SUMMER TRAINING REPORT

Solid State Physics Laboratory(SSPL)

Topic – Convolution Neural Networks (CNN) for Detecting Signals using Hydrophone

SWRAJ KUMAR
NETAJI SUBHAS INSTITUTE OF DELHI
UNIVERSITY OF DELHI

Acknowledgement

I would like to thank Defense Research and Development Organization (DRDO) for providing necessary guidance and facility to undertake my research project.

Special thanks to team of researchers at hydrophone division at Solid State Physics Laboratory (SSPL). Scientist Sanjeev Verma has provided continued support and has contributed hugely towards the ideation and implementation of the project.

Last but not the least I will thank Dr Devendra Verma, Head, Hydrophone Division, SSPL, DRDO for his guidance and constant support. I will like to make laboratory assistants at hydrophone division for helping me to conduct experiments with underwater sensors. DRDO provided conducive environment for carrying out my research. It has been truly a wonderful experience to carry out research work at SSPL, DRDO.

Index

Introduction

Detection of signals collected from hydrophone is a difficult task. Signal need to be processed using techniques like Dynamic Time Wrapping (DTW) and Fourier transform before something meaningful can be extracted from the data. Further methods such as Mel Frequency Cepstral Coefficient (MFCC) are calculated for the identifying the signal. But MFCC doesn’t represent the signal in its entirety. But with the advancement in Deep Learning it is possible for us to detect signal more accurately. Clustering algorithms such as K means clustering and Gaussian Mixture Models has been used for speaker identification. But more recently Recurrent Neural networks (RNN) have also show significant improvement in results. But due to much greater computational requirements of RNN, they cannot be scaled to make a general model for identifying signals such an Inception or VCG type networks which are based on Convolutional neural networks (CNN). The project uses CNN to make a whale call up detection system. Due to restricted use of sensitive naval data, I have to rely on bio acoustic data such as whale data set to create same environment setting. But the same idea can be incorporated for naval usage such as in submarine and Autonomous Underwater Vehicles (AUV). The data set doesn’t contain cases where there is echo effects involved as the data was collected in deep Atlantic Ocean. Scaling the model to a production level system will require a better dataset which can’t be arranged through open source.

Hydrophone

A Hydrophone is a microphone designed to be used underwater for recording or listening to underwater sound. Most hydrophones are based on a piezoelectric transducer that generates electricity when subjected to a pressure change. Such piezoelectric materials, or transducers, can convert a sound signal into an electrical signal since sound is a pressure wave. Some transducers can also serve as a projector, but not all have this capability, and some may be destroyed if used in such a manner.

A hydrophone can "listen" to sound in air but will be less sensitive due to its design as having a good acoustic impedance match to water, which is a denser fluid than air. Likewise, a microphone can be buried in the ground, or immersed in water if it is put in a waterproof container, but will give similarly poor performance due to the similarly bad acoustic impedance match.

Types of Hydrophone

1) Omnidirectional Hydrophone - records signals from all direction with equal sensitivity. BII- 700 is an omnidirectional hydrophone used in Autonomous Underwater Vehicles (AUV) and submarines.

2) Directional Hydrophone – Higher sensitivity to signals from a particular direction. Focused hydrophone can detect signal in a particular direction in which it is held. Array Hydrophone is also another example of this category which is just a combination of array of hydrophones in a line formation.

Problem Statement

Project uses Convolutional Neural networks for identifying whale up call sounds. Due to limited availability of naval hydrophone data, bio acoustic data has been taken to replicate the environment though the conceptualization will remain the same if the research idea is extrapolated to defense. Navy usage.

The data set used in this project is open source and can be downloaded from Kaggle website page hosting Cornell University’s data set. Cornell University's Bioacoustics Research Program has extensive experience in identifying endangered whale species and has deployed a 24/7 buoy network to guide ships from colliding with the world's last 400 North Atlantic right whales.

Right whales make a half-dozen types of sounds, but the characteristic up-call is the one identified by the auto-detection buoys. The up-call is useful because it’s distinctive and right whales give it often. A type of “contact call,” the up-call is a little like small talk--the sound of a right whale going about its day and letting others know it’s nearby. In this recording, the up-call is easy to hear--a deep, rising “whoop” that lasts about a second: Right whale up-call.

Training and testing data are in the form of AIFF files.

Previous State of the Art

During enrollment, speech from a speaker is passed through the front-end processing steps described above and the feature vectors are used to create a speaker model. Desirable attributes of a speaker model are:
(1) a theoretical underpinning so one can understand model behavior and mathematically approach extensions and improvements.
(2) generalizable to new data so that the model does not over fit the enrollment data and can match new data.
(3) parsimonious representation in both size and computation.

There are many modeling techniques that have some or all of these attributes and have been used in speaker verification systems. The selection of modeling is largely dependent on the type of speech to be used, the expected performance, the ease of training and updating, and storage and computation considerations. A brief description of some of the more prevalent modeling techniques is given next.

Template Matching

In this technique, the model consists of a template that is a sequence of feature vectors from a fixed phrase. During verification, a match score is produced by using dynamic time warping (DTW) to align and measure the similarity between the test phrase and the speaker template. This approach is used almost exclusively for text-dependent applications.

Nearest Neighbor

In this technique, no explicit model is used; instead all features vectors from the enrollment speech are retained to represent the speaker. During verification, the match score is computed as the cumulated distance of each test feature vector to its k nearest neighbors in the speaker's training vectors. To limit storage and computation, feature vector pruning techniques are usually applied.

Neural Networks

The particular model used in this technique can have many forms, such as multi-layer perceptions or radial basis functions. The main difference with the other approaches described is that these models are explicitly trained to discriminate between the speaker being modeled and some alternative speakers. Training can be computationally expensive and models are sometimes not generalizable.

Hidden Markov Models

This technique uses HMMs, which encode the temporal evolution of the features and efficiently model statistical variation of the features, to provide a statistical representation of how a speaker produces sounds. During enrollment, HMM parameters are estimated from the speech using established automatic algorithms. During verification, the likelihood of the test feature sequence is computed against the speaker's HMMs. For text-dependent applications, whole phrases or phonemes may be modeled using multi-state left-to-right HMMs. For text-independent applications, single state HMMs, also known as Gaussian Mixture Models (GMMs), are used. From published results, HMM based systems generally produce the best performance

Support Vector Machine (SVM)

Support vector machines (SVMs) have proven to be a powerful technique for pattern classification. SVMs map inputs into a high-dimensional space and then separate classes with a hyperplane. A critical aspect of using SVMs successfully is the design of the inner product, the kernel, induced by the high dimensional mapping. We consider the application of SVMs to speaker and language recognition. A key part of our approach is the use of a kernel that compares sequences of feature vectors and produces a measure of similarity.

Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

Historical Background

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras.

Many important advances have been boosted by the use of inexpensive computer emulations. Following an initial period of enthusiasm, the field survived a period of frustration and disrepute. During this period when funding and professional support was minimal, important advances were made by relatively few researchers. These pioneers were able to develop convincing technology which surpassed the limitations identified by Minsky and Papert. Minsky and Papert, published a book (in 1969) in which they summed up a general feeling of frustration (against neural networks) among researchers, and was thus accepted by most without further analysis. Currently, the neural network field enjoys a resurgence of interest and a corresponding increase in funding.

The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits. But the technology available at that time did not allow them to do too much.

Feed Forward Networks

Feed-forward networks have the following characteristics:

1. Perceptrons are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world, and hence are called hidden layers.
2. Each perceptron in one layer is connected to every perceptron on the next layer. Hence information is constantly "fed forward" from one layer to the next., and this explains why these networks are called feed-forward networks.
3. There is no connection among perceptrons in the same layer.

How these networks classify?

A single perceptron can classify points into two regions that are linearly separable. Now let us extend the discussion into the separation of points into two regions that are not linearly separable. Consider the following network:

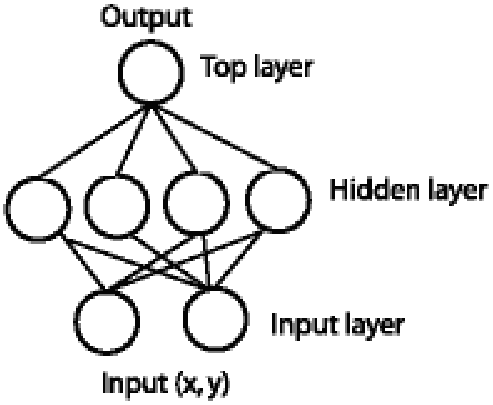


Fig : A feed-forward network with one hidden layer.

The same (x, y) is fed into the network through the perceptrons in the input layer. With four perceptrons that are independent of each other in the hidden layer, the point is classified into 4 pairs of linearly separable regions, each of which has a unique line separating the region.

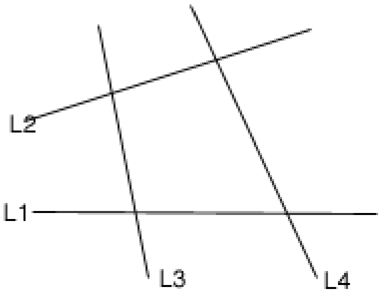


Fig : 4 lines each dividing the plane into 2 linearly separable regions.

The top perceptron performs logical operations on the outputs of the hidden layers so that the whole network classifies input points in 2 regions that might not be linearly separable. For instance, using the AND operator on these four outputs, one gets the intersection of the 4 regions that forms the center region.

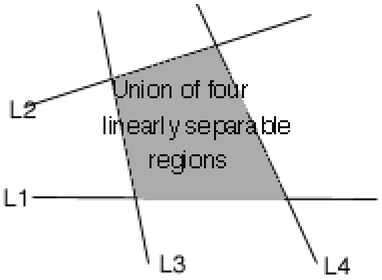


Fig : Intersection of 4 linearly separable regions forms the center region.

By varying the number of nodes in the hidden layer, the number of layers, and the number of input and output nodes, one can classification of points in arbitrary dimension into an arbitrary number of groups. Hence feed-forward networks are commonly used for classification.

Backpropagation -- learning in feed-forward networks:

Learning in feed-forward networks belongs to the realm of supervised learning, in which pairs of input and output values are fed into the network for many cycles, so that the network 'learns' the relationship between the input and output.

We provide the network with a number of training samples, which consists of an input vector i and its desired output o. For instance, in the classification problem, suppose we have points (1, 2) and (1, 3) belonging to group 0, points (2, 3) and (3, 4) belonging to group 1, (5, 6) and (6, 7) belonging to group 2, then for a feed-forward network with 2 input nodes and 2 output nodes, the training set would be:

{ i = (1, 2) , o =(0, 0)
i = (1, 3) , o = (0, 0)
i = (2, 3) , o = (1, 0)
i = (3, 4) , o = (1, 0)
i = (5, 6) , o = (0, 1)
i = (6, 7) , o = (0, 1) }

The basic rule for choosing the number of output nodes depends on the number of different regions. It is advisable to use a unary notation to represent the different regions, i.e. for each output only one node can have value 1. Hence the number of output nodes = number of different regions -1.

In backpropagation learning, every time an input vector of a training sample is presented, the output vector o is compared to the desired value d.

The comparison is done by calculating the squared difference of the two:

$$\mathbf{Err = (d\text{-}o)^2}$$

The value of Err tells us how far away we are from the desired value for a particular input. The goal of backpropagation is to minimize the sum of Err for all the training samples, so that the network behaves in the most "desirable" way.

$$\text{Minimize } \sum \mathbf{Err = (d\text{-}o)^{2}}$$

We can express Err in terms of the input vector (i), the weight vectors (w), and the threshold function of the perceptions. Using a continuous function (instead of the step function) as the threshold function, we can express the gradient of Err with respect to the w in terms of w and i.

Given the fact that decreasing the value of w in the direction of the gradient leads to the most rapid decrease in Err, we update the weight vectors every time a sample is presented using the following formula:

$$w_{new} = w_{old} - n \frac{\delta \, Err}{\delta \, w}$$ where n is the learning rate
(a small number ~ 0.1)

Using this algorithm, the weight vectors are modified so that the value of Err for a particular input sample decreases a little bit every time the sample is presented. When all the samples are presented in turns for many cycles, the sum of Err gradually decreases to a minimum value, which is our goal as mentioned above.

Convolutional Neural Networks (CNN)

Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representations.

Sparse Interaction is accomplished by making the kernel smaller than the input. For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency.

What inspired Convolutional Networks?

CNNs are biologically-inspired models inspired by research by D. H. Hubel and T. N. Wiesel. They proposed an explanation for the way in which mammals visually perceive the world around them using a layered architecture of neurons in the brain, and this in turn inspired engineers to attempt to develop similar pattern recognition mechanisms in computer vision.

In their hypothesis, within the visual cortex, complex functional responses generated by "complex cells" are constructed from more simplistic responses from "simple cells'.

For instances, simple cells would respond to oriented edges etc, while complex cells will also respond to oriented edges but with a degree of spatial invariance.

Receptive fields exist for cells, where a cell responds to a summation of inputs from other local cells.

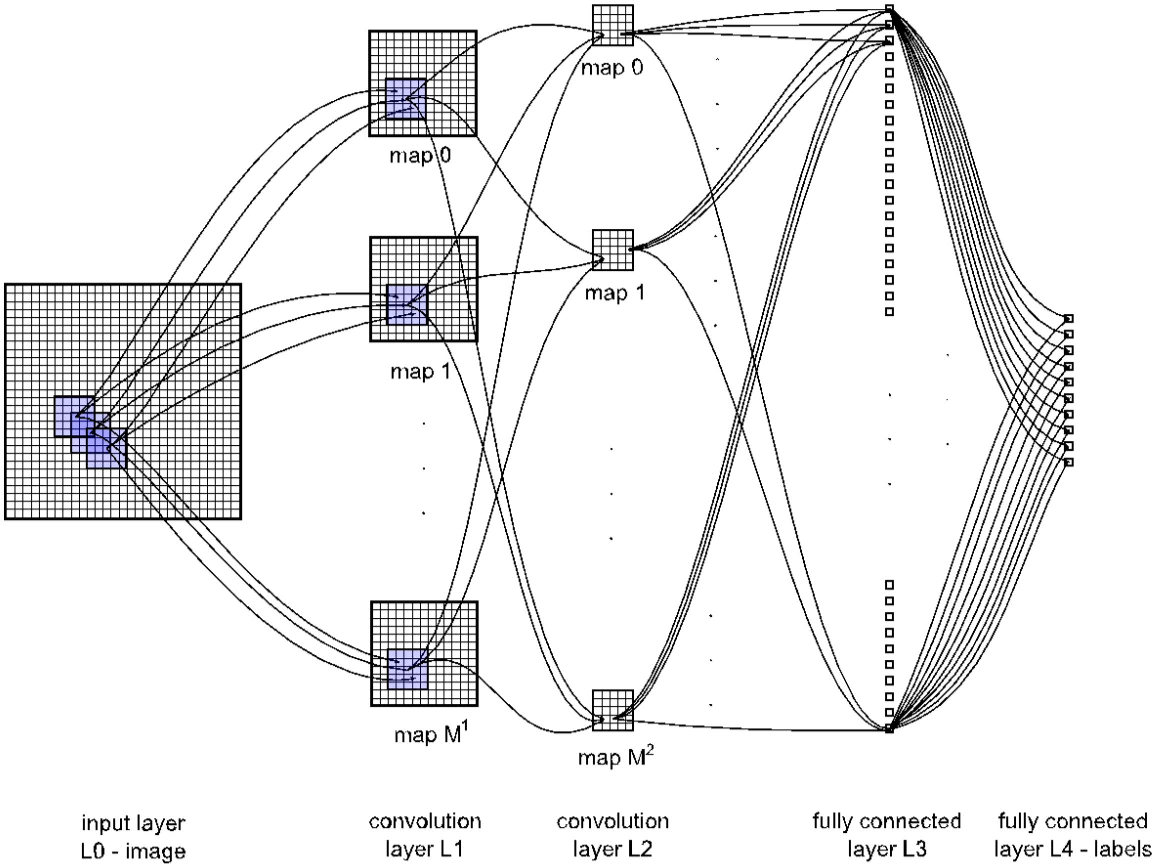
The architecture of deep convolutional neural networks was inspired by the ideas mentioned above

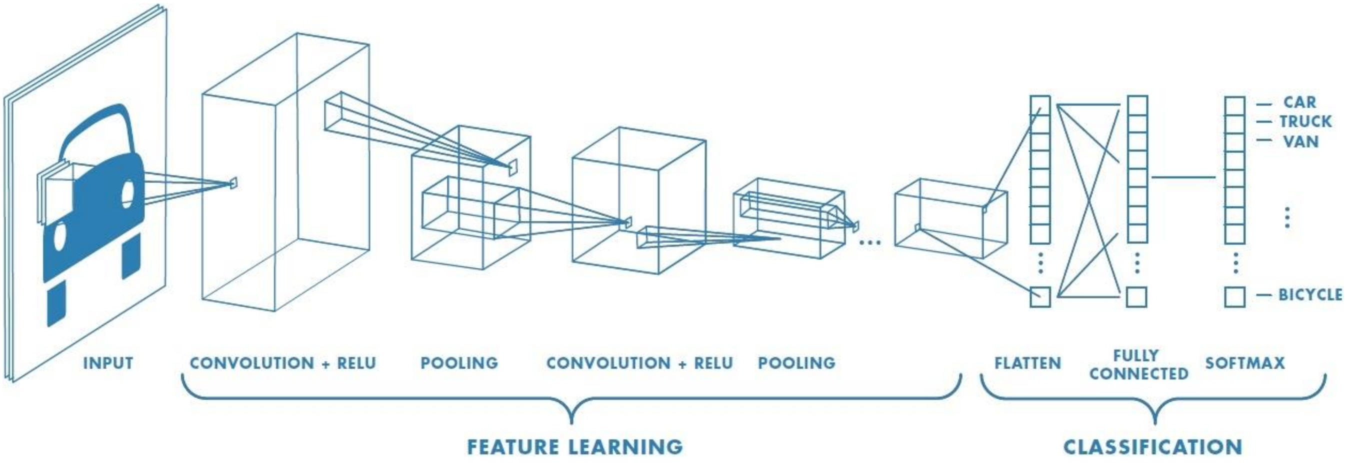
- local connections
- layering
- spatial invariance (shifting the input signal results in an equally shifted output signal. , most of us are able to recognize specific faces under a variety of conditions because we learn abstraction These abstractions are thus invariant to size, contrast, rotation, orientation

However, it remains to be seen if these computational mechanisms of convolutional neural networks are similar to the computation mechanisms occurring in the primate visual system

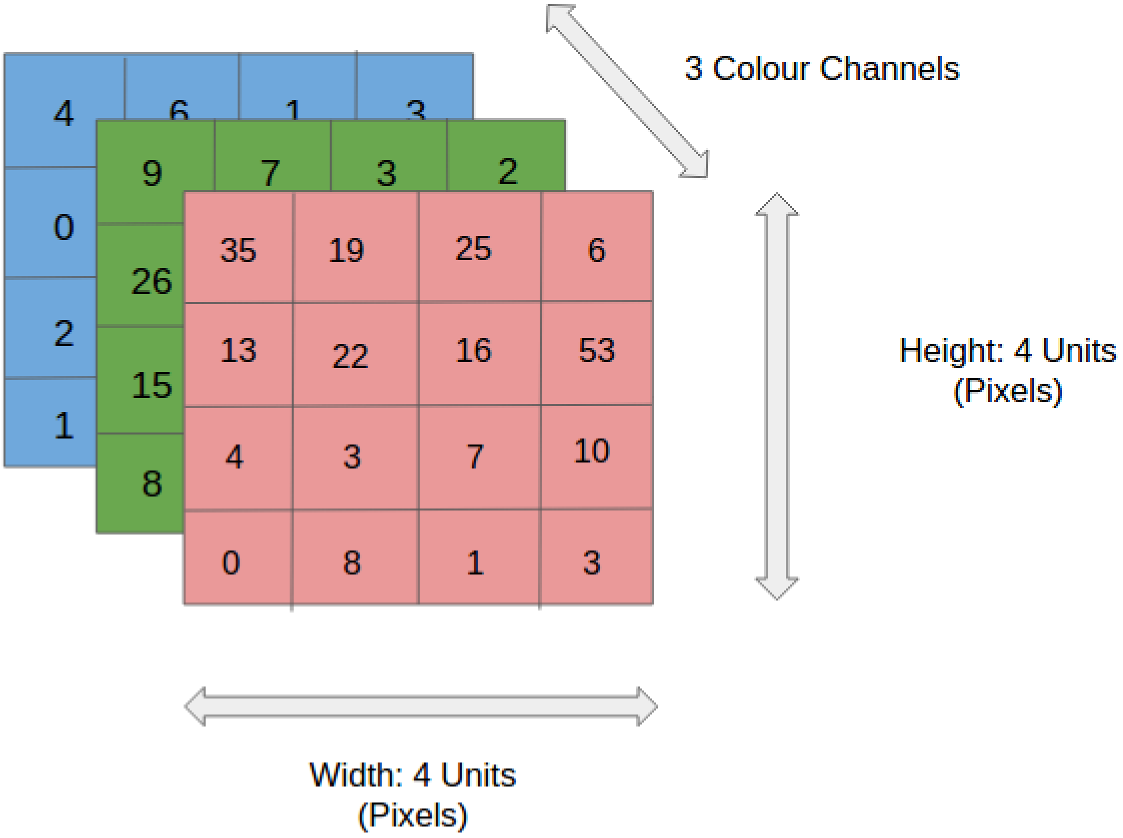
- convolution operation
- shared weights
- pooling/subsampling

How does it work?



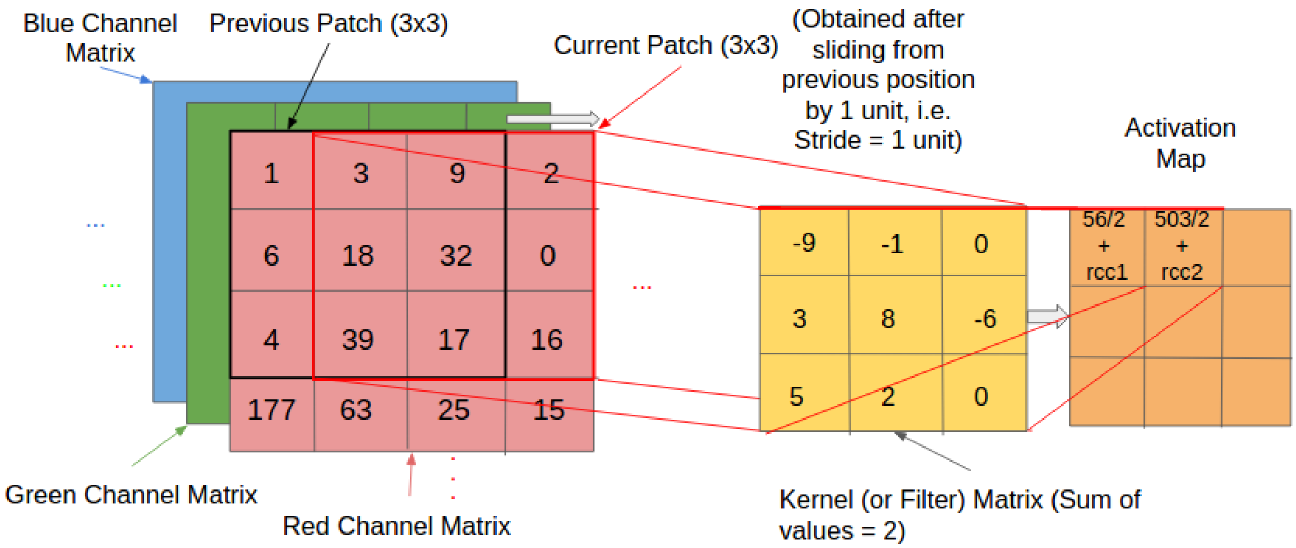


Step 1 - Prepare a dataset of images

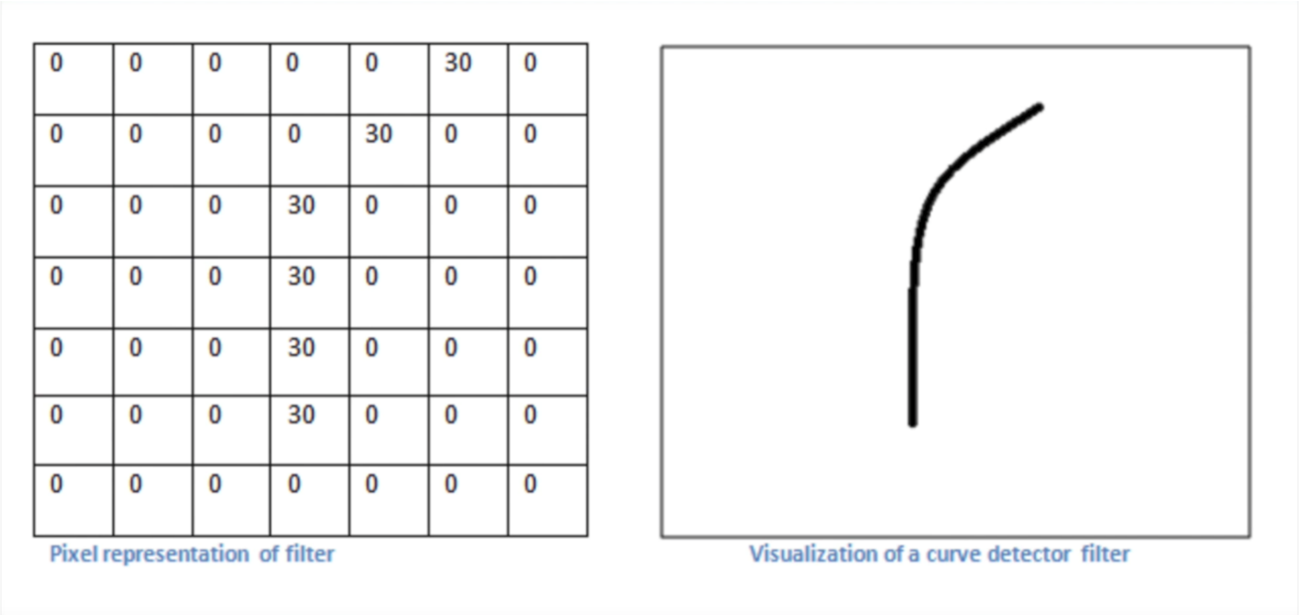


- Every image is a matrix of pixel values.
- The range of values that can be encoded in each pixel depends upon its bit size.
- Most commonly, we have 8 bit or 1 Byte-sized pixels. Thus the possible range of values a single pixel can represent is [0, 255].
- However, with colored images, particularly RGB (Red, Green, Blue)-based images, the presence of separate color channels (3 in the case of RGB images) introduces an additional ‘depth’ field to the data, making the input 3-dimensional.
- Hence, for a given RGB image of size, say 255×255 (Width x Height) pixels, we’ll have 3 matrices associated with each image, one for each of the color channels.
- Thus the image in it’s entirety, constitutes a 3-dimensional structure called the Input Volume (255x255x3).

Step 2 - Convolution



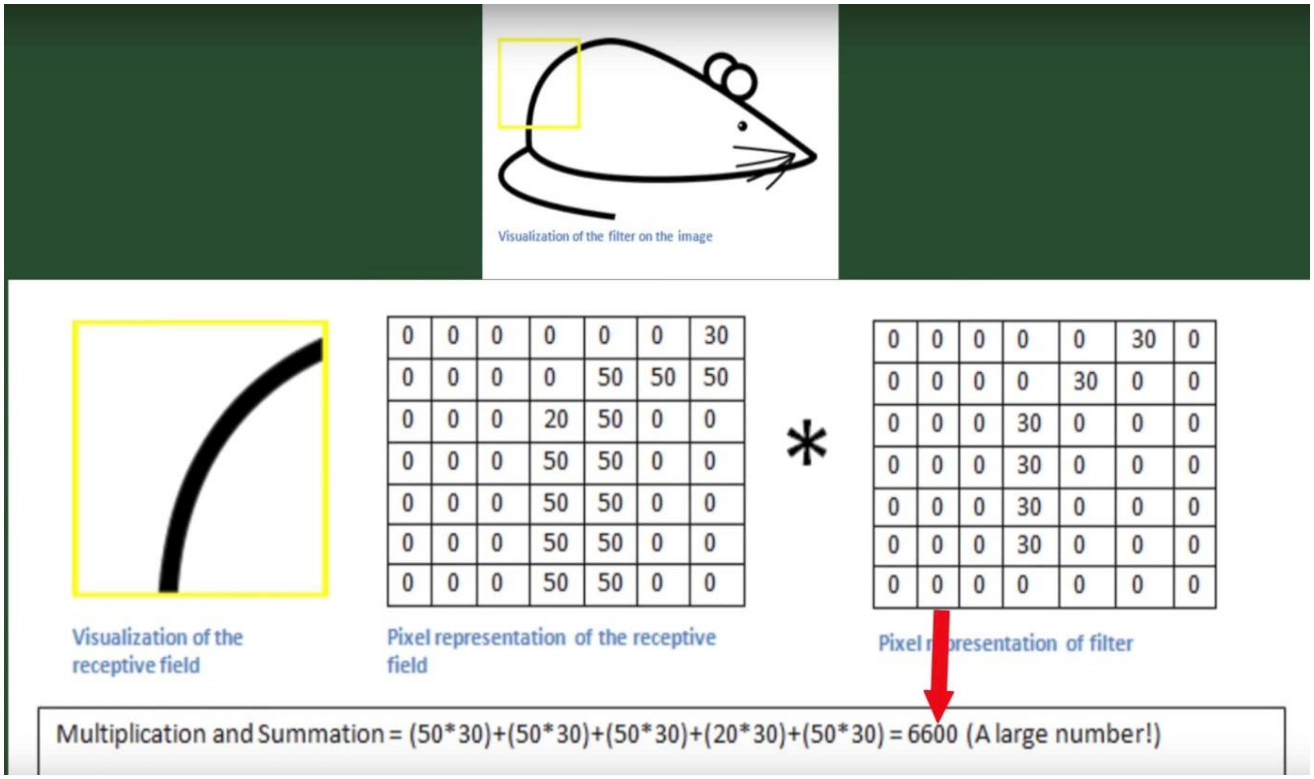
- A convolution is an orderly procedure where two sources of information are intertwined.
- A kernel (also called a filter) is a smaller-sized matrix in comparison to the input dimensions of the image, that consists of real valued entries.
- Kernels are then convolved with the input volume to obtain so-called ‘activation maps’ (also called feature maps).
- Activation maps indicate ‘activated’ regions, i.e. regions where features specific to the kernel have been detected in the input.
- The real values of the kernel matrix change with each learning iteration over the training set, indicating that the network is learning to identify which regions are of significance for extracting features from the data.
- We compute the dot product between the kernel and the input matrix. -The convolved value obtained by summing the resultant terms from the dot product forms a single entry in the activation matrix.
- The patch selection is then slided (towards the right, or downwards when the boundary of the matrix is reached) by a certain amount called the ‘stride’ value, and the process is repeated till the entire input image has been processed. - The process is carried out for all color channels.
- instead of connecting each neuron to all possible pixels, we specify a 2 dimensional region called the ‘receptive field[14]’ (say of size 5×5 units) extending to the entire depth of the input (5x5x3 for a 3 color channel input), within which the encompassed pixels are fully connected to the neural network’s input layer. It’s over these small regions that the network layer cross-sections (each consisting of several neurons (called ‘depth columns’)) operate and produce the activation map. (reduces computational complexity)



0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Visualization of a curve detector filter



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

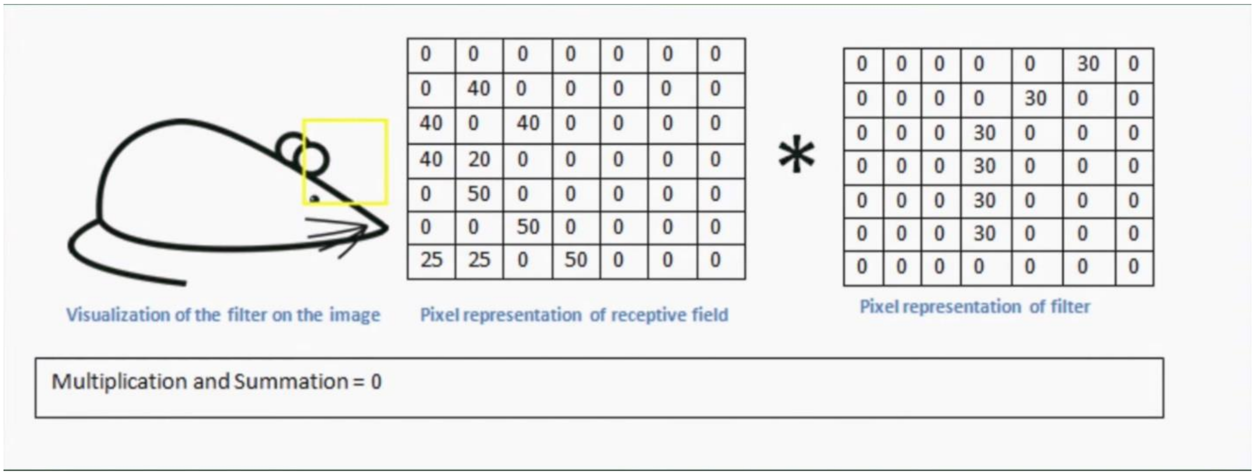
Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)



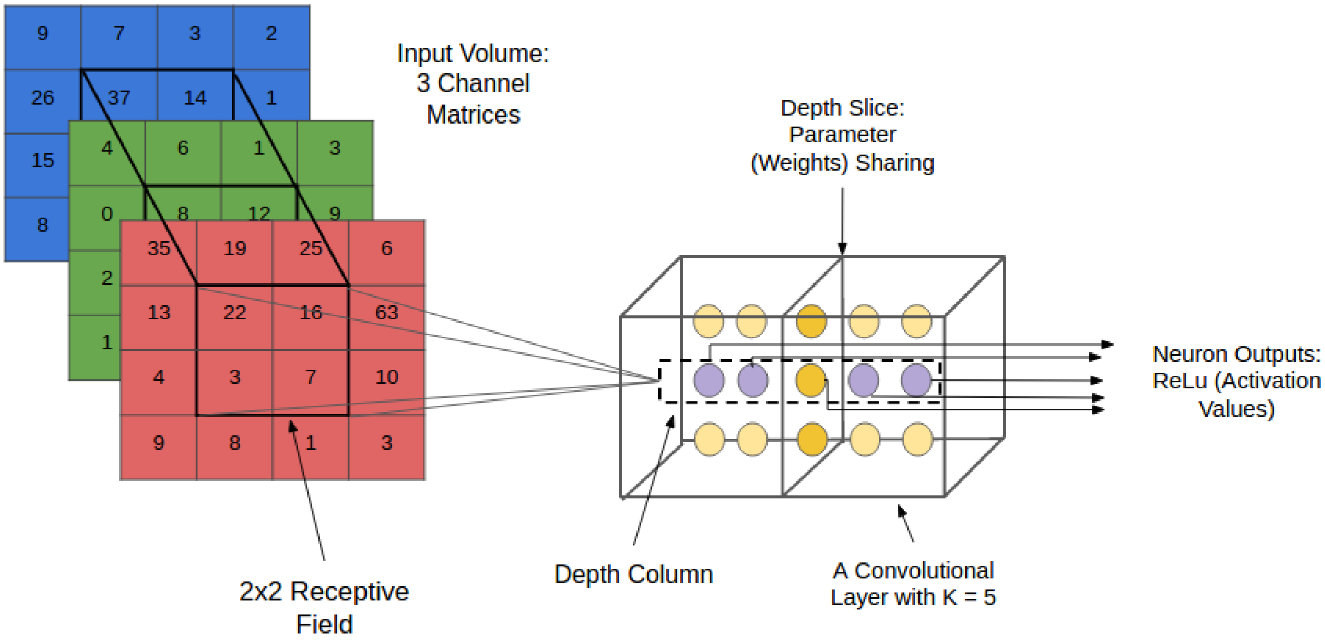
0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

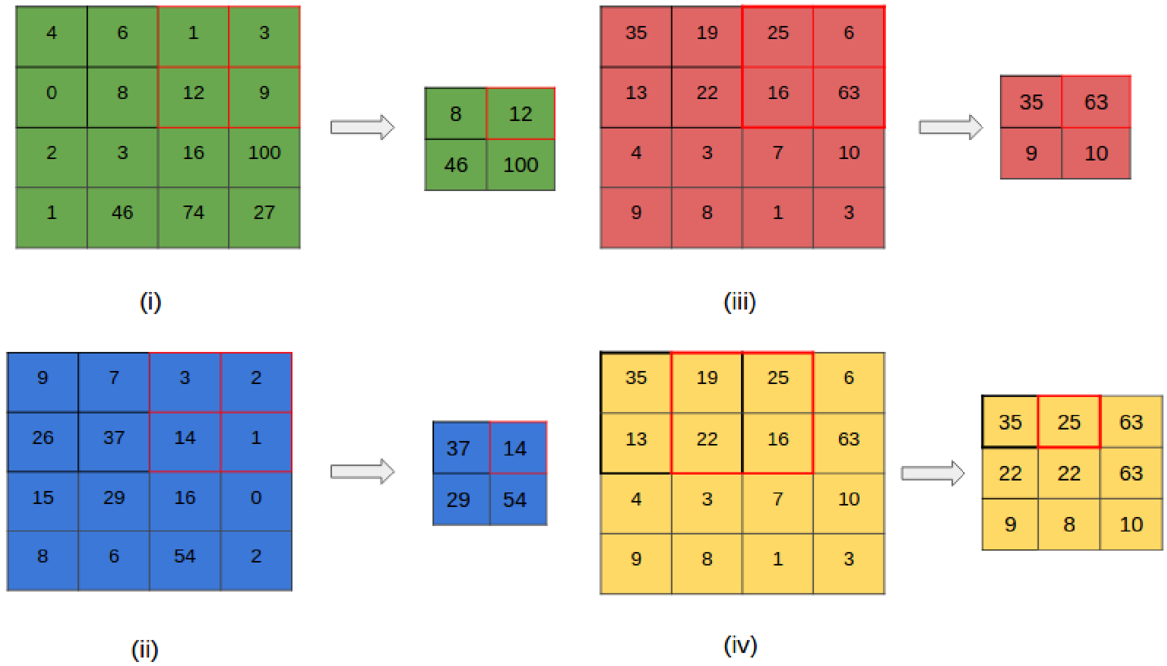
Visualization of the filter on the image
Pixel representation of receptive field
Pixel representation of filter

Multiplication and Summation = 0



- When the sliding window is over the area where the values are non-zero then the dot product of the matrix representing the window and the image portion will be a large number but if the sliding window if over the area of the image where it didn't learn a feature then the dot product will be zero or a smaller number compared with the previous value obtained.

Step 3 – Pooling



4	6	1	3
0	8	12	9
2	3	16	100
1	46	74	27

8	12
46	100

(i)

35	19	25	6
13	22	16	63
4	3	7	10
9	8	1	3

35	63
9	10

(iii)

9	7	3	2
26	37	14	1
15	29	16	0
8	6	54	2

37	14
29	54

(ii)

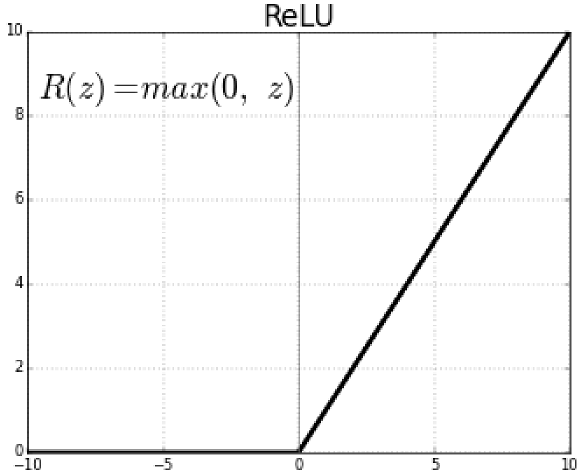
35	19	25	6
13	22	16	63
4	3	7	10
9	8	1	3

35	25	63
22	22	63
9	8	10

(iv)

- Pooling reducing the spatial dimensions (Width x Height) of the Input Volume for the next Convolutional Layer. It does not affect the depth dimension of the Volume.
- The transformation is either performed by taking the maximum value from the values observable in the window (called 'max pooling'), or by taking the average of the values. Max pooling has been favoured over others due to its better performance characteristics.
- also called downsampling

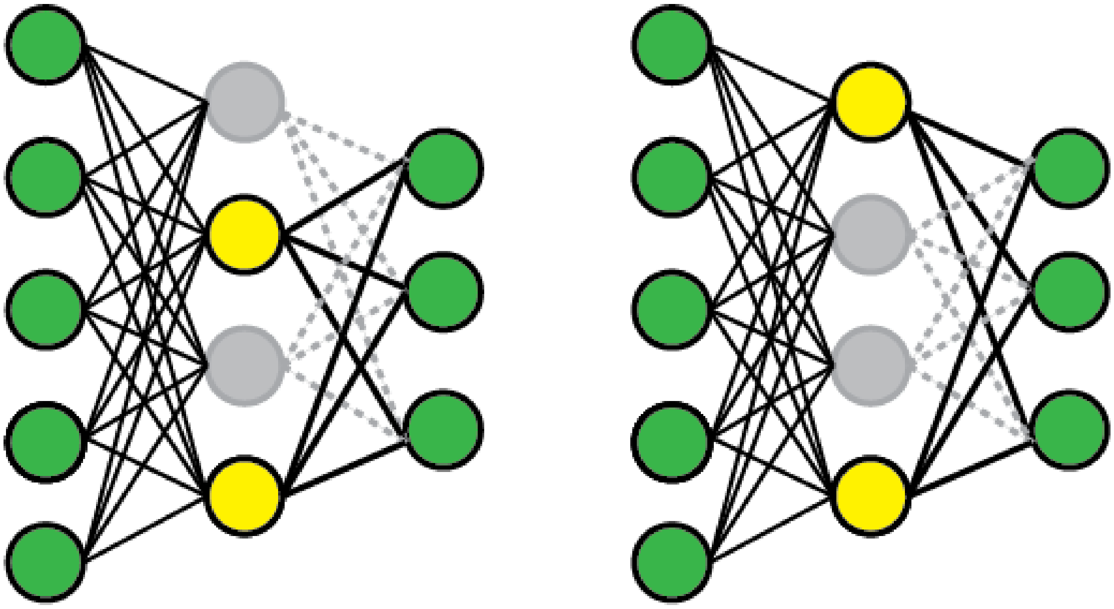
Step 4 - Normalization (ReLU in our case)



Normalization (keep the math from breaking by turning all negative numbers to 0) (Rectifier Linear Unit or ReLU) a stack of images becomes a stack of images with no negative values.

Repeat Steps 2-4 several times. More, smaller images (feature maps created at every layer)

Step 5 – Regularization



- Dropout forces an artificial neural network to learn multiple independent representations of the same data by alternately randomly disabling neurons in the learning phase.
- Dropout is a vital feature in almost every state-of-the-art neural network implementation.
- To perform dropout on a layer, you randomly set some of the layer's values to 0 during forward propagation.

Step 6 - Probability Conversion

At the very end of our network (the tail), apply a softmax function to convert the outputs to probability values for each class.

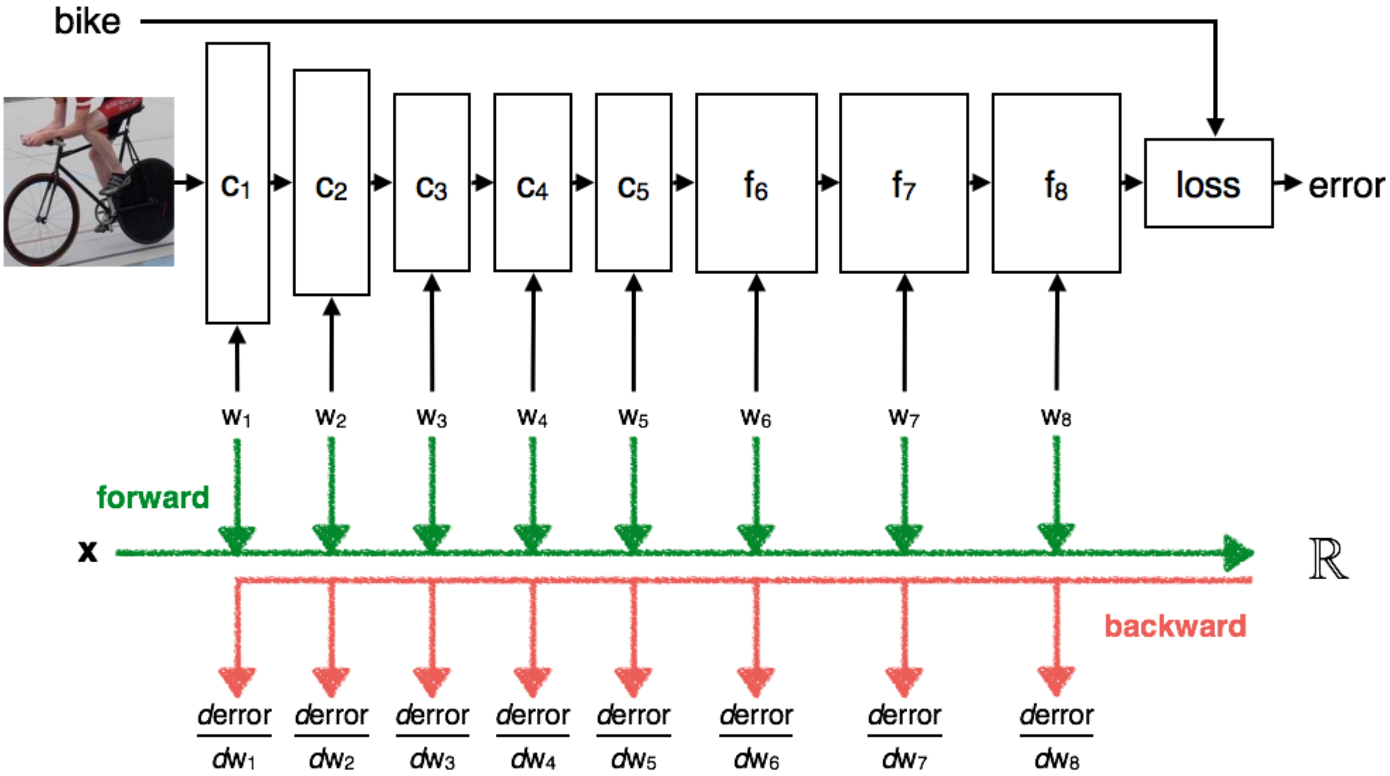
$$\boxed{\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}}$$

Step 7 - Choose most likely label (max probability value)

Equation : argmax(Softmax outputs)

Learning weights

- We can learn features and weight values through backpropagation

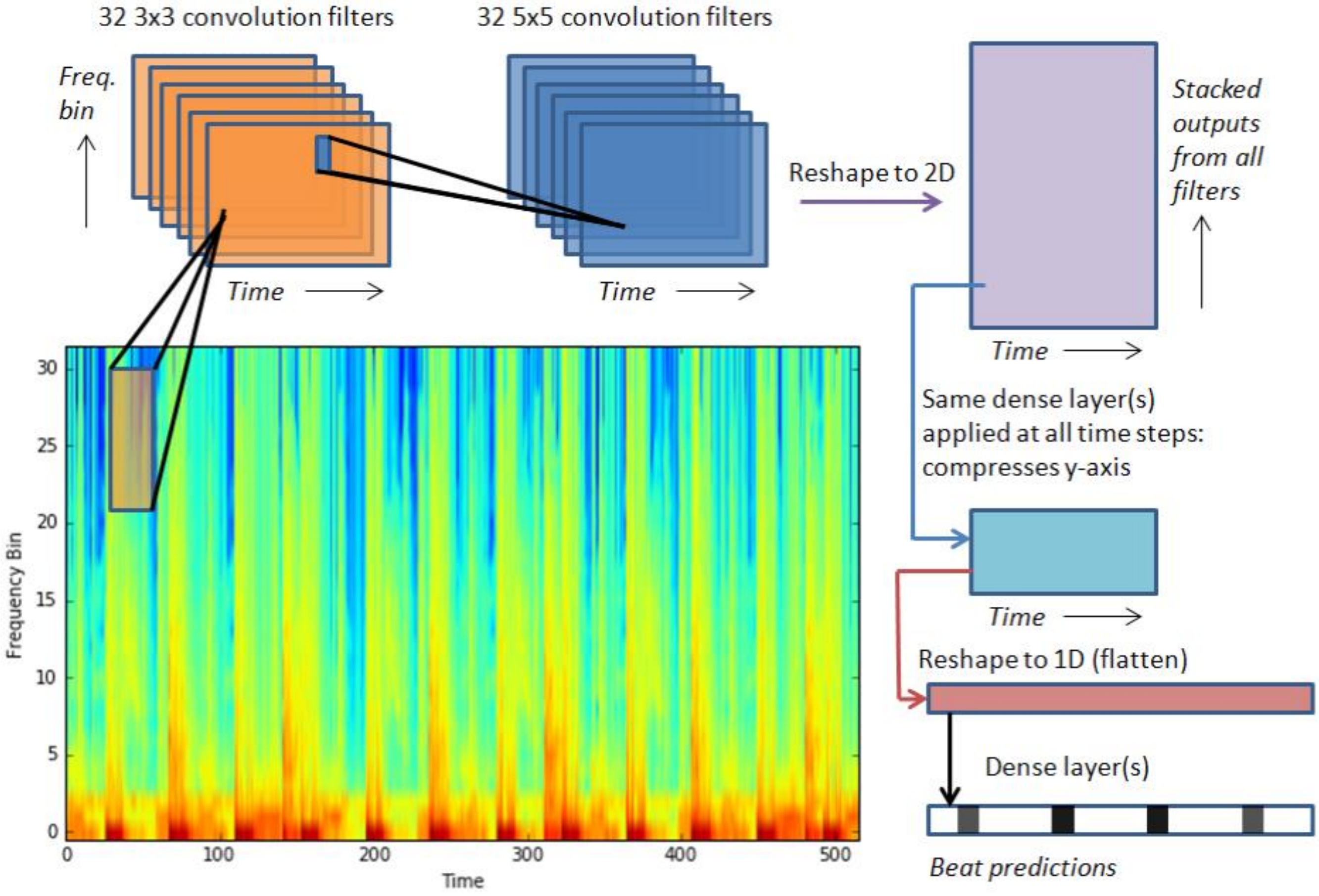


The other hyperparameters are set by humans and they are an active field of research (finding the optimal ones)

i.e - number of neurons, number of features, size of features, poooling window size, window stride

When is a good time to use it?

- To classify images
- To generate images

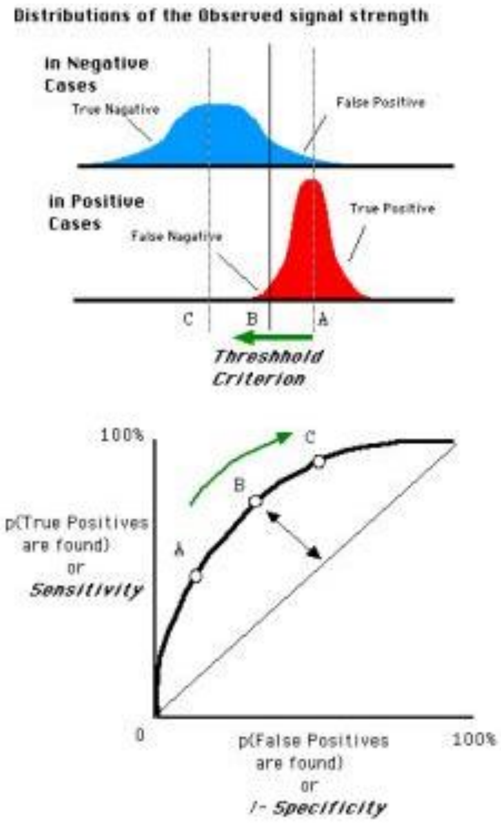

32 3x3 convolution filters
32 5x5 convolution filters
Freq. bin
Time
Time
Reshape to 2D
Stacked outputs from all filters
Time
Same dense layer(s) applied at all time steps: compresses y-axis
Time
Reshape to 1D (flatten)
Dense layer(s)
Beat predictions
Frequency Bin
0
5
10
15
20
25
30
0
100
200
300
400
500
Time

But can also be applied to any spatial 2D or 3D data. Images. Even sound and text. A rule of thumb is if data is just as useful if you swap out the rows and columns, like customer data, then you can't use a CNN.

Metrics

The main evaluation metric for this project will be that used in the Kaggle competition, this is the Area Under the Curve (AUC), where the Curve is the ROC curve.

The receiver operating characteristic (ROC) curve is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection. The false-positive rate is also known as the fall-out or probability of false alarm. The ROC curve is thus, the sensitivity as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from −∞ to the discrimination threshold) of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability in x-axis. Other interesting tool can be the confusion matrix, which is a more detailed version of the ROC curve. The confusion matrix is a table that shows the predicted labels for each of the true input labels. Hence, this table shoes true positives (TPs), false positives (FPs), true negatives (TNs) and false negatives (FNs). Each prediction result or instance of the confusion matrix represents a point in the ROC space.


Distributions of the Observed signal strength
In Negative Cases
True Negative
False Positive
In Positive Cases
False Negative
True Positive
C
B
A
Threshhold Criterion
100%
C
B
A
p(True Positives are found) or Sensitivity
0
p(False Positives are found) or 1- Specificity
100%

Another important measure can be the error rate vs iterations for different batch sizes. The error rate used can be the percentage of wrong classified samples. This metric is widely used in literature for ConvNets as it gives an insight of the state of training of the network. It is quite common to detect over-training. When this phenomenon occurs, the training error keeps decreasing over time, but the test error goes through a minimum and then starts increasing after a certain number of iteration

Whale Detection

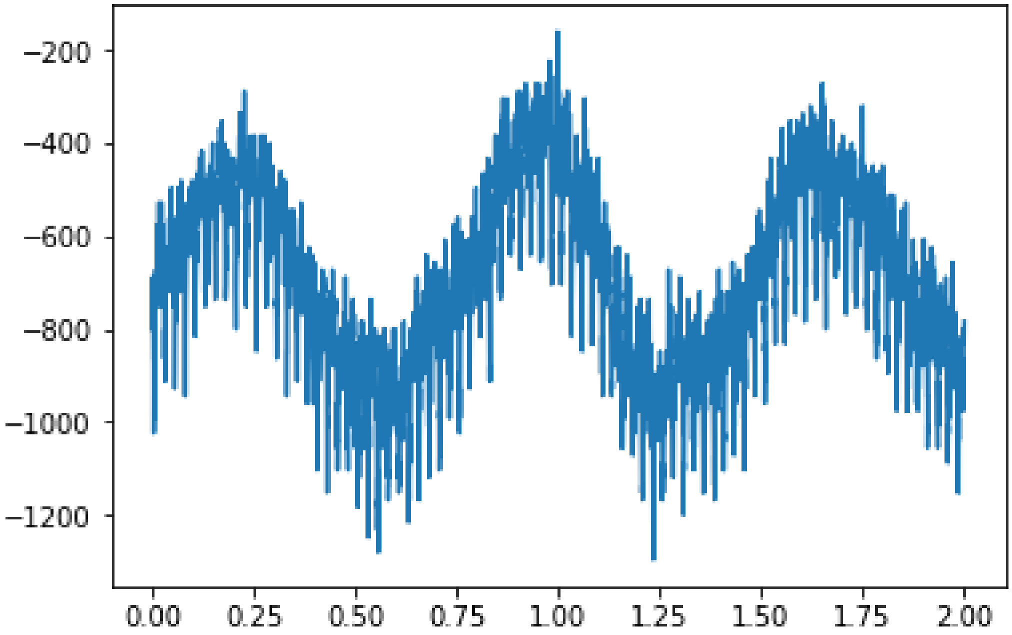
Data statistics

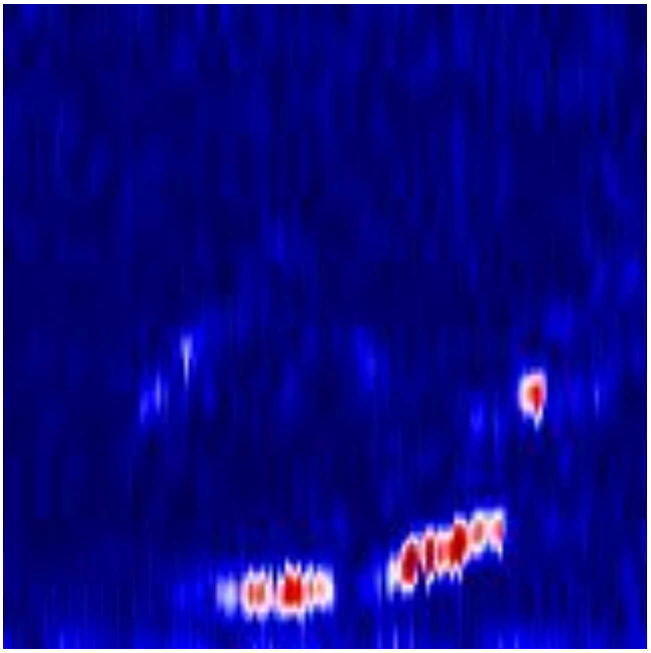
- Training and testing data has been taken from ICML 2013 competition held on Kaggle.com
- Training data consists of 47,842 AIFF files of 2 seconds recording using hydrophone.
- Testing data consists of 25,407 AIFF files of 2 seconds recording using hydrophone.

Conversion to wav file format

- It is much simple to convert the AIFF files to wave file format.
- As the project involves training from large numbers of files, one by one conversion of file is not an effective solution.
- Python script was written for carrying out the conversions using FFMPEG software.

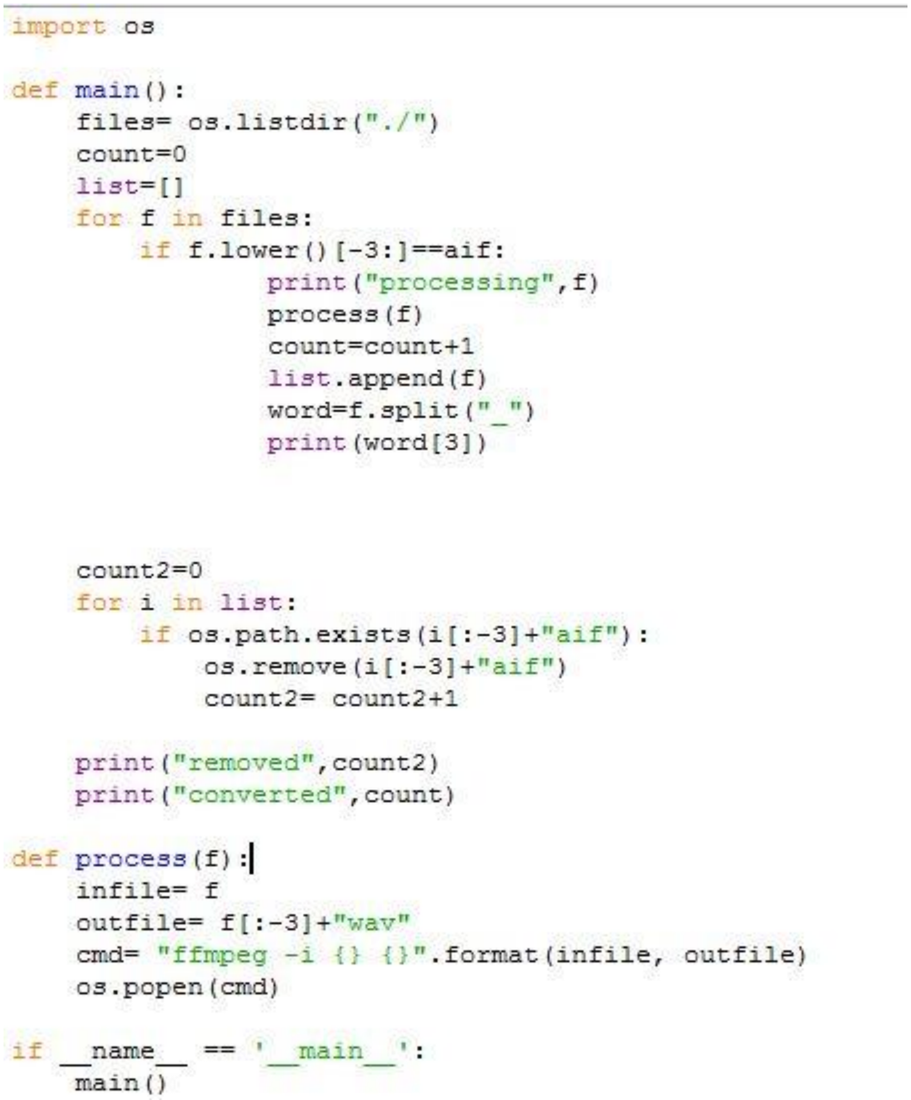
The following is a sample of recoded clip.





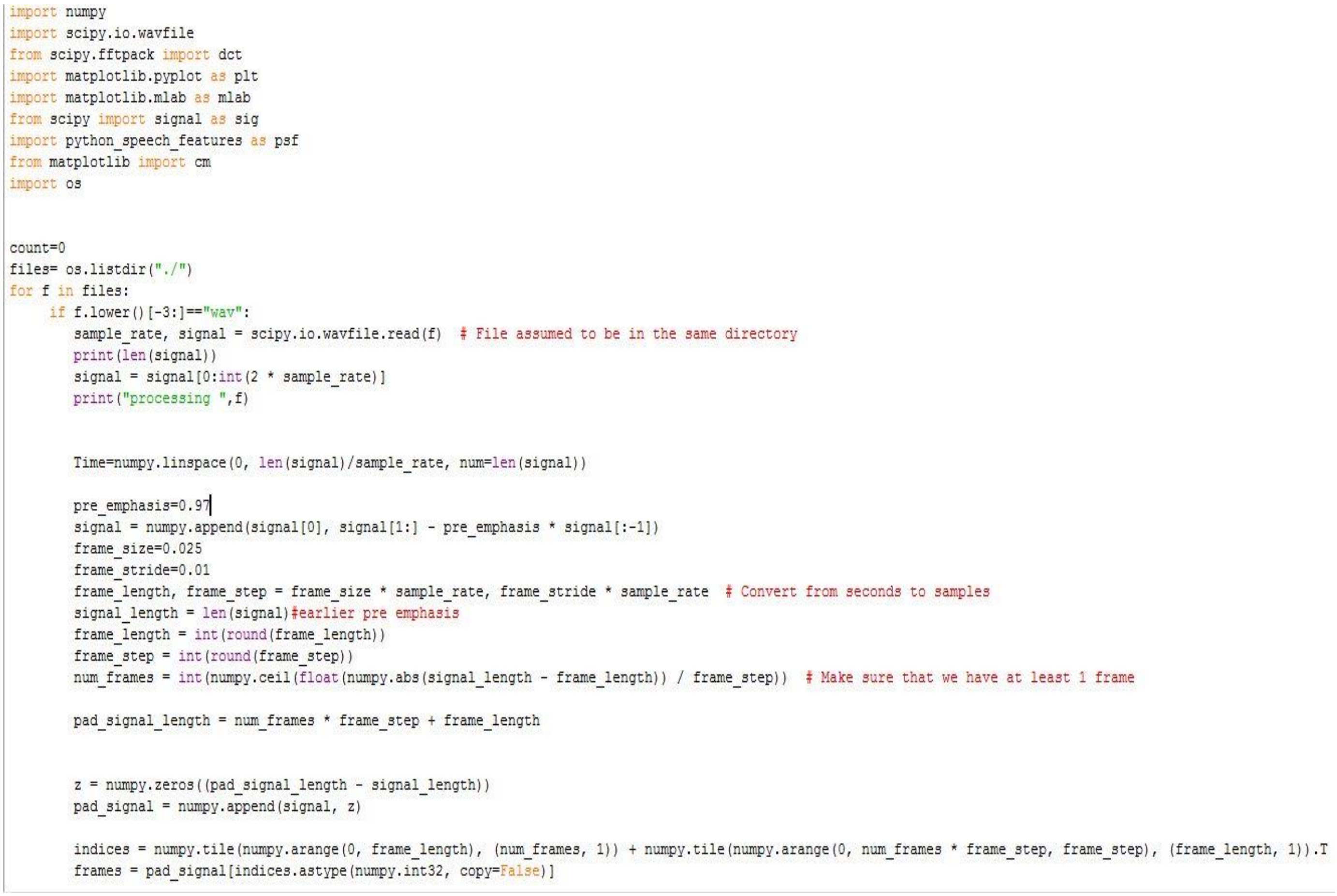
Spectrogram representing whale up call

Code for converting Audio Interchangeable Format file to wave file using FFMPEG

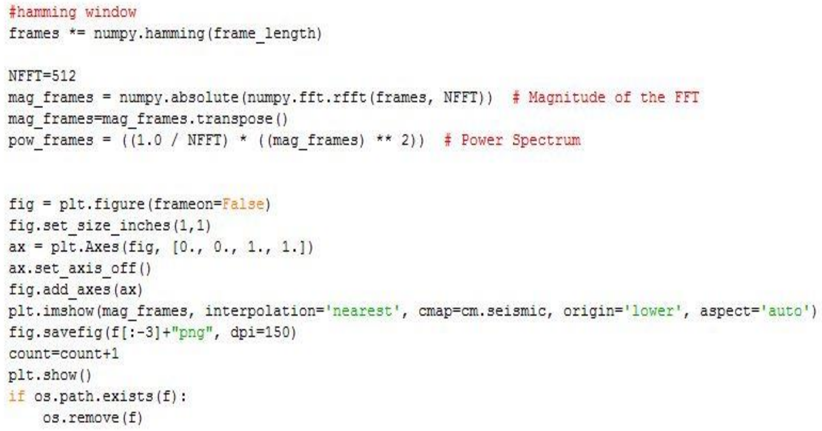


```
import os

def main():
    files= os.listdir("./")
    count=0
    list=[]
    for f in files:
        if f.lower()[-3:]==aif:
                print("processing",f)
                process(f)
                count=count+1
                list.append(f)
                word=f.split("_")
                print(word[3])



    count2=0
    for i in list:
        if os.path.exists(i[:-3]+"aif"):
            os.remove(i[:-3]+"aif")
            count2= count2+1

    print("removed",count2)
    print("converted",count)

def process(f):
    infile= f
    outfile= f[:-3]+"wav"
    cmd= "ffmpeg -i {} {}".format(infile, outfile)
    os.popen(cmd)

if __name__ == '__main__':
    main()
```

Plotting spectrogram



```python
import numpy
import scipy.io.wavfile
from scipy.fftpack import dct
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from scipy import signal as sig
import python_speech_features as psf
from matplotlib import cm
import os


count=0
files= os.listdir("./")
for f in files:
    if f.lower()[-3:]=="wav":
        sample_rate, signal = scipy.io.wavfile.read(f)  # File assumed to be in the same directory
        print(len(signal))
        signal = signal[0:int(2 * sample_rate)]
        print("processing ",f)


        Time=numpy.linspace(0, len(signal)/sample_rate, num=len(signal))

        pre_emphasis=0.97
        signal = numpy.append(signal[0], signal[1:] - pre_emphasis * signal[:-1])
        frame_size=0.025
        frame_stride=0.01
        frame_length, frame_step = frame_size * sample_rate, frame_stride * sample_rate  # Convert from seconds to samples
        signal_length = len(signal)#earlier pre emphasis
        frame_length = int(round(frame_length))
        frame_step = int(round(frame_step))
        num_frames = int(numpy.ceil(float(numpy.abs(signal_length - frame_length)) / frame_step))  # Make sure that we have at least 1 frame

        pad_signal_length = num_frames * frame_step + frame_length


        z = numpy.zeros((pad_signal_length - signal_length))
        pad_signal = numpy.append(signal, z)

        indices = numpy.tile(numpy.arange(0, frame_length), (num_frames, 1)) + numpy.tile(numpy.arange(0, num_frames * frame_step, frame_step), (frame_length, 1)).T
        frames = pad_signal[indices.astype(numpy.int32, copy=False)]
```



```python
        #hamming window
        frames *= numpy.hamming(frame_length)

        NFFT=512
        mag_frames = numpy.absolute(numpy.fft.rfft(frames, NFFT))  # Magnitude of the FFT
        mag_frames=mag_frames.transpose()
        pow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2))  # Power Spectrum


        fig = plt.figure(frameon=False)
        fig.set_size_inches(1,1)
        ax = plt.Axes(fig, [0., 0., 1., 1.])
        ax.set_axis_off()
        fig.add_axes(ax)
        plt.imshow(mag_frames, interpolation='nearest', cmap=cm.seismic, origin='lower', aspect='auto')
        fig.savefig(f[:-3]+"png", dpi=150)
        count=count+1
        plt.show()
        if os.path.exists(f):
            os.remove(f)
```

Benchmark

I will try to compare the performance of popular ConvNets (i.e. LeNet-5 proposed by Lecun or AlexNet, the winner of the 2010 and 2012 ImageNet Large Scale Visual Recognition Competition (ILSVRC) proposed by Krizhevsky) with the performance of the winning model of the competition which is based on Gradient Boosting and the Daniel Nouri's model based on Krizhevsky's 2012 ILSVRC ConvNet model , which first inspired this work. The Area Under the Curve (AUC) (see the Evaluation metrics section I.III) of these models in the public leaderboard was:

- SluiceBox: 0.98410 (1st position)
- Nouri: 0.98061 (6th position with 1/4 times the submission of the winner)

Nevertheless, I will not be able to compare the performance of my models to these results. The reason is that I do not have the test labels and also, the public leaderboard data test used is slightly different for each participant. I will try two different approaches:

1. assuming that there are enough complete data samples (train dataset), trying to increase the accuracy as much as possible

2. assuming the predictions generated by the winning model as test labels and them as reference to compare our model with Consequently, it will not be easy to truly compare the performance of this model with the ones in the competitions. Taking into consideration all the factors explained above, 0.95 AUC can be a really good value to achieve.

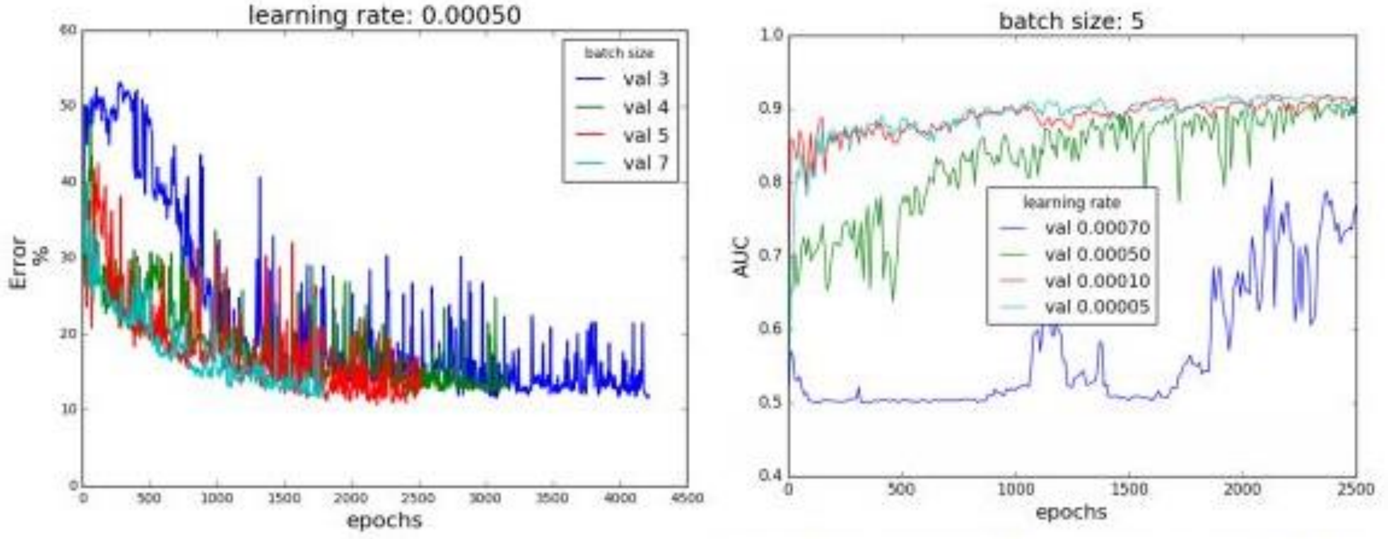
Model

- INPUT layer: 32x32 pixel image in gray scale represented with 8-bit number (0-255 levels).
- C1 - CONV layer: F = 5x5, S = 1, D = 6
- S2 - POOL layer: F = 2x2, S = 2, D = 6
- C3 - CONV layer: F = 5x5, S = 1, D = 16
- S4 - POOL layer: F = 2x2, S = 2, D = 16
- C5 - CONV layer: F = 5x5, S = 1, D = 120
- F5 - FC layer: neurons= 120 x 2, one per label (considering label 0 and label 1)

The training is performed using mini-batch gradient descent, which is a version of the true gradient descent (combines batch and stochastic gradient descent), used when the data amount is quite high. It iterates over batches of n samples in order to approach the minimum of the cost function step by step (epochs). Mini-batch gradient descent reduces the variance of the parameter updates, which can lead to more stable convergence. It also can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient with respect to a mini-batch very efficient. The gradient descent main parameter is the learning rate (α). The learning rate expresses the speed of convergence of the gradient descent. Large learning rates lead to faster convergence but it may miss the minimum and not converge properly. Low learning rates lead to a better convergence point, but it is slower, requiring more steps and more memory allocation. A good compromise is to use a decaying learning rate: high values for the first epochs to accelerate the initial convergence and then smaller ones to slow it down. Regularization is a common way to prevent over-fitting and the most used method is L2 regularization. This type of regularization penalizes the square magnitude of all parameters, adding the term ½*$\lambda\omega^2$ to the prediction, for λ the regularization strength. In this work L2 regularization will be used to control the over-fitting. The network has been implemented making use of Tensorflow.

Refinement

The process followed has been one the iteration over different parameters to obtain the best combination. Select batch-size in order to select the proper batch size simulations have been performed for different learning rates. After many simulations, I have observed that there must be a trade-off between batch-size and instability. The bigger the batch, more stable the curves, but the poorer the performance since the number of epochs is smaller. The batch size must be big enough to provide a less noisy curve but small enough to give good values of prediction. From figure below it seems that a good compromise value is a batch size of 5.

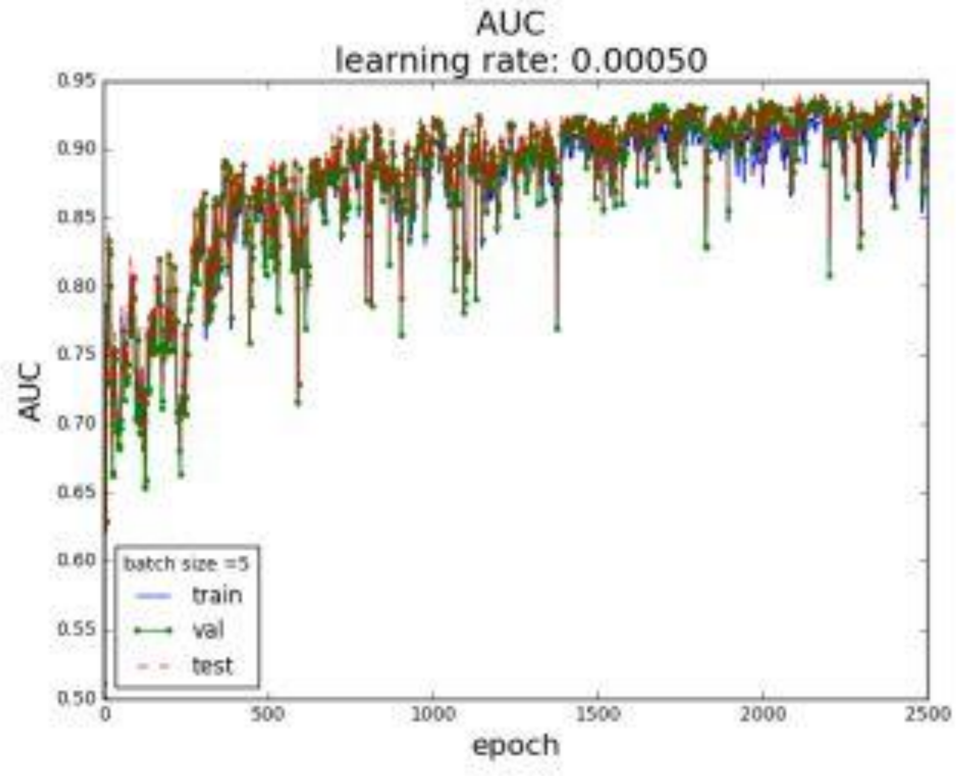


(a) Error curve for the validation dataset, for different batch sizes and a learning rate α=0.0005

(b) AUC for the validation dataset, for different learning rates and a batch size = 5

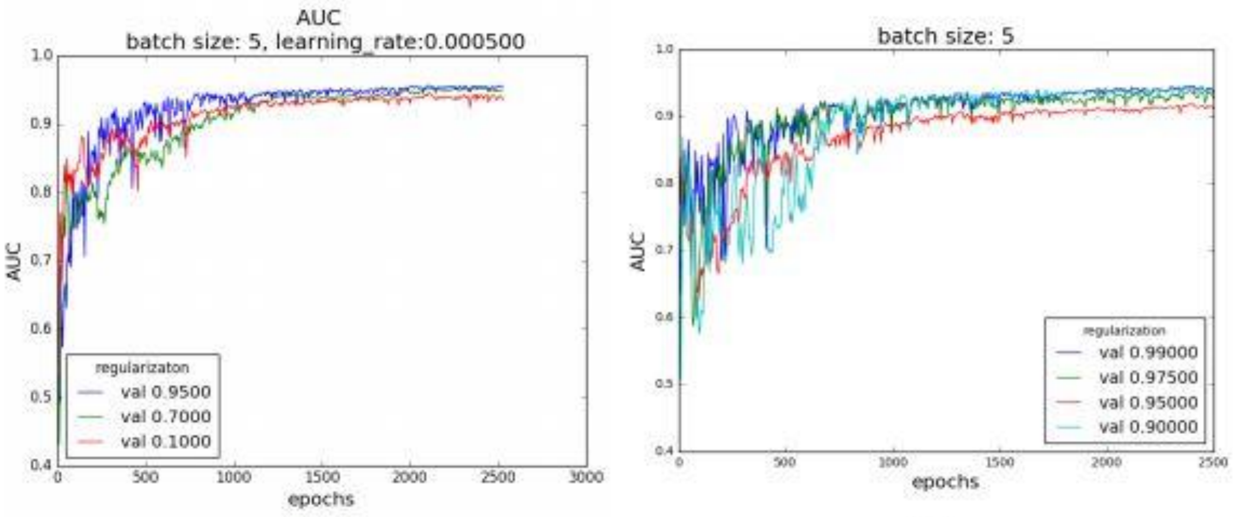
Select learning rate

Secondly, for a fixed batch size, the cost has been calculated for different learning rates. Learning rates too big may not find the minimum and converge too fast. Small learning rate may be too slow and not fast enough for a small dataset like the one in this work. A compromise value must be chosen, showing a slope good that guarantees convergence. Figure 12b shows that for the selected batch size of 5, a learning rate α of 0.0005 seems like a good value. Figure 13 shows that there is not over-fitting, since there is no gap between the training curve and the validation and test curves.



Select regularization

Finally, for fixed batch size and learning rate, the AUC has been calculated for regularization parameters. Regularization is a good way to limit over-fitting, allowing the model to generalize better. Figure below shows the AUC curve for different values of the regularization parameters. Figure shows that a good value for the regularization parameter can is near 0.95, and figure 14b shows that for the selected batch size of 5 and learning rate of 0.0005 a good regularization value is 0.97, since it allows to reach a greater AUC.



(a)

(b)

Fig : AUC in the validation dataset for different regularization values, batch size = 5 and learning rate α=0.0005

Results

Model Evaluation and Validation

The model used is the LeNet-5. Figure below shows the AUC and the error for different epochs, achieving 0.944 AUC in the test set (and up to 0.958 depending on the test split) and an error of 12% when reaching 2500 epochs, batch size = 5, learning rate α=0.0005 and regularization parameter = 0.97. Figure shows there is no over-fitting, since validation and test curve follow the training curve. This is clearly a good result given the limitations related to data in this work previously explained in section II.I and is very close to the objective of 0.95 AUC.

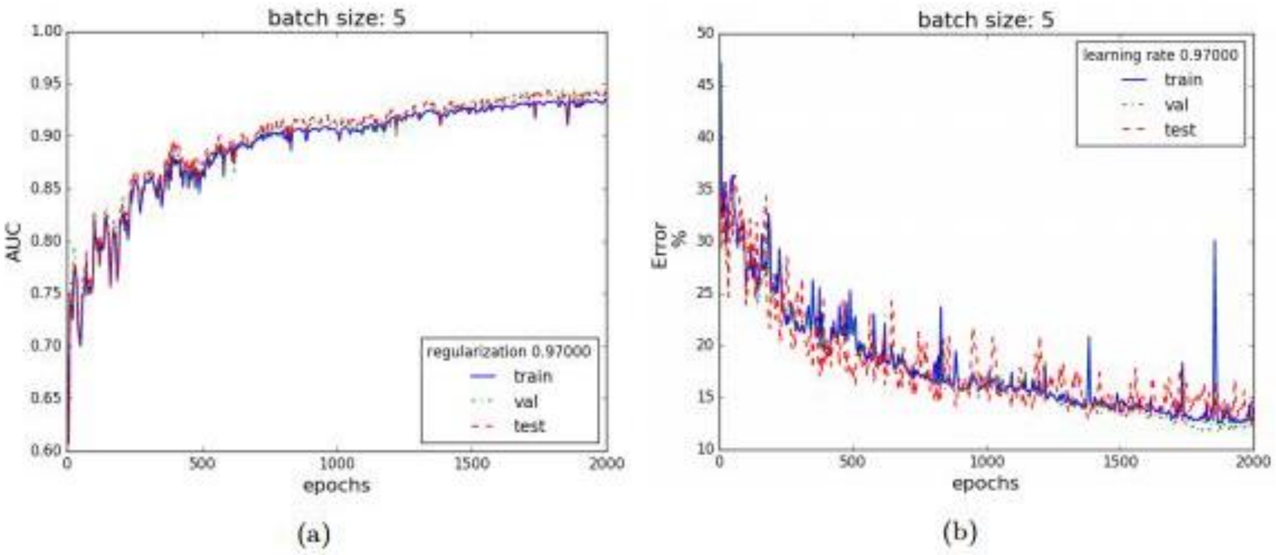


Fig : (a) AUC and (b) error for the train validation and test dataset, for the final model batch size = 5, learning rate α=0.0005and regularization parameter = 0.97

One problem I faced when training the model was the limitation in terms of data samples for training which reduced the maximum AUC obtained. In order to solve it, labels for the test dataset have been obtained using the evaluations from the winning model as true labels (SluiceBox: 0.98410 AUC). This has allowed o increase the volume of data samples available to train the model and hence, increase its performance. I have taken this decision after taking into consideration the processed of obtaining the samples. The sounds were firstly recorded when a buoy detected an up-all in the area. Afterwards, samples were labeled by human ear. This leads to a lot of mislabeled samples, and gives the intuition that using the labels from the prediction of a good model as test dataset, freeing samples for training, can be more good than harm. This has allowed to extend the available data for training (from around 8000 to 12500 samples from the balanced dataset of 14000 samples) and have available 54503 samples from the unbalanced testing dataset for testing, though it was used 10000 random samples for computational issues. This dataset was not artificially balanced just to try how the training would perform in a real world with little samples from one label when compared to the other.

Justification

This work has shown the use of simple ConvNet for audio recognition, obtaining good performance with an AUC of up to 0.958, whereas more complex models as SluiceBox obtained 0.98410 AUC and Nuori 0.98061. One of the advantages of simpler networks relies upon the reduction of time required for training the model, resulting in fewer computer requirements. With a vast dataset, the system would increase significantly the performance and equal those with more complex structure, making it a simple, yet robust choice.

Conclusion

Free form visualization

In this section, I will try to give an insight of the network, presenting, as it is frequently done, the characteristics of the first layer. Figure below it is presented the evolution of the filters of the first CONV layer (F=5x5) with depth 6 (D=6). These filters are the ones convoluting and sweeping through the image and contain the weights fond the CONV layer neurons. It is clear how, due to backpropagation and minimization of the cost function, weights are changed and they evolve to new and more adequate values for prediction. In fact, the six filters start very undefined and they adapt to their final value. Figure below shows the evolution of the biases with the number of epochs and how they tend to stabilize. These biases are the biases of the CONV layer and they change to minimize the cost function.

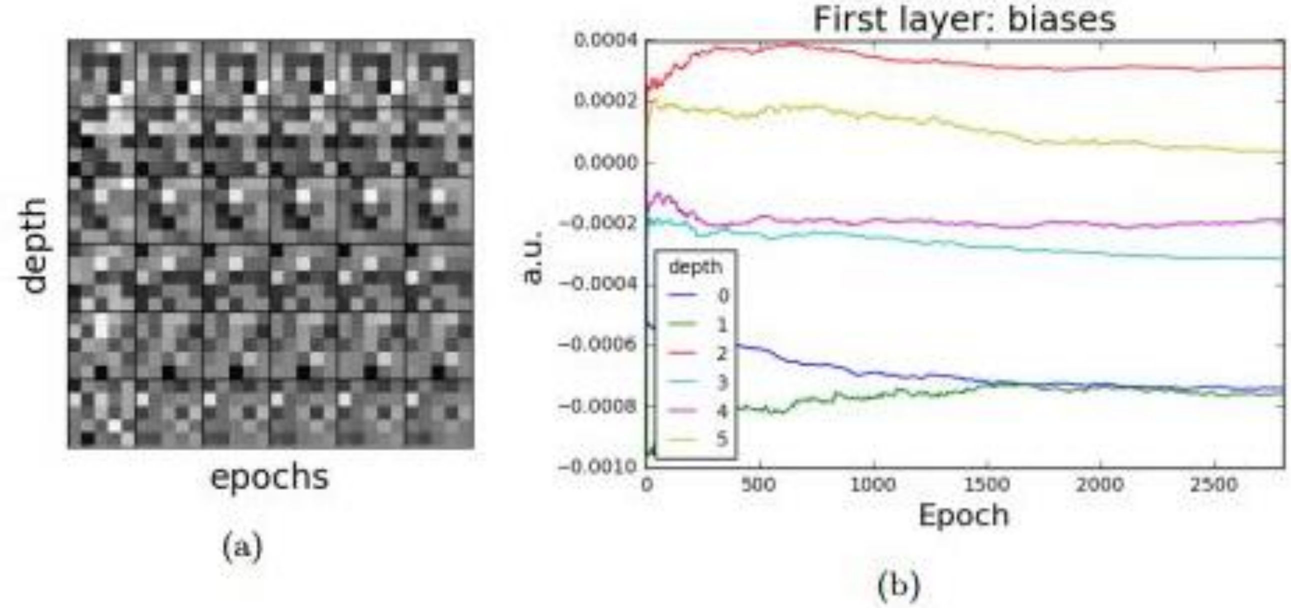


Fig: (a) Weights and (b) biases of the first layer with 6 filters of depth every 500 epochs for batch size=5, learning rate α=0.0005and regularization parameter= 0.97

Finally, figure below shows the output every 500 epochs for some samples. It is not clear, but in some of the squares it can be seen the shape of the up-call.

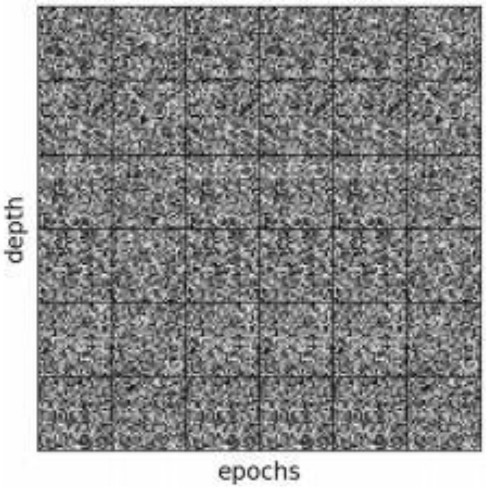


fig: Activation of the first layer with 6 filters of depth every 500 epochs for batch size=5 and learning rate α=0.0005

Reflection

In this project, it is presented how simple ConvNets can be used to label data from the image of the spectrogram, as it in many classification problems. More complex networks may achieve better results, but for an image of 32x32px, I prefer to take the strategy: the simpler, the better. In this case, the use of LeNet-5 has given a 0.44-0.958 AUC, which is really good given the low complexity of the network.

Improvement

this work, the problem is to recognize a simple and very specific spectrogram pattern from in a form of an image. A good rule of thumb for this is the simpler, the better and LeNet-5 achieves good performance with a simple structure. The complexity of the model should not be the focus, since like in any Machine Learning problem, the most important factor is the amount of data. Collecting more samples and extending the training would make LeNet-5 a good competitor against other models achieving better performance with over-complicated models. Once resolved the issue with the amount of data, more complex models are good start points to increase the performance and achieve better results and compare the trade-off between complexity with results. Good examples of models to try are AlexNet or GoogLeNet. Dropout has not been applied in this work, since the network has lots of max-pooling layers and regularization, and in these cases, the use of dropout is not clear. Introducing dropout could be a good option to try. Also, methods to make faster training, as Batch Normalization may lead to better results.

Alex Net

AlexNet is the winning solution of IMAGENET Challenge 2012. This is one of the most reputed computer vision challenge and 2012 was the first time that a deep learning network was used for solving this problem.

Also, this resulted in a significantly better result as compared to previous solutions.

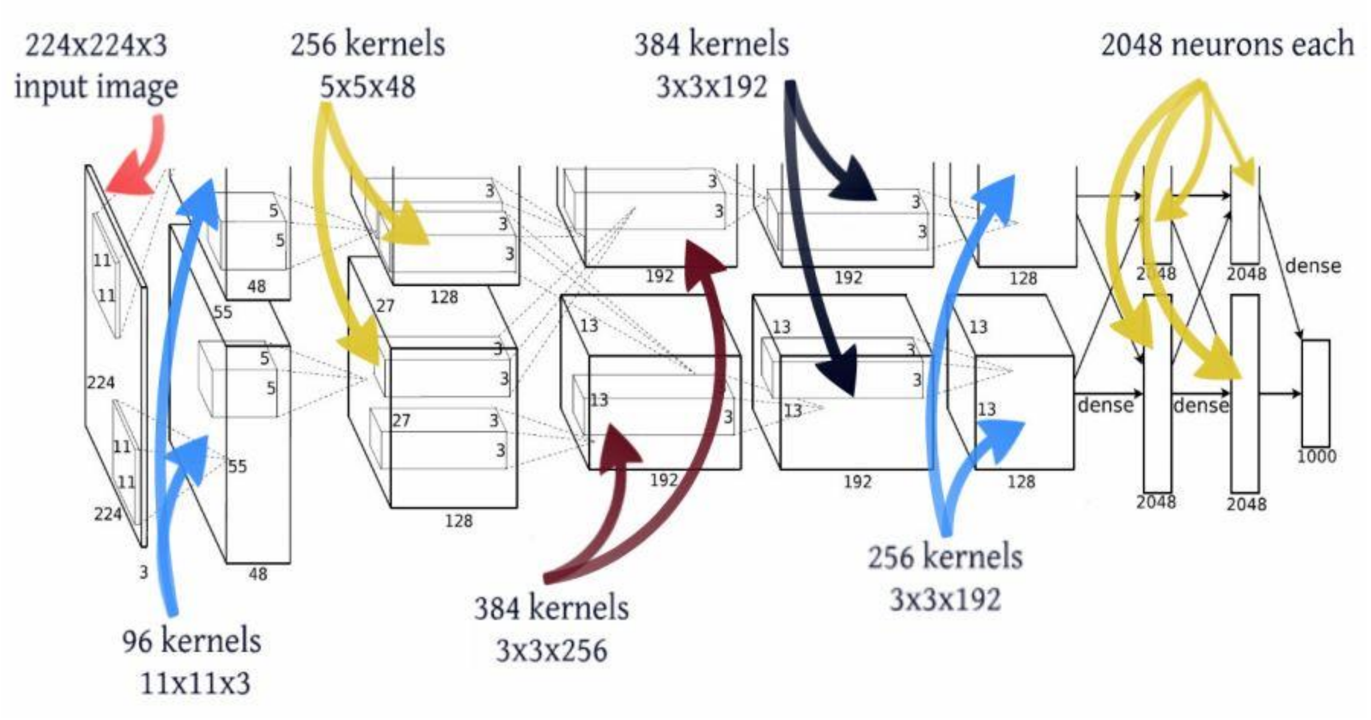
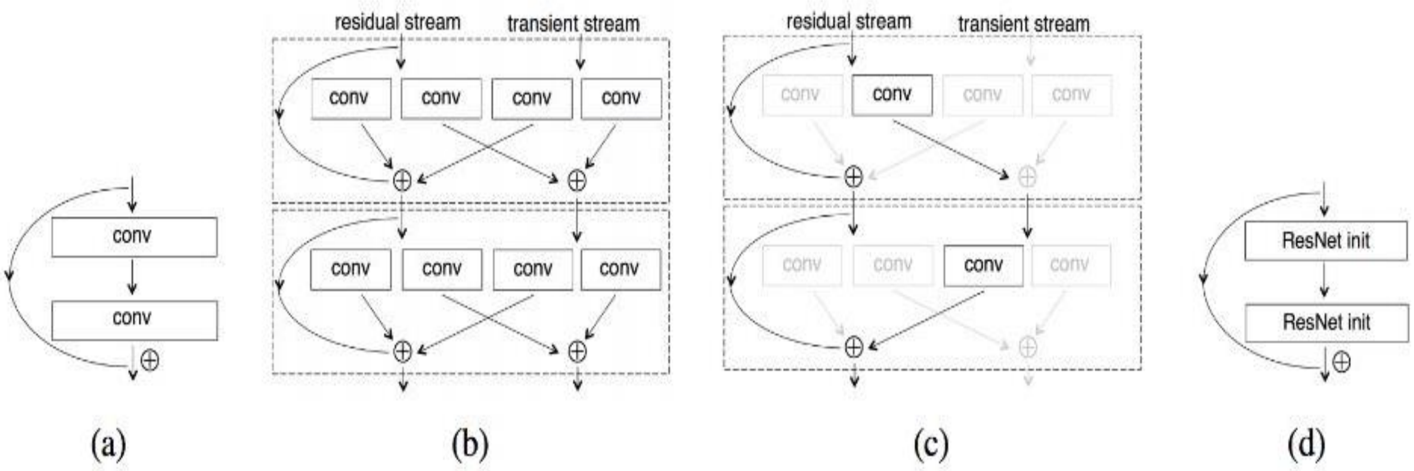


Fig : Model Architecture of Alex Net

ResNets (Residual Neural Networks)

Residual neural networks, or ResNets (Deep Residual Learning for Image Recognition), are a technique Microsoft introduced in 2015. The ResNet technique allows deeper neural networks to be effectively trained. ResNets won the ImageNet competition in December with a **3.57%**error score. Recently, researchers have published several papers augmenting the ResNet model with some interesting improvements.

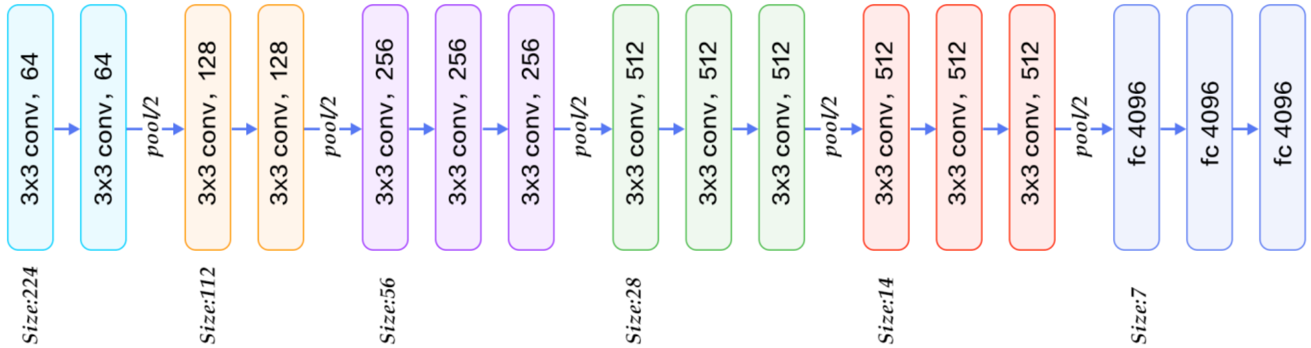
ResNets tweak the mathematical formula for a deep neural network. They tweak the layers' equations to introduce identity connections between them. The identity function is simply **id(x) = x**; given an input **x** it returns the same value **x** as output. A layer in a traditional neural network learns to calculate a function **y = f(x)**. A residual neural network layer approximately calculates **y = f(x) + id(x) = f(x) + x**. Identity connections enable the layers to learn incremental, or residual, representations. The layers can start as the identity function and gradually transform to be more complex. Such evolution occurs if the parameters for the **f(x)** part begin at or near **zero**.



ResNet in ResNet fractal structure.

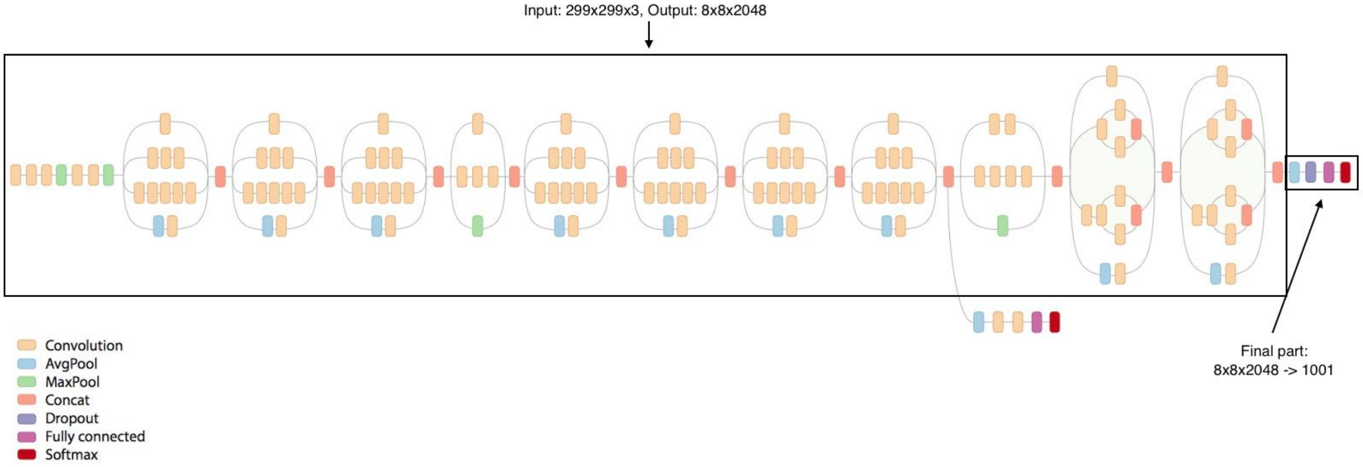
VGG 16 (Oxford net)

Developed by researchers of Visual Geometry Group (VGG) at Department of Engineering Science, University of Oxford.



This is the Keras model of the 16-layer network used by the VGG team in the ILSVRC-2014 competition.

Inception



The Inception v3 model takes weeks to train on a monster computer with 8 Tesla K40 GPUs and probably costing $30,000 so it is impossible to train it on an ordinary PC. We will instead download the pre-trained Inception model and use it to classify images.

The Inception v3 model has nearly 25 million parameters and uses 5 billion multiply-add operations for classifying a single image. On a modern PC without a GPU this can be done in a fraction of a second per image.

Future Scope

A ship can be identified from a submarine from a hydrophone data using signal processing as the noises produced lies in different frequency ranges but distinguishing Scorpene class submarine from a Khalid class is a complex task as they lie in same frequency range. Spectrograms are used as a tool for audio visualisation but with deep learning they can be used to map frequency-time pattern to different classes. This paper uses Convolutional Neural Network(CNN) to classify hydrophone data for whale calls. CNN has outperformed every other machine learning algorithm for image recognition task. A Spectrogram was plotted corresponding to each data sample and further image processing was done. These spectrograms were fed as input samples to a three-layered CNN. Due to limited availability of open source hydrophone data, classification was done on bio acoustics hydrophone data instead of naval data. However, the results obtained shows that a CNN model can be used for identifying signatures of naval vessels and detecting underwater activities. This paper focuses on binary classification, due to non-availability of multi class classification dataset.

References

[1] Chao Li, Xiaokong Ma, Bing Jiang, Xiangang Li, Xuewei Zhang, Xiao Liu, Ying Cao, Ajay Kannan, Zhenyao Zhu (Baidu Inc.), Deep Speaker: an End-to-End Neural Speaker Embedding System

[2] Navinda Kottege, Raja Jurdak, Frederieke Kroon, Dean Jones, Classification of Underwater Broadband Bio-acoustics Using Spectro-Temporal Features

[3] Urmila Shrawankar, Dr. Vilas ThakareTECHNIQUES FOR FEATURE EXTRACTION IN SPEECH RECOGNITION SYSTEM : A COMPARATIVE STUDY

[4] Ignacio Moreno, Samy Bengio, Noam Shazeer, Google, End-to-End Text-Dependent Speaker Verification

[5] Manish P. Kesarkar, FEATURE EXTRACTION FOR SPEECH RECOGNITON

[6] Daniel Nouri Blog post [http://danielnouri.org]

[7] Kaggle [www.kaggle.com]

[8] Siraj Raval [https://github.com/llSourcell]

[9] **Haytham Fayek, Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between**