  
--input_arrays=normalized_input_image_tensor \  
--output_arrays=TFLite_Detection_PostProcess, \  
                 TFLite_Detection_PostProcess:1, \  
                 TFLite_Detection_PostProcess:2, \  
                 TFLite_Detection_PostProcess:3 \  
--allow_custom_ops \  
--graph_def_file=/frozen_tflite/imageaiguitar.pb \  
--output_file=TFLITE/detect.tflite
```

# Appendix C

## Important links

- Music Instrument Localization repository  
<https://tinyurl.com/MusicInstrumentLoc>
- Music instrument localization demo from android app  
<https://www.youtube.com/watch?v=XyNMDB4iNjw>
- Localization UI/UX Google Colab Notebook  
<https://tinyurl.com/UIUXColabLoc>
- 2D images annotated data set  
<https://tinyurl.com/PPMIAnnotatedImages>
- 360 images annotated data set  
<https://tinyurl.com/360annotatedimages>
- Audio Data Set  
<https://tinyurl.com/AudioDataSet>

# Appendix D

## Abbreviations

**AP** - Average Precision

**ASR** - Automatic Speech Recognition

**BASS** - Blind Audio Source Separation

**BoF** - Bag of Freebies

**BoS** - Bag of Specials

**CNN** - Convolution Neural Networks

**COCO** - Common Objects in Context

**DAP** - Deep Audio Prior

**DCASE** - Detection and classification of Acoustic Scenes and Events

**DoA** - Distance of Arrival

**FN** - False Negative

**FP** - False Positive

**FPN** - Feature Pyramid Network

**GPU** - Graphics Processing Unit

**HMD** - Head Mounted Device

**IRMAS** - Instrument Recognition in Music Audio Signals

**IoU** - Intersection over Union

**mAP** - mean average precision

**MFC** - Mel-Frequency Cepstrum

**NLP** - Natural Language Processing

**PAN** - Path Aggregation Network

**PPMI** - People Playing Music Instruments

**Pascal VOC** - Visual Object Classes

**RMSProp** - Root Mean Square Propagation

**ReLU** - Rectified Linear Unit

**SAM** - Spatial Attention Modules

**SSD** - Single Shot Detector

**SVM** - Support Vector Machine

**TN** - True Negative

**TP** - True Positive

**VR** - Virtual Reality

**VoTT** - Visual Object Tagging Tool

**YOLO** - You Only Look Once