B. TECH. PROJECT REPORT

ON

Sound Source Localization in Image using Cross Modal Learning

By:

PREM KUMAR SEETHANABOYINA

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

December, 2019

Sound Source Localization in Image using Cross Modal Learning

A PROJECT REPORT

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

in the

Discipline of Electrical Engineering

Submitted by:

PREM KUMAR SEETHANABOYINA (160002042)

Guided by:

Dr. PUNEET GUPTA

Assistant Professor, Discipline of Computer Science and Engineering, IIT Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE

December, 2019

Declaration of Authorship

I hereby declare that the project entitled **"Sound Source Localization in Image using Cross Modal Learning"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in the Discipline of Electrical Engineering and completed under the supervision of **Dr. PUNEET GUPTA**, Assistant Professor, Discipline of Computer Science and Engineering, IIT Indore is an authentic work.

Furthermore, I declare that I have not submitted this work for the award of any other degree elsewhere.

Signature:

_____

Date:

_____

Certificate

This is to certify that the thesis entitled “Sound Source Localization in Image using Cross Modal Learning” and submitted by Prem Kumar Seethanaboyina, Roll No. 160002042, in partial fulfilment of the requirements for EE 493 B.Tech Project embodies the work done by him under my supervision. It is certified that the declaration made by the student is correct to the best of my knowledge.

_____

Supervisor

Dr. PUNEET GUPTA
Assistant Professor,
Indian Institute of Technology Indore
Date:

Abstract

Sounds are accompanied by visual scenes in nature and possess a strong correlation between them. Human brain is capable of understanding this correlation as it is continuously confronting audio and image samples and hence is able to localize the source of sound. The objective of this project is to localize the sound source in an image using machine learning methods. This problem involves two modals i.e. visual and audio. Therefore, I included two pipelines, one for visual features and another for audio features. These two pipelines are then combined using a cross-modal interference model. The cross-modal interference model then computes the correspondence score for each pixel and finally we get to know which pixels are contributing to the sound source.

Acknowledgements

I would like to thank my B.Tech Project supervisors **Dr. Puneet Gupta** for his guidance and constant support in structuring the project and his valuable feedback throughout the course of this project. His overseeing the project meant there was a lot that I learnt while working on it. I thank him for his time and efforts.

I am really grateful to the Institute for the opportunity to be exposed to systemic research.

Lastly, I offer my sincere thanks to everyone who helped me to complete this project, whose name I might have forgotten to mention.

Contents

List of Figures

List of Abbreviations

PSPnet	**Pyramid Scene Parsing Network**
SGD	**Stochastic Gradient Descent**
VGG	**Visual Geometry Group**

Chapter 1

Introduction

Generally, in day to day life, Humans observe a large number of visual-audio examples combined together and learn the relation between them throughout their whole life unknowingly. Due to the correlation between the audio and the visual events, humans are smart to understand the object or the particular event which generates sound and deduce and in addition to it localize the sound source even without any relevant education. Generally, the visual effect and also the corresponding sound effect occur naturally in synchronized fashion. Typically, these events of visuals are coherent with sounds. Visual events are integrated and om addition to this it also corresponds to the coherence property of sound. When we come across a moving car, we hear the engine sound at the same time. Sound carries important data of the spatial and temporal signals of the source within a visual scene. As shown below in figure 1, the source is identified based on the sound from engine. From this we can conclude that sound is not only corresponding to the visual information, but also associated to visual events.

In this project we attempt to locate the source of audio or which pixels are contributing to the source of sound for a pair of image/video and audio. Our model is designed using a two-stream network, in our case the two streams correspond to visual data and sound where particular network facilitates each modularity and localised module which integrates the attention mechanism.

In our daily lives, we generally encounter a huge visual and audio content. We can ask a similar question to machine whether it can identify and correlate the audio and video, and localize the sound source only with the knowledge of audio and visual content by understanding the relation between both events similar to humans? Nowadays there has been a lot of interest in using images and applying cross-modelling techniques in it. This project is based on learning the correspondence between visual scene and audio and try to localize the sound source.

Chapter 2

Literature survey

The given problem statement involves inputs of two different modals. Image which belongs to visual context and sound which belongs to audio context are the two inputs. Methods have been researched over for this problem statement which involve two different networks for each of the modalities which finally merge into one network for the detection of part of image or the pixels that contribute to the sound source There are many networks for the processing of images like Image-net, Alex-net etc. which are developed by different developers all over the world. These networks make the network required to process the visual modal. While there are many successful networks for the processing of images or the visual scene, sound networks pose a trouble in this problem. However, there are some proposed audio network structures that give significantly better results when included as a feature extractor or as a start of the network. The features extracted by the visual and audio network need to be combined so as to arrive at our result. In the localization module we get features from audio network as well as from visual network which need to be somehow mapped together so as that this module can identify which pixels contribute to the sound source through this mapping learnt by the model. Different functions have been proposed for the localization module such as Euclidean distance method, cross entropy etc. all these methods of localization are intended at calculating the correspondence of audio and each of the pixel so as to identify the pixels that contribute to the sound source. Euclidean distance method calculates the distance which is an analogy of dissimilarity between the audio vector and visual vector at every pixel. The cross entropy function also follows a similar approach in finding the correspondence between the two modals. Visual networks implemented are a series of convolution layers, pooling layers etc. depending on which image classification network to use and the audio networks are also built with convolution layers and pooling layers. The localization module predominantly has mathematical operations which need to be computed for large matrix. The output highlights the source of the sound. The model was trained on 1,00,000 samples in an unsupervised way and semi supervised way.

Chapter 3

Model design

Some of the layers and optimizers used in the model are explained below:

3.1 Layers

Pooling

There are two types of pooling layers, local or global pooling layers which can be incorporated in Convolutional networks. Pooling layers are mainly used to bring down the dimensions of the information carried by the neurons from one layer to another. It reads data from several neurons and gives a combined information through another neuron hence reducing the number of dimensions. Small clusters are combined using local pooling layers which generally use 2 x 2 sized kernel. While local pooling acts on small sized windows, global pooling layers are implemented to act upon all the information carrying neurons of the previous layers. Moreover pooling layers are capable of computing on average and max. Average pooling layer calculates the average of the incoming neurons' data while max-pooling layer is capable of taking the maximum value among all the neurons in a cluster.

Convolutional layer

The main building block for any convolution neural network is the convolution layer. The parameters for the layer include filters or commonly known as kernels which can learn and have small receptive field, still is capable of extending through complete input volume. As part of the process, dot product has to be calculated between the input and the filter values which is done by convolving or sliding the filter across the dimensions of the input layer and produces an activation map having 2-dimensions for the filter. Finally , the network then learns from the filters that activate once it detects specific pattern of features at some position within the input.

Activation maps produced from all the filters are then stacked up along a dimension which produces the convolution layer's final output. Each entry within the output volume produced can be understood as the information from a neuron that studies a small area in the input layer and communicates with other neurons from the same activation layer to share information.

Fully Connected Layer **(dense)**

Once the data is processed using convolution and max pool layers, fully connected layers come into the picture. The higher degree information is learnt by these layers. From the preceding layers connection is established to the Fully connected layers through a neuron. Thus the activations of a fully connected layer is computed in a parallel mannered transformation which can be calculated using matrix multiplication and adding the bias offset.

Lambda Layer

Keras provides a technique to make our own custom layers known as lambda layers. Using lambda layers one can build his own layers which are not inbuilt in keras. These lambda layers don't have associated weights with them. There could be some functionality that need to be performed on the input from the previous layer like keras backend functions etc. which are not provided as layers by keras. Hence a lambda layer comes into the picture in such instances.

Activation Function

Activation function keeps in check which neurons are ought to be activated by scheming the weighted sum and bas addition to it. This inhibits a notion of non linearity into the data carried by a neuron.

Softmax Function:

- **Nature:** non-linear
- **Uses:** Usually deployed when multiple classes need to be handled. The softmax function outputs a value for each class that ranges between 0 and 1 and also divide by the sum of the outputs.
- **Output:** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

3.2 Optimising Techniques:

Stochastic Gradient Descent (SGD):

The process or a system which is linked to a random probability is the meaning of stochastic. This is the main reason in stochastic gradient descent, random selection is used for few samples without considering total dataset for each iteration. A term batch is defined in gradient descent which corresponds to total number of samples among the complete data which is used for calculating gradient for respective iteration. The batch is considered to be as whole dataset in case of batch gradient descent which is a typical gradient descent optimization. The actual problem occurs when there is a huge dataset, but in some cases the complete data is used to obtain minimum in a less noisy manner.

In case of a typical gradient descent technique of optimization, if we consider a dataset of 1 million images we need to use the complete set of data while iterating for optimising and finding the minimum. In order to perform this it takes a lot of computational cost. To reduce the computational cost we use the stochastic gradient descent. In SGD a simple sample is inputted to the iteration i.e. batch size of 1 is used. The sample is selected among the complete data in a random manner.

SGD algorithm:

The algorithm is based in the recursive execution of the equation

$$\theta_j = \theta_j - \alpha\ (\hat{y}^i - y^i)x^i_j$$

In case of SGD, we calculate the slope of the cost function corresponding to every sample in every loop, instead of the sum of gradients of cost function corresponding to all samples.

In SGD, only one sample is selected at random among the complete data at each loop, the alsorithm takes a path to reach the minimum which is noiser than the ouput of the previous typically gradient descent algorithm. This does not have a major impact because the path of the algorithm is not dependent, our goal is to reach the minimum with less computational cost and shorter training time.

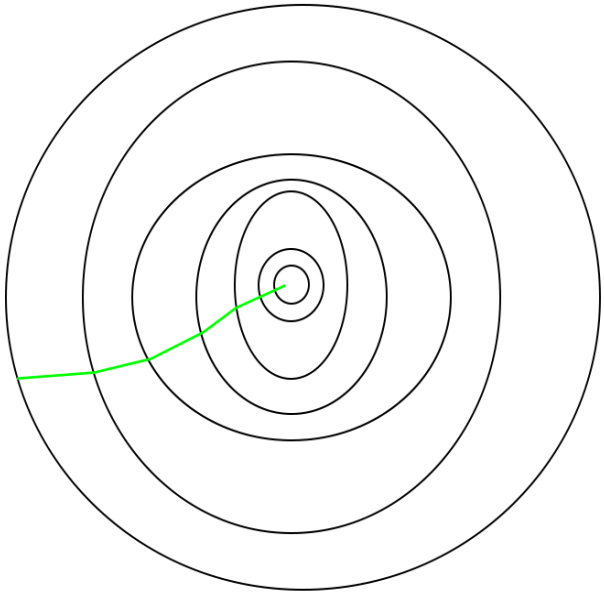


Figure 1: Path Taken by Batch Gradient Dedcent

(source: google images)

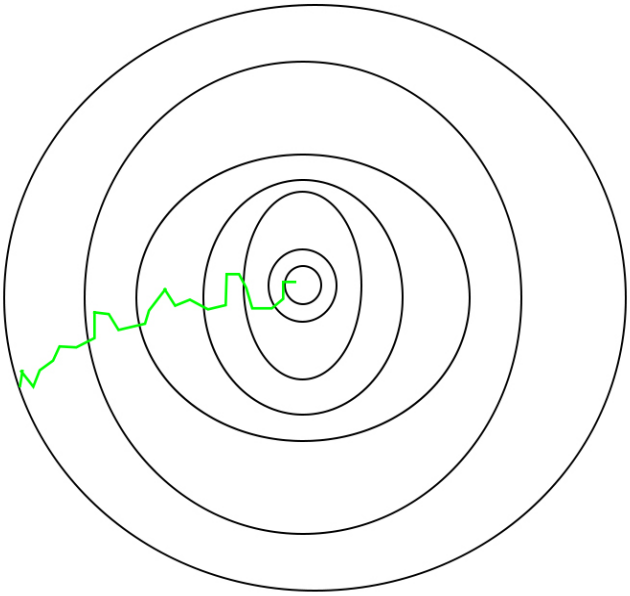


Figure 2 : Path taken by Stochastic Gradient Descent

(source: google images)

One fact which we have to consider is, as we consider stochastic gradient descent is noisier than the typical gradient descent, generally this takes a higher number of loops to reach the minimum than the typically used gradient, this has a very less computational cost than gradient descent. Hence due to this we generally prefer SDG over normal gradient descent.

After reading the papers objects that sound and sound source localization in visual scenes i came up with the following approach and implemented as described below.

Approach: We have two inputs, one image and other audio and the output is depending on these two inputs. Hence we included a pipeline for image and one for audio in the network architecture (as explained below). Based on the features generated by these pipelines a localising module is included to jointly learn the interdependency / mapping between the two sets of features.

3.3 Visual Network

Semantic segmentation

Deep learning as well as computer vision has seen a significant development in the field of image classification. We expect an image classification model to deduce what an object or the visual scene belong to. Image classification mainly deals with high level information. While image classification has gained a lot of recognition, object detection identifies, localizes and classify multiple objects detected within the image and draws a bounding box that circumferences these objects. Though detection in able to identify or locate the object and also classify it, still it is a mid-level process as we don't exactly get an idea of the actual shape and boundary of the object.
While image classification and object detection models provide us some information about the visual context, semantic segmentation provides us with a lot of information. semantic segmentation aims at classifying each pixel in the visual or image as shown below. At lot of research has been done on these task using deep learning methods.

Basic structure

The primary structure of semantic segmentation models the contemporary methods. This eases the usage of the methods for different ones, this can be done because the all type of problems have a similar setup to be implemented and also the backbone is used.

Great illustrations can be done of this structure, U-Net. Any type of extraction of feature network can be used for image classification for the left side. This includes different type of network like DenseNets, VGGNet, ResNets, NASNets and MobileNets.

The first fact which we need to consider for choosing classification network for feature extraction is to consider the tradeoffs. ResNet152 producecs a high accuracy but this cant be liable in case of speed as compared to MobileNet. The tradeoffs that we consider in case of classification are also applicable even for segmentation. The important which we need to note is the important conditions which we consider will have the major impact force in designing or selecting the segmentation network.

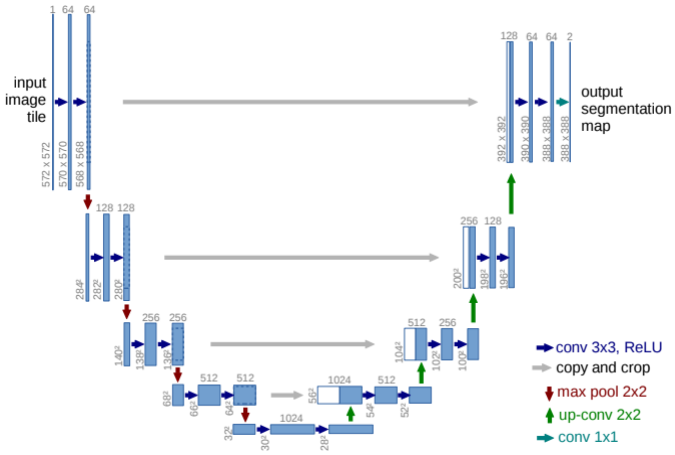


Figure 3 : U-Net model for segmentation

(source: google images)

The processing of extracted features are done at different scales after the features have been extracted. Two fold is the main reason for the previous step. In our complete data set there is a lot of diversity corresponding to the scale of the input, processing these diverse inputs will allow the network to have a good learning of different inputs.

One trade-off is considered in case of semantic segmentation. For a better accuracy of the results when a large dataset is used we need to process of a very high level features because these are very good for better understanding of the features and contain more semantic information. In case if we perform only deep features that leads to bias for better results for only complex samples which leads to less localisation in case of low resolution.

The recent contemporary best method has followed the previously discussed methods for feature extraction by multi scale processing. By using this many models can be easily implemented and trained. The selection criteria depend the use case of what we want to reduce the accuracy or the speed or memory. Many methods have been put forward to understand the trade-off, while not effecting the accuracy.

In the following we have discussed about the latest methods, since these methods will be most useful for the future workers in order to understand the primary structure.

This method can be used in different problems few of them include autonomous vehicle, robotics, computer human interaction and photo editing tools. If we consider semantic segmentation is very crucial in self driven four wheelers and robotics to understand the neighbourhood habitat.

Pyramid Scene Parsing Network (PSPNet)

The FRRN (semantic segmentation model) did an excellent job of directly executing the multi-scale processing. However, doing significant amount of processing at each step puts on a lot of pressure for computation, particularly when it has to carry out some processing at full resolution as well. This problem is aptly confronted by the PSPnet which proposes to use multiple pooling scales. The initial layers do the feature extraction work which are generally ResNet, DenseNet etc. It then collects features from the down sampling layers for further processing.

The aim of the PSPnet being the multi scale information of the image, this network involves 4 non-similar window sizes and strides. This is capable of capturing high level feature information using the 4 available scales by passing the huge computational load. A light loaded convolution task is enough to compute the features after upsampling which will be at same resolution. Finally, these all can be concatenated. We have now successfully concatenated feature maps of all the different scales available. All these tasks can be carried on input even with lower resolution with a great speed. To bring the resultant output to the required dimensions bilinear interpolation comes in handy to upscale. Various state-of-art networks have already focussed on this technique of upsampling after all processing has been done.

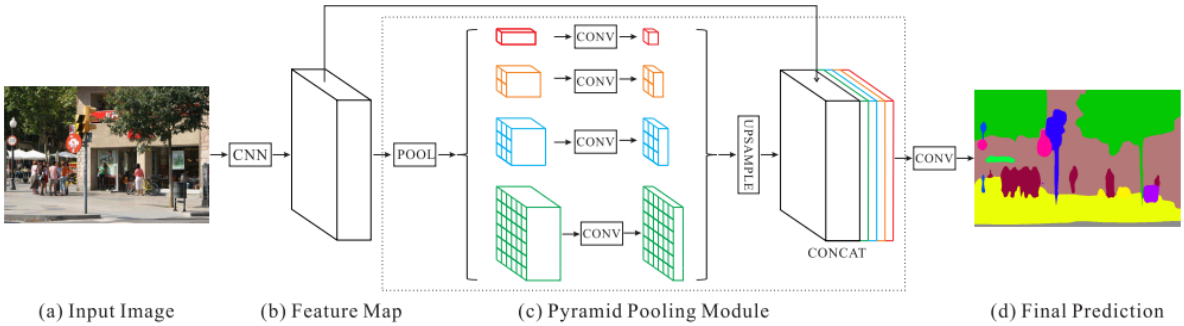


Figure 4 : PSPnet model structure

(source: google images)

Visual network is based on the structure of semantic segmentation network widely known as PSPnet. The PSPnet is capable of identifying 150 different classes such as (human, car, animal etc.). We inhibit the whole PSPnet except the last few layers after softmax. Now this network gives an output consisting of a vector of size 150 for each pixel which has the probabilities of the pixel belonging to the class.

Model: "model_3"

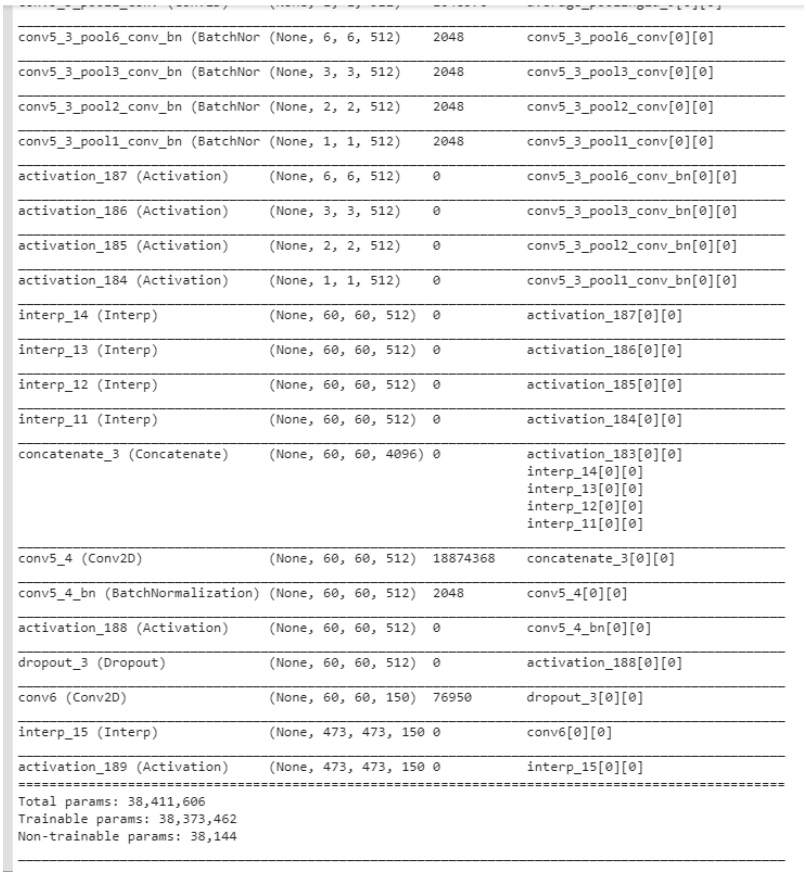
Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 473, 473, 3)	0	
conv1_1_3x3_s2 (Conv2D)	(None, 237, 237, 64)	1728	input_3[0][0]
conv1_1_3x3_s2_bn (BatchNormali	(None, 237, 237, 64)	256	conv1_1_3x3_s2[0][0]
activation_150 (Activation)	(None, 237, 237, 64)	0	conv1_1_3x3_s2_bn[0][0]
conv1_2_3x3 (Conv2D)	(None, 237, 237, 64)	36864	activation_150[0][0]
conv1_2_3x3_bn (BatchNormalizat	(None, 237, 237, 64)	256	conv1_2_3x3[0][0]
activation_151 (Activation)	(None, 237, 237, 64)	0	conv1_2_3x3_bn[0][0]
conv1_3_3x3 (Conv2D)	(None, 237, 237, 128	73728	activation_151[0][0]
conv1_3_3x3_bn (BatchNormalizat	(None, 237, 237, 128	512	conv1_3_3x3[0][0]
activation_152 (Activation)	(None, 237, 237, 128	0	conv1_3_3x3_bn[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 119, 119, 128	0	activation_152[0][0]
activation_153 (Activation)	(None, 119, 119, 128	0	max_pooling2d_3[0][0]
conv2_1_1x1_reduce (Conv2D)	(None, 119, 119, 64)	8192	activation_153[0][0]
conv2_1_1x1_reduce_bn (BatchNor	(None, 119, 119, 64)	256	conv2_1_1x1_reduce[0][0]
activation_154 (Activation)	(None, 119, 119, 64)	0	conv2_1_1x1_reduce_bn[0][0]
zero_padding2d_44 (ZeroPadding2	(None, 121, 121, 64)	0	activation_154[0][0]
conv2_1_3x3 (Conv2D)	(None, 119, 119, 64)	36864	zero_padding2d_44[0][0]
conv2_1_3x3_bn (BatchNormalizat	(None, 119, 119, 64)	256	conv2_1_3x3[0][0]
activation_155 (Activation)	(None, 119, 119, 64)	0	conv2_1_3x3_bn[0][0]
conv2_1_1x1_increase (Conv2D)	(None, 119, 119, 256	16384	activation_155[0][0]
conv2_1_1x1_proj (Conv2D)	(None, 119, 119, 256	32768	activation_153[0][0]
conv2_1_1x1_increase_bn (BatchN	(None, 119, 119, 256	1024	conv2_1_1x1_increase[0][0]
conv2_1_1x1_proj_bn (BatchNorma	(None, 119, 119, 256	1024	conv2_1_1x1_proj[0][0]
add_44 (Add)	(None, 119, 119, 256	0	conv2_1_1x1_increase_bn[0][0] conv2_1_1x1_proj_bn[0][0]

			conv2_1_1x1_proj_bn[0][0]
activation_156 (Activation)	(None, 119, 119, 256	0	add_44[0][0]
conv2_2_1x1_reduce (Conv2D)	(None, 119, 119, 64)	16384	activation_156[0][0]
conv2_2_1x1_reduce_bn (BatchNor	(None, 119, 119, 64)	256	conv2_2_1x1_reduce[0][0]
activation_157 (Activation)	(None, 119, 119, 64)	0	conv2_2_1x1_reduce_bn[0][0]
zero_padding2d_45 (ZeroPadding2	(None, 121, 121, 64)	0	activation_157[0][0]
conv2_2_3x3 (Conv2D)	(None, 119, 119, 64)	36864	zero_padding2d_45[0][0]
conv2_2_3x3_bn (BatchNormalizat	(None, 119, 119, 64)	256	conv2_2_3x3[0][0]
activation_158 (Activation)	(None, 119, 119, 64)	0	conv2_2_3x3_bn[0][0]
conv2_2_1x1_increase (Conv2D)	(None, 119, 119, 256	16384	activation_158[0][0]
conv2_2_1x1_increase_bn (BatchN	(None, 119, 119, 256	1024	conv2_2_1x1_increase[0][0]
add_45 (Add)	(None, 119, 119, 256	0	conv2_2_1x1_increase_bn[0][0] activation_156[0][0]
activation_159 (Activation)	(None, 119, 119, 256	0	add_45[0][0]
conv2_3_1x1_reduce (Conv2D)	(None, 119, 119, 64)	16384	activation_159[0][0]
conv2_3_1x1_reduce_bn (BatchNor	(None, 119, 119, 64)	256	conv2_3_1x1_reduce[0][0]
activation_160 (Activation)	(None, 119, 119, 64)	0	conv2_3_1x1_reduce_bn[0][0]
zero_padding2d_46 (ZeroPadding2	(None, 121, 121, 64)	0	activation_160[0][0]
conv2_3_3x3 (Conv2D)	(None, 119, 119, 64)	36864	zero_padding2d_46[0][0]
conv2_3_3x3_bn (BatchNormalizat	(None, 119, 119, 64)	256	conv2_3_3x3[0][0]
activation_161 (Activation)	(None, 119, 119, 64)	0	conv2_3_3x3_bn[0][0]
conv2_3_1x1_increase (Conv2D)	(None, 119, 119, 256	16384	activation_161[0][0]
conv2_3_1x1_increase_bn (BatchN	(None, 119, 119, 256	1024	conv2_3_1x1_increase[0][0]
add_46 (Add)	(None, 119, 119, 256	0	conv2_3_1x1_increase_bn[0][0] activation_159[0][0]
activation_162 (Activation)	(None, 119, 119, 256	0	add_46[0][0]
conv3_1_1x1_reduce (Conv2D)	(None, 60, 60, 128)	32768	activation_162[0][0]
conv3_1_1x1_reduce_bn (BatchNor	(None, 60, 60, 128)	512	conv3_1_1x1_reduce[0][0]

activation_163 (Activation)	(None, 60, 60, 128)	0	conv3_1_1x1_reduce_bn[0][0]
zero_padding2d_47 (ZeroPadding2	(None, 62, 62, 128)	0	activation_163[0][0]
conv3_1_3x3 (Conv2D)	(None, 60, 60, 128)	147456	zero_padding2d_47[0][0]
conv3_1_3x3_bn (BatchNormalizat	(None, 60, 60, 128)	512	conv3_1_3x3[0][0]
activation_164 (Activation)	(None, 60, 60, 128)	0	conv3_1_3x3_bn[0][0]
conv3_1_1x1_increase (Conv2D)	(None, 60, 60, 512)	65536	activation_164[0][0]
conv3_1_1x1_proj (Conv2D)	(None, 60, 60, 512)	131072	activation_162[0][0]
conv3_1_1x1_increase_bn (BatchN	(None, 60, 60, 512)	2048	conv3_1_1x1_increase[0][0]
conv3_1_1x1_proj_bn (BatchNorma	(None, 60, 60, 512)	2048	conv3_1_1x1_proj[0][0]
add_47 (Add)	(None, 60, 60, 512)	0	conv3_1_1x1_increase_bn[0][0] conv3_1_1x1_proj_bn[0][0]
activation_165 (Activation)	(None, 60, 60, 512)	0	add_47[0][0]
conv3_2_1x1_reduce (Conv2D)	(None, 60, 60, 128)	65536	activation_165[0][0]
conv3_2_1x1_reduce_bn (BatchNor	(None, 60, 60, 128)	512	conv3_2_1x1_reduce[0][0]
activation_166 (Activation)	(None, 60, 60, 128)	0	conv3_2_1x1_reduce_bn[0][0]
zero_padding2d_48 (ZeroPadding2	(None, 62, 62, 128)	0	activation_166[0][0]
conv3_2_3x3 (Conv2D)	(None, 60, 60, 128)	147456	zero_padding2d_48[0][0]
conv3_2_3x3_bn (BatchNormalizat	(None, 60, 60, 128)	512	conv3_2_3x3[0][0]
activation_167 (Activation)	(None, 60, 60, 128)	0	conv3_2_3x3_bn[0][0]
conv3_2_1x1_increase (Conv2D)	(None, 60, 60, 512)	65536	activation_167[0][0]
conv3_2_1x1_increase_bn (BatchN	(None, 60, 60, 512)	2048	conv3_2_1x1_increase[0][0]
add_48 (Add)	(None, 60, 60, 512)	0	conv3_2_1x1_increase_bn[0][0] activation_165[0][0]
activation_168 (Activation)	(None, 60, 60, 512)	0	add_48[0][0]
conv3_3_1x1_reduce (Conv2D)	(None, 60, 60, 128)	65536	activation_168[0][0]
conv3_3_1x1_reduce_bn (BatchNor	(None, 60, 60, 128)	512	conv3_3_1x1_reduce[0][0]
activation_169 (Activation)	(None, 60, 60, 128)	0	conv3_3_1x1_reduce_bn[0][0]

zero_padding2d_49 (ZeroPadding2	(None, 62, 62, 128)	0	activation_169[0][0]
conv3_3_3x3 (Conv2D)	(None, 60, 60, 128)	147456	zero_padding2d_49[0][0]
conv3_3_3x3_bn (BatchNormalizat	(None, 60, 60, 128)	512	conv3_3_3x3[0][0]
activation_170 (Activation)	(None, 60, 60, 128)	0	conv3_3_3x3_bn[0][0]
conv3_3_1x1_increase (Conv2D)	(None, 60, 60, 512)	65536	activation_170[0][0]
conv3_3_1x1_increase_bn (BatchN	(None, 60, 60, 512)	2048	conv3_3_1x1_increase[0][0]
add_49 (Add)	(None, 60, 60, 512)	0	conv3_3_1x1_increase_bn[0][0] activation_168[0][0]
activation_171 (Activation)	(None, 60, 60, 512)	0	add_49[0][0]
conv3_4_1x1_reduce (Conv2D)	(None, 60, 60, 128)	65536	activation_171[0][0]
conv3_4_1x1_reduce_bn (BatchNor	(None, 60, 60, 128)	512	conv3_4_1x1_reduce[0][0]
activation_172 (Activation)	(None, 60, 60, 128)	0	conv3_4_1x1_reduce_bn[0][0]
zero_padding2d_50 (ZeroPadding2	(None, 62, 62, 128)	0	activation_172[0][0]
conv3_4_3x3 (Conv2D)	(None, 60, 60, 128)	147456	zero_padding2d_50[0][0]
conv3_4_3x3_bn (BatchNormalizat	(None, 60, 60, 128)	512	conv3_4_3x3[0][0]
activation_173 (Activation)	(None, 60, 60, 128)	0	conv3_4_3x3_bn[0][0]
conv3_4_1x1_increase (Conv2D)	(None, 60, 60, 512)	65536	activation_173[0][0]
conv3_4_1x1_increase_bn (BatchN	(None, 60, 60, 512)	2048	conv3_4_1x1_increase[0][0]
add_50 (Add)	(None, 60, 60, 512)	0	conv3_4_1x1_increase_bn[0][0] activation_171[0][0]
activation_174 (Activation)	(None, 60, 60, 512)	0	add_50[0][0]
conv5_1_1x1_reduce (Conv2D)	(None, 60, 60, 512)	262144	activation_174[0][0]
conv5_1_1x1_reduce_bn (BatchNor	(None, 60, 60, 512)	2048	conv5_1_1x1_reduce[0][0]
activation_175 (Activation)	(None, 60, 60, 512)	0	conv5_1_1x1_reduce_bn[0][0]
zero_padding2d_51 (ZeroPadding2	(None, 68, 68, 512)	0	activation_175[0][0]
conv5_1_3x3 (Conv2D)	(None, 60, 60, 512)	2359296	zero_padding2d_51[0][0]
conv5_1_3x3_bn (BatchNormalizat	(None, 60, 60, 512)	2048	conv5_1_3x3[0][0]

conv5_1_3x3_bn (BatchNormalizat	(None, 60, 60, 512)	2048	conv5_1_3x3[0][0]
activation_176 (Activation)	(None, 60, 60, 512)	0	conv5_1_3x3_bn[0][0]
conv5_1_1x1_increase (Conv2D)	(None, 60, 60, 2048)	1048576	activation_176[0][0]
conv5_1_1x1_proj (Conv2D)	(None, 60, 60, 2048)	1048576	activation_174[0][0]
conv5_1_1x1_increase_bn (BatchN	(None, 60, 60, 2048)	8192	conv5_1_1x1_increase[0][0]
conv5_1_1x1_proj_bn (BatchNorma	(None, 60, 60, 2048)	8192	conv5_1_1x1_proj[0][0]
add_51 (Add)	(None, 60, 60, 2048)	0	conv5_1_1x1_increase_bn[0][0] conv5_1_1x1_proj_bn[0][0]
activation_177 (Activation)	(None, 60, 60, 2048)	0	add_51[0][0]
conv5_2_1x1_reduce (Conv2D)	(None, 60, 60, 512)	1048576	activation_177[0][0]
conv5_2_1x1_reduce_bn (BatchNor	(None, 60, 60, 512)	2048	conv5_2_1x1_reduce[0][0]
activation_178 (Activation)	(None, 60, 60, 512)	0	conv5_2_1x1_reduce_bn[0][0]
zero_padding2d_52 (ZeroPadding2	(None, 68, 68, 512)	0	activation_178[0][0]
conv5_2_3x3 (Conv2D)	(None, 60, 60, 512)	2359296	zero_padding2d_52[0][0]
conv5_2_3x3_bn (BatchNormalizat	(None, 60, 60, 512)	2048	conv5_2_3x3[0][0]
activation_179 (Activation)	(None, 60, 60, 512)	0	conv5_2_3x3_bn[0][0]
conv5_2_1x1_increase (Conv2D)	(None, 60, 60, 2048)	1048576	activation_179[0][0]
conv5_2_1x1_increase_bn (BatchN	(None, 60, 60, 2048)	8192	conv5_2_1x1_increase[0][0]
add_52 (Add)	(None, 60, 60, 2048)	0	conv5_2_1x1_increase_bn[0][0] activation_177[0][0]
activation_180 (Activation)	(None, 60, 60, 2048)	0	add_52[0][0]
conv5_3_1x1_reduce (Conv2D)	(None, 60, 60, 512)	1048576	activation_180[0][0]
conv5_3_1x1_reduce_bn (BatchNor	(None, 60, 60, 512)	2048	conv5_3_1x1_reduce[0][0]
activation_181 (Activation)	(None, 60, 60, 512)	0	conv5_3_1x1_reduce_bn[0][0]
zero_padding2d_53 (ZeroPadding2	(None, 68, 68, 512)	0	activation_181[0][0]
conv5_3_3x3 (Conv2D)	(None, 60, 60, 512)	2359296	zero_padding2d_53[0][0]
conv5_3_3x3_bn (BatchNormalizat	(None, 60, 60, 512)	2048	conv5_3_3x3[0][0]
activation_182 (Activation)	(None, 60, 60, 512)	0	conv5_3_3x3_bn[0][0]



conv5_3_pool6_conv_bn (BatchNor	(None, 6, 6, 512)	2048	conv5_3_pool6_conv[0][0]
conv5_3_pool3_conv_bn (BatchNor	(None, 3, 3, 512)	2048	conv5_3_pool3_conv[0][0]
conv5_3_pool2_conv_bn (BatchNor	(None, 2, 2, 512)	2048	conv5_3_pool2_conv[0][0]
conv5_3_pool1_conv_bn (BatchNor	(None, 1, 1, 512)	2048	conv5_3_pool1_conv[0][0]
activation_187 (Activation)	(None, 6, 6, 512)	0	conv5_3_pool6_conv_bn[0][0]
activation_186 (Activation)	(None, 3, 3, 512)	0	conv5_3_pool3_conv_bn[0][0]
activation_185 (Activation)	(None, 2, 2, 512)	0	conv5_3_pool2_conv_bn[0][0]
activation_184 (Activation)	(None, 1, 1, 512)	0	conv5_3_pool1_conv_bn[0][0]
interp_14 (Interp)	(None, 60, 60, 512)	0	activation_187[0][0]
interp_13 (Interp)	(None, 60, 60, 512)	0	activation_186[0][0]
interp_12 (Interp)	(None, 60, 60, 512)	0	activation_185[0][0]
interp_11 (Interp)	(None, 60, 60, 512)	0	activation_184[0][0]
concatenate_3 (Concatenate)	(None, 60, 60, 4096)	0	activation_183[0][0] interp_14[0][0] interp_13[0][0] interp_12[0][0] interp_11[0][0]
conv5_4 (Conv2D)	(None, 60, 60, 512)	18874368	concatenate_3[0][0]
conv5_4_bn (BatchNormalization)	(None, 60, 60, 512)	2048	conv5_4[0][0]
activation_188 (Activation)	(None, 60, 60, 512)	0	conv5_4_bn[0][0]
dropout_3 (Dropout)	(None, 60, 60, 512)	0	activation_188[0][0]
conv6 (Conv2D)	(None, 60, 60, 150)	76950	dropout_3[0][0]
interp_15 (Interp)	(None, 473, 473, 150	0	conv6[0][0]
activation_189 (Activation)	(None, 473, 473, 150	0	interp_15[0][0]

Total params: 38,411,606
Trainable params: 38,373,462
Non-trainable params: 38,144

Figure 5 : Visual Network Summary

3.4 Audio Network:

VGG16:

A. Zisserman and K. Simonyan from the University of Oxford have a modelled a convolution neural network named VGG16 in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model was tested using ImageNet, and the accuracy was 92.7% , the data set which includes different types of data classified into thousand classes and complete data to be around 140 lakh images. After developing the model, it was submitted to ILSVRC-2014 , it also been improved by using huge kernels by replicating huge kernel sized filters with a lot of 3×3 kernel-sized filters placing them in a serial fashion. The training of VGG16 was done by using NVIDIA Titan black GPU’S.

The architecture depicted below is VGG16.

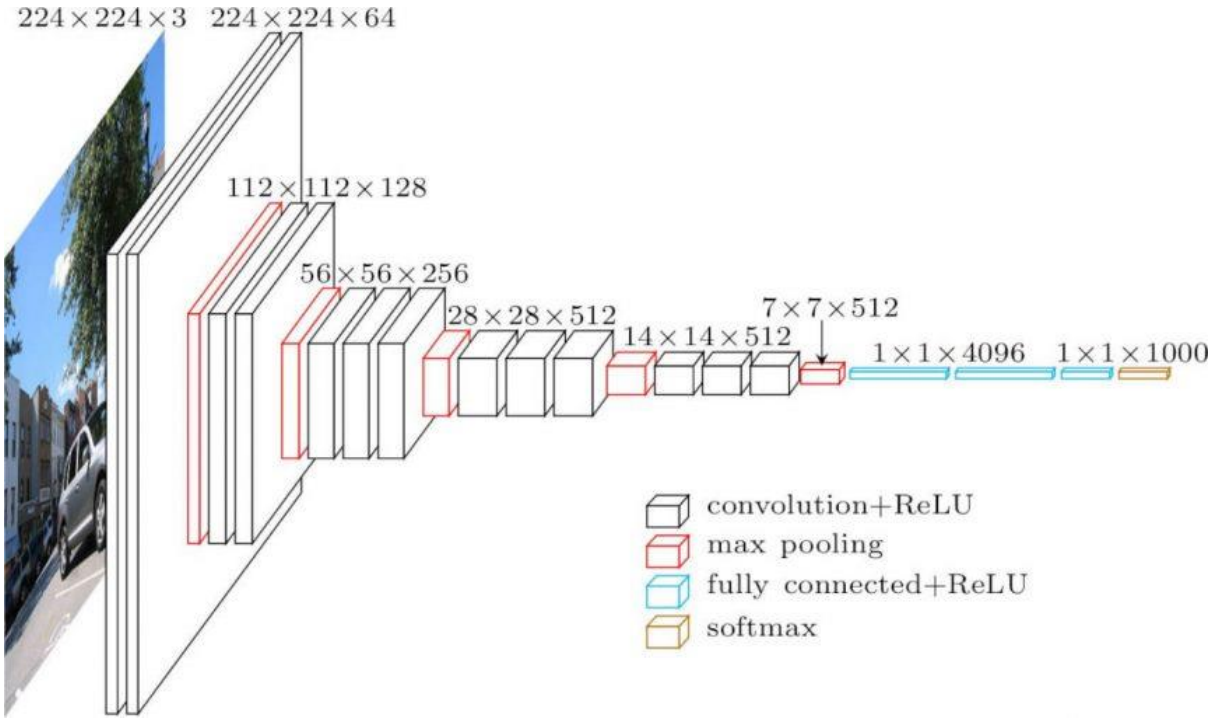


Figure 6 : VGG-16 Architecture

(source: google images)

The first convolution layer is inputted with an image of size 224 x 224 colour image(RGB). A series of convolution layers were created an image is passed through them, in these layers filters are used in each layer in addition to them a small receptive filed is also used: 3x3 (this being the minimalistic size to capture the configuration). Among these configurations, it uses 1x1 convolution filter in one of them, in these a linear transformation of channels in input are observed. The stride is set to be 1; the convolution layer's spatial padding is set so that the property od spatial resolution is preserved even after performing convolution i.e. the padding for 3x3 convolution layer is 1 pixel. For the five corresponding max pool layers spatial pooling is carried out. For the max pooling, we have considered a stride 2 and performed on 2x2 pixel window.

Three fully connected layers are used where, the first two layers contain 4096 channels each and the last one is used to perform 1000-way ILSVRC classification and totally have 1000 channels. The last layer corresponds to soft max layers. In all the fully connected layers the configuration is same for all networks.

Rectification (ReLU) non-linearity is incorporated in all the hidden layers. One of the assumptions is that none of the networks contain Local response normalisation (LRN), these types of

normalization has no effect on improving the performance of ILSVRC dataset, this leads to increase in the memory consumption and computation time.

VGGish

Architecture

One of the variants of the VGG model is VGGish, the difference between VGGish and VGG as a configuration A with 11 weight layers. Few of the changes are mentioned below:

- In a log mel spectrogram audio inputs size is changed to 96x64.
- Four of the five convolution/maxpool were considered, the last layer was not considered.
- We have included fully connected layer of 128 wide, instead of a fully connected layer of 1000 wide. It corresponds to a compact embedding layer. The model definition mentioned here defines the layers up to and includes the embedding layer of 128 wide.

VGGish was trained with audio features computed as follows:

- Complete audio data is resampled to 10 kHz.
- A spectrogram is mapped to 64 mel bins which covers almost a quiet range 125-7500 Hz. This is used in mel spectrogram.
- Magnitudes of Short-Time Fourier transform is computed with a window size of 25ms, a window hop of 10ms and a periodic Hann window.
- A log (mel-spectrum + 0.01) where offset is avoided in taking log of 0 is used to stabilize log mel spectrogram.
- Each of the examples corresponding to these features are framed into non-overleaping examples of 0.96 seconds, each has 96 frames and 64 mel bands of 10 ms each.

The well-known VGGish network is incorporated into this network. VGGish network being adopted from VGG -16 classifying model, mainly consists of convolution layers, pooling layers, and dense (fully connected) layers. The final output of this network is a 128 dimensional vector which has highly reliable features. These features are then sent into a dense layer which in turn outputs a 150 (it is the number of classes PSPnet identifies) dimensional vector. The basic structure is shown in the figure.

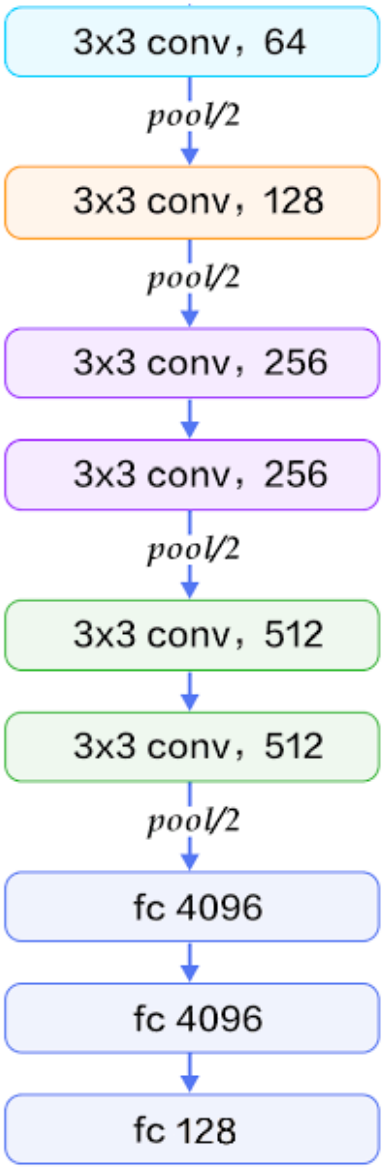


Figure 7 : Audio Network Flowchart

(source: google images)

```
Model: "VGGish"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1audmodel (InputLayer) (None, 96, 64, 1)         0
_________________________________________________________________
conv1audmodel (Conv2D)       (None, 96, 64, 64)        640
_________________________________________________________________
pool1audmodel (MaxPooling2D) (None, 48, 32, 64)        0
_________________________________________________________________
conv2audmodel (Conv2D)       (None, 48, 32, 128)       73856
_________________________________________________________________
pool2audmodel (MaxPooling2D) (None, 24, 16, 128)       0
_________________________________________________________________
conv3/conv3_1audmodel (Conv2 (None, 24, 16, 256)       295168
_________________________________________________________________
conv3/conv3_2audmodel (Conv2 (None, 24, 16, 256)       590080
_________________________________________________________________
pool3audmodel (MaxPooling2D) (None, 12, 8, 256)        0
_________________________________________________________________
conv4/conv4_1audmodel (Conv2 (None, 12, 8, 512)        1180160
_________________________________________________________________
conv4/conv4_2audmodel (Conv2 (None, 12, 8, 512)        2359808
_________________________________________________________________
pool4audmodel (MaxPooling2D) (None, 6, 4, 512)         0
_________________________________________________________________
flatten_audmodel (Flatten)   (None, 12288)             0
_________________________________________________________________
vggish_fc1/fc1_1audmodel (De (None, 4096)              50335744
_________________________________________________________________
vggish_fc1/fc1_2audmodel (De (None, 4096)              16781312
_________________________________________________________________
vggish_fc2audmodel (Dense)   (None, 128)               524416
=================================================================
Total params: 72,141,184
Trainable params: 0
Non-trainable params: 72,141,184
```

Figure 8 : Audio Network Summary

3.5 Cross modal interference :

1. A softmax layer is present at the end of visual and audio layers which ensures that maximum sum of all elements in the vector (be that audio vector or any vector at the pixel) is 1.

2. Dot product is calculated between audio features vector and vector corresponding to each pixel of the image producing a single value corresponding to each pixel of the image which is the score of that pixel.

3. Contribution of each pixel is given by the score corresponding to that pixel which was obtained in the previous step. This score has a max value of 1 when dot product is taken which is scaled to the range of 0 to 255 for ease of colour coding.

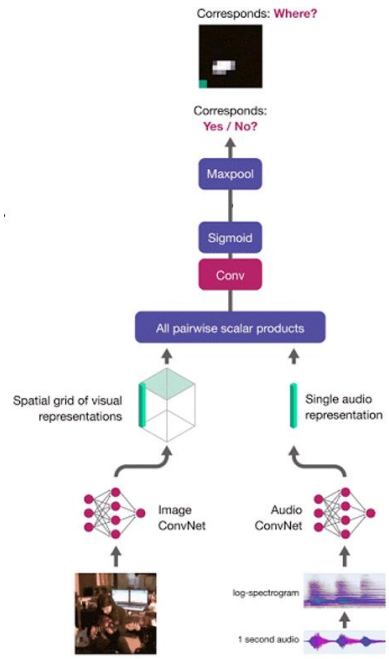


Figure 9 : Cross Modal Interface Overview

(source: google images)

3.6 Intuition behind Cross-modal interface:

1. Visual network generates a vector corresponding to each pixel of the image. The size of the vector is the number of classes the PSPnet is capable of detecting.
2. Audio network generates a vector of size equal to the number of classes the PSPnet is capable of detecting.

When dot product is calculated between the audio features and visual network features for a pixel a score is produced. If the pixel is contributing to the sound source, both audio and visual vector will have high value at same class index producing higher score.

For example:

Consider the following audio and video vectors:

Audio vector	Visual vector (at a pixel)
0.005	0.005
0.01	0.01
0.95	0.94
0.01	0.003
0.01	0.01
0.002	0.01
0.01	0.017
0.003	0.005

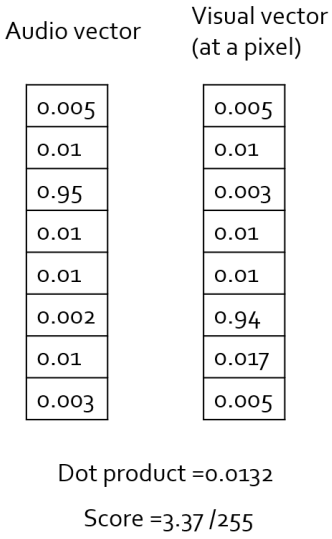
Dot product =0.893

Score =227.7 / 255

(a)

Since the score is high that pixel is contributing to the sound source.

Consider another audio and visual vectors:



(b)

Figure 10 : Examples for (a) Corresponding, (b) Non-Corresponding Cases

Since the score is low that pixel is not contributing to the sound source.

Chapter 4

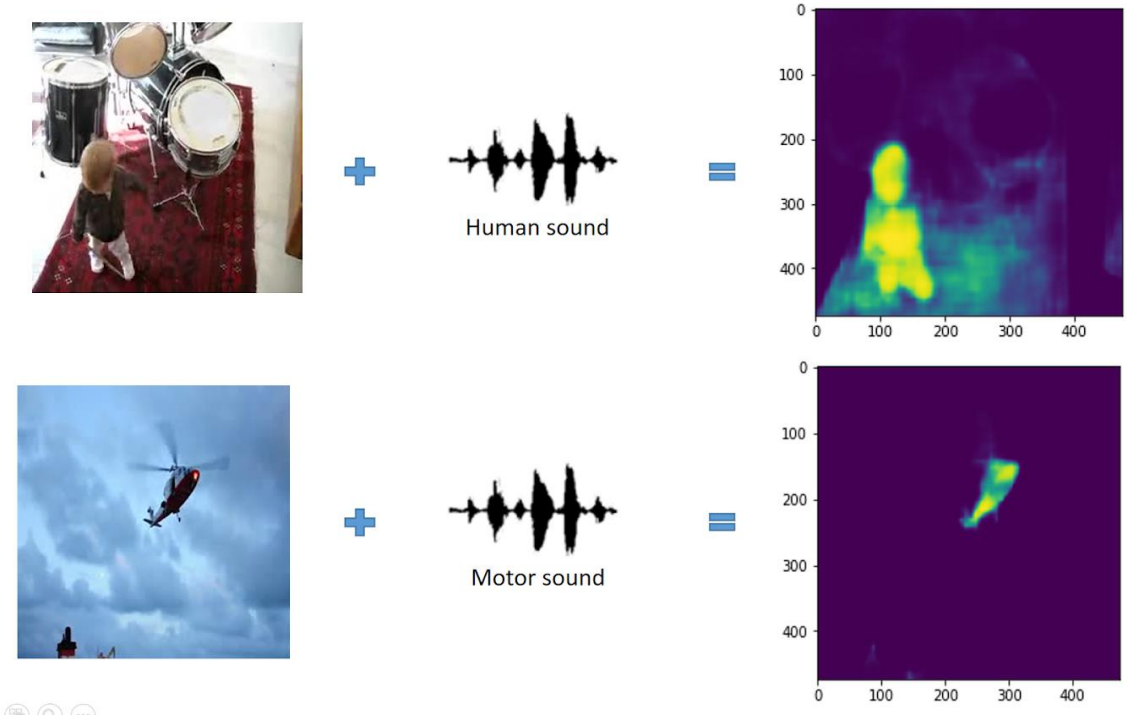
Experiments

Training and Dataset

The dataset used has 5000 samples of pairs of images and audio recordings. Bounding boxes for the samples were also available which were used as labels for the image-sound pair. Each image was pre-processed to resize it to the required dimensions (473 x 473). Audio features were extracted from the sound file using mfcc concept and a resultant 2-dimensional matrix of size 96 x 64 containing features was generated. The model was then trained on this dataset with batch size 15 for nearly 30 epochs. While feature extraction took nearly 6 hrs training took a significant time on a G.P.U. The results obtained are shared in next section.

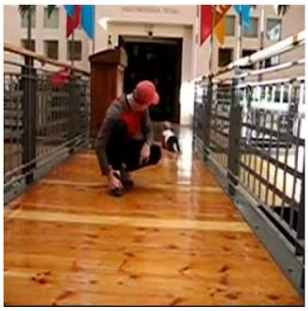
Results

Positive results



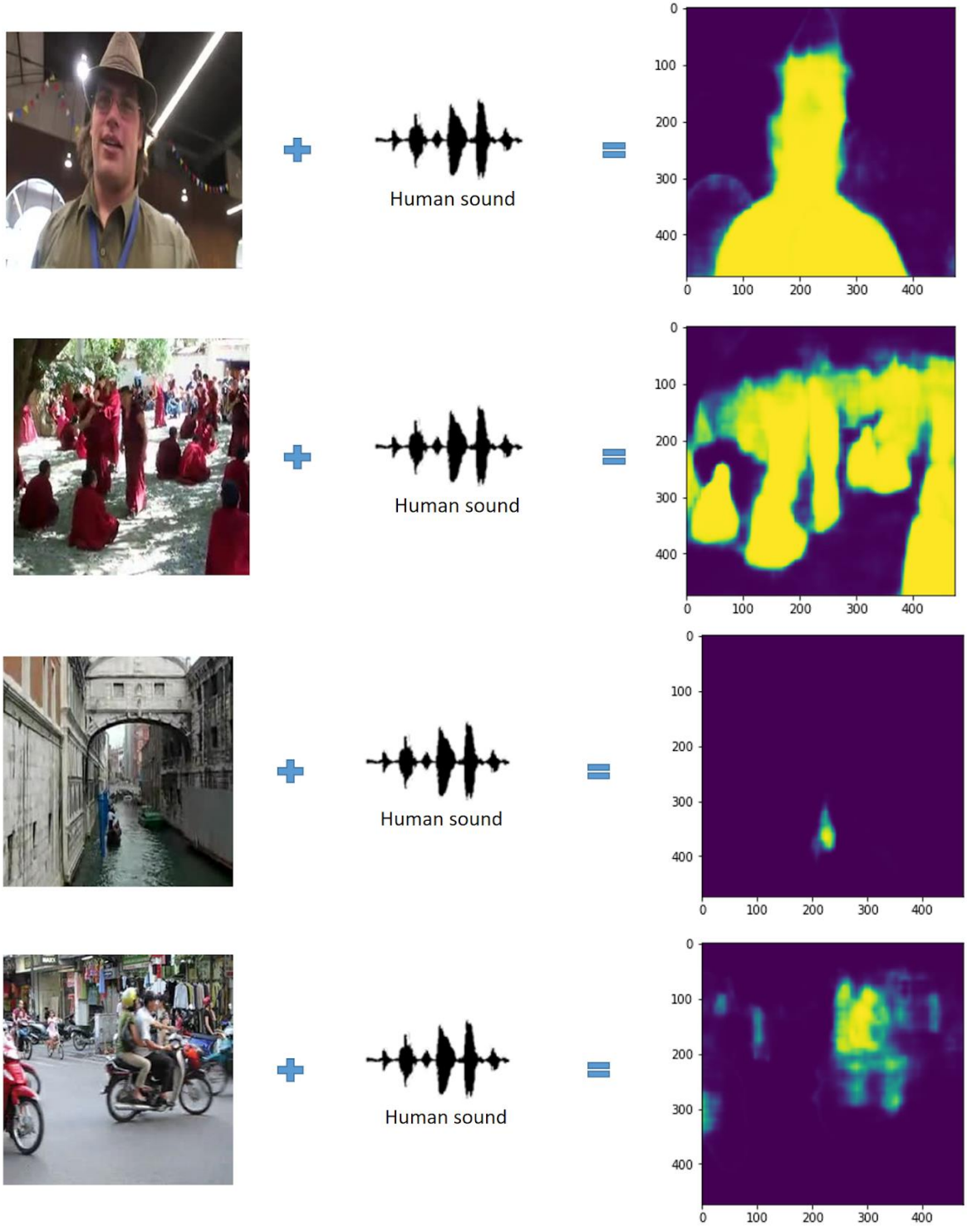
Human sound

Human sound



Human sound

Motor sound


Human sound
Human sound
Human sound
Human sound
0
100
200
300
400

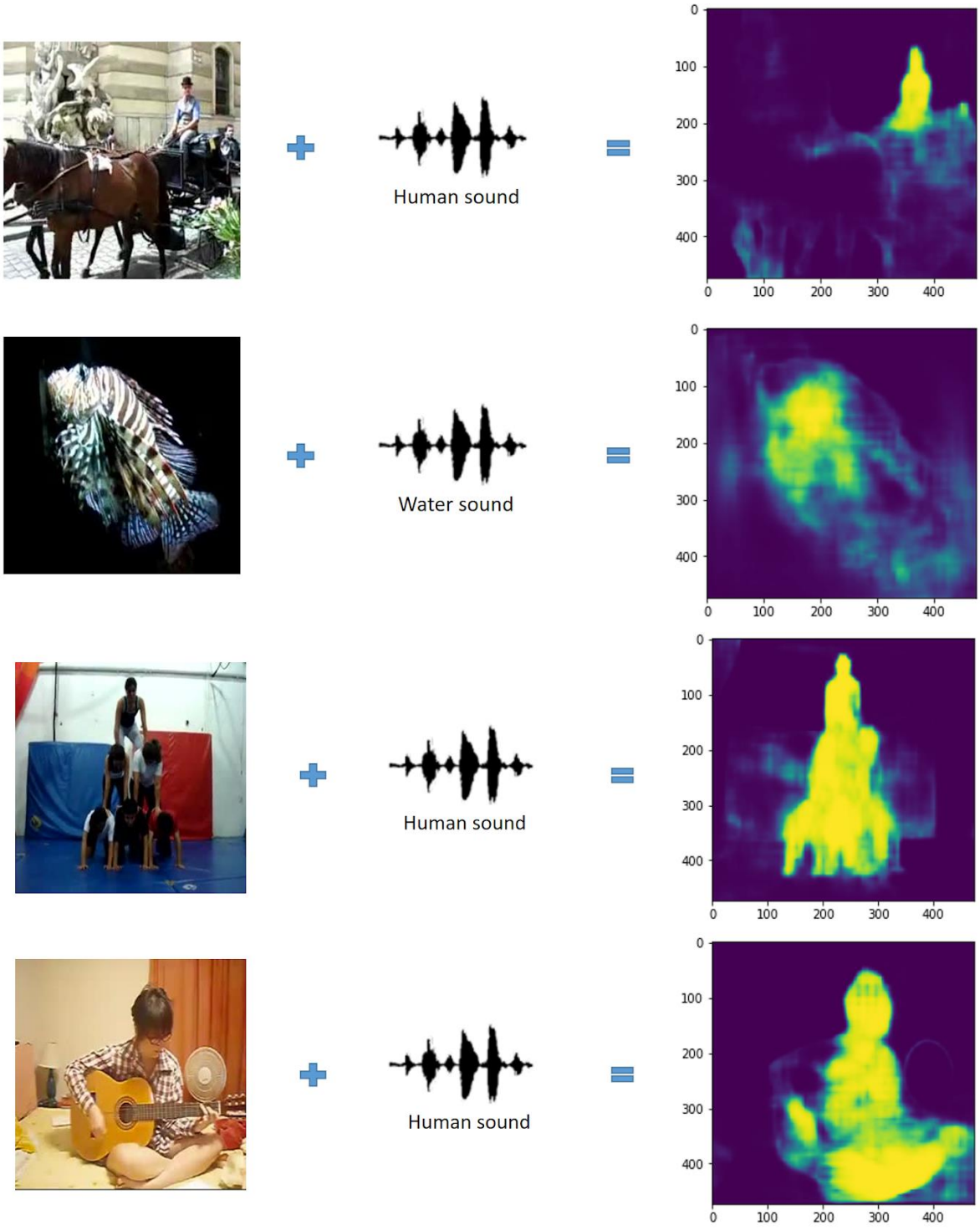


Figure 11 : Positive Results

Moderate results

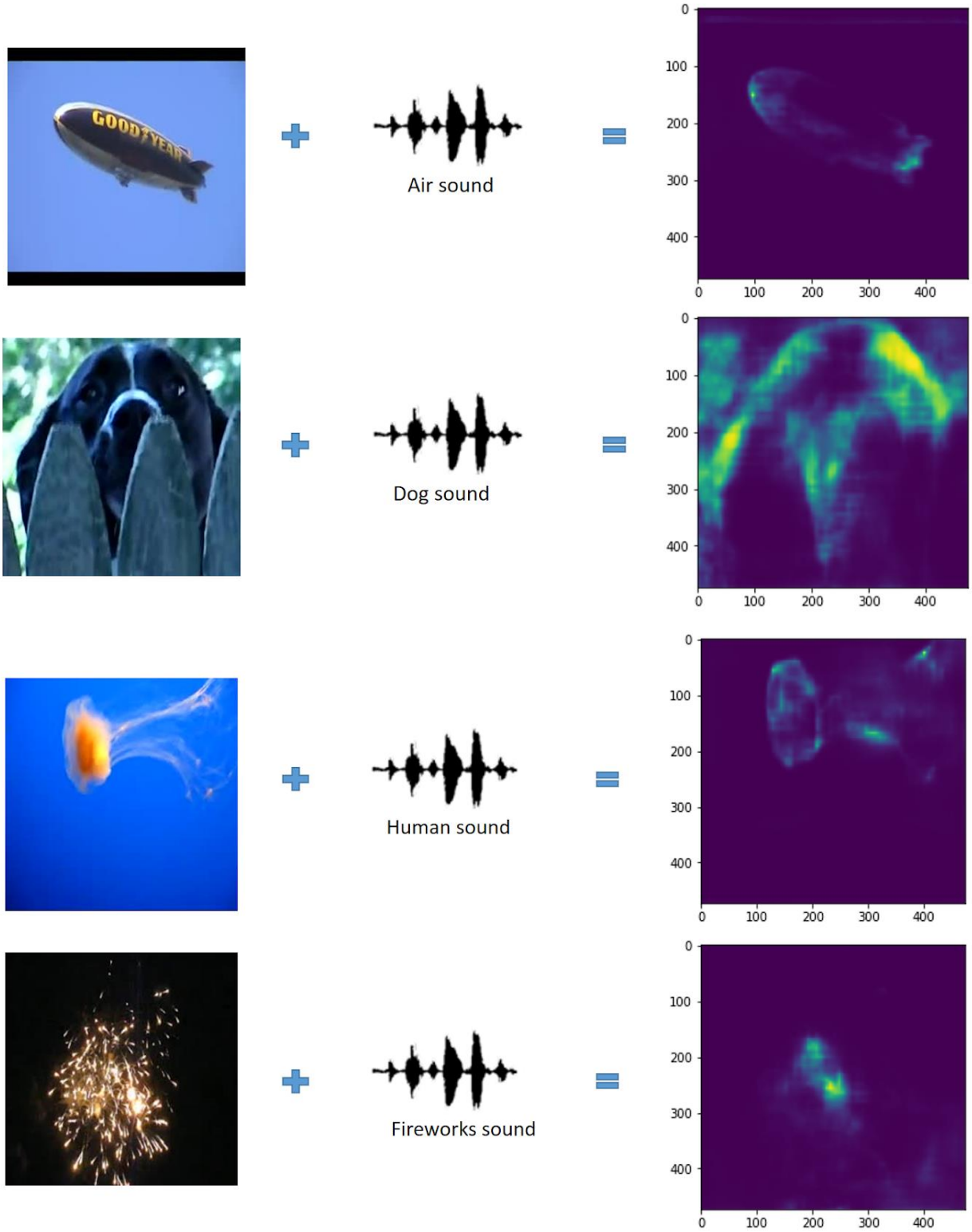


Figure 12 : Moderate Results

Negative results

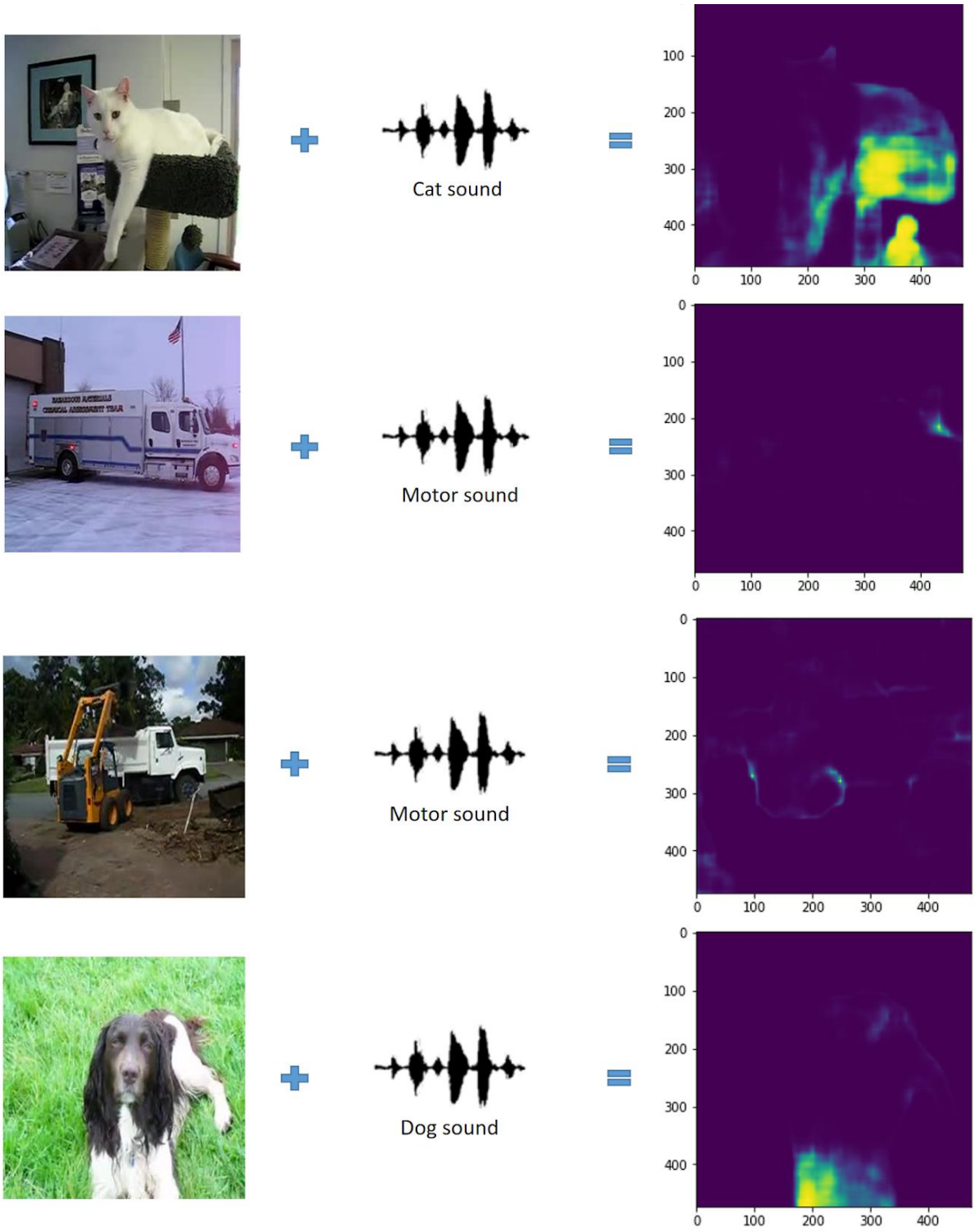


Figure 13 : Negative Results

Chapter 5

Future Scope

In this project a model has been developed to localize the sound source in a visual context. The visual network has a semantic segmentation pipeline (PSPnet) and the audio network has inhibited VGGish structure and has few other layers. The localization module uses the concept of dot product in finding out the correspondence between the audio and visual vectors. This model was trained on 5000 samples each having a pair of image and audio associated to it and a label containing the information about the bounding box of the sound source. Had the label contain more precise information like exact boundary of the source the model's performance can be enhanced.

The dataset majorly contained examples of human image-sound pairs which resulted in good results for such samples while others examples being less showed poor outcomes. By including good ratio of samples for all categories of pairs would improve the results.

These methods can also be used on video-sound pairs for the same task which can be used for real time sound source detection in any visual context.

References

- R. Arandjelovic and A. Zisserman. Look, listen and learn. In IEEE International Conference on Computer Vision, 2017.
- Relja Arandjelovi´c and Andrew Zisserman. Objects that Sound.
- Arda Senocak1 Tae-Hyun Oh, Junsik Kim, Ming-Hsuan Yang and In So Kweon. Learning to Localize Sound Source in Visual Scenes.
- B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In IEEE Conference on Computer Vision and Pattern Recognition, 2016.
- Y. Aytar, C. Vondrick, and A. Torralba. Soundnet: Learning sound representations from unlabeled video. In Neural Information Processing Systems, 2016.
- A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov, et al. Devise: A deep visual-semantic embedding model. In Neural Information Processing Systems